CHIP SPECIAL

19,– DM

FISCHER-TECHNIK UND COMPUTER

Roboter in drei Ausbaustufen ● Messen mit Sensoren ● Bilder erkennen ● Motoren und Magnete steuern ● Regelkreise ● Ein Buggy erkennt die Umwelt



Programme für

- **Atari ST**
- **Commodore 64/128**
- **Schneider CPC**
- **IBM PC und Kompatible**

© CHIP Communications GmbH

aktuelle Sonderhefte von CHIP gibt es auf www.chip-kiosk.de

ECHTE VORBILDER

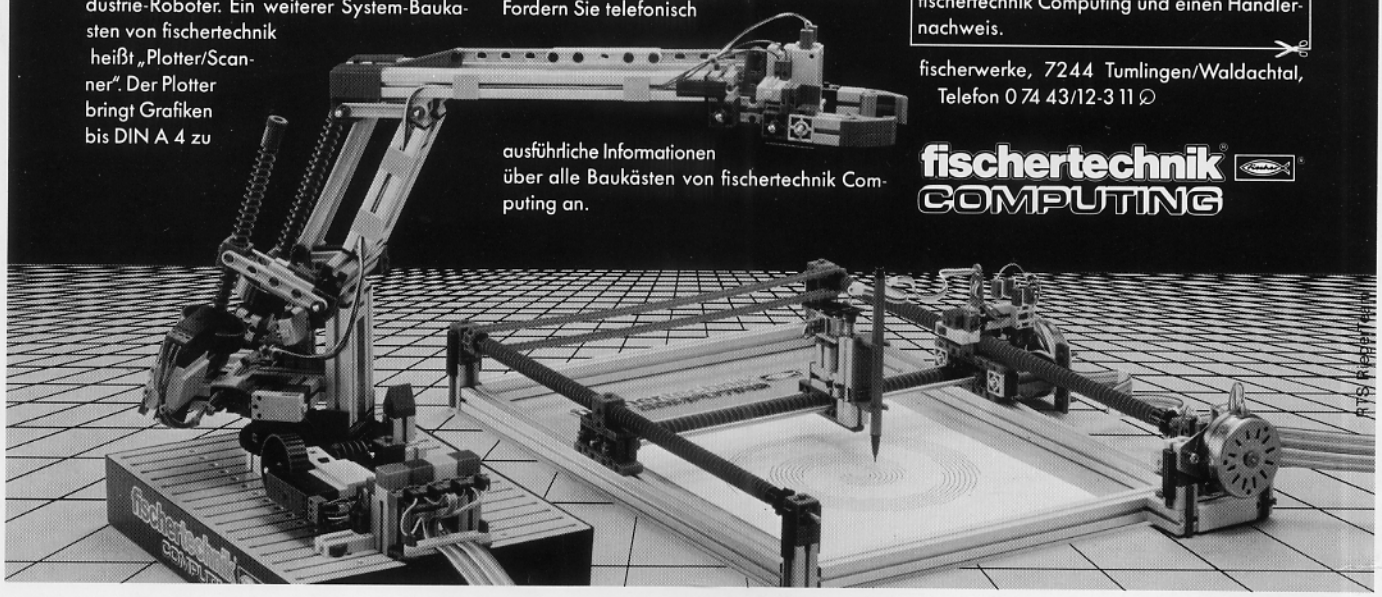
Setzen Sie sich mit einem wirklichkeitsnahen Roboter auseinander. Programmieren und steuern Sie Arbeitsschritte und Bewegungen. Der „Trainings-Roboter" ist eine dreiachsige Knickarm-Konstruktion, genau wie echte Industrie-Roboter. Ein weiterer System-Baukasten von fischertechnik heißt „Plotter/Scanner". Der Plotter bringt Grafiken bis DIN A 4 zu Papier, mit dem Scanner werden Vorlagen digital abgetastet. Beide Baukästen enthalten Software und Programmieranleitung. Zum Anschluß an die meisten Home- und Personalcomputer gibt es Interfaces von fischertechnik. Fordern Sie telefonisch ausführliche Informationen über alle Baukästen von fischertechnik Computing an.

Oder Coupon ausschneiden, auf eine Postkarte kleben (Absender nicht vergessen!) und an untenstehende Adresse senden:

Bitte schicken Sie mir Ihren Farbprospekt über fischertechnik Computing und einen Händlernachweis.

fischerwerke, 7244 Tumlingen/Waldachtal, Telefon 0 74 43/12-3 11

fischertechnik COMPUTING



RTS Friedrichsen

CHIP SPECIAL 28,– DM

ATARI

Programmer's Guide für Atari ST Pascal plus 2.0

- Systemroutinen unter Pascal plus: XBIOS, BIOS, GEMDOS, VDI, AES
- Weiterführende Dialog- und Windowtechniken
- Einfaches Event-Management
- Professionelle Grafikprogrammierung
- Perfekte GEM-Einbindung
- Universelle Bibliotheksmodule

CHIP SPECIAL

BESSER PROGRAMMIEREN

AMIGA

Amiga-Handbuch Grafik-Anwendungen

Drittes Amiga-SPECIAL

AMIGA 500
AMIGA 1000
AMIGA 2000

Grafikerzeugung auf dem Amiga · Die großen Möglichkeiten · Digitizer und Genlock-Interface · Lightpen · Grafikhardware Plotten und Drucken · Grafik-Software und IFF-Standard · Grafik-Programmierung für Einsteiger und Fortgeschrittene · Amiga intern

Alle Grafikprogramm-Beispiele auf Diskette erhältlich

28,– DM · 82005/87006

Der Computer spielt mit



Wenn ein Computer Mühle spielt . . .

Sogar das Mühlespielen wird jetzt von Elektronengehirnen besorgt: Auf der Computer-Messe Systems 87 stellen Fachverlage Software-Programme vor, mit denen der Besitzer ein Mühlespiel oder eine Modelleisenbahn steuern kann.

Zu den Kuriosa rund um die Computerschau zählt auch das Bildtelefon, das die Bundespost in einer Sonderschau vorführt. Frühestens 1989 soll die Anlage aus Telefonapparat, Monitor und Kamera für etwa 2000 Mark im Handel erhältlich sein.

Sie ist vor allem für den Gebrauch in Betrieben gedacht: Wenn mehrere Geräte zusammengeschlossen sind, lassen sich sogar ganze Videokonferenzen einberufen.

mm

Auf der Systems gesehen: Otto Mett (23) am Computer-Mühlespiel.

Am dritten Messetag, während der Computer-Messe Systems '87, erfuhren die rund 253 000 Leser der Münchner Abendzeitung, was die Mitglieder der Redaktion CHIP-SPECIAL längst wußten: "Sogar das Mühlespielen wird jetzt vom Elektronengehirn besorgt!" Besucht wurde diese alle zwei Jahre stattfindende Schau von 150 000 Menschen.

Was AZ-Redakteur Ludwig Hübl nur von den Schachweltmeistern Kasparow und Karpow kannte, hatte das Programmierteam der SPECIAL-Redaktion am Vogel-Stand mit so viel Grips wie nötig und so wenig Technik wie möglich verwirklicht: Das von SPECIAL-Mitarbeiter Reinhard Tchorz in der Programmiersprache Turbo-Pascal entwickelte, von SPECIAL-Team-Mitglied Bernd Edlinger an die Schnittstelle des Fischertechnik-Computing-Systems angepaßte Programm ist in der Lage, den von Gerhard Bader, dem Autor dieser Ausgabe, konstruierten Portalkran so zu steuern, daß der sensible Greifer jeden mit Tastatur eingegebenen Zug - der sich am Bildschirm verfolgen läßt - exakt auszuführen vermag. Die Logik des Mühlespiels unterscheidet sich nicht von der eines Schachprogramms.

Zweifellos sind Bewegungsmodelle der aktuelle Hit bei Computer-Freaks. Der Sprung aus der Bildschirmebene in die realistische dritte Dimension ist ebenso reizvoll wie lehrreich. Das Fischer-Technik-Interface schafft die Verbindung zwischen Computersystemen und den Schaltern, Sensoren, Motoren und Magneten der Fischertechnik-Baukästen. Was sich damit alles verwirklichen läßt, wird in dieser Ausgabe anhand von nachvollziehbaren Beispielen und Programmen aufgezeigt, die vier Rechner unterstützen (ST, C64/128, CPC und IBM-PC). Zu diesen faszinierenden Beipielen gehört das erwähnte Mühlespiel gegen den Computer, das aufgrund seiner Unschlagbarkeit so manchem unserer Team-Mitglieder schlaflose Nächte bereitete.

Daß Ihnen das Gespann 'Computer und Fischertechnik' nicht nur schlaflose Zeiten, sondern mit Hilfe dieser Ausgabe viel Spaß und Grundwissen vermittelt, das wünscht Ihnen im Auftrag des SPECIAL-Teams

Ihre Redaktion CHIP-SPECIAL

Armin Schwarz

RTS RiegerTeam

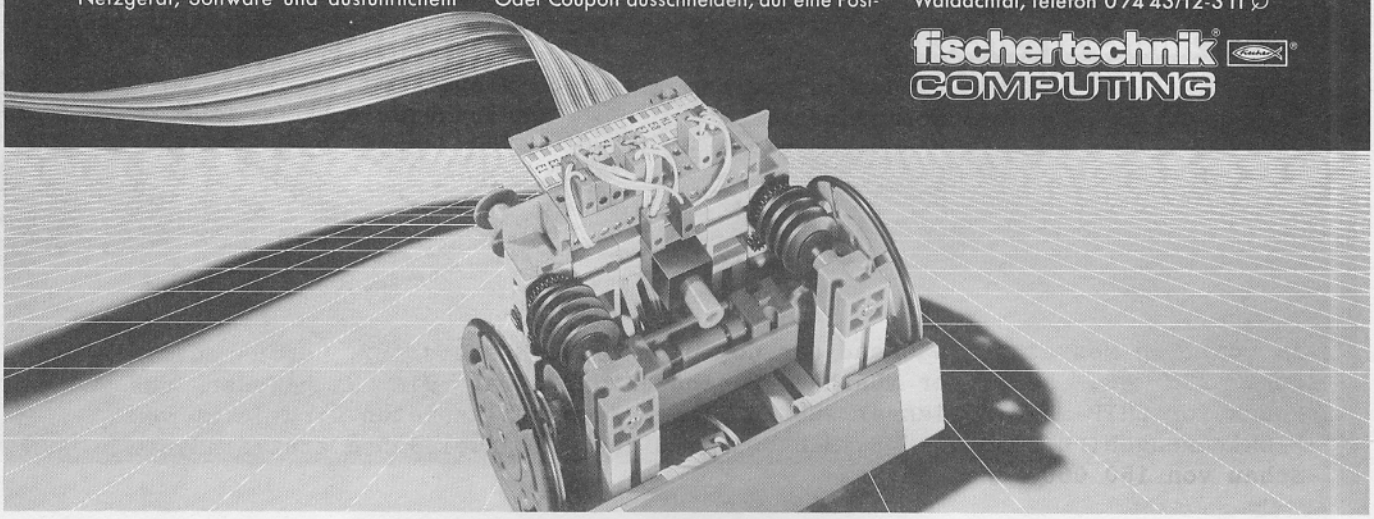
KEINE EXPERIMENTE

NEU Steigen Sie ein in die faszinierende Welt des Experimentierens. Erleben Sie, wie Motoren über Computer angesteuert, wie Fahrroboter programmiert oder Meßdaten grafisch dargestellt werden. Mit Computing Experimental, dem System-Baukasten von fischertechnik. Komplett mit Interface, Netzgerät, Software und ausführlichem Experimentierhandbuch. Sie können insgesamt 16 verschiedene Modelle zu den Themen Messen, Steuern, Regeln und Robotik zusammenbauen. Also keine Experimente: Computing Experimental. Fordern Sie telefonisch ausführliche Informationen über alle Baukästen von fischertechnik Computing an.
Oder Coupon ausschneiden, auf eine Postkarte kleben (Absender nicht vergessen!) und an untenstehende Adresse senden:

Bitte schicken Sie mir Ihren Farbprospekt über fischertechnik Computing und einen Händlernachweis.

fischerwerke, 7244 Tumlingen/
Waldachtal, Telefon 0 74 43/12-3 11

fischertechnik COMPUTING



CHIP SPECIAL 82005/87012 28,– DM

Ausgabe 7

PASCAL

Turbo-Pascal
Programmier-Praxis

- Textanalyse und -verarbeitung
- Dynamische Strings
- HGC-Toolbox
- Bausatz Editor
- Präsentationsgrafik
- Automatisches Stichwortverzeichnis
- Neue Benutzeroberfläche – selbst gestaltet

Alles über Turbo-Pascal 4.0

Sämtliche Programme für alle MS-DOS- und CP/M-Computer auf Datenträger erhältlich

CHIP SPECIAL 28,– DM

APPLE GS

APPLE

Was der Apple II GS besser kann

- Die CPU 65816 intern
- Geheimnisse der Toolbox
- Der neue Ensonique Musik-Chip
- Die leistungsfähige Interruptstruktur
- Der Desktop-Bus
- APW-Shell für C und Pascal
- TML-Pascal für Profis
- GS-Programme selbstgemacht
- Midi mit GS und Synthesizer
- GS-Hard- und Software

Alle Programme auf Diskette erhältlich

Inhaltsverzeichnis

Editorial	3	Der Computer spielt mit
Schlaues Gespann	7	Computer lernen sehen und fühlen
	10	Schaltungen zur Aus- und Eingabe
Steuern mit Licht	13	Wir steuern ein Transportband
Regeln Schritt für Schritt	19	Grundlagen der digitalen Regelung
	26	Dreipunkt-Regelung
Wenn sich die Zustände ändern	29	Nachführ-Regelung
	32	Niveauregelung bei bewegten Flüssigkeiten
Bewegungsabläufe kontrollieren	34	Interface-System Bert
	34	Mechanische Antriebe positionieren
Tempo im Griff	37	Geschwindigkeits-Steuerung
	41	Positionierung mit Codierung
	45	Die Bert-Kommandos für den Schrittmotor
Grafik mit Pfiff	47	Gerätesimulation mit dem Plotter
	52	HP-GL-Simulation in BASIC
Turbo-Pascal steuert Interface	66	Roboter-Steuerung
Rekursionen und Bewegungen	70	Mühle gegen den Computer
	71	Interface-Steuerung unter Turbo-Pascal
	77	Mühle Algorithmen in Turbo-Pascal
	90	Roboter-Steuerung mit Turbo-Pascal
Umbau und Alternativen	93	So koppeln Sie Interfaces
Nützliches und Wissenswertes	97	Bestellkarten
	99	Impressum

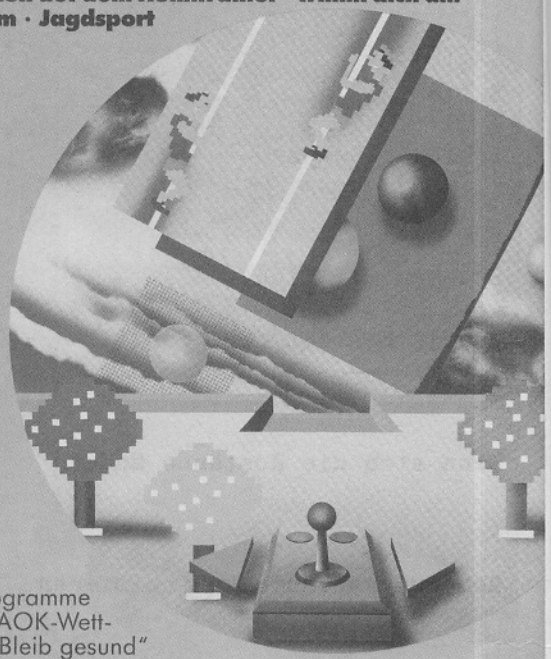
In dieser Ausgabe

Tennis	**Boris**
Werfen	**Frisbee für zwei**
Fitneß-Training	**Squash**
Boxsport	**Ring frei**
Siegerprogramm	**Bleib' fit am Bildschirm**
Radrennen	**Tour de France**
Bakterienjagd	**Karlchen, der Antikörper**
Sportspiel	**Athlet**
Golf	**Bernhard**
Tanzsport	**Tanz' Dich fit**
Waldlauf	**Der Aktivpfad**
Geschicklichkeits-Training	**AOK-Jumpman** **Krankenschwester**
Gedächtnis-Training	**Töne merken**
Gymnastik	**Vorturner**
Kegeln	**Gut Holz**
Dreikampfsport	**Joggen, Armbrustschießen, Radrennen**
Zielschießen	**Duck Shoot**
Werkzeug	**Programmierhilfe-Karte** **Fehlerkiller** **Hex-Lader**
Rudern	**Bootfahren**

CHIP-SPECIAL 81890/86003 · 24,– DM

Die schönsten Sportspiele

Tennis · Golf · Squash · Kegeln · Boxen · Jogging · Wettrennen auf dem Heimtrainer · Trimm dich am Bildschirm · Jagdsport



Siegerprogramme aus dem AOK-Wettbewerb „Bleib gesund"

Alle Programme für C 64/128 auf Datenträger erhältlich

In dieser Ausgabe

Serviceteil	**Programmierhilfe**
Tips und Tricks	**Umbau des 600 XL zum 800 XL**
	Vier neue Funktionstasten
	Programmieren des ANTIC-CHIP's
	Wichtige Speicherstellen
	Patches für DOS 2.0
	Programmprotector
	Hyper DOS II
	ROM-Fehler im Atari XL
	Schnelle Player/Missiles
	Quick Loader
	Super Load
	B-Directory
	Pfeil-Cursor
Utility	**ROMRAM**
	Freie Sektoren auslesen
	Move
	Tastendruck genügt
	Data-Zeilen-Generator
	File Mapper
	Autorun
	Keyboard
	Disketten-Name
	Menü-Dos
	Color-Controler
Werkzeug	**BASIC-Manager**
Fenstertechnik	**Window-Handler**
Hardcopy	**Draw-Pen**
	Graphics 9
Drucken	**Hardcopy**
Schnittstelle	**RS232-Treiber**
Zeit	**Digital-Uhr**
Spekulation	**Aktienanalyse**
Grafik	**Plotprint**
	Lucas
Maßeinheiten	**Umwandlung**
Musik	**Minicomposer**
	RAM-Sound
Spiel	**Danger-Snake**
Geschicklichkeit	**Gumfi**
Kartenspiel	**Mau-Mau**
Actionspiel	**Starmission**
Geschicklichkeit	**Mausi**
	Crackout
	Helium
Mathematik	**Multi-Regression**
Schnittstelle	**Interface-Tool**
Grafik	**Grafik-Show**
CHIP-SPECIAL-Serviceteil	**Große PEEK- und POKE-Liste**

Außerdem stellen wir in dieser Ausgabe unter der Rubrik „Programme per Post" weitere zwanzig Programme vor, die Sie per Bestellkarte beziehen können.

CHIP SPECIAL 18,– DM

ATARI

The Best of Atari XL/XE

Das vierte XL-SPECIAL

22 Listings zum Abtippen ● Weitere 20 Programme nur auf Datenträger erhältlich ● Freesoft ● Große Toolbox ● Autorun ● Bankswitching ● Hardcopy ● Color- und Window-Technik ● Disk-Utilities ● Grafik-Tools ● Tips und Tricks ● Bauanleitungen ● ROM-Fehler erkennen ● Super-Games ● Starmission ● Die geheimnisvolle Insel ● Small Business ● Text und Kalkulation ● Allround-Archiv ● Datenbank

Großer Leser-Wettbewerb
Gewinnen Sie Atari-Software-Pakete

Computer lernen sehen und fühlen

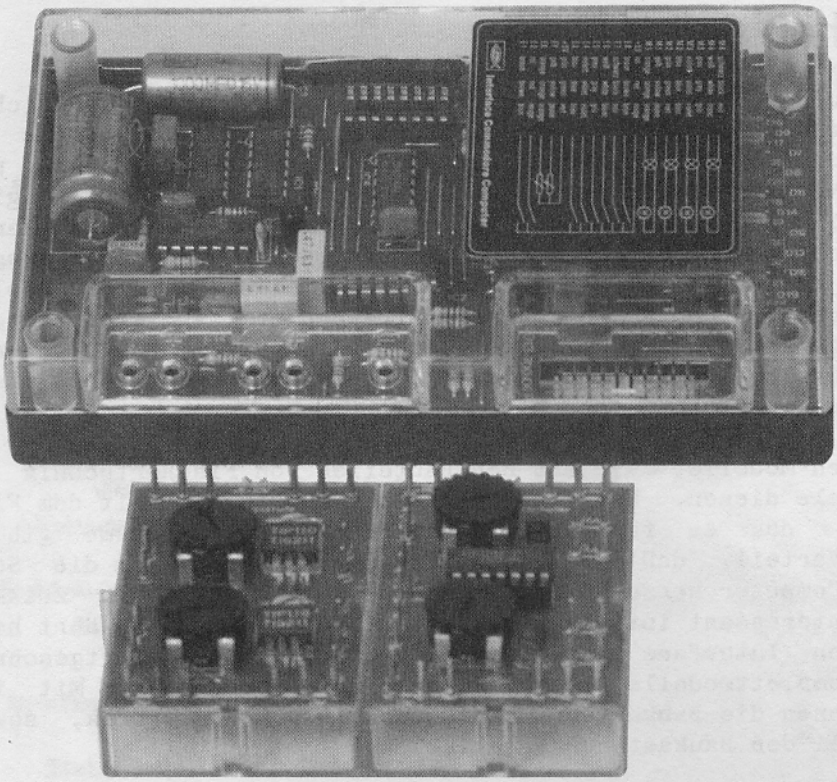
Technische Modelle zu steuern, ist eine Bereicherung für Modellbauer. Gleichzeitig erweitert es den Umgang mit dem Computer um eine weitere Dimension, so wie das Modell eines Schiffs, eines Autos oder eines Flugzeugs erst durch die Funkfernsteuerung perfekt wird. Zur Steuerung vom Modellen, die mit Elektromotoren betrieben werden, gibt es mehrere Möglichkeiten. Bei Homecomputern und bei einigen Personal-Computern kann der Spieleport und die Druckerschnittstelle herangezogen werden, um Signale zu erfassen und mit Impulsen zu steuern. In IBM-kompatible PCs können spezielle Erweiterungsplatinen eingesteckt werden, auf denen I/O-Ports und schnelle A/D-Wandler ins System integriert werden. Eine preiswerte Alternative ist das Fischer-Technik-Interface, das nicht nur den preiswerten Anschluß von Sensoren ermöglicht, sondern bereits die Leistungsstufen enthält, um Motoren und Magnete direkt anzusteuern.

In diesem CHIP Special sollen Modelle, die mit den Bauteilen von Fischertechnik gebaut werden können, als Beispiele dienen. Praktisch alle Steuerungen sind mit dem Fischer-Technik-Interface möglich, das es für alle populären Computersysteme gibt. Das Computing- System hat den Vorteil, daß alle wesentlichen Baugruppen, um die Schnittstelle zwischen Modell und Computer herzustellen, in preiswerten Baukästen zusammengefaßt sind. Für Einsteiger interessant ist das neue System Experimental, bewährt hat sich die separate Lieferung von Interface und Computing-Grundkasten, fortgeschrittene Techniken eröffnen die Komplettmodelle Plotter/Scanner und Lernroboter. Mit in das System integriert werden können die Baukästen Elektromechanik und Elektronik, sowie mit einem speziellen Magnetventil der Baukasten Pneumatik.

Daneben sollen Möglichkeiten zum individuellen Anschluß von Motoren und Sensoren gegeben werden. Die universellste Möglichkeit bietet sicher der Steuerrechner BERT, der über die serielle Schnittstelle mit praktisch jedem Rechnersystem gekoppelt werden kann. Er stellt viele I/O-Leitungen zur Verfügung. Die Leistungsstufen müssen jedoch selbst gebaut werden. Auch hierfür sind in diesem Heft Anregungen enthalten.

Das Interface

Das Interface bietet acht Bit zur Ausgabe, um vier Motoren in zwei Drehrichtungen oder acht Magnete oder Lampen zu steuern. Acht Bit breit ist die Eingabe, daneben stehen zwei Analogeingabekanäle. Mit dem Interface wird Software für die jeweilen Computer geliefert. Zur Zeit gibt es das Interface für den Apple II, Atari ST, Commodore C 64, IBM PC und die Schneider-CPC-Rechner. Die Software besteht neben den Beispielprogrammen für die Modelle aus den binären Treiberprogrammen, um die Schnittstelle zwischen den BASIC-Interpretern und dem Interface zu schaffen. Von der Idee lehnt sich das Konzept von Fischertechnik an die HP-Labor- und Industrierechner-Konzeption an. Bei diesen Systemen erfolgt die Steuerung von Peripherie über den HP-IB-Bus, der voll in das komfortbale HP-BASIC eingebunden ist. Die ft-Konzeption stellt für die BASIC-Interpreter schnelle Maschinensprache-Routinen zur Verfügung, die in das BASIC eingebunden werden. Der Aufruf der Assemblerroutinen erfolgt mit SYS- (CALL_) oder USR-Befehlen, mit denen die entsprechenden Parameter übergeben werden. Wir stellen die Einbindung des Interface in Turbo-Pascal 3.0 für den IBM PC vor.



Die ft-Steuer-Spracherweiterungen wurde so komfortabel ausgelegt, daß Motoren über ihre Nummer und die Drehrichtung aufgerufen werden. Durch entsprechende Voreinstellungen im Interface-Rumpf können im Programm übersichtliche Befehle benutzt werden. Dasselbe gilt für das Einlesen der acht digitalen Kanäle und der beiden Poti-Eingänge. Vom Grundkonzept lehnt sich die ft-Konzeption an die Programmierung speicherprogrammierbarer Steurungen an.

Die Ausgabe-Bits sind paarweise zur Steuerung von Motroen zusammengefaßt, sodaß vier Motoren gesteuert werden können. Die Ausgabe erfolgt über Leistungsverstärker. Die Ausgänge erlauben eine Dauerbelastung von 1 Ampere.

Als Sicherung ist eine Tote-Mann-Schaltung integriert, das nach einer bestimmten Zeit, einer halben Sekunde, die Leistungselektronik abschaltet, wenn keine weiteren Befehle übertragen werden. Damit ist sichergestellt, daß bei einem Programmfehler oder beim Programmstop das Modell nicht weiterläuft. Alle Ausgaberegister bleiben gesetzt, sodaß keine Informationen verlorengehen. Eine LED signalisiert den datenfluß zwischen Rechner und Interface, sodaß die Schnittstellenaktivität optisch kontrolliert werden kann.

Die Eingangsgatter sind so beschaltet, daß ein offener Eingang mit 0 ausgelesen wird. Wird ein Mikroschalter betätigt, wird der Eingang gegen +5 Volt geschaltet. Dann lifert die Schaltung eine Eins. Die Verbindung zwischen Modell und Interface wird über ein 20-poliges Flachbandkabel hergestellt, das über die Farbcodierung leicht angeschlossen werden kann. Ideal zum Anschluß ist die neue 20 polige Steckerleiste die mit ft-Experimental geliefert wird und auch den wechselnden Anschluß von verschiedenen Modellen über die Standard-Stecker leicht macht.

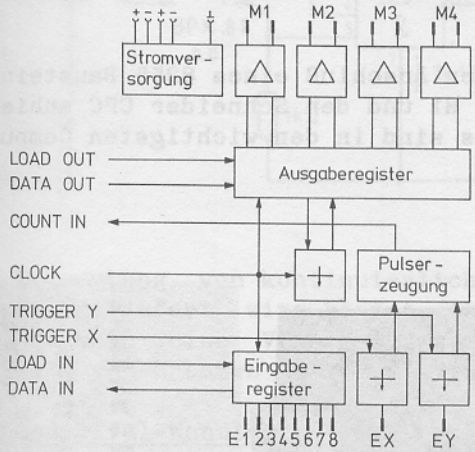
So arbeitet das Interface

Der Trick beim ft-Interface besteht aus der parallel-seriell-Wandlung mit anschließender seriell-parallel-Wandlung. Das Interface kommt damit mit acht Leitungen aus. Diese sind beim VC-64 über den user-Port, beim Apple II über den Game-Port, beim Schneider CPC mit dem systembus und beim IBM und Atari mit der parallelen Druckerschnittstelle verbunden.

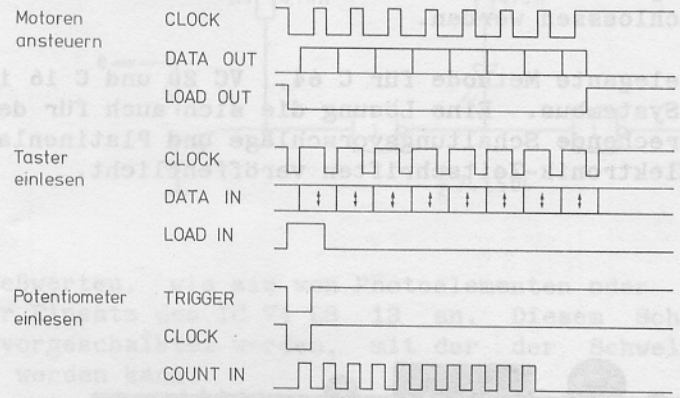
Erläutert werden soll die Funktionsweise am C64, alle anderen Rechner arbeiten mit demselben Prinzip.

Das Bitmuster der vier Motoren ist als Variable AVAR gespeichert. Beim Aufruf eines Motorsteuerkommandos werden entsprechend der Motornummer die dazugehörenden Bitpositionen herausmaskiert und dann die Drehrichtung eingesetzt. Dieses acht Bit Wort wird nun durch Schibebefhele Bit für Bit an die Leitung PB2 (DATA OUT) gelegt. Sie ist

mit dem Dateneingang eines Schieberegisters im Interface verbunden. Das Einlesen in das Schieberegister wird durch Impulsflanken am takteingang gesteuert. Das Taktsignal wird vom Grundprogramm auf der Leitung PB3 (CLOCK) erzeugt. Siand alle acht Bit übertragen wird die Leitung PB0 (LOAD OUT) gesetzt. Dieses Signal sorgt dafür, daß die daten des schieberegisters in ein acht Bit Speicherregister übernommen werden, die dann wiederum die Verstärkerstufen ansteuern.



Blockschaltbild des fischertechnik Interface



Impulsdiagramme des fischertechnik Interface

Das Taktsignal wird auch zur Ansteurung eines nachtriggerbaren Monoflops verwendet, das die Tote-Mann-Schaltung aktiviert.

Die digitalen Eingänge arbeiten analog der Ausgabe. Zunächst werden über PB1 (LOAD IN) die Daten in das Schieberegister übernommen. Durch die gleiche Taktleitung (PB§ CLOCK) werden dann die Daten des Schiberegisters an dessen Ausgang gelebt. Dieser ist mit PB7(DATA IN) verbunden, der vom Programm geprüft wird. Anschließend wird das abgefragte Bit herausmaskiert und an das Programm der Wert Null oder Eins übergeben.

Dadurch, daß immer dieselbe Taktleitung benutzt wird, ergibt sich als Nebeneffekt die ständige Aktivierung der Tote-Mann-Schaltung. In der Praxis sorgt dies für einen kontinuerlichen Interface-betrieb, da nach dem Start eines Motor ein Ereignis, das Auslösen eines Kontakts abgefragt wird.

Das Interface verfügt auch über zwei Analogeingänge. Wie im Joystick-Port wird kein A/D-Wandler sondern ein Monoflop verwendet. Die jeweiligen Ports (Trigger X) werden über PB4 oder PB5 (TRIGGER Y) getriggert. Die Laufzeit des Monoflops hängt dann vom Kondensator im Interface und dem Widerstandswert des Potentiometers ab. Normalerweise würde es genügen die Laufzeit auszuzählen. Eine Eigenart des C64 verlangt jedoch die Benutzung des Zählereingangs CNT2, über den die Leitung COUNT IN geführt wird.

Das Fischer-Technik-Interface ist so aufgebaut, daß über einen IC-Sockel, der nachträglich eingelötet werden muß alle Signale zur Verfügung stehen, um ein weiteres Interface anzusteuern. Ein entsprechendes Maschinenprogramm gibt es von ft für den C64, unser Pascal-Programm kann am IBM PC ebenfalls zwei Interfaces steuern. Als zweites Interface wird immer das Apple-Inteface verwendet. Die Analog-Eingänge der weiteren, gekoppelten Interface können nicht angesteuert werden. Statt acht Bit werden dann 16 Bit übertragen, bzw eingelesen.

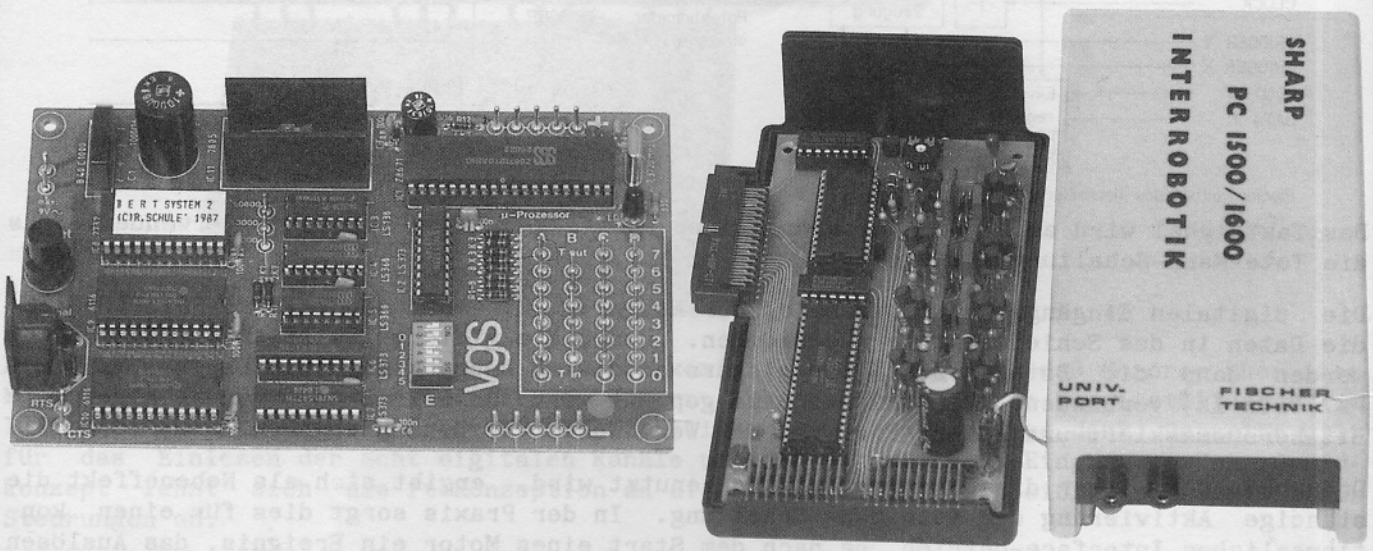
Alternative Schnittstellen

Um Sensoren an Computer anzuschließen und Motoren zu steuern, gibt es außer dem ft-Interface nach eine ganze Reihe anderer Möglichkeiten. Wer einen IBM PC kompatiblen Rechner hat kann auf eine ganze Reihe von Erweiterungskarten zurückgreifen. Es gibt I/O Karten mit zwei 8255 Bausteinen, die je 24 I/O leitungen bereitstelllen. Es gibt A/D Wandler Platinen mit verschiedener Auflösung und Wandlungsgeschwindigkeit und Relaiskarten um direkt Lastströme zu steuern.

Für den C 64 gibt es eine Reihe von Hardware-Erweiterungen, die über den User-Port eine ganze Reihe von I/O Kanälen bereitstellen. Interessant ist das Labor-System von Conrad Elektronic, das allerdings noch nicht über eine digitale Eingabe-Platine verfügt. Für den legendären ZX-81 gibt es ebenfalls bei Conrad eine digitale I/O Platine und eine Platine mit einem A/D-Wandler.

Die einfachste Möglichkeit dem Rechner die Modellwelt zu erschließen ist die Nutzung der Spiele-Ports und der parallelen Schnittstelle. An den C 64 können so vier Ptentiometer und acht Schalter zur Rückmeldung, über den User-Port acht Ausgänge angeschlosssen werden. Auch an den IBM-PC können an den Game-Port je nach Ausbaustufe bis zu acht Eingänge, vier Potis und an die bis zu drei Druckerschnittstellen je acht Ausgabekanäle angeschlosssen werden.

Eine elegante Methode für C 64, VC 20 und C 16 ist der Anschluß eines 8255 Bausteins an den Systembus. Eine Lösung die sich auch für den ZX 81 und den Schneider CPC anbietet. Entsprechende Schaltungsvorschläge und Platinenlayouts sind in den wichtigsten Computer- und Elektronik-Zeitschriften veröffentlicht.



Eine andere universelle Möglichkeit bietet BERT.

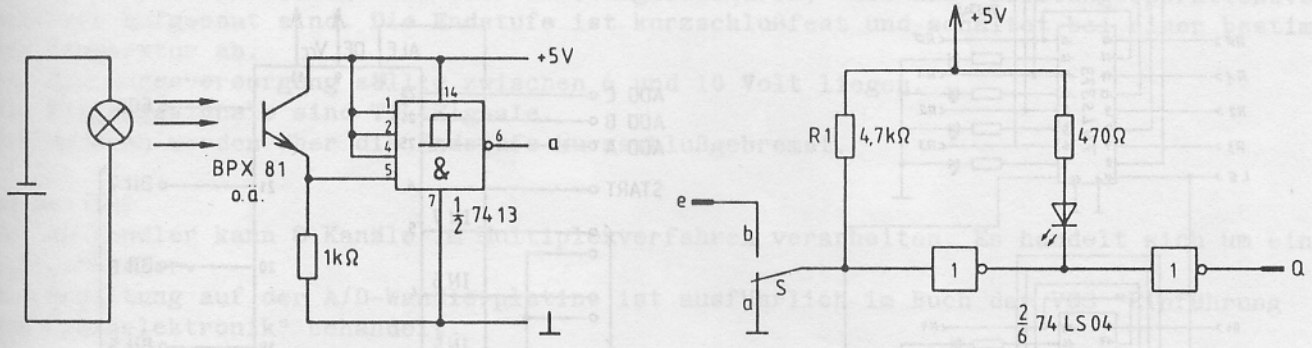
Daneben gibt es noch zwei andere Lösungen um mit Computern zu steuern. Eine Möglichkeit beitet das Interface für den Sharp PC 1500 von Luellmann. Eine andere bietet die Interface-Platinen zum Micro-Prozessor 8-Bit-Entwicklungssystem von Multitech. Badele Elektronik hat hier ein komplettes Schnittstellenmodul erarbeitet, zu dem entsprechende Ausbildungsunterlagen zur Programmierung in Assembler und Software für die Modelle des Fischertechnik Computing Grundkastens gehören.

Schaltungen zur Aus- und Eingabe

Wer eigene Schnittstellen schafft muß auch sichere Möglichkeiten zur Datenübernahme und zur Leistungssteuerung bereitstellen. Wir stellen eine Eingabe- eine Ausgabe- und eine AD-Wandlerpaltine vor. Das Layout ist nicht auf geringen Platzbedarf optimiert sondern so gestaltet, daß es in die Dimensionen des ft-Systems paßt.

Zur Ein- und Ausgabe sind spezielle Schaltunsgelektroniken notwneig, wenn man andere Systeme als das Fischer-technik-Interface verwendet.

Zur Eingabe von Informationen gibt es eine ganze Reihe von Möglichkeiten. Das einfachte Möhlichkeit besthet darin Schalter abzufragen. Dazu gibt es mehere Möglichkeiten. Die eingachste ist einen Schalten, der über einen 2,7 KOhm Widerstand gesichert ist mit dem Eingang zu verbinden. Wesentlich komfortabler ist eine tastenentprellung, die mit dem Baustein 74 LS 00 aufgebaut wird. Über zwei LEDs wird die Schalterstellung angezeigt. Prellfreie Schalter empfehlen sich dann, wenn Bewegungen erfaßt werden, bei denen beispielsweise ein Dreieckstein einen Minitaster auslöst.

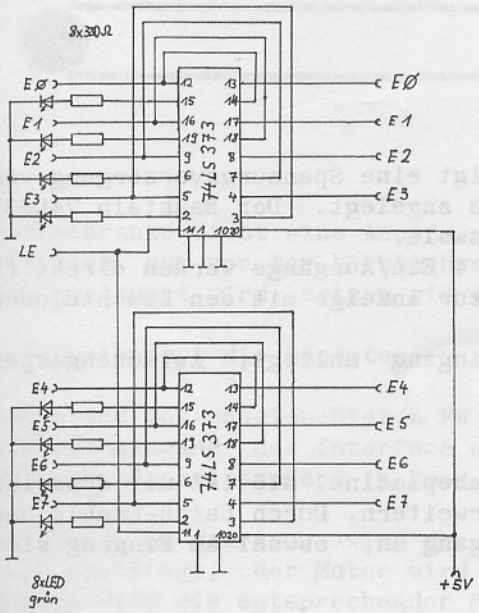


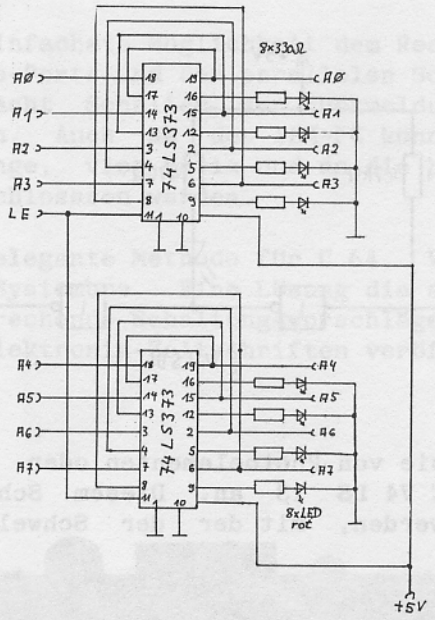
Zur Auswertung von kontinuierlichen Meßwerten, wie sie von Photoelementen oder Heißleitern geliefert wird bietet sich der Einsatz des IC 74 LS 13 an. Diesem Schmitt-Trigger kann eine Verstärkungsstufe vorgeschaltet werden, mit der der Schwellwert innerhalb von Grenzen indivuell gewählt werden kann.

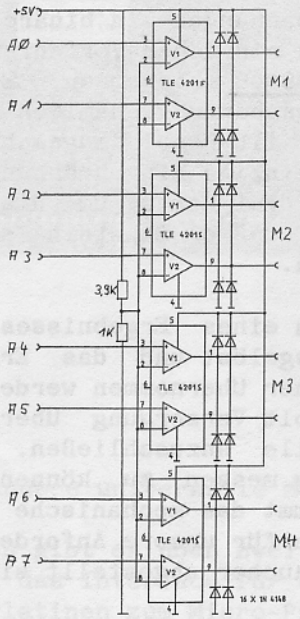
Analog-Digital-Wandler

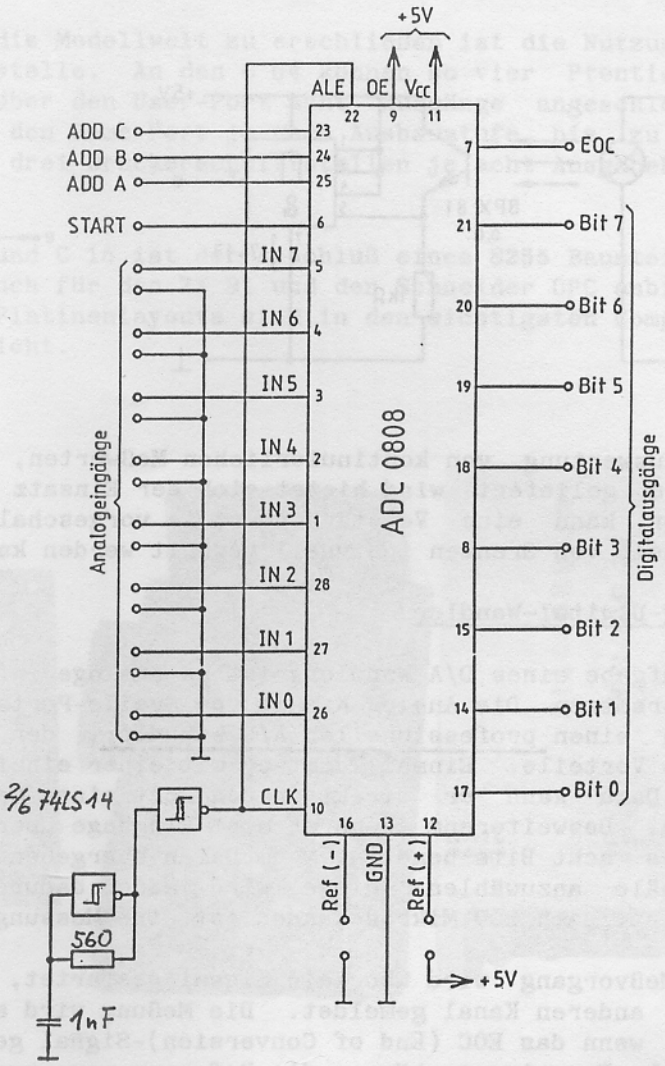
Die Aufgabe eines D/A Wandlers ist es analoge Informationen, Spannungen- in binäre Werte zu übersetzen. Die Analog-Kanäle von Speile-Ports arbeiten nach einfachen Verfahren. Wir wollen einen professionellen A/D Wabndler, den Baustein ADC 08008 einsetzen. Er hat mehere Vorteile. Einmal kommt er mit einer einzigen Versorgungsspannung, nämlich 5 Volt aus. Dazu kann er direkt an den Port eines Compputers mit TTL-Pegel angeschlossen werden. Desweiterenm kann er acht Eingänge überwachen. Die einzige Einschränkung ist, daß es acht Bits benötigt, um Daten übergeben zu können und drei Ports um die acht Meßkanäle anzuwählen, diese wird jedoch dadurch aufgewogen, daß er Baustein schnell arbeitet, nach 100 Mikrosekunden ist eine Messung abgeschlossen.

Der Meßvorgang wird über ein Signal gestartet, das Vorliegen eines Ergebnisses über einen anderen Kanal gemeldet. Die Meßung wird also disket ausgelöst und das Ergebnis kann, wenn das EOC (End of Conversion)-Signal gesetzt ist sicher übernommen werden. Für einfach Experimente können die Referenzspannungen aus der 5 Volt Versorgung übernommen werden. Für höhere Ansprüche ist eine Referenzspannungsquelle anzuschließen. Unser Schaltungsvorschlag reicht aus, um Potentiometer genau genug messen zu können. Die üblichen Bauteile haben ein Toleranz von zehn Prozent. Dazu kommt das mechanische Spiel. Im übrigen arbeiten die einfachen A/D-WAndler im ft- Interface für unsere Anforderungen genügend genau, wenn erst dann gemessen wird, wenn alle Verbraucher abgestellt sind und sich die Versorgungsspannung stabilisiert hat.









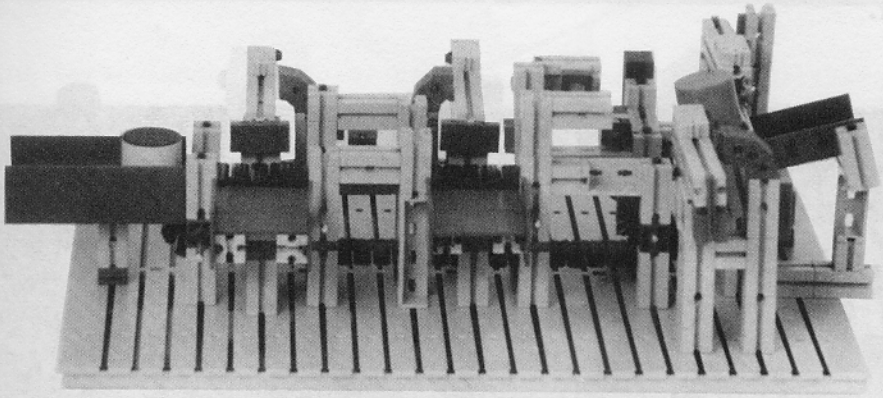
Die universellen Schnittstellenplatinen

Eingabeplatine:
Die Platine ist in TTL-Technik aufgebaut. Sie benötigt eine Spannungsversorgung von +5V. Die digitalen Signale werden an den Baustein 74LS373 angelegt. Der Baustein 74LS373 ist ein 8-Bit-Latchbaustein mit Latchenable und Outputenable.
Die Platine ist mit 2 IC 74LS373 bestückt. Jeweils 4 Ein/Ausgänge werden direkt für die digitalen Signale benutzt, die anderen vier dienen zur Anzeige mit den Leuchtdioden.
Output-Enable ist ständig freigegeben.
Durch Latch-Enable kann das Bitmuster, das am Eingang anliegt, zwischengespeichert werden. Latch-Enable ist einzeln herausgeführt.

Ausgabeplatine:
Die Ausgabeplatine arbeitet im Prinzip wie die Eingabeplatine. Sie ist mit denselben ICs aufgebaut. Hier bietet es sich an, die Ausgabe zu erweitern. Durch Latch-Enable kann das Ausgabesignal gespeichert bleiben und liegt am Ausgang an, obwohl am Eingang sich was völlig anderes abspielt.

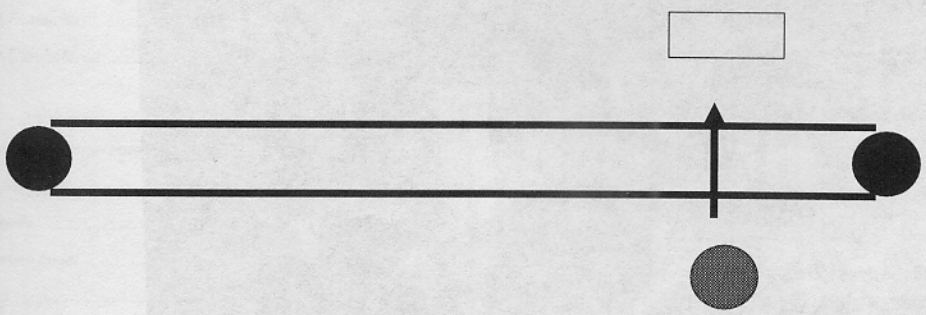
Motorplatine:
Die Motorplatine besteht aus vier Leistungsendstufen, die aus Leistungsoperationsverstärkern aufgebaut sind. Die Endstufe ist kurzschlußfest und schaltet bei einer bestimmten Temperatur ab.
Die Spannungsversorgung sollte zwischen 6 und 10 Volt liegen.
Die Eingangssignale sind TTL-Signale.
Die Motoren werden über die Endstufe kurzschlußgebremst.

AD-Wandler
Der AD-Wandler kann 8 Kanäle im Multiplexverfahren verarbeiten. Es handelt sich um einen 8-Bit-Wandler.
Die Schaltung auf der A/D-Wandlerplatine ist ausführlich im Buch der VGS "Einführung in die Mikroelektronik" behandelt.



Wir steuern ein Transportband

Genug der Vorbereitungen. Wir wollen ein Transportband steuern. Wenn Sie ein universellen Modell bauen, das nach und nach mit Sensoren und Ein- Ausgabebühnen bestückt wird, können Sie vielfältige Steuerungsaufgaben mit einem Grundmodell ausprobieren. Das Transportband soll im ersten Schritt zerbrechliche Güter, z.B. Porzellanpferde, zur Verpackungsstelle befördern. Solche Gegenstände können nur vorsichtig aufs Band gesetzt und wieder abgenommen werden; dabei muß das Band natürlich stillstehen. Die Lösung der Aufgabe heißt: Kontrolle durch Lichtschranke. Dazu können die Poti-Eingänge herangezogen werden. Eleganter ist es den Schwellwertschalter aus dem ft-Elektronik-Baukasten einzusetzen.



Als Lichtschranke dient eine Anordnung mit FW und zugehöriger Lichtquelle. Damit der FW nun wirklich nur von der Lichtschrankenlampe und nicht auch noch vom "Fremdlicht" der Umgebung gesteuert wird, setzen wir ihm die 1-mm-Störlichtkappe auf.

Und so funktioniert die Bandsteuerung:

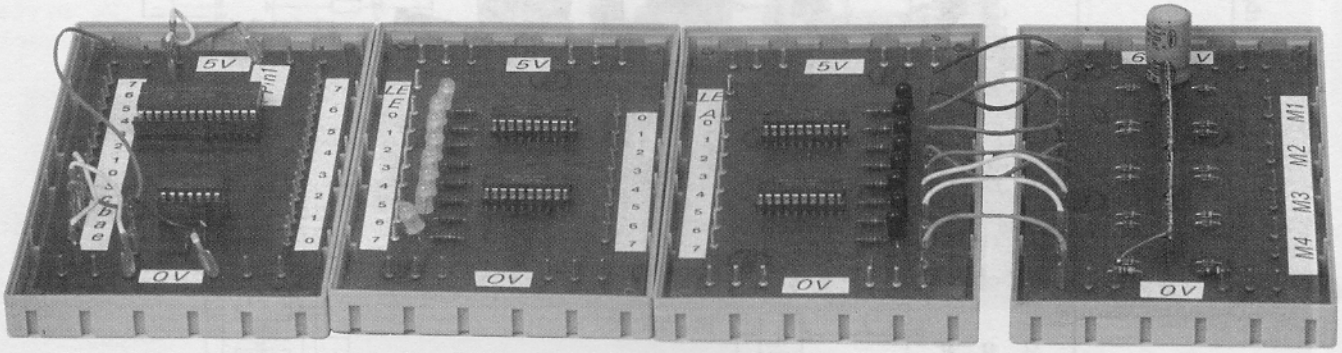
Der Widerstand des unbeleuchteten FW mit aufgesetzter 1-mm-Kappe ist viel zu groß, um am Potentiometereingang des Interface einen vernünftigen Meßwert erhalten zu können. Es kann also ein Porzellanpferd in Form eines Bausteins 15 aufs Band gelegt werden.

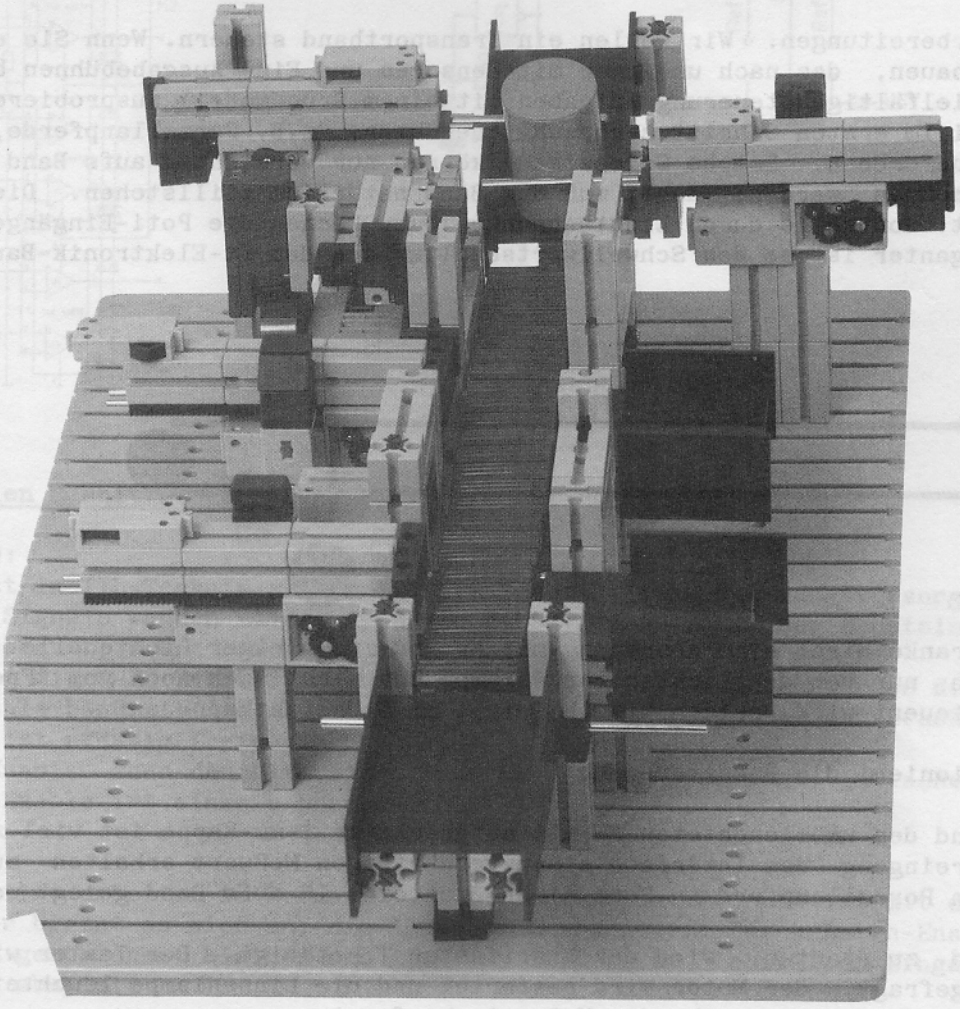
Um das Band zu starten, wird der Starttaster T betätigt. Der Taster wird über das Interface abgefragt, der Motor wird gestartet und die Linsenlampe leuchtet hell auf. Am Potieingang wird ein entsprechender Meßwert abgefragt.

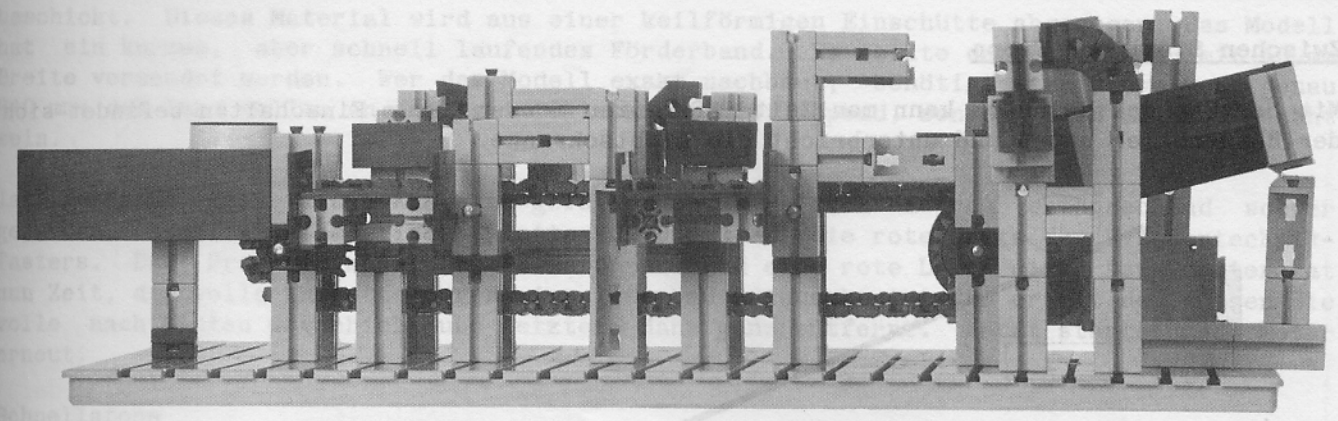
Das Band läuft nun so lange, bis das Fördergut die Lichtschranke unterbricht und dabei den FW abdunkelt. Der Wert am Potieingang ändert sich und der Motor wird durch das Programm gestoppt. Das Fördergut kann in Ruhe abgenommen werden. Erneuter Start mit Taster T.

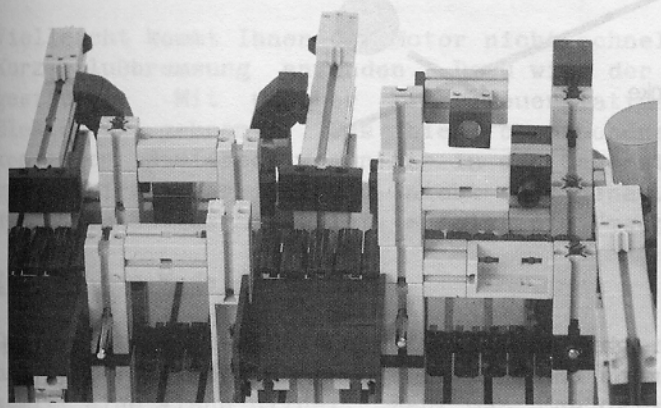
Größenkontrolle

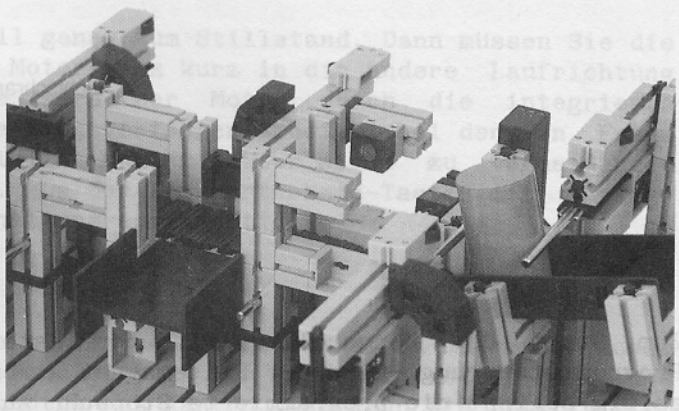
Wir können die Lichtschranke aber auch sehr gut zur Größenkontrolle von Bausteinen benutzen:
Der Baustein zwischen Lichtschranke und Band kann beliebig verändert werden. Auf diese Weise können wir die Größe der Bausteine bestimmen, welche ungehindert in einen Abfüllkorb fallen dürfen. Ein größerer Baustein wird von der Lichtschranke abgefangen und kann von Hand entfernt werden.











Anzeige

In dieser Ausgabe

Autor: Gerhard Bader

Grundwissen	**Start**
Digital-Eisenbahn	**Märklin Digital HO**
Digital-Eisenbahn	**Die Komponenten des Märklin-Digital-HO-System**
Praxis	**Anwendungsspiele**
Speicherprogrammierbare Steuerungen	**SPS**
Möglichkeiten	**Die Vorteile des Digital-Betriebs**
Schnittstelle	**Das Märklin-Interface 6050**
Steuern und Regel	**Grundlegende Steuerungen**
Ordnen	**Sortieren in Sekundenschnelle**
Künstliche Intelligenz	**Turbo-Prolog stellt die Weichen**
Fahrplan	**Fahrwegoptimierung**
Steuerungs-Programm	**ESPV3 steuert die Modellbahn**
train-ing	**Spielend lernen**
Gleisbild-Editor	**Ein interaktives Universalprogramm**
Sortieren	**Loksort**

4 003634 009537

CHIP SPECIAL 19,– DM

COMPUTER STEUERT MÄRKLIN-MODELL-BAHN

Anwendungen für Märklin-Digital-HO ● Hardware ● Software ● Fahrstraßen ● Automatikbetrieb ● Modellzüge simulieren Prozesse ● "train-ing" – ein System für die Ausbildung

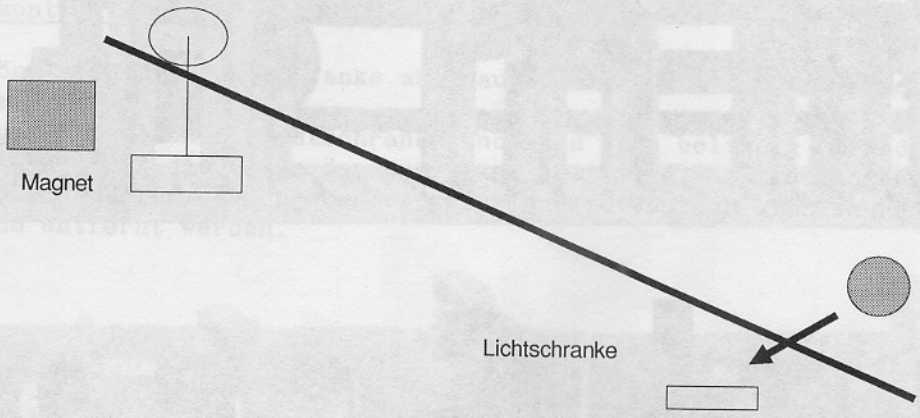


Programme für

- Atari ST
- Amiga
- Commodore 64/128
- IBM PC und Kompatible
- und alle Computer mit serieller Schnittstelle

Zwischen Start und Stopp

Mit der Uhr des Rechners kann man Zeitdifferenzen messen. Beim Einschalten befindet sich der Hängeroller unten und unterbricht die Lichtschranke.



Zum Start des Hängerollers wird der Taster ganz kurz betätigt: Über Interface und Programmsteuerung wird der Haltemagnet stromlos und gibt den Hängeroller frei - gleichzeitig beginnt die programmierte Stoppuhr zu laufen.

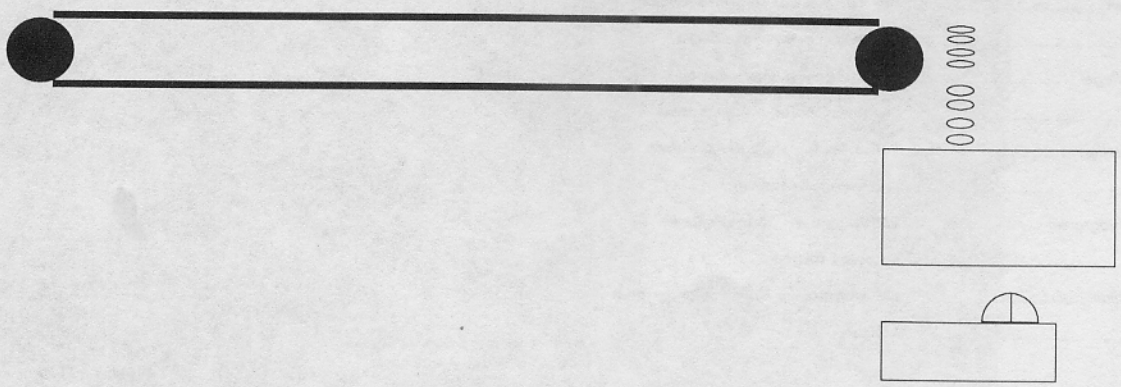
Beim Einfahren des Hängerollers in die Lichtschranke ändert sich der Meßwert, die Uhr wird gestoppt.
Zur Kontrolle wird die Zeit für den Durchlauf des unbelasteten Hängerollers noch ein paarmal gestoppt.

Und nun das gleiche mit dem belasteten Hängeroller.

Vergleichen wir die gemessenen Zeiten, so ergibt sich eindeutig:
Das Gewicht des Hängerollers hat keinerlei Einfluß auf die Geschwindigkeit - vorausgesetzt, daß die Neigung der "schiefen Ebene" nicht verändert wird.

Übrigens: Die Erklärung dieses Sachverhalts kann man in jedem Physik-Schulbuch nachlesen.

Halbautomatische Waage



Prinzip

Bei einer einfachen Waage, steht auf dem Wiegetisch, an genau fixierter Stelle, eine Kassette. Sie wird über ein Förderband mit rieselfähigem Material, z.B. Reiskörnern,

beschickt. Dieses Material wird aus einer keilförmigen Einschütte abgezogen. Das Modell hat ein kurzes, aber schnell laufendes Förderband. Es sollte dafür Ripsband mit 40 mm Breite verwendet werden. Wer das Modell exakt nachbaut, benötigt eine Länge von genau 240 mm; das Band sollte an der Nahtstelle nicht überlappend, sondern Stoß an Stoß genäht sein.

Ist genügend Reis in die Kassette gefallen und der Tisch dadurch entsprechend schwer geworden, senkt sich der Tisch (Platte 90x90 mm) auf die rote Taste des Fischertechnik-Tasters. Das Programm schaltet den Motor aus und eine rote Lampe ein. Der Roboter hat nun Zeit, die volle Kassette gegen eine leere auszutauschen, indem er mit der leeren die volle nach hinten wegschiebt und letztere dann ganz entfernt. Jetzt startet der Motor erneut.

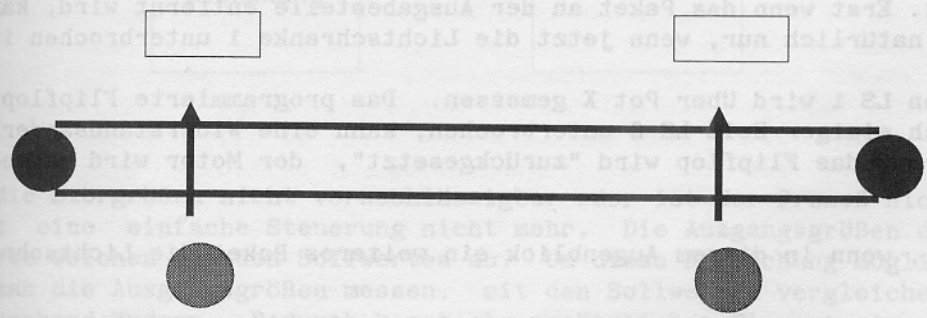
Schnellstopp

Vielleicht kommt Ihnen der Motor nicht schnell genug zum Stillstand. Dann müssen Sie die Kurzschlußbremsung anwenden. Dazu wird der Motor ganz kurz in die andere Laufrichtung gesteuert. Mit unserer Motorsteuerplatine wird der Motor durch die integrierte Elektronik gebremst. Das hier besprochene Wiegeverfahren arbeitet bei den in Frage kommenden Gewichten wegen der in den Lagerstellen und am Taster nicht zu vermeidenden unterschiedlichen Reibung nicht sehr genau. Da der Fischertechnik-Taster mit Springkontakten ausgestattet ist, erfolgt die Umschaltung schlagartig.

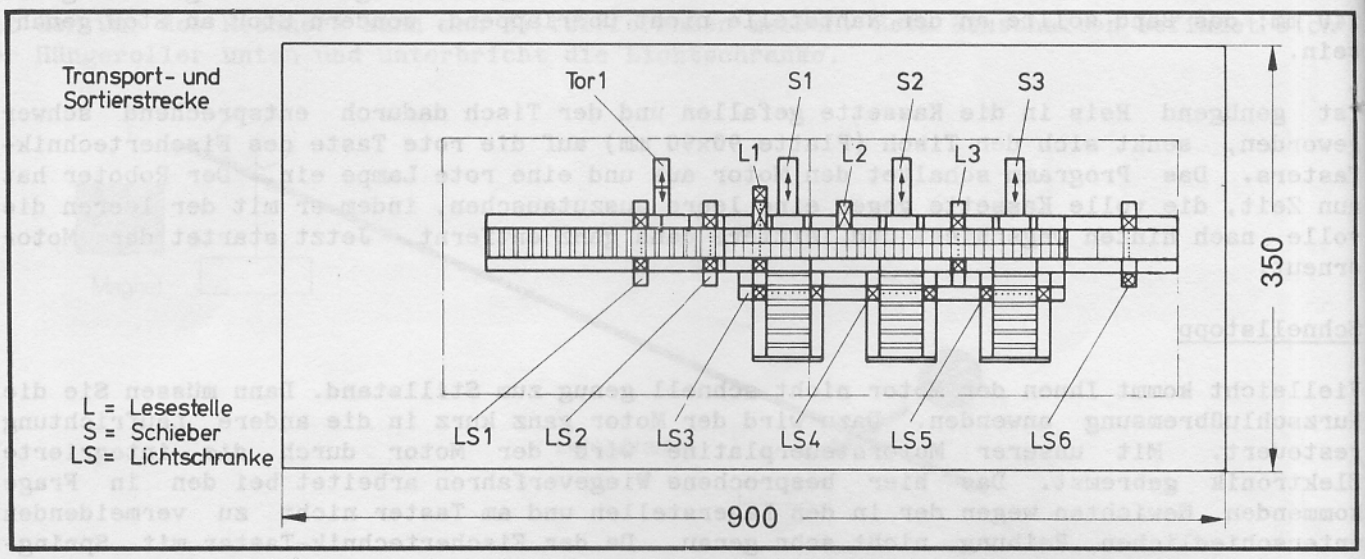
Zum Bau des Modells

Das Band sollte relativ straff auf den Reifen sitzen und die zwei Reifen auf einer Achse den größtmöglichen Abstand haben, damit das Band sich in der Mitte einsatteln kann und somit eine Rinne bildet. Baut man die Einschütte aus Karton, so kann sie relativ viel Vorrat aufnehmen. In diesem Fall empfiehlt es sich aber unter das Band eine Stütze zu bauen, so daß es nicht durchhängt.

Paket-Förderanlage
Steuerung mit 2 Lichtschranken



Lageplan



Sie haben gelernt, wie man mit Impulsen, die von Lichtschranken kommen, einen Motor ein- und ausschalten kann. Ein neues Modell benötigen Sie zur Erprobung nicht; Sie können z.B. nach der Prinzipskizze das Modell mit 2 Lichtschranken versehen. Diese Anlage könnte zum Befördern von Paketen dienen.

Die Lichtschranke 1 wird an der "Paket-Aufgabe" montiert. Damit sie einen großen Bereich erfaßt, geht der Lichtstrahl nicht senkrecht, sondern schräg über das Band. Wird auf der von ihr erfaßten Fläche ein Paket aufgelegt, so soll der Motor starten. Er soll sich selbst stillsetzen, wenn das Paket an der "Ausgabestelle" die Lichtschranke 2 erreicht hat, unabhängig davon, ob die Lichtschranke 1 schon wieder durch ein neues Paket unterbrochen ist oder nicht. Erst wenn das Paket an der Ausgabestelle entfernt wird, kann der Motor erneut starten, natürlich nur, wenn jetzt die Lichtschranke 1 unterbrochen ist.

Bei Unterbrechung von LS 1 wird über Pot X gemessen. Das programmierte Flipflop wird gesetzt. Wird nun nach einiger Zeit LS 2 unterbrochen, kann eine Widerstandsänderung an Pot Y gemessen werden und das Flipflop wird "zurückgesetzt", der Motor wird abgeschaltet.

Dies ist auch der Fall, wenn in diesem Augenblick ein weiteres Paket die Lichtschranke 1 unterbrechen sollte.

Wird das Paket vom Band genommen und damit die Lichtschranke 2 freigegeben, so bleibt der Motor stehen.
Erst beim Auflegen eines neuen Paketes im Bereich der Lichtschranke 1 beginnt der Motor sich erneut zu drehen. Liegen bereits weitere Pakete auf dem Band, läuft er jedoch nur so lange, bis das nächste Paket die Ausgabestelle erreicht hat.

Speicherung des Einschaltimpulses

Die zuletzt erprobte Paketförderanlage mit 2 Lichtschranken kann noch verbessert werden. Das Band sollte nach der Abnahme des Paketes weiterlaufen, falls sich noch weitere Pakete auf der Strecke zwischen den beiden Lichtschranken befinden.

Grundlagen der digitalen Regelung

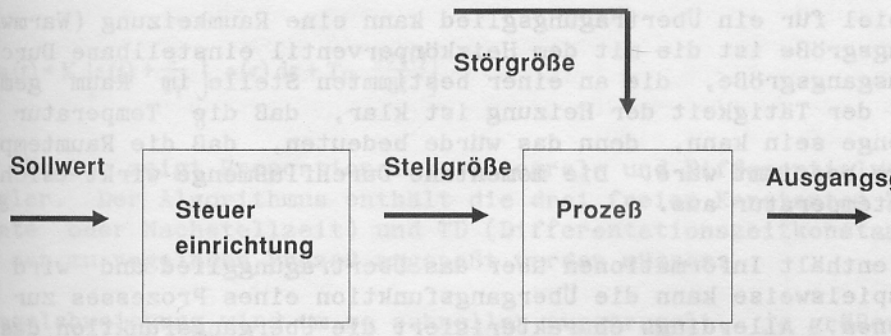
Analoge Regelung

In der Technik gilt es, bestimmte Größen eines Prozesses gezielt zu beeinflussen. Schematisch kann ein Prozeß durch einen Block dargestellt werden. Auf der einen Seite wirken Eingangsgrößen auf ihn ein; die Reaktion des Prozesses darauf wird durch die Ausgangsgrößen beschrieben.

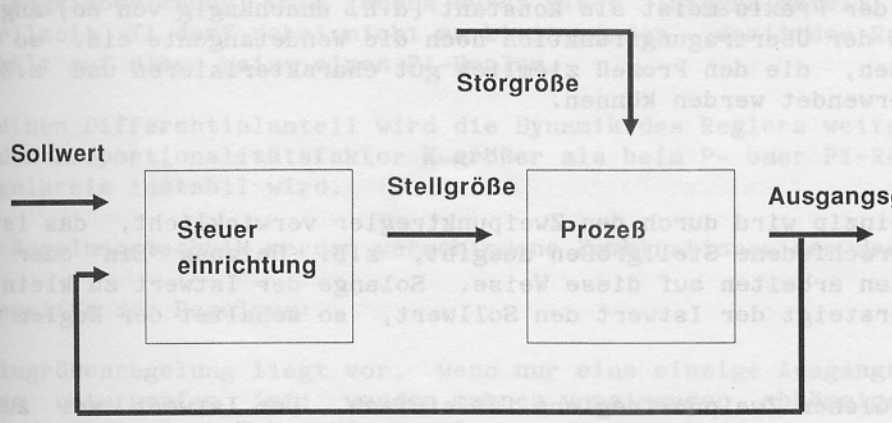
Bei jedem technischen Prozeß gibt es Eingangsgrößen, die man gezielt - von Hand oder automatisch - vorgibt; diese werden Stellgrößen genannt. Alle anderen Eingangsgrößen, meist nicht vorhersehbar, heißen Störgrößen.

Wenn der Prozeß genau bekannt ist, braucht man nur die Stellgrößen entsprechend vorzugeben, um die gewünschten Ausgangsgrößen zu erhalten. Eine solche Anordnung bezeichnet man als Steuerung. Die Steuereinrichtung bestimmt aus den gewünschten Ausgangsgrößen (den Sollwerten oder Führungsgrößen) die erforderlichen Stellgrößen und gibt sie an den Prozeß weiter.

In der Praxis lassen sich Störgrößen nicht ausschließen. In vielen Fällen kann ihr Einfluß vernachlässigt werden; man denke etwa an das Schalten von Signallampen. Bei diesen interessiert in erster Linie nicht ihre Helligkeit, sondern ihr Zustand, d.h. ob sie ein oder aus sind, wohl aber, ob sie funktionsfähig sind. Deshalb gibt es Kontrollschaltungen, die eine Verkehrsampel abschalten, wenn die Rotlichter nicht mehr funktionieren.



Sind die Störgrößen nicht vernachlässigbar oder ist der Prozeß nicht genau bekannt, so genügt eine einfache Steuerung nicht mehr. Die Ausgangsgrößen des Prozesses oder der Istwerte weichen von den Sollwerten ab. Um diese Abweichung möglichst klein zu machen, muß man die Ausgangsgrößen messen, mit den Sollwerten vergleichen und die Stellgrößen entsprechend ändern. Dadurch kommt als zusätzliches Element eine Rückführung hinzu. Aus der Steuerung wird eine Regelung; die Steuereinrichtung wird durch einen Regler ersetzt. Das wesentliche Kennzeichen jeder Regelung ist der geschlossene Wirkungskreis.



Bei einer Regelung bezeichnet man die Ausgangsgrößen des Prozesses auch als Regelgrößen; der Prozeß heißt auch Regelstrecke.

Die wichtigsten Eigenheiten der Regelung zeigen sich bereits dann, wenn nur ein Istwert geregelt und daher auch nur eine Stellgröße vorgegeben wird. Im allgemeinen braucht der Regler Ist- und Sollwert nicht getrennt zu bearbeiten, sondern es genügt die Regelabweichung als Maß für den Fehler des Istwertes.

Als Abkürzungen werden verwendet:
w Sollwert
y Istwert
u Stellgröße
e :=w-y Regelabweichung

Übertragungsglieder

Um das Verhalten eines Regelkreises beschreiben und verstehen zu können, muß man zunächst seine Elemente untersuchen. Äußerlich betrachtet haben Regler und Prozeß dieselbe Struktur. Aus einer vorgegebenen Eingangsgröße entsteht durch eine Gleichung beschreibbar, eine Ausgangsgröße. Dies wird durch das Übertragungsglied charakterisiert.

Die in der Technik vorkommenden Übertragungsglieder sind sehr vielfältig. Für allgemeine Untersuchungen ist es gleichgültig, von welcher Art (Druck, Temperatur, Durchflußmenge, elektrische Spannung usw.) die Ein- und Ausgangsgrößen sind; es kommt nur auf ihren Zusammenhang an. Ein Übertragungsglied ist daher vollständig beschrieben, wenn man zu jedem Eingangssignal u(t) das zugehörige Ausgangssignal y(t) angibt.

Als anschauliches Beispiel für ein Übertragungsglied kann eine Raumheizung (Warmwasserheizung) dienen: Eingangsgröße ist die mit dem Heizkörperventil einstellbare Durchflußmenge des Wassers, Ausgangsgröße, die an einer bestimmten Stelle im Raum gemessene Temperatur. Auf Grund der Tätigkeit der Heizung ist klar, daß die Temperatur keine Funktion der Durchflußmenge sein kann, denn das würde bedeuten, daß die Raumtemperatur durch die Durchflußmenge bestimmt wäre. Die momentane Durchflußmenge wirkt sich erst viel später auf die Raumtemperatur aus.

Die Übergangsfunktion enthält Informationen über das Übertragungglied und wird daher häufig verwendet. Beispielsweise kann die Übergangsfunktion eines Prozesses zur Optimierung des Reglers dienen. Allerdings charakterisiert die Übergangsfunktion das Übertragungsglied nicht vollständig: Während nämlich durch den Operator die Übergangsfunktion eindeutig bestimmt ist, kann man umgekehrt nicht in allen Fällen aus der Übergangsfunktion schließen, wie das Übertragungsglied auf beliebige Eingangssignale reagieren wird.

Prozesse mit Ausgleich

Besonders häufig treten Prozesse mit Ausgleich auf: Legt man an einen solchen Prozeß eine Sprungfunktion an, so nähert sich das Ausgangssignal (das ist in diesem Fall die Übertragungsfunktion) asymptotisch einem konstanten Wert. Der Verstärkungsfaktor des Prozesses, Kp, kann in der Praxis meist als konstant (d.h. unabhängig von uo) angesehen werden. Zeichnet man in der Übertragungsfunktion noch die Wendetangente ein, so erhält man insgesamt 3 Kenngrößen, die den Prozeß ziemlich gut charakterisieren und z.B. für die Regleroptimierung verwendet werden können.

Zweipunktregler

Das einfachste Reglerprinzip wird durch den Zweipunktregler verwirklicht, das ist ein Regler, der nur zwei verschiedene Stellgrößen ausgibt, z.B. Heizung "Ein" oder "Aus". Alle Bimetallthermostaten arbeiten auf diese Weise. Solange der Istwert zu klein ist, wird eingeschaltet; übersteigt der Istwert den Sollwert, so schaltet der Regler wieder aus.

Das Verhalten eines solchen Zweipunktreglers ist einfach. Der Istwert sei zunächst gleich Null. Wird zur Zeit t=0 eingeschaltet, so steigt der Istwert gemäß der Übergangs-

funktion des Prozesses an. Erreicht er den Soll-Wert, so schaltet der Regler ab. Wegen der Trägheit des Prozesses steigt jedoch der Istwert noch etwas weiter an und beginnt dann zu sinken. Bei einem Zweipunktregler oszilliert daher der Istwert um den Sollwert. Dieser Effekt ergibt sich unmittelbar aus der Trägheit des Prozesses und ist auch dann vorhanden, wenn man die Hysterese zu Null macht. Da Oszillationen des Istwerts in den wenigsten Fällen tragbar sind, müssen Verfahren gefunden werden, die die Dynamik des Prozesses berücksichtigen und damit bessere Regelungen ermöglichen.

PID-Regler

Jeder Regler kann als Übertragungsglied mit der Eingangsgröße e (Regelabweichung) und der Ausgangsgröße u (Stellgröße) aufgefaßt werden. Der Zweipunktregler kennt nur die zwei Stellgrößen "Ein" und "Aus". Das ist eine sehr grobe Art, auf einen Prozeß einzuwirken. Wesentlich bessere Regler kann man erhalten, wenn man kontinuierliche Stellgrößen zuläßt. In der Praxis ist der Stellbereich, d.h. der Bereich, in dem die Stellgröße variieren kann, selbstverständlich begrenzt.

Ähnliches gilt für die Information, die dem Regler über den Istwert zur Verfügung steht. Der Zweipunktregler kennt nur drei verschiedene Eingangssignale: Der Istwert ist entweder zu groß oder zu klein, oder er liegt im erlaubten Bereich. Diese Information gestattet nicht, einen kontinuierlichen Stellbereich sinnvoll auszunützen, da sie zu ungenau ist. Man braucht vielmehr ein quantitatives Maß für die Abweichung des Istwertes vom Sollwert; dazu dient die Regelabweichung.

Ein Regler ist ein Übertragungsglied und daher vollständig bestimmt, wenn zu jeder möglichen Regelabweichung e(t) die resultierende Stellgröße u(t) angegeben wird. Dieser Zusammenhang heißt Regelalgorithmus.

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^t e(\tau)\,d\tau + T_D\,\frac{de(t)}{dt}\right]$$

Dieser Regler zeigt Proportional-, Integral- und Differentialverhalten und heißt daher PID-Regler. Der Algorithmus enthält die drei freien Konstanten K, T1 (Integrationszeitkonstante oder Nachstellzeit) und TD (Differentationszeitkonstante oder Vorhaltzeit), die an den zu regelnden Prozeß angepaßt werden müssen.

Die Regelabweichung wird um so schneller ausgeregelt, je größer der Proportionalitätsfaktor K ist. Allerdings darf K nicht zu groß gewählt werden, weil der Regelkreis sonst schwingt.

Ein wesentlicher Nachteil des P-Reglers zeigt sich, wenn man den eingeschwungenen Zustand betrachtet, in dem der Istwert und daher auch die Stellgröße einen stationären Wert annehmen. Um den Sollwert zu erreichen, ist normalerweise eine von Null verschiedene Stellgröße u erforderlich. Ein P-Regler weist also eine bleibende Regelabweichung auf, die wegen der Beschränkung von K nach oben nicht beliebig klein gemacht werden kann.

Diese Regelabweichung kann jedoch durch einen Integralanteil reduziert werden. Die Nachstellzeit T1 darf dabei nicht zu klein werden, damit der Regelkreis stabil bleibt. Man erhält auf diese Weise einen PI-Regler.

Durch einen Differentialanteil wird die Dynamik des Reglers weiter verbessert. Man kann dann den Proportionalitätsfaktor K größer als beim P- oder PI-Regler wählen, ohne daß der Regelkreis instabil wird.

In der Regelungstechnik werden verschiedene Kombinationen der drei Anteile verwendet.

Grundbegriffe der Regelung:

Eine Eingrößenregelung liegt vor, wenn nur eine einzige Ausgangsgröße des Prozesses der Regelung unterworfen ist; werden mehrer voneinander abhängige Größen geregelt, so spricht man von einer Mehrgrößenregelung.

Ein Übertragungsglied ist ein Teil eines Systems, der aus der ihm vorgegebenen Eingangsgröße auf gesetzmäßige Weise eine Ausgangsgröße erzeugt. Beispiele für Übertragungsglieder sind Prozesse und Regler.

Die Übergangsfunktion eines Übertragungsgliedes ist durch die Ausgangsgröße gegeben, die man dann erhält, wenn man an den Eingang eine Sprungfunktion gibt. Sie ist eine wichtige Eigenschaft des Übertragungsgliedes, charakterisiert es aber nicht vollständig.

Ein Prozeß mit Ausgleich liegt vor, wenn sich seine Übergangsfunktion im Laufe der Zeit asymptotisch einem konstanten Wert nähert.

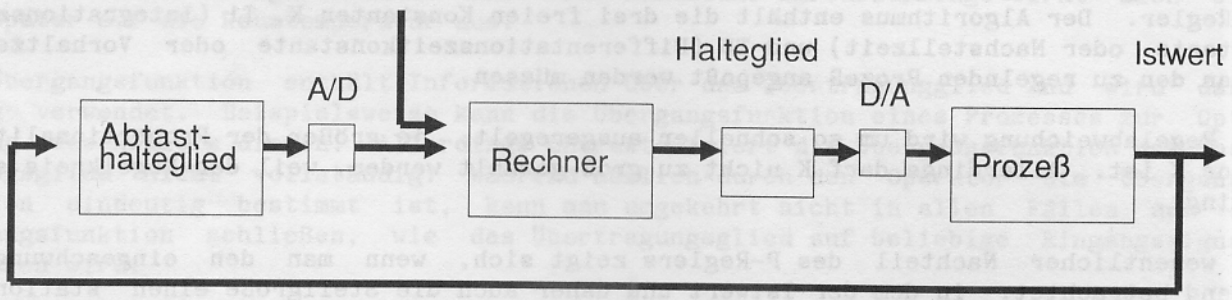
Unter Regelalgorithmus versteht man die Vorschrift, die aus der Eingangsgröße des Reglers (der Regelabweichung) seine Ausgangsgröße (die Stellgröße) erzeugt.

Ein Zweipunktregler kennt nur zwei verschiedene Stellgrößen (Ein/Aus). Von der Regelabweichung berücksichtigt er nur das Vorzeichen (Istwert zu groß oder zu klein), nicht aber den absoluten Wert. Das führt zu Schwingungen des Istwertes um den Sollwert.

Die Nachteile des Zweipunktreglers kann man vermeiden, wenn man eine kontinuierliche Stellgröße zuläßt und zugleich den Betrag der Regelabweichung mit berücksichtigt. Besonders verbreitet ist der PID-Regler.

Digitale Regelung

Der Analogregler ist dadurch gekennzeichnet, daß seine Ein- und Ausgangssignale analoge Größen sind. Der Regelalgorithmus wird durch physikalische Maßnahmen wie Bimetallschalter oder analoge elektronische Schaltungen realisiert. Die variablen Eingangsgrößen liegen ständig am Reglereingang an; ebenso stehen die Ausgangsgrößen ständig am Reglerausgang zur Verfügung. Ferner reagiert der Regler sofort auf die Eingangssignale gemäß seinem Regelalgorithmus; Verzögerungen treten nur insoweit auf, als sie durch den Algorithmus bedingt sind.



Dieses Bild ändert sich wesentlich, wenn man anstelle eines analogen Reglers einen Digitalregler verwendet, d.h., wenn der Regelalgorithmus durch einen Digitalrechner realisiert wird. Der Rechner muß die Eingangssignale durch eine Reihe von Operationen in die zugehörigen Ausgangssignale umwandeln; dazu benötigt er eine gewisse Rechenzeit. Daher reagiert der Digitalrechner grundsätzlich verzögert.

Solange die Rechenoperationen andauern, kann der Regler keine neuen Eingangsinformationen zur Kenntnis nehmen. Das hat zur Folge, daß die Eingangsgrößen nur zu diskreten Zeitpunkten in den Rechner eingelesen und die Stellgrößen nur zu diskreten Zeitpunkten ausgegeben werden. Man spricht daher von einer Abtastregelung. Die Abtastregelung, die Zeit zwischen zwei Einlesevorgängen, darf auf keinen Fall kleiner als die Rechenzeit sein. Nach Möglichkeit macht man die Abtastzeit sogar groß gegen die Rechenzeit.

Amplitudenquantisierung

Die Ausgangsdaten eines Rechners liegen grundsätzlich in digitaler Form vor. Prozesse benötigen jedoch, von Ausnahmen abgesehen, analoge Stellgrößen. In den meisten Fällen muß daher dem Regler ein Digital/Analog-Wandler (abgekürzt D/A-Wandler) nachgeschaltet werden. Aus der digitalen Form des Reglerausgangs folgt, daß die möglichen Stellgrößen den Stellbereich nicht mehr kontinuierlich ausfüllen, sondern nur endlich viele diskrete Werte annehmen können; es liegt eine Amplitudenquantisierung vor. Während jedoch die Zeitquantisierung durch die Abtastzeit To eine wesentliche Eigenschaft des Digitalreglers ist, kann die Amplitudenquantisierung durch eine genügend große Wortlänge des Rechners beliebig verfeinert werden. Im allgemeinen genügen 12 Bits.

Die Eingangsgrößen müssen dem Rechner ebenfalls in digitaler Form vorliegen. Da die meisten Prozesse analoge Istwerte aufweisen, ist dem Regler ein Analog/Digital-Wandler vorzuschalten. Der Sollwert wird dem Rechner zumeist als Inhalt eines Speicherplatzes zur Verfügung stehen, so daß die Regelabweichung zweckmäßigerweise im Rahmen des Regelalgorithmus berechnet wird.

Durch die Digitalisierung des Istwertes geht Information verloren, da Zwischenwerte nicht erkannt werden. Ähnlich wie bei der Stellgröße kann man jedoch die Genauigkeit durch entsprechende Wahl der Wortlänge erhöhen.

Halteglieder

Sobald der Rechner die Stellgröße berechnet hat, stellt er sie kurzzeitig an seinem Ausgang zur Verfügung. Der Prozeß benötigt jedoch die Stellgröße auch zwischen den Abtastzeitpunkten. Zwischen Rechner und Prozeß muß daher ein Halteglied eingefügt werden, das die Stellgröße bis zum nächsten Ausgabezeitpunkt festhält; die leistet z.B. das Fischer-Technik-Interface. Die Ausgabebausteine von Mikrocomputern haben in der Regel die Eigenschaft, daß sie ihren Ausgangszustand so lange beibehalten, bis sie von der CPU umprogrammiert werden. Für Mikrocomputersysteme ist das die nächstliegende und einfachste Lösung.

Wenn mehrere analoge Größen auszugeben sind, kann es aus Kostengründen günstig sein, nur einen einzigen Ausgabeport mit angeschlossenem D/A-Wandler einzusetzen. Unmittelbar vor der Wandlung muß daher der Istwert abgetastet und zumindest so lange festgehalten werden, bis die Wandlung beendet ist. Solche Einrichtungen heißen Abtast-Halte-Glieder. Bei genügend langsamen Prozessen ändern sich die Istwerte während der Wandlungszeit praktisch nicht, so daß sich in vielen Fällen ein Abtast-Halte-Glied vor dem A/D-Wandler erübrigt.

Nachteile der digitalen Regelung

Ein digitaler Rechner weist gegenüber einem analogen eine Reihe von Nachteilen auf. An erster Stelle stehen die Folgen, die sich aus der zeitlichen Quantisierung ergeben. Bei schnellen Regelungen spielt neben der Rechenzeit auch die Wandlungszeit des A/D-Wandlers eine Rolle.

Weit weniger schwerwiegend sind die Nachteile, die sich aus der Amplitudenquantisierung von Istwert und Stellgröße sowie aus den Rundungsfehlern im Rechner ergeben, da sich diese Fehler durch Vergrößerung der Wortlängen beliebig klein machen lassen.

Störanfälligkeit

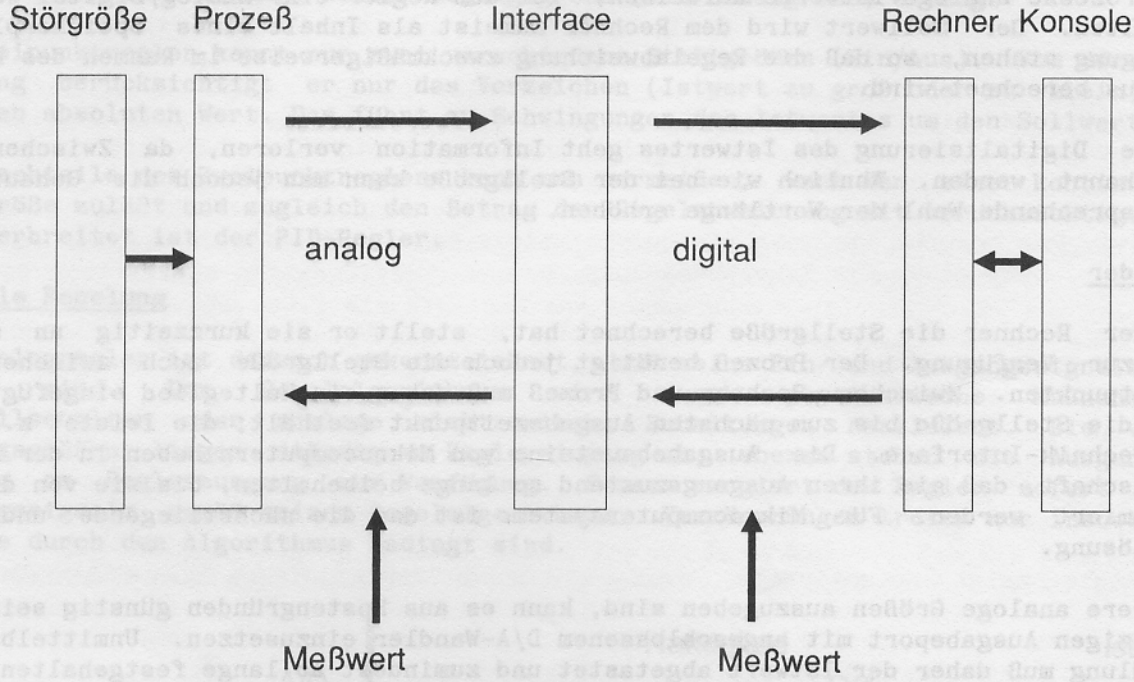
Beim Systementwurf muß auch die Störanfälligkeit des Mikrocomputersystems berücksichtigt werden. Ein ungünstiger Störimpuls kann den Prozessor aus dem Programm herauswerfen; ohne Hilfe von außen findet er nicht mehr hinein. Systeme, die in einer störenden Umgebung arbeiten sollen, müssen daher mit einer Hardwareeinrichtung versehen werden, die einen solchen Absturz des Programms erkennt und Gegenmaßnahmen einleitet.

Bei einem Ausfall der Versorgungsspannung reagiert ein Mikrocomputersystem in unkontrollierbarer Weise. Das kann zu erheblichen Schäden beim geregelten Prozeß führen. Daher muß das System einen drohenden Versorgungsausfall so rechtzeitig erkennen, daß die

erforderlichen Schutzmaßnahmen noch getroffen werden können. Ein digitaler Regler benötigt daher meist eine Netzausfallerkennung, die nur durch Hardware realisiert werden kann. Da bei einem Spannungsausfall die RAM-Inhalte verloren gehen, müssen kritische RAM-Bereiche durch eine Pufferbatterie gesichert werden. Dazu kommt ein automatischer Anlauf nach einem Systemausfall.

Komplexität digitaler Regler

Schließlich ist zu berücksichtigen, daß ein digitales Minimal-Regelsystem bereits ziemlich umfangreich ist. Während man bei einem einfachen Analogrechner mit ein paar Operationsverstärkern, Kondensatoren und Widerständen auskommt, besteht ein Mikrocomputersystem mindestens aus Prozessor, ROM, RAM, Ein-/Ausgabe-Baustein, A/D- und D/A-Wandler oder aus einem Einchip-Mikrocomputer.



Neuartige Reglertypen

Durch Digital-Rechner können Reglertypen verwirklicht werden, die in Analogtechnik nicht oder nur mit unverhältnismäßig großem Aufwand möglich sind. Der analoge PID-Regler, der als "Standardregler" angesehen werden kann, enthält drei frei wählbare Parameter. Es leuchtet ein, daß damit keine optimale Anpassung an komplizierte Prozesse möglich ist. Bei einem analogen Regler ist die sinnvolle Einführung weiterer Parameter sehr umständlich, während der zusätzliche Programmaufwand beim Digitalregler unerheblich ist.

Im allgemeinen können die Reglerparameter nicht so festgelegt werden, daß sie für alle Phasen des Prozesses optimal sind. Für den Rechner ist es jedoch kein Problem, die Parameter für den jeweiligen Sollwert geeignet zu wählen. Die Abhängigkeit der optimalen Parameter vom Sollwert muß dem Rechner bekannt sein, was einigen Syntheseaufwand erfordert, so daß sich diese Art der Optimierung nicht in allen Fällen lohnt.

Eine häufig vorkommende Regelaufgabe besteht darin, den Istwert konstant zu halten. Die Reglerparameter werden dann so bestimmt, daß die Regelgüte bei den normalerweise vorkommenden Störungen optimal wird. Für andere Phasen des Prozesses, wie etwa An- und Abfahren, können andere Parameter oder sogar andere Reglertypen optimal sein. Der Digitalrechner kann ohne weiteres zwischen den beiden verschiedenen Reglern umschalten.

Schließlich kann der Rechner noch dazu verwendet werden, die Reglerparameter an die zeitlich veränderlichen Eigenschaften des Prozesses anzupassen. Solche Regler heißen adaptive Regler.

Bei vielen Prozessen sind die Sollwerte in Abhängigkeit von der Zeit von vornherein fest vorgegeben. Das ist etwa bei Raumheizungen der Fall, wo die Temperaturen z.B. nachts und an den Wochenenden und Feiertagen abgesenkt werden soll.

Realisierung mehrerer Regler durch einen Rechner

Wenn die Abtastzeit genügend groß gewählt werden kann, ist ein Mikroprozessor ohne weiteres in der Lage, eine größere Zahl von Reglern zugleich zu realisieren. Bei mehreren unabhängigen Regelkreisen ist es allerdings meist einfacher, für jeden Kreis einen eigenen Prozessor zu verwenden, da beim Einsatz eines einzigen Prozessors ein Betriebssystem zur Verwaltung der einzelnen Kreise erforderlich ist, wodurch der Programmieraufwand beträchtlich erhöht wird. Zudem sind beim Ausfall des Prozessors alle Regelkreise betroffen.

Wird jedoch der zusätzliche Programmieraufwand in Kauf genommen, so können komplizierte Systeme durch einen einzigen Mikroprozessor bedient werden. Genannt seien etwa Mehrgrößenregler, Kaskadenregler und Störgrößenaufschaltung sowie Stellgliedsteuerung und -regelung.

Tageslichtgesteuerte Tunnelbeleuchtung

Problem
Beim Einfahren in einen dunklen, nicht beleuchteten Tunnel hat der Fahrzeuglenker nachts keine Probleme, das Licht seiner Scheinwerfer reicht völlig aus. Am Tage hat er jedoch trotz eingeschalteter Scheinwerfer Schwierigkeiten, weil das menschliche Auge einige Zeit braucht, um sich an die viel geringere Helligkeit zu gewöhnen, zu "adaptieren", wie der Fachausdruck lautet. Bei hellem Sonnenschein ist dieser Effekt viel stärker ausgeprägt als bei trübem Wetter. Abhilfe schafft man mit einer vom Tageslicht gesteuerten Tunnelbeleuchtung. Bei langen Tunnels genügt es, wenn man sich auf die Tunneleingänge beschränkt. Die mittlere Lampenreihe ist ständig eingeschaltet, die Hälfte der Lampen der linken und rechten Reihe wird bei normalem Tageslicht automatisch zugeschaltet. Nur bei hellstem Tageslicht sind alle Lampen angeschaltet.

Modell
Ein Tunnel-Modell aus Pappe ist leicht herzustellen, die Anzahl der maximal leuchtenden Lampen hängt von der Größe Ihres Lampenparkes und der zur Verfügung stehenden Stromversorgung (zweites Netzgerät oder Batterien) ab. Wir wollen uns hier auf die Darstellung des Prinzips beschränken und mit maximal sechs Lampen arbeiten.

Man kann einen oder zwei Fotowiderstände einsetzen. Die diskrete Regelung weist im Programm dem ersten Widerstand, angeschlossen über Pot X einen Schaltwert, dem anderen über Pot Y angeschlossen, einen anderen Schalt- oder Schwellwert zu. Programmgesteuert kann man auch nur einen Fotowiderstand einsetzen. Im Programm sind zwei "Schaltfenster" vorgesehen, die jeweils Lampenreihen zuschalten.

Keine Regelung!
Ganz gleich, wie groß Sie "Schaltfenster" machen, die Beleuchtung im Tunnel ist nicht geregelt, sondern nur gesteuert. Die tatsächliche Helligkeit im Tunnel hat nämlich keinen Einfluß auf den Schaltzustand. So wird z.B. nachts beim Ausfall aller Lampen der ständig leuchtenden Gruppe die zweite Lampengruppe nicht eingeschaltet. Bei einem Regelverfahren müßte diese Gruppe die Aufgabe der ausgefallenen Gruppe übernehmen. Wie man das verwirklichen kann, werden Sie später erfahren.

Dreipunkt-Regelung

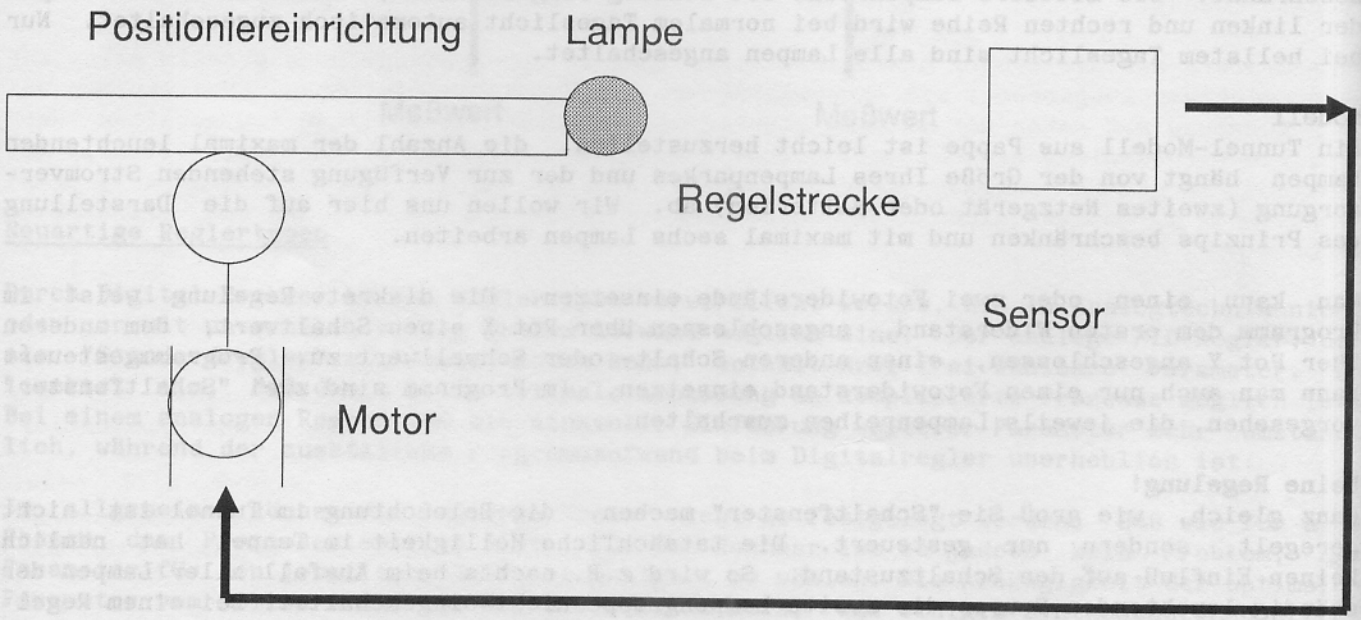
Für die Regelung der Temperatur oder der Bodenfeuchtigkeit kann man sich dank der Trägheit des Wärmefühlers bzw. der langsamen Verdunstung des Wassers im Boden mit einer sogenannten Zweipunkt-Regelung begnügen. Dabei wird unterstellt, daß bei einer Schaltgrenze ein Vorgang ausgelöst wird, bis eine Abschaltgrenze erreicht wird. Durch die Umweltbedingungen wird dann automatisch das untere Schaltniveau erreicht. Das Beispiel ist eine Temperaturregelung. Ein Heißleiter wird erwärmt, bis eine obere Schaltschwelle erreicht wird. Durch die kühlere Umgebung wird der untere Auslösungspunkt automatisch erreicht.

Für schnelle Vorgänge, z.B. die Regelung der Beleuchtung am Arbeitsplatz oder für fotografische Zwecke genügt das Zweipunkt-Verfahren nicht. Wir brauchen dazu mindestens eine Dreipunkt-Regelung.

Auch bei diesem Verfahren wird, wie bei der Zweipunkt-Regelung, durch einen Meßfühler die zu regelnde Größe, z.B. die Beleuchtungsstärke, gemessen und laufend gegen den Sollwert, z.B. den angestrebten Beleuchtungsstärkewert von 200 Lux, "verglichen". Dieser "Vergleich" erfolgt programmgesteuert.

Es sind drei Vergleichsergebnisse möglich:
Istwert entspricht Sollwert;
Istwert ist kleiner als Sollwert;
Istwert ist größer als Sollwert.

Stimmt der Istwert (= gemessener Wert) und der Sollwert überein, braucht die tatsächliche Beleuchtungsstärke (= Istwert) nicht verändert zu werden. Ist der Istwert dagegen kleiner (oder größer) als der Sollwert, muß die Regeleinrichtung den Istwert in Richtung Sollwert so lange verändern, bis beide übereinstimmen.



Prinzip-Modell

Bauen Sie zur ersten Erprobung das Modell. Es entspricht dem Modell, das wir für die Erprobung der - für diesen Zweck ungeeigneten - Zweipunkt-Regelung benutzt haben. Zur Stromversorgung des Motors benötigen Sie eine eigene Energiequelle, z.B. eine 4,5-V-Batterie oder ein zweites Netzgerät. Nur dann arbeitet die Schaltung rückwirkungsfrei. Den Fotowiderstand denken wir uns als Teil einer Arbeitsfläche, die Sie aus einem Stück

Karton leicht herstellen können. Zur Erprobung des Prinzips brauchen Sie diese Fläche aber nicht. Die Lampe schließen Sie an die Gleichspannungsbuchsen des Netzgerätes an, so daß Sie Ihre Betriebsspannung beliebig einstellen können. Den verschiebbaren Reiter, unter dem der Fotowiderstand montiert ist, lassen Sie zunächst - etwa in der Mitte des Bereiches - unverändert stehen. Die Verschiebemöglichkeit wollen wir erst später ausnutzen.

Die zu lösende Aufgabe lautet: Die Arbeitsfläche und damit der Fotowiderstand soll - unabhängig wie hoch die Betriebsspannung der Lampe ist - stets gleich stark beleuchtet werden. Das bedeutet, daß bei kleiner werdender Lampenspannung, d.h. bei Verminderung des Gesamtlichtstromes der Lampe, die Entfernung zwischen Lampe und Fotowiderstand verkleinert werden muß. Es wird somit ein größerer Auffang-Winkel erfaßt. Umgekehrt muß beim Ansteigen der Betriebsspannung, und damit bei Vergrößerung des Gesamtlichtstromes der Lampe, der Abstand zwischen Lampe und Fotowiderstand größer werden, damit ein kleinerer Auffang-Winkel entsteht.

Die Lösung dieser Aufgabe ist nicht schwer. Wir verwenden einen Fotowiderstand. Eine Regelung entsteht erst, wenn ein Motor, der den Abstand zwischen Lampe und Fotowiderstand verändern kann, so geschaltet wird, daß folgende Aufgabe erfüllt ist: Bei erreichen des Sollwerts muß der Motos stillstehen, ist der Sollwert nicht erreicht muß er die Lampe an den Fotowiderstand annähern, bzw. wegfahren. Der Sollwert ist der Widerstandwert im Regelprogramm.

Die Steuerung des Motors erfolgt programmgesteuert. Die zusätzlichen Forderung, daß der Motor bei Ausfall der Elektronik stillstehen muß, wird automatisch erreicht.

Sollwert-Vorgabe

Nun müssen wir noch den Sollwert der Beleuchtungsstärke im Programm festlegen. Wir beschließen z.B., daß die Sollbeleuchtung dann entsteht, wenn die Lampe bei voll aufgedrehtem Netzgerät aus 12 cm Entfernung den Fotowiderstand beleuchtet. Zum "Eingeben" dieses Sollwertes in die Regeleinrichtung schalten Sie den Motor ab und stellen die genannten Bedingungen her. Dann messen Sie mit dem Diagnoseprogramm den Wert für den zugeordneten Potentiometereingang. Jetzt ist der Sollwert eingestellt. Denken Sie bitte daran, daß Sie den Drehknopf nun nicht wieder verändern dürfen, Sie würden ja einen anderen Sollwert wählen!

Prüfung

Erst jetzt dürfen Sie den Motor wieder anschalten und die Regelung erproben. Verkleinern Sie die Betriebsspannung der Lampe, dann wird der Motor die Lampe näher an den Fotowiderstand heranfahren (Sollte er sie weiter entfernen, so müssen Sie den Motor umpolen).

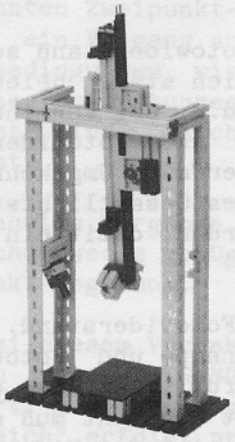
Schalten Sie eine zweite Lampe dazu, so muß der Motor die Lampen weiter weg transportieren. Tauschen Sie die Lampe gegen eine heller oder dunkler leuchtende Lampe aus, so wird sich wieder ein anderer Abstand ergeben. Bei Alterung der Lampe wird der Kolben geschwärzt, Sie können das mit Hilfe eines Butterbrotpapiers oder eines transparenten Zeichenpapiers "simulieren". Jetzt wird sich der Abstand erneut ändern.

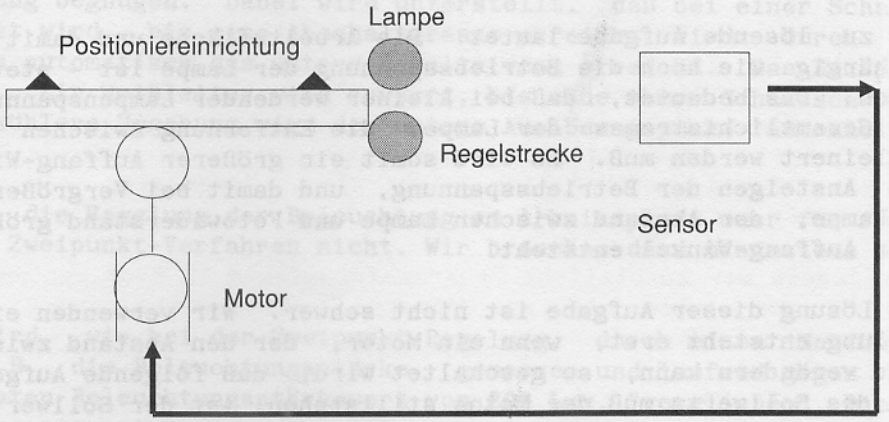
Zum Nachweis, daß die Regelung unabhängig von der Stellung des Fotowiderstandes ist, verschieben Sie nun den Fotowiderstand. Sofern Sie den Drehknopf des Grundbausteins nicht verändern, muß sich (angenähert) stets der gleiche Abstand Lampe-Fotowiderstand ergeben.

Leuchtdichte-Regelung

Beim Modell zur Erprobung der Dreipunkt-Regelung fiel stets gleichviel Licht auf den in die beleuchtete Arbeitsfläche eingebauten Fotowiderstand. Es ist leicht einzusehen, daß diese Beleuchtungsmethode nicht richtig ist, wenn auf dieser Arbeitsfläche einmal helle und dann wieder sehr dunkle Gegenstände bearbeitet und betrachtet werden sollen. Ein einfacher Versuch wird Sie davon überzeugen: Es seien zwei gleiche Zeichnungen, einmal auf weißem, einmal auf grauem Papier gezeichnet. Sorgen Sie dafür, daß beide Flächen gleich stark beleuchtet werden, etwa dadurch, daß sie gleichweit von einer über dem

Tisch angeordneten Beleuchtungslampe entfernt sind. Trotzdem werden Sie die Striche auf dem weißen Papier viel besser erkennen können als die auf dem grauen Papier. Letzteres "schluckt" mehr Licht als die weiße Fläche. Erst wenn Sie die graue Fläche noch stärker beleuchten, z.B. durch Annähern an die Lampe, läßt sich die Zeichnung ebenfalls gut erkennen. Bei dieser Beleuchtung würde weißes Papier vielleicht zu grell erscheinen. Zumindest wäre in diesem Fall der Energieverbrauch unnötig hoch.





Prinzip

Es kommt somit bei diesem "Zeichentisch" in erster Linie darauf an, daß Ihr Auge stets etwa den gleichen Helligkeitseindruck vom Papier bekommt. Das vom Papier reflektierte Licht sollte in diesem Fall mit einer Regeleinrichtung annähernd konstant gehalten werden. Es muß also die Leuchtdichte, die Ihr Auge wahrnimmt, auf einen konstanten Wert geregelt werden.

An einem Modell soll dies erprobt werden. Der Fotowiderstand erfaßt - wie Ihr Auge - das vom Papier reflektierte Licht. Zur Regelung werden die beiden Lampen vom Motor mehr oder weniger weit dem Arbeitstisch angenähert. Die Regelschaltung unterscheidet sich im Prinzip nicht von der vorher erprobten, so daß auf die Beschreibung der Wirkungsweise hier verzichtet werden kann.

Endschalter

Da es bei Ihrem Modell vorkommen kann, daß bei mattschwarzem bzw. weißem Papier die Lampen nicht nahe bzw. weit genug gefahren werden können, bauen wir aus Sicherheitsgründen zwei Endschalter in die Schaltung ein. T1 begrenzt die Bewegung der Lampenschiene nach oben, T2 nach unten. Beim Austausch des auf dem Arbeitstisch aufliegenden Materials (schwarzes - graues - weißes Papier) muß sich automatisch der Abstand der Lampe zum Tisch ändern. Das gilt auch, wenn plötzlich zusätzliches Tageslicht auf den Arbeitstisch fällt oder wenn eine der zwei Lampen ausfällt oder die Lampenspannung sich verändert.

Größerer Regelbereich

Noch interessanter wird die Schaltung, wenn Sie die Taster, die von der Lampenschiene betätigt werden, nicht als Wegbegrenzer verwenden, sondern in die Regelschaltung miteinbeziehen, um einen größeren Regelbereich zu erhalten. In diesem Fall schalten die beiden Taster programmgesteuert eine 2. Lampe. Der Regelbereich beginnt dann mit: Schiene ganz oben - 1 Lampe an. Er endet mit: Schiene ganz unten - 2 Lampen an. Mit dieser Schaltung wird also bei Auflegen von hellem Papier sogar noch Strom gespart.

An der Regelschaltung selbst ändert sich nichts. Der Regelvorgang verläuft dann z.B. folgendermaßen: Leuchtet nach einem Papierwechsel gerade nur eine Lampe und reicht deren Licht aber nicht aus, fährt die Schiene nach unten. Reicht am Ende des Weges das Licht immer noch nicht, so schaltet die Schiene über den Taster T2 die zweite Lampe ein. Dadurch erhält der Fotowiderstand mehr Licht als es dem Sollwert entspricht. Der Motor wird daher sofort umgepolt und die Schiene so weit nach oben geführt, bis Ist- und Sollwert übereinstimmen.
Steigt irgendwann das Tageslicht stark an oder wird ein sehr helles Zeichenpapier aufgelegt, so entfernt der Motor die beiden Lampen immer weiter vom Arbeitstisch. Ist am Ende des Weges die Ist-Leuchtdichte immer noch größer als der Sollwert, so wird über den Taster T1 die Zusatzlampe abgeschaltet.

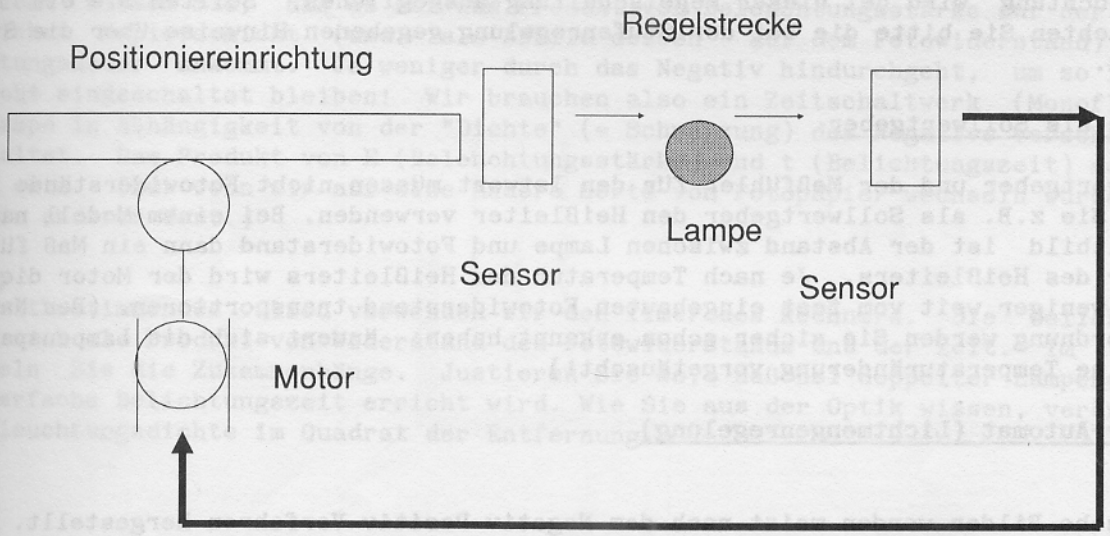
Nachführ-Regelung

Bisher haben Sie Regeleinrichtungen kennengelernt, bei denen der Sollwert auf einem bestimmten Wert "fest" eingestellt wurde. Der Sollwert bleibt also konstant, es sei denn, er wird neu vorgegeben.

Sollwertgeber

Bei manchen Regelaufgaben, vor allem in der Verfahrenstechnik, ist der Sollwert aber kein "Festwert", sondern ein Wert, der von einer sich laufend ändernden physikalischen Größe, z.B. von der beim Verarbeitungsprozeß sich einstellenden Temperatur, abhängig ist. An dem einfachsten Modell sollen die Probleme bei dieser sogenannten "Nachführ-Regelung" aufgezeigt werden.

Prinzip-Modell



Links von der Lampe (mit opaker Kappe) ist ein Bügel mit einem Fotowiderstand (FW 1) auf 2 Führungsschienen verschiebbar. Rechts von der Lampe ist auf einer waagrecht liegenden Zahnstange ein Fotowiderstand (FW 2) montiert. FW 1 kann von Hand verschoben werden, FW 2 wird mit Hilfe eines Motors an die Lampe angenähert bzw. entfernt.

Wir wollen nun ein Programm für die Motorsteuerung entwerfen, die automatisch dafür sorgt, daß der Abstand Lampe-FW 2 stets annähernd so groß ist wie der Abstand Lampe-FW 1. Diese "Nachführ"-Regelung entspricht der erprobten 3-Punkt-Regelung, jedoch mit dem Unterschied, daß der Sollwert nicht vorgegeben wird. Als Sollwertgeber wirkt der FW 1!

Wie schon bei der 3-Punkt-Regelung soll der Motor stillstehen (G1 = 1 und G2 = 0 oder umgekehrt), wenn der Widerstandswert der beiden den Spannungsteiler für E1 bildenden Widerstände gleich groß ist. Während bei der Festwertregelung einer der beiden Widerstände mit dem Drehknopf eingestellt werden kann und somit konstant bleibt, ist er bei der Nachführregelung vom Widerstandswert des FW 1 und damit von dessen Beleuchtung abhängig. Überzeugen Sie sich, daß die Nachführung tatsächlich richtig funktioniert. (Richtige Polung des Motors ist natürlich Voraussetzung.) Der Antriebsmotor muß durch eine unabhängige Energiequelle versorgt werden (zweites Netzgerät oder Batterie).

Außerdem darf der Motor nicht zu schnell laufen, weil sonst beim Regelvorgang ein sogenanntes "Überschwingen" oder "Pendeln" auftritt. Die Regelung kommt dann unter Umständen nie zur Ruhe.

Bei dieser Nachführ-Regelung entsteht, wie bei der Festwert-Regelung, der "Abgleich" (Istwert = Sollwert), wenn beide Meßfühler gleiche Widerstandswerte haben, da die Spannungsteiler für die zwei Eingänge E2 der beiden Potentiometereingänge gleiche Widerstandswerte haben. Da die Lichtquelle unserer Versuchsanordnung auf beiden Seiten etwa gleich viel Licht ausstrahlt und die beiden Fotowiderstände vom gleichen Typ sind (und Sie gleiche Kappen auf die Fotowiderstände aufgesetzt haben!), wird der Motor dafür sorgen, daß der Abstand I2 stets annähernd so groß ist wie der Abstand I1.

Ungleiche Kappen
Wie ist es aber, wenn die Fotowiderstände mit unterschiedlichen Kappen oder mit Tuben unterschiedlicher Öffnung versehen werden? Überlegen Sie, bevor Sie weiterlesen.

Wollen Sie, daß der Abstand I2 stets um einen annähernd konstanten Faktor kleiner ist als I1, dann muß der FW 2 die Kappe mit der kleineren Bohrung erhalten. Soll I2 stets größer als I1 sein, muß die Kappe mit der kleineren Bohrung auf den FW 1 gesetzt werden.

Zwei Lampen

Die gerade erprobte Nachführ-Regelung arbeitet mit einer einzigen Lichtquelle zur Beleuchtung des Sollwertgebers und des Istwertfühlers. Sie funktioniert natürlich auch, wenn Sie zwei getrennte Lichtquellen verwenden. So können sie z.B. als Lichtquelle für den Sollwertgeber das Tageslicht verwenden und damit über einen Stufenschalter die Beleuchtung in einem Tunneleingang (oder einer sogenannten "Lichtschleuse") zwischen einem völlig dunklen Raum und dem Freien regeln. Im Gegensatz zur tageslichtgesteuerten Tunnelbeleuchtung wird bei dieser Regelschaltung ausgeglichen. Sollten Sie ein Modell bauen, beachten Sie bitte die bei der Stufenregelung gegebenen Hinweise über die Stufung der Lampen.

Heißleiter als Sollwertgeber

Der Sollwertgeber und der Meßfühler für den Istwert müssen nicht Fotowiderstände sein, so können Sie z.B. als Sollwertgeber den Heißleiter verwenden. Bei einem Modell nach dem Blockschaltbild ist der Abstand zwischen Lampe und Fotowiderstand dann ein Maß für die Temperatur des Heißleiters. Je nach Temperatur des Heißleiters wird der Motor die Lampe mehr oder weniger weit vom fest eingebauten Fotowiderstand transportieren. (Den Nachteil dieser Anordnung werden Sie sicher schon erkannt haben: ändert sich die Lampenspannung, so wird eine Temperaturänderung vorgetäuscht!)

Filmkopier-Automat (Lichtmengenregelung)

Negativ
Fotografische Bilder werden meist nach dem Negativ-Positiv-Verfahren hergestellt. Der in der Aufnahmekamera befindliche lichtempfindliche Film wird belichtet. Durch die chemische Entwicklung des Filmes entsteht ein "Negativ-Bild". Die Stellen, die in der Natur hell erscheinen (Schnee - Sand), sind im Negativ dunkel und umgekehrt.

Kontaktkopie
Das Papierbild wird im einfachsten Fall (Kontaktkopie) durch Auflegen seiner lichtempfindlichen Schicht auf das Negativ und Beleuchten des Negativs von der Rückseite her gewonnen. Zur Vereinfachung der optischen Probleme ist ein Kontakt-Kopierverfahren bewährt. Bei diesem hat das Papierbild die gleiche Größe wie das Negativ. Das Prinzip ändert sich jedoch nicht, wenn zwischen Negativ und Positiv eine Vergrößerungsoptik eingesetzt ist. Der Diffusor wirkt lichtstreuend, so daß Kratzer im Negativ-Film sich im Papier-Positiv-Bild nicht auswirken. Wir deuten den Diffusor durch mattweiße Kappen vor den Lampen an.

Negativschwärzung
Je nach der Länge der Belichtungs- oder Entwicklungszeit können die Negative normal belichtet, überbelichtet oder unterbelichtet sein. Man erhält damit unter Umständen von ein und demselben Aufnahmeobjekt verschieden stark "geschwärzte" Negative.

Kopierlicht
Die lichtempfindliche Schicht des Fotopapiers wird an den Stellen, an denen der Film keine Lichtstrahlen durchläßt, nicht belichtet und deshalb nach der fotochemischen Entwicklung hell bleiben. Hat man für normal belichtete Negative die richtige Kopierbelichtung (Beleuchtungsstärke und Kopierzeit) gefunden, wird sich - ohne Automatik - bei Verwendung eines zu dunklen Negativs ein insgesamt zu helles, d.h. überbelichtetes Papierbild ergeben. Umgekehrt ergäbe ein zu helles Negativ bei der gleichen Kopierbelichtung ein zu dunkles, d.h. unterbelichtetes Papierbild. Der Kunde, der einen Film mit unterschiedlicher Schwärzung der Negative zum Kopieren, also zur Herstellung von Positivbildern gibt, verlangt natürlich, daß die mittlere Schwärzung des

Bildes so ist, wie er es am Original gesehen hat. Damit nun nicht für jedes Negativ einzeln durch Versuche die richtige Belichtung des Kopierbildes gesucht werden muß, baut man Kopier-Automaten, die nach Ablauf der richtigen Belichtungszeit das Kopierlicht selbständig abschalten.

Kopierlicht-Automat

Das Fotopapier ist etwas lichtdurchlässig. Deshalb fällt ein Teil des Lichtes, der durch das Negativ auf das Fotopapier gelangt, durch dieses hindurch. Es kann von einem Fotowiderstand aufgefangen werden. Da die Transparenz des verwendeten Papiermaterials stets gleich ist, ist die Beleuchtungsstärke auf dem Fotowiderstand in erster Linie von der Schwärzung des Negativs abhängig. Bei einem "dichten" Negativ ist der Fotowiderstand weniger beleuchtet, bei einem "flauen" Negativ erhält er mehr Licht. Der absolute Wert der Beleuchtungsstärke läßt sich ohne weiteres mit einem Fotowiderstand und einem Strommesser feststellen. Solche Messungen finden Sie im hobby-Labor 2. Uns interessiert hier jedoch, wie wir automatisch stets richtig belichtete Papierbilder erhalten.

Vielleicht wissen Sie, daß es auf das Produkt von Beleuchtungsstärke auf der lichtempfindlichen Papierschicht (und - als Abbild dessen - auf dem Fotowiderstand) und der Belichtungsdauer ankommt. Je weniger durch das Negativ hindurchgeht, um so länger muß das Licht eingeschaltet bleiben! Wir brauchen also ein Zeitschaltwerk (Monoflop), das die Lampe in Abhängigkeit von der "Dichte" (= Schwärzung) des Negativs verschieden lang einschaltet. Das Produkt von E (Beleuchtungsstärke) und t (Belichtungszeit) soll stets gleich sein. (Nur wenn wir auf eine andere Sorte von Fotopapier wechseln würden, ergibt sich ein anderer Wert.)

Prinzip

Als zeitbestimmendes Glied verwenden wir den Timer des Rechners. Die Belichtungszeit ergibt sich als Produkt von Widerstand des Fotowiderstands und der Zeit. Im Vorversuch ermitteln Sie die Zusammenhänge. Justieren Sie so, daß bei doppelter Lampenentfernung die vierfache Belichtungszeit erricht wird. Wie Sie aus der Optik wissen, verändert sich die Beleuchtungsdichte im Quadrat der Entfernung.

Modell

Mit dieser Kenntnis können wir nun schon einen Belichtungsautomaten konstruieren. Sie können wahlweise 1 oder 2 Lampen einschalten. Die Negative mit verschiedener Dichte simulieren Sie mit Butterbrotpapier (Format 58x58 mm). Je mehr Papierblätter Sie auf den Arbeitstisch legen, um so dichter ist Ihr Negativ. Der FW berücksichtigt nicht die gesamte Fläche des Bildes, sondern nur die Bildmitte. Die durchgehenden Lichtstrahlen werden von der Linse (f=3,5 cm) erfaßt und über den Spiegel an den Fotowiderstand weitergeleitet. Das Programm simuliert ein Monoflop, dessen Kippzeit von der Beleuchtungsstärke auf dem Fotowiderstand abhängt.

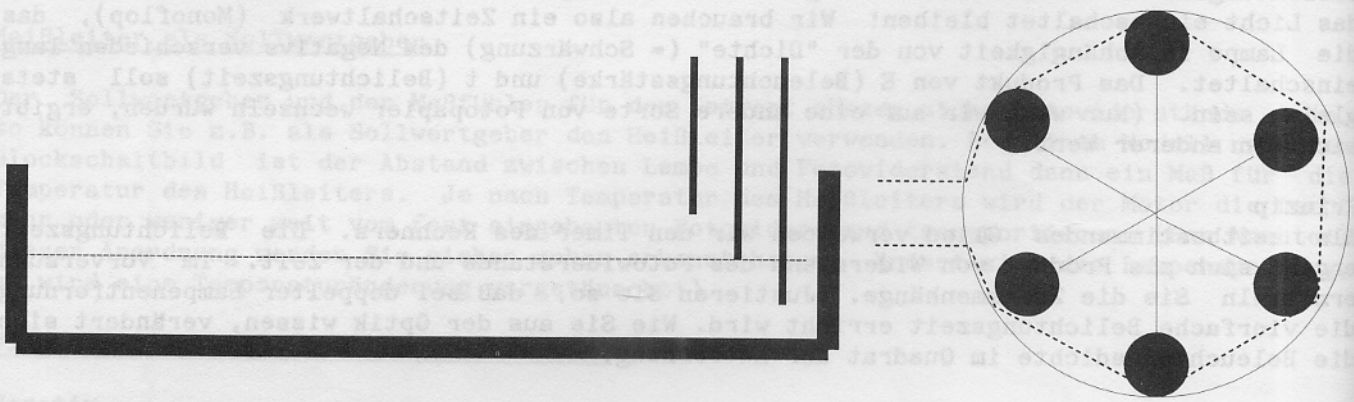
Verbesserung

Die Schaltung hat einen kleinen Mangel: Sollten Sie während der Belichtungszeit den Schalter auf "Aus" zurückstellen, erlischt das Kopierlicht zu früh. Ferner stört ein wenig, daß der Schalter S vor jedem erneuten Belichtungsvorgang auf "Aus" zurückgestellt werden muß, bevor er erneut auf "Ein" geschaltet werden kann. Deshalb ersetzen wir den Schalter durch einen Taster. Der Übersichtlichkeit halber ist nur eine Kopierlampe eingezeichnet. Jetzt beginnt der Meßvorgang, wenn Sie den Taster drücken.

Niveauregelung bei bewegten Flüssigkeiten

2 Meßfühler

Eine einfache Niveauregelung in einer elektrisch leitenden Flüssigkeit arbeitet mit einem Meßfühler. Diese Methode ist jedoch nicht besonders geeignet, wenn es sich um einen großen Behälter handelt, bei dem die Oberfläche durch Wind oder durch die Wirkung eines Rührwerks nicht ruhig bleibt, sondern bewegt ist. Bei Entstehung von Wellen würde der Antriebsmotor für das Sperrventil oder der Pumpenmotor - falls wie beim Modell die Flüssigkeitszufuhr durch eine Pumpe erfolgt - laufend ein- und ausgeschaltet werden. Man müßte wegen der bei jedem Einschalten auftretenden hohen Anlaufströme und der damit verbundenen starken Wärmeentwicklung einen übergroßen, d.h. teueren Motor verwenden. Deshalb wendet man hier ein Regelverfahren mit zwei Pegelstandsfühlern an. Man benötigt also drei Tauchelektroden. Die längste Elektrode taucht ständig in die Flüssigkeit, z.B. in Wasser, das mit etwas Salz leitend gemacht wird. Sie ist mit "-" verbunden. Deshalb führt jeder Meßfühler, der in die Flüssigkeit taucht, 1-Signal. Er ist "unbeschaltet", wenn seine Spitze nicht in die Flüssigkeit taucht.



Pumpenmodell

Wer nicht noch einmal mit dem einfachen Modell arbeiten möchte, bei dem der Zufluß des Wassers aus einem höher liegenden Vorratsbehälter in das niveaugeregelte Becken mit einem Absperrventil gesteuert wird, der kann eine sogenannte "Schlauchpumpe" bauen, die Wasser in das niveaugeregelte Becken pumpen kann. Sie benötigen dazu allerdings 6 breite Räder. Als Schlauchleitung dient ein möglichst weicher und außen sehr glatter Schlauch, z.B. ein Silikonschlauch mit 5 mm Durchmesser, wie er für Aquarien als Luftschlauch Verwendung findet. Er sollte möglichst weich sein, eine Wandstärke von höchstens 0,5 mm (sehr wichtig) und eine Länge von 1 Meter haben. Als Wasserbehälter, dessen Niveau geregelt werden soll, empfiehlt sich ein ft-1000-Kasten.
Das Prinzip der Schlauchpumpe zeigt das Bild. Einerseits drücken die Rollen das Wasser in Laufrichtung durch den Schlauch, andererseits entsteht auf der anderen Seite jeder Rolle Unterdruck; deshalb wird das Wasser von niederem Niveau angesaugt. Die Auflage für den Schlauch, der Schlauch und die Rollen werden ringförmig angeordnet. Beachten Sie beim Bau des Modells bitte, daß der Schlauch gut fixiert wird, damit er nicht etwa von den Rollen mitgenommen wird. (Daher die Forderung, daß die Schlauchaußenseite möglichst glatt sein soll.)

Soll die Anlage längere Zeit in Betrieb bleiben, darf das Niveau des Behälters, aus dem die Pumpe absaugt, nicht tiefer als die Pumpe selbst liegen, weil sich sonst der Schlauch im Laufe der Zeit entleert und unter Umständen die Pumpe dann trocken läuft. Aus dem gleichen Grund soll das Schlauchende nicht in das Wasser im niveaugeregelten Becken tauchen. Zur Wasserentnahme wird das Ventil von Hand geöffnet. Der tief in das Gefäß eintauchende Schlauch muß vorher mit Wasser gefüllt werden (Prinzip der kommunizierenden Röhren). Durch die Dimensionierung der Schläuche muß dafür gesorgt werden, daß mehr Wasser zu- als abfließt.

Regelprinzip

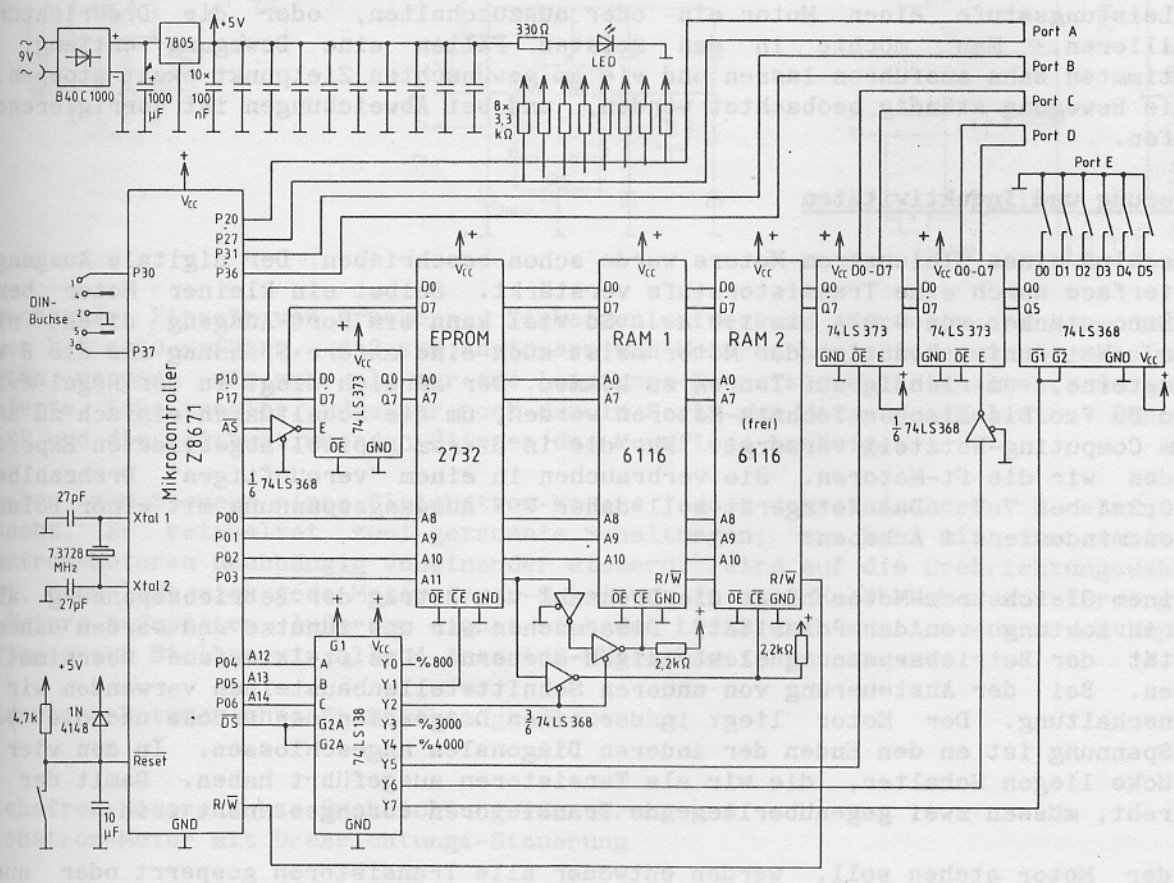
Beginnen wir mit der Betrachtung des Regelprinzips bei einem mittleren Pegelstand. Die untere Sonde B taucht ein, die obere Sonde A aber nicht. Die Pumpe arbeitet, es läuft

Wasser zu - unabhängig, ob das Auslaufventil geöffnet ist oder nicht. Der Pegel steigt also im Behälter. Nach einiger Zeit ist das Niveau soweit gestiegen, daß auch der obere Fühler, die Sonde A, in das Wasser taucht, also 1-Signal führt. Der Motor muß durch dieses 1-Signal abgeschaltet werden und abgeschaltet bleiben, auch wenn infolge "Wellengangs" die Sonde A kurzzeitig immer wieder freiliegen sollte.
Sinkt nun durch Wasserentnahme der Pegel, so wird zunächst die obere Sonde A auch bei Wellenschlag nicht mehr vom Wasser erreicht, sie ist "unbeschaltet". Das darf zu keiner Änderung des Schaltzustandes des Motors führen. Erst wenn das Wasser auch die untere Sonde B erstmals freigibt, die Sonde B also ebenfalls unbeschaltet ist, muß der Motor sich selbständig wieder einschalten. Der Motor muß weiterlaufen, auch wenn die Sonde B kurzzeitig (infolge Wellenschlags) oder ständig (als Folge der beginnenden Auffüllung des Beckens) in die Flüssigkeit taucht.

Von den Signalen her gesehen, lautet die zu lösende Aufgabe: Das von der unteren Sonde B kommende Signal: "Sonde nicht eingetaucht = nicht beschaltet" soll den Motor einschalten. Das von der oberen Sonde abgegebene Signal: "Sonde taucht ein = 1-Signal" soll den Motor ausschalten. Hat sich das Signal einer Sonde ausgewirkt, so darf nur noch das richtige Signal der anderen Sonde den Schaltzustand des Motors rückgängig machen.

Leider liefert uns nur die Sonde A ein zum Setzen und Rücksetzen notwendiges 1-Signal. Das von der Sonde B kommende Signal "unbeschaltet" (= Sonde B nicht eingetaucht) müssen wir zu einem 1-Signal umformen. Damit haben wir erreicht, daß der Motor eingeschaltet wird, solange der Pegel unter das Niveau der Sonde B gesunken ist. Da wir den Impulseingang des Flipflops benutzt haben, spielen die weiteren Vorgänge an der Sonde B keine Rolle mehr.

Von der Sonde A aus muß das Rücksetzen erfolgen, wenn diese zum ersten Mal mit der Flüssigkeit in Berührung kommt. Bei bewegter Oberfläche wird die Sonde schon kurze Zeit später wieder freigelegt, dies darf jedoch nicht zu einem erneuten Einschalten des Motors führen. Sobald die Sonde Berührung mit der Flüssigkeit hat, führt sie 1-Signal.



Interface-System Bert

Von VGS Köln wurde Bert als universelle Interface-Platine für Experimente und für anspruchsvolle Steueraufgaben entwickelt. Gegenüber anderen Schnittstellen-Konzepten hat Bert den Vorteil, daß er über eine serielle Schnittstelle an jeden Computer angeschlossen werden kann. Hinzu kommt leistungsfähige Software für Steueraufgaben und 32 Ein-/Ausgabe-Leitungen, die ausreichen, um auch komplexe Steueraufgaben zu bewältigen. Die Programmierung ist einfach, da ein Z8-Mikroprozessor verwendet wird, der über einen integrierten BASIC-Interpreter verfügt. Dazu kommt eine spezielle Steuerfirmware. Ein weiterer Pluspunkt von Bert ist, daß die Programme in EPROM gebrannt werden können und das System dadurch auch autonom arbeiten kann.

Der BASIC-Interpreter des Z8761 umfaßt 15 Befehle, die ausreichen, um Steuerprogramme zu erstellen. Dazu kommen eine Vielzahl von ROM-Routinen, die so komfortabel ausgelegt sind, daß auch die Programmierung komplexer Vorgänge einfach wird. Es gibt Befehle, um Zeitgeber und Analog-Eingänge zu überwachen und Motoren, auch Schrittmotoren, komfortabel zu steuern.

Beim Einsatz von Bert kann der Anwender sich durch die Gestaltung seiner Programme zwischen zwei Betriebsarten entscheiden. Das System kann als intelligente Interface-Platine genutzt werden, wobei die Programme im PC ablaufen; oder als eigenständiger Computer, wobei Programme bis zu vier KByte möglich sind. Wer noch tiefer einsteigen will, kann den Z8761 auch in Assembler programmieren.

Mechanische Antriebe positionieren

Mit der Fähigkeit, etwas zu bewegen, wächst der Computer über seine Schranken hinaus, die ihm Bildschirm und Tastatur auferlegen. Für viele Anwendungen genügt es nicht über eine Leistungsstufe einen Motor ein- oder auszuschalten, oder die Drehrichtung zu kontrollieren. Man möchte in den meisten Fällen eine Bewegung entlang einer vorbestimmten Bahn ausführen lassen und sie am gewünschten Zielpunkt exakt stoppen. Dazu muß die Bewegung ständig beobachtet werden, und bei Abweichungen ist korrigierend einzugreifen.

Ansteuerung und Induktivitäten

Der Anschluß eines Gleichstrom-Motors wurde schon beschrieben. Der digitale Ausgang wird im Interface durch eine Transistorstufe verstärkt. Selbst ein kleiner Motor benötigt eine Stromstärke von 0,2 A bis 1,0 A . So viel kann ein Port-Ausgang direkt niemals abgeben. Weiterhin benötigt der Motor meist auch eine andere Spannung als die 5 V der TTL-Bausteine, um richtig auf Touren zu kommen. Der Bereich liegt in der Regel zwischen 3 V und 20 V. Die Fischer-Technik-Motoren werden, um die Schaltungen einfach zu halten, aus dem Computing-Netzteil versorgt. Für die in diesem Kapitel abgebildeten Experimente verwenden wir die ft-Motoren. Sie verbrauchen in einem "vernünftigen" Drehzahlbereich etwa 0,3 A bei 7 V. Das Netzgerät soll daher 7 V Ausgangsspannung mit einer Belastbarkeit von mindestens 1 A haben.

Bei einem Gleichstrom-Motor hängt die Drehzahl vom Betrag der Betriebsspannung ab und die Drehrichtung von der Polarität. Dies machen wir uns zunutze und werden daher die Polarität der Betriebsspannung elektronisch steuern. Das ft-Interface übernimmt alle Aufgaben. Bei der Ansteuerung von anderen Schnittstellenbausteinen verwenden wir eine Brückenschaltung. Der Motor liegt in der einen Diagonalen der Brücke und die Versorgungs-Spannung ist an den Enden der anderen Diagonalen angeschlossen. In den vier Ästen der Brücke liegen Schalter, die wir als Tansistoren ausgeführt haben. Damit der Motor sich dreht, müssen zwei gegenüberliegende Transistoren durchgesteuert sein.

Wenn der Motor stehen soll, werden entweder alle Transistoren gesperrt oder nur die beiden oberen oder nur die beiden unteren. Auf keinen Fall dürfen die beiden linken, die beiden rechten oder alle vier Transistoren gleichzeitig durchgesteuert sein, da dies einen Kurzschluß der Stromversorgung bewirkt.

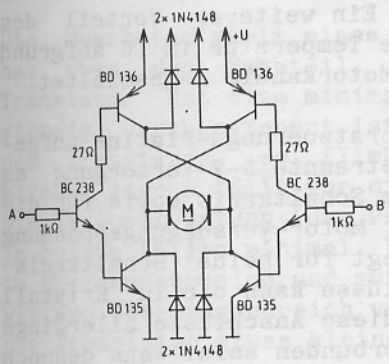
Durch geeignete Schaltungsmaßnahmen können die vier Brückenäste über nur zwei Eingänge angesteuert werden. Die Vorverstärkerstufen (Transistoren BC 238) besorgen nicht nur jeweils die gleichzeitige Durchsteuerung gegenüberliegender Brückenäste, sondern setzen auch die Belastung der betreffenden Steuerleitung herab. Für die vier Kombinationen der Steuersignale ergeben sich folgende Zusammenhänge:

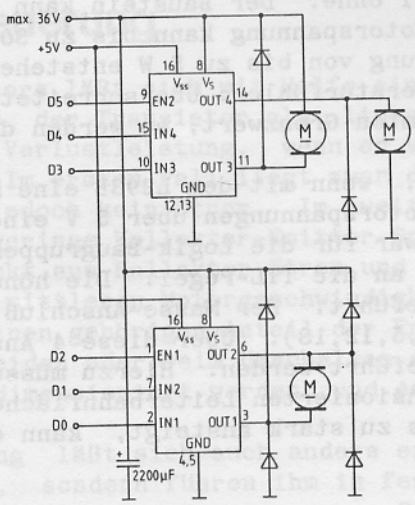
A	B	Motor
low	low	Stillstand
low	high	Rechtslauf
high	low	Linkslauf
high	high	nicht erlaubt - Kurzschluß!

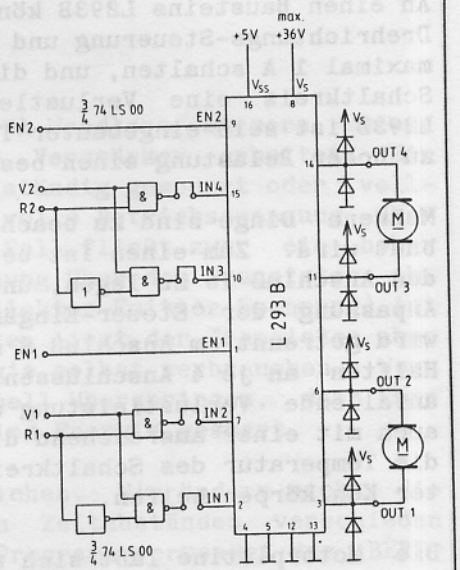
Damit können wir nun einen Gleichstrom-Motor in beiden Drehrichtungen steuern. Der Schaltungsentwurf berücksichtigt, daß ein Motor kein rein ohmischer Widerstand ist, sondern auch eine Induktivität besitzt.

Das hier Gesagte gilt auch für andere Magnete, z.B. in Relaisspulen, Magnetventilen oder Solenoidspulen; bei letzteren wird die magnetische Energie über den bewegten Kern in mechanische umgesetzt.

Ein Schaltungsvorschlag zur Motorsteuerung







Der wachsende Einsatz von Druckern, Diskettenlaufwerken, Servosteuerungen und dergleichen hat dazu geführt, daß zum Ansteuern von Motoren heute integrierte Schaltkreise angeboten werden, die mehr als nur die Leistungs-Transistoren beinhalten. So gibt es integrierte Schaltregler, die nur noch den in Serie zum Motor geschalteten Fühlwiderstand RF und die zeitbestimmenden Glieder des Monoflops benötigen.

Wir haben zum Steuern eines Gleichstrom-Motors jedoch den einfacheren Baustein L293B ausgesucht. Er beinhaltet zwei getrennte Schaltungen; man kann mit ihm somit zwei Gleichstrom-Motoren unabhängig voneinander steuern. Wird auf die Drehrichtungswahl verzichtet, dann läßt sich jede Motorsteuer-Schaltung in zwei Halbbrücken zur Versorgung je eines Motors aufspalten. Außerdem eignet sich der L293B auch zum Ansteuern von bipolaren Schrittmotoren. Stellen wir alle 4 Anschluß-Möglichkeiten zusammen:

4 Gleichstrom-Motoren ohne Drehrichtungs-Steuerung

oder:
2 Gleichstrom-Motoren ohne Drehrichtungs-Steuerung und
1 Gleichstrom-Motor mit Drehrichtungs-Steuerung

oder:
1 bipolarer Schrittmotor

Das Bild zeigt das Innenschaltbild einer Hälfte des Bausteins L293B, und zwar für verschiedene Schaltzustände. Man erkennt die Brückenschaltung der Motorverstärker; außerdem sind die vier extern anzuschließenden Freilaufdioden und der Motor (als Induktivität) eingezeichnet. Die Ausgangsverstärker werden durch die beiden Eingänge IN1 und IN2 (bzw. IN3 und IN4 in der anderen IC-Hälfte) gesteuert. Die Logikgatter sorgen dafür, daß die Schalter in der Brücke niemals so geschlossen werden, daß ein Kurzschluß entsteht. Außerdem muß für den Betrieb des Motors der Eingang EN1 (bzw. für die andere Schaltkreishälfte EN2) auf H-Pegel liegen; EN bedeutet enable, d.h. soviel wie Freigabe. Werden die beiden Eingänge IN1 und IN2 (bzw. IN3 und IN4) gemeinsam auf L-Pegel gelegt, so sind die beiden unteren Transistoren durchgeschaltet. Der Induktions-Strom fließt nun, je nach Polarität, über einen der beiden Transistoren sowie die extern anzubringenden Freilaufdioden. In diesem Fall läuft der Motor aus, da der Induktions-Strom immer wieder durch die Spule des Motors zirkuliert. Das gleiche geschieht auch, wenn beide Steuer-Eingänge IN1 und IN2 auf H-Pegel liegen: Dann fließt der Induktions-Strom über einen der beiden oberen Transistoren und die zugehörige Freilaufdiode.

Wird das Freigabe-Signal EN1 auf L-Pegel geschaltet, dann sind alle vier Transistoren der Brücke gesperrt, und der Induktions-Strom wird über die Freilaufdioden in die Versorgungs-Spannung zurückgespeist; das hat zur Folge, daß der Motor sehr schnell zum Stehen kommt.

An einen Bausteins L293B können drei Gleichstrom-Motoren angeschlossen werden: einer mit Drehrichtungs-Steuerung und zwei ohne. Der Baustein kann je Brücke eine Stromstärke von maximal 1 A schalten, und die Motorspannung kann bis zu 36 V betragen. Insgesamt darf im Schaltkreis eine Verlustleistung von bis zu 5 W entstehen. Ein weiterer Vorteil des L293B ist sein eingebauter Temperaturfühler: Überschreitet die Temperatur im IC aufgrund zu hoher Belastung einen bestimmten Grenzwert, so werden die Motorkanäle abgeschaltet.

Mehrere Dinge sind zu beachten, wenn mit dem L293B eine Motorsteuerungs-Platine aufgebaut wird. Zum einen ist bei Motorspannungen über 5 V eine getrennte 5-V-Versorgung an den Anschluß 16 zu legen, und zwar für die Logik-Baugruppen im Schaltkreis sowie für die Anpassung der Steuer-Eingänge an die TTL-Pegel. Die höhere Motor-Versorgungsspannung wird getrennt am Anschluß 8 zugeführt. Der Masse-Anschluß liegt für beide Schaltkreis-Hälften an je 4 Anschlüssen (4,5,12,13). Über diese 4 Anschlüsse kann die im Kristall anfallende Verlustleistung abgeführt werden. Hierzu müssen diese Anschlüsse allerdings auch mit einer ausreichend dimensionierten Leiterbahnfläche verbunden sein. Wenn dennoch die Temperatur des Schaltkreises zu stark ansteigt, kann ein auf das Gehäuse aufgeklebter Kühlkörper helfen.

Die Motorplatine läßt sich nicht nur durch die Signale IN1 und IN2 (bzw. IN3 und IN4) für Rechts- und Linkslauf ansteuern. In späteren Experimenten wird es sich als nützlich erweisen, an einem Eingang über einen Pegel die Drehrichtung festzulegen und an einem anderen Eingang den Motorlauf ein- und auszuschalten. Zu diesem Zweck sind dem Motorsteuer-IC L293B noch TTL-Gatter vorgeschaltet. Mit R1 bzw. R2 wird die Drehrichtung festgelegt, während V1 bzw. V2 zum Ein- und Ausschalten des Motors dient.

Wenn der Eingang V1 (V2) auf H-Pegel liegt, der Motor also eingeschaltet ist, bewirkt ein H-Pegel am Eingang R1 (R2), daß IN1 (IN3) auf H-Pegel liegt. Ein L-Pegel an R1 (R2) aktiviert dagegen IN2 (IN4). Ist der Motor abgeschaltet (V1 bzw. V2 auf L-Pegel), so befinden sich beide Steuereingänge IN1 und IN2 (bzw. IN3 und IN4) auf H-Pegel.

Damit beide Betriebsarten verfügbar bleiben, sind der L293B und die Logikschaltung nur über Steckbrücken verbunden, die nach Wahl zu öffnen oder zu schließen sind.

Ein erstes Programm zur Motorsteuerung

Um uns von der Funktionsfähigkeit der Motorsteuer-Platine zu überzeugen, benutzen wir ein einfaches Programm. Wir können die Drehrichtung des Motors über IN1 und IN2 steuern. Durch Drücken der Pfeiltasten (Cursortasten) am PC wird der Motor in Links- bzw. Rechtslauf versetzt. Dabei ist die Angabe "links" bzw. "rechts" zweifellos willkürlich, weil die beobachtete Laufrichtung vom Einbau des Motors in die Apparatur und von der Blickrichtung abhängt. In einem Modell ist die Drehrichtung natürlich wichtig; dann müssen gegebenenfalls die Anschlüsse des Motors vertauscht werden, um die korrekte Drehrichtung zu erzielen.

```
100 REM MOTORSTEUERUNG
102 REM ==============
110 REM CURSOR LINKS = LINKSLAUF
120 REM CURSOR RECHTS= RECHTSLAUF
130 REM A = ABSCHALTEN MIT NACHLAUF
140 REM LEERTASTE = SCHNELLSTOP
150 REM ANSCHLUSS DER MOTORPLATINE
160 REM D0 - IN 1
170 REM D1 - IN 2
180 REM D2 - EN 1
200 S=0 :REM ANFANGSSTELLUNG
210 PRINT"GLEICHSTROMMOTOR-STEUERUNG"
220 PRINT
230 PRINT"CURSOR LINKS  = LINKSLAUF"
240 PRINT"CURSOR RECHTS = RECHTSLAUF"
250 PRINT"TASTE A       = ABSCHALTEN"
260 PRINT"LEERTASTE     = SCHNELLSTOP"
270 GET A$ : IF A$="" THEN GOTO 270
280 IF A$=CHR$(157) THEN S=6 :REM LINKS
290 IF A$=CHR$(29) THEN S=5 :REM RECHTS
300 IF A$=" " THEN S=0 :REM SCHNELLSTOP
310 IF A$="A" THEN S=4 :REM NACHLAUF
320 PRINT#2,"D=";S
330 GOTO 270
```

Durch Drücken der Taste A wird der Motor gestoppt. Das Abschalten erfolgt mit Nachlauf, indem beide Steuereingänge des Schaltkreises auf L-Pegel gelegt werden. Anders verhält es sich mit dem Schellstopp-Befehl, der durch Drücken der Leertaste ausgelöst wird. Auch hier werden beide Steuereingänge auf L-Pegel gelegt. Da aber der Freigabe-Eingang ebenfalls auf low geht, sind alle Transistoren gesperrt, und es erfolgt Rückspeisung des Induktions-Stroms in das Netzgerät.

Geschwindigkeits-Steuerung

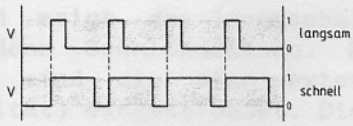
Die Geschwindigkeit eines Motors läßt sich mit Hilfe eines DA-Wandlers steuern. Diese hat aber den Nachteil, daß der Transistor als linearer Verstärker arbeitet. Ein Transistor hat eine minimale Verlustleistung, wenn er vollständig gesperrt oder vollständig durchgesteuert ist. Im ersten Fall liegt zwar die volle Betriebsspannung über dem Transistor, es fließt jedoch kein Strom. Im zweiten Fall fließt zwar ein hoher Strom, jedoch fällt nur die geringe Kollektor-Emitter-Spannung über dem Transistor ab. Die Verlustleistung (das Produkt aus Kollektor-Strom und Kollektor-Emitter-Spannung) ist in beiden Fällen minimal. Bei mittleren Motorgeschwindigkeiten liegt der Transistor aber im Übergangsbereich und muß einen gehörigen Anteil der Energie selbst verbrauchen. Wenn wir den Übergangsbereich vermeiden oder beim Umschalten schnell überspringen, dann kann der Transistor etwas kleiner dimensioniert werden, und es wird Energie gespart.

Die Geschwindigkeits-Steuerung läßt sich auch anders erreichen. Wir ändern nicht die Betriebsspannung des Motors, sondern führen ihm in festen Zeitabständen verschieden lange "Energiepakete" mit stets gleicher Spannung zu. Das Programm verwendet das BERT-Kommando I = expr.

Das Programm wird über die Cursortasten bedient. In Zeitabständen, die durch die Arbeitsgeschwindigkeit des PC-BASIC-Interpreters und der V24-Übertragung gegeben sind, durchläuft das Programm immer wieder den Ausgabebefehl in Zeile 280. Dort wird nun ein Impuls an BERTs Zeitgeber-Ausgang B6 = Iout ausgelöst. Wenn der Impuls lang und die Datenübertragung recht schnell ist (z.B. 4800 Bd), dann kann die Impulsdauer durchaus bis zum nächsten Durchlauf der Ausgabe-Zeile reichen. An Tout steht dann durch Nachtriggerung ein ständiges H-Signal an. Wird die Impulsdauer kürzer gewählt, so entsteht ein Rechtecksignal, dessen Tastverhältnis oder duty cycle (Verhältnis von Pulsdauer zu Periodendauer) per Programm einstellbar ist. Im Extremfall erscheinen an B6 nur noch kurze Nadelimpulse.

Führen wir die gepulsten Signale dem Motor zu, so wird er keineswegs im Takt rucken, sondern vielmehr aufgrund des Nachlaufs die Pulse "vermitteln". Er läßt sich daher in seiner Geschwindigkeit recht glatt steuern, wenn der zeitliche Abstand der Pulse nicht zu groß ist.

Wie schon in den Kapiteln zuvor bei zeitkritischen Anwendungen, werden wir dieses Problem lösen, indem wir BERT mit dieser Aufgabe betrauen. Das BERT-Kommando DC = expr, geschrieben in BASIC als:




```
100 REM IMPULS-STEUERUNG
102 REM ================
110 REM CURSOR LINKS = LINKSLAUF
120 REM CURSOR RECHTS = RECHTSLAUF
130 REM LEERTASTE = ABSCHALTEN
140 REM ANSCHLUSS DER MOTORPLATINE
150 REM B6 - V1
160 REM B5 - R1
170 REM EN 1 AN H-PEGEL
200 REM ANFANGSSTELLUNG
210 VI=128:V0=0:VM=16384-VI
220 PRINT"GLEICHSTROMMOTOR-STEUERUNG"
230 PRINT
240 PRINT"CURSOR LINKS  = LINKSLAUF"
250 PRINT"CURSOR RECHTS = RECHTSLAUF"
260 PRINT"LEERTASTE     = SCHNELLSTOP"
270 GET A$
280 IFA$=""THEN PRINT#2,"I=";V0:GOTO270
285 REM LINKS:
290 IFA$=CHR$(157)THENIFV>-VMTHENV=V-VI
295 REM RECHTS:
300 IF A$=CHR$(29)THEN IFV<VMTHENV=V+VI
310 IF A$=" " THEN V=0 :REM SCHNELLSTOP
320 V0=ABS(V) : D=-(V>0)
330 PRINT#2,"B5=";D
340 PRINT"GESCHW.";V,"RICHTUNG";D
350 GOTO 270
```

erzeugt an Ausgang B6 eine Impulsfolge, deren Tastverhältnis sich in Bruchteilen der Zahl 256 einstellen läßt, also vom 0/256 bis 255/256. Als expr im BERT-Kommando ist dabei lediglich der Zähler in den obigen Brüchen anzugeben; expr reicht somit von 0 bis 256. Die 0-1-Flanken des Signals an B6 folgen ca. 56mal pro Sekunde aufeinander. Man sagt auch: Die Impulsfrequenz beträgt ca. 56 Hz. Sie ist also optimal an die Frequenz-Charakteristik von Gleichstrom-Motoren angepaßt. Durch die impulsförmige Ansteuerung wird den Motoren ein wesentlich besseres Anlaufverhalten verliehen, als dies durch eine Spannungs-Steuerung möglich wäre.

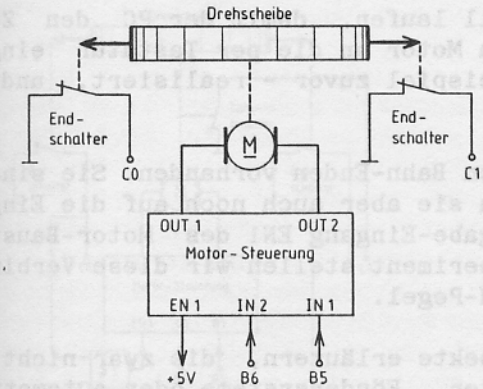
Dadurch, daß der PC sich jetzt nicht mehr um die Impuls-Erzeugung kümmern muß, stört es auch nicht, wenn er eine längere Befehlsfolge zu verarbeiten hat. Der Motor dreht derweil mit der gleichen Geschwindigkeit seine Runden, weil er eine gleichmäßige, ausreichend dichte Pulsfolge erhält.

Positionierung

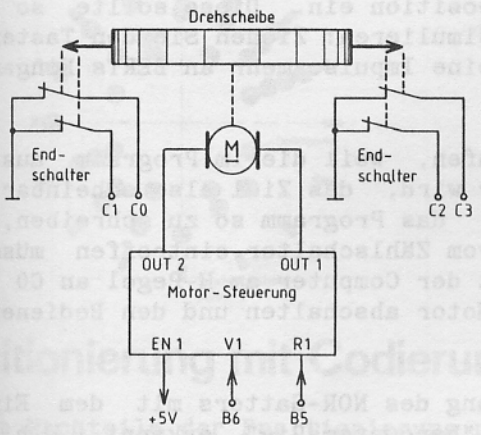
Meist genügt es nicht, einen Motor nur in Bewegung zu setzen. Meist wird mit ihm ein Stellglied angetrieben, das danach eine ganz bestimmte Lage einnehmen soll. Der Motor muß so abgeschaltet werden, daß der Mechanismus genau in der gewünschten Position zur Ruhe kommt.

Nehmen wir das Modell einer Drehscheibe. Mit einem Programm soll die Scheibe mit ihrem Zeiger im Uhrzeigersinn um eine halbe Umdrehung gedreht werden. Dort soll sie stehenbleiben und sich erst nach Tastendruck wieder in entgegengesetzter Richtung um den gleichen Winkel drehen. Am anderen Ende wiederholt sich das gleiche Spiel. Um diese Aufgabe zu lösen, könnten wird die Zeitdauer der Bewegung abstoppen. Das Programm würde dann den Antriebsmotor für diese Zeitspanne aktivieren. Bei allereinfachsten Ansprüchen mag solch ein Verfahren ausreichend sein. Sobald jedoch die Versorgungsspannung des Motors schwankt oder unterschiedliche Reibungsverluste auftreten, wird die einmal ausgemessene Zeit nicht mehr stimmen. Sollte die Scheibe gar in der einen Richtung leichter laufen als in der anderen, so wird die mittlere Position des Zeigers immer weiter in einer Richtung verschoben.

Es führt also kein Weg an einer exakten Positionsmeldung vorbei. Mit zwei Mikroschaltern werden die Endpositionen festgelegt.




```
100 REM DREHSCHEIBE 1
102 REM =============
110 REM DIE DREHSCHEIBE DREHT SICH ZWI-
120 REM SCHEN DEN BEIDEN ENDSTELLUNGEN.
130 REM DIE BEWEGUNG STARTET MIT RETURN
140 REM ANSCHLUSS DER MOTORPLATINE:
150 REM B5 - IN 1
160 REM B6 - IN 2
170 REM EN 1 AN H-PEGEL
180 REM C0 - ENDPOSITION LINKS
190 REM C1 - ENDPOSITION RECHTS
200 PRINT#2,"B=0" :REM MOTOR AUS
210 PRINT"STEUERUNG EINER DREHSCHEIBE"
220 PRINT
230 PRINT"START MIT >RETURN<";
240 INPUT DU$
250 PRINT#2,"B6=1" :REM NACH LINKS
260 PRINT"DREHSCHEIBE NACH LINKS!"
270 PRINT#2,"C0?":INPUT#2,ETL
280 IF ETL=0 THEN GOTO 270
290 PRINT#2,"B6=0" :REM MOTOR AUS
300 PRINT"START MIT >RETURN<";
310 INPUT DU$
320 PRINT#2,"B5=1" :REM NACH RECHTS
330 PRINT"DREHSCHEIBE NACH RECHTS!"
340 PRINT#2,"C1?":INPUT#2,ETR
350 IF ETR=0 THEN GOTO 340
360 PRINT#2,"B5=0" :REM MOTOR AUS
370 GOTO 230
```



Anhand der Pegel C0 und C1 ist im Programm abzufragen, ob die Endpositon erreicht ist. Da wir keine Geschwindigkeitssteuerung einsetzen, ist das Programm übersichtlich. Es besteht aus zwei getrennten Teilen für Hin- und Rücklauf. Wegen der Kurzschlußbremsung mit Hilfe der Motorsteuerplatine bleibt die Scheibe exakt an den vorbestimmten Stellen stehen.

Würde mit einer solchen Steuerung ein Karussell betrieben, dann würden sich die Passagiere sicherlich über das abrupte Abschalten beschweren. Aber auch der Mechanik in einer Förderstrecke oder überhaupt bei Maschinen-Steuerungen bekommt es nicht gut, wenn sie schlagartig abgebremst wird. Das läßt sich vermeiden, wenn man den Motor schon ein Stück vor der angestrebten Halte-Position langsamer schaltet. Auch der Motornachlauf nach dem Halt-Befehl wird bei der geringeren Ausgangsgeschwindigkeit kürzer sein. Das Bild zeigt den veränderten Aufbau mit jeweils zwei Mikroschaltern am Bahn-Ende. Beim Erreichen des ersten Schalters soll die Geschwindigkeit vermindert und beim zweiten Schalter der Motor gestoppt werden. Die Schalter werden an die Eingänge C0 bis C3 angeschlossen. Bei dieser Steuerung muß also wieder eine variable Geschwindigkeit möglich sein.

Positionierung mit Zählern

Das bisher beschriebene Verfahren der Positionierung mit Mikroschaltern ist ausreichend, wenn es nur um eine kleine Anzahl von Positionen geht. Oft aber muß entlang einer Bahn jede beliebige Position erreichbar sein - allerdings mit Einschränkungen, wie sie durch den Quantisierungsfehler gegeben sind. Offensichtlich ist diese Aufgabe nicht mit Positions-Schaltern zu lösen: Einerseits lassen sie sich nicht beliebig dicht aneinanderreihen, und andererseits würde die Anzahl der benötigten Leitungen und Port-Eingänge bei dichter Staffelung jedes vernünftige Maß überschreiten. Wir müssen daher eine Information über den längs der Bahn zurückgelegten Weg verschaffen.

Ein Maß für den zurückgelegten Weg ist die Anzahl der Umdrehungen der Motorwelle oder irgendeiner anderen Welle in der Kraftübertragung. In unserem Versuchsaufbau bauen wir einen Mikroschalter ein, der pro Umdrehung der Antriebswelle zweimal geöffnet wird. Die Zähl-Impulse werden auf den BERT-Anschluß C2 gegeben. Der Zählerstand gibt uns somit die Umdrehungs-Anzahl des Schneckenrades und damit - über die Zähnezahl der Drehscheibe -

deren Position. Die Scheibe darf nicht zu schnell laufen, damit der PC den Zähler korrekt abfragen kann. Das Programm steuert nun den Motor an die per Tastatur eingegebene Stelle. Auch eine Bremsphase wird - wie im Beispiel zuvor - realisiert, und zwar als Brems-Intervall kurz vor der Soll-Position.

Allerdings sind nach wie vor die Mikroschalter an den Bahn-Enden vorhanden. Sie sind mit den Eingängen C0 und C1 verbunden. Zusätzlich werden sie aber auch noch auf die Eingänge eines NOR-Gatters geführt, dessen Ausgang den Freigabe-Eingang EN1 des Motor-Bausteins steuert. Für das nachfolgend beschriebene erste Experiment stellen wir diese Verbindung noch nicht her und legen EN1 statt dessen fest auf H-Pegel.

Wir sollten an dieser Stelle einige Sicherheitsaspekte erläutern, die zwar nicht für unser Modell, wohl aber für große Anlagen wie Roboter, Fördersysteme oder automatische Lager-Einrichtungen wichtig sind.

Verwenden Sie das Programm und geben Sie eine Zielposition ein. Diese sollte so weit entfernt liegen, daß Sie Zeit haben, den Defekt zu simulieren: Ziehen Sie den Taster des Umdrehungszählers von der Scheibe weg. Nun kommen keine Impulse mehr an BERTs Eingang C2 an.

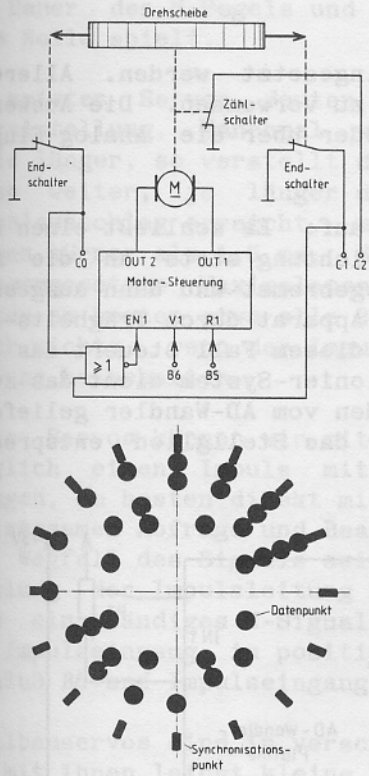
Infolgedessen wird der Motor immerfort weiter laufen, weil die im Programm aus dem Zählerstand abgeleitete Wegstrecke nicht mehr größer wird, das Ziel also scheinbar nie erreicht wird (Wir haben es hier bewußt unterlassen, das Programm so zu schreiben, daß immer innerhalb vorgegebener Zeiten neue Impulse vom Zählschalter eintreffen müssen). Erreicht der Zeiger nun den Endschalter, so erkennt der Computer am H-Pegel an C0 bzw. C1 das Erreichen des Bahn-Endes; er kann nun den Motor abschalten und den Bediener der Anlage durch eine Bildschirm-Meldung warnen.

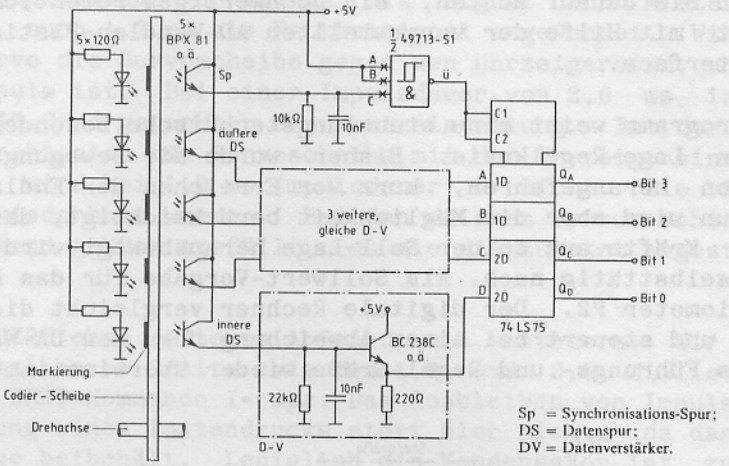
Im nächsten Experiment verbinden wir nun den Ausgang des NOR-Gatters mit dem Eingang EN1. Damit wird das Erreichen des Bahn-Endes auch "hardwaremäßig" erkannt, d.h. der Schutz funktioniert auch noch bei einem Ausfall des Computersystemes. Bringen Sie die Anlage wieder in die Ausgangsposition und starten Sie das gleiche Programm wie zuvor. Bevor die Zielposition erreicht ist, drücken Sie die BREAK- oder STOP-Taste des PCs, um so einen Ausfall des Computers zu simulieren. Der Motor läuft nun unkontrolliert weiter. Sobald jedoch das Bahn-Ende erreicht ist, wird er durch das Signal des Endschalters abgeschaltet.

Übrigens ist der Betrieb des Motors damit verriegelt. Selbst wenn das Programm wieder in Betrieb genommen wird, kann der Motor wegen EN1 = 0 nicht aus dieser Lage loslaufen, auch nicht in der anderen Richtung. Nur durch manuellen Eingriff kann die Endlage wieder verlassen werden. Ein solches Verhalten ist in manchen Anlagen durchaus erwünscht, um zusätzlichen Schaden durch einen vorschnellen Neustart zu verhindern. Wenn BERT ausfällt - Sie können dies durch Drücken seiner Reset-Taste herbeiführen - bleibt in den allermeisten Fällen der Motor ebenfalls stehen.

Nun zu einer dritten Simulation eines Fehlers: Sie starten auch hier wieder die ganze Anlage mit einer entfernten Zielposition. Stellen Sie sich nun vor, die Signal-Leitung des Umdrehungszählers würde beschädigt und ein Kabel vom Endschalter herausgerissen. Probieren Sie aus, was dabei geschieht: Die Anlage reagiert genauso, als hätte die Drehscheibe den Endtaster erreicht. Nun wird auch klar, warum alle Endschalter als Öffner ausgeführt sind: Bei Kabelschäden resultiert ebenso ein H-Pegel wie bei der normalen Kontaktangabe.

Bei der hier vorgestellten Anlage werden die durch die Antriebswelle ausgelösten Impulse gezählt. Weil dabei ein Zähler hochgezählt - inkrementiert - wird, nennt man dieses Verfahren auch Positionierung per Inkremental-Geber. Es hat den Nachteil, daß kein Zählimpuls durch einen kurzzeitigen Ausfall oder unsichere Impulsflanken verloren gehen oder doppelt gezählt werden darf. Ein solcher Fehler kann vom System nicht erkannt werden. Daher ist es angebracht, daß die Anlage nach einer gewissen Zahl von Positionier-Vorgängen stets wieder eine Nullposition anfährt, die sich unabhängig vom Inkremental-Geber durch einen Mikroschalter erkennen läßt. An dieser Stelle wird dann der Zähler wieder auf Null gesetzt. Vorher kann man auch den Zählerstand abfragen und bei zu großen Abweichungen eine Warnmeldung am Bildschirm ausdrucken.





Sp = Synchronisations-Spur;
DS = Datenspur;
DV = Datenverstärker.

Positionierung mit Codierung

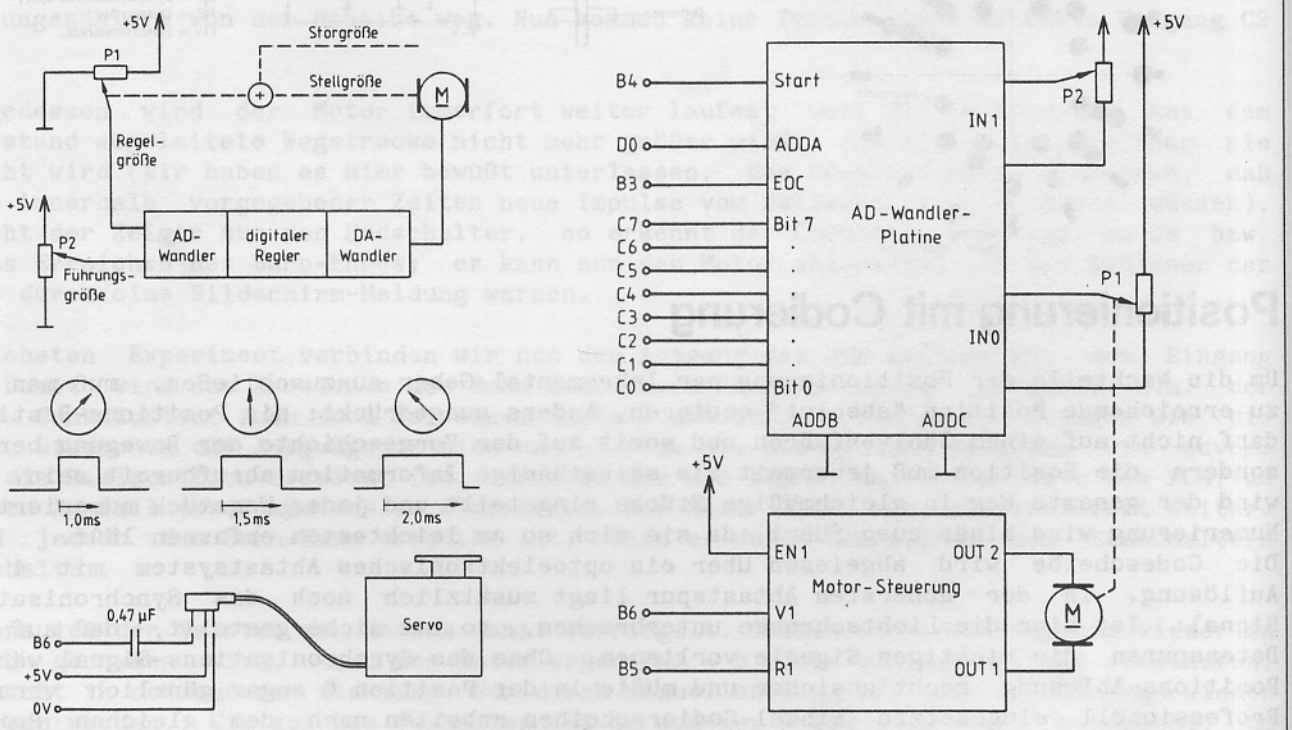
Um die Nachteile der Positionierung per Inkremental-Geber auszuschließen, muß man jede zu erreichende Position "absolut" codieren. Anders ausgedrückt: die Positions-Bestimmung darf nicht auf einem Zählverfahren und somit auf der Vorgeschichte der Bewegung beruhen, sondern die Position muß jederzeit als selbständige Information abrufbereit sein. Dazu wird der gesamte Weg in gleichmäßige Stücke eingeteilt und jedes Wegstück numeriert. Die Numerierung wird binär ausgeführt, da sie sich so am leichtesten erfassen läßt.
Die Codescheibe wird abgelesen über ein optoelektronisches Abtastsystem mit 4 Bits Auflösung. In der äußersten Abtastspur liegt zusätzlich noch das Synchronisations-Signal: Ist hier die Lichtschranke unterbrochen, so ist sichergestellt, daß auf den 4 Datenspuren die richtigen Signale vorliegen. Ohne das Synchronisations-Signal wäre die Positions-Ablesung recht unsicher und müßte in der Position 0 sogar gänzlich versagen. Professionell eingesetzte Winkel-Codierscheiben arbeiten nach dem gleichen Prinzip, jedoch mit wesentlich feinerer Teilung, die bis zu 12 Bits Auflösung gehen kann.

Der Schaltplan zeigt die Auswerte-Schaltung für die Lichtschranken an der Code-Scheibe zu sehen. Das Signal der Fototransistoren wird verstärkt. Ist eine Lichtschranke durch einen aufgeklebten Punkt unterbrochen, dann resultiert am Speicher-Eingang ein L-Pegel; entsprechend erzeugt eine Leerstelle ein H-Signal. Anders ist das bei der Synchronisations-Spur. Der NAND-Schmitt-Trigger-Baustein invertiert das Signal, so daß der Synchronisations-Punkt letztlich einen kurzen H-Impuls zur Übernahme in den Speicherbaustein 74LS75 erzeugt. Wenn dieses H-Signal erscheint, dann liegen die 4 Datensignale bereits stabil an den Speicher-Eingängen an, weil die Klebepunkte auf den Datenspuren größer sind als die Synchronisationspunkte. Der Speicherbaustein hält die Daten bis zum nächsten Synchronisations-Signal. Damit wird der PC bezüglich seiner Arbeitsgeschwindigkeit entlastet, da er die Position während der ganzen Sektordauer, also bis zum Erreichen des nächsten Synchronisations-Signals, abfragen kann. Die Position kann jetzt direkt abgelesen werden. Die Bremsphase wird hier einen Sektor vor Erreichen der Zielposition eingeleitet.
Eine Codescheibe nach diesem Prinzip wird in der Praxis oft mit einem Inkremental-Verfahren kombiniert. Dabei dient sie der Auflösung von Zwischenwerten für die Dauer einer ganzen Umdrehung, während die Umdrehungen separat gezählt werden. Bei unserem Modell beträgt der Vorschub bei einer Wellenumdrehung etwa 6,2 Grad. Mit Hilfe der Codescheibe mit ihren 16 Sektoren können wir somit einen Vorschub von ca. 0,4 Grad auflösen.

Ein Lage-Regelkreis

Als Winkelsensor kann beispielsweise ein Drehwiderstand eingesetzt werden. Allerdings sollten Sie darauf achten, ein hochwertiges Potentiometer zu verwenden. Die Auswertung erfolgt mit Hilfe der vorgestellten AD-Wandler-Platine, oder über die Analog-Eingänge des Interface.

Das Programm weist eine steuerungstechnische Besonderheit auf: Es schließt einen sogenannten Lage-Regelkreis. Bisher wurde die Bewegungs-Einrichtung stets an die Soll-Position herangefahren, kurz vor Erreichen der Endlage abgebremst und dann ausgeschaltet. Nun wird aber die Möglichkeit berücksichtigt, daß der Apparat durch Trägheits- oder äußere Kräfte aus seiner Soll-Lage herausbewegt wird. In diesem Fall steuert das Programm selbsttätig nach. Als Sollwert-Vorgabe für das Positionier-System dient das zweite Potentiometer P2. Der digitale Rechner vergleicht die beiden vom AD-Wandler gelieferten Werte und steuert bei einer Abweichung über den DA-Wandler das Stellglied entsprechend an, bis Führungs- und Regel-Größe wieder übereinstimmen..



Wir bilden hier per Software das Verhalten einer sogenannten Servo-Steuerung nach, wie sie auch im Modellbau verwendet wird. Dabei wird die Motorgeschwindigkeit immer kleiner gewählt, je näher die Ist-Position der Soll-Position kommt. Wenn beide Positionswerte übereinstimmen, wird der Motor schließlich abgeschaltet. Um Regelschwingungen zu verhindern, empfiehlt es sich jedoch, den Motor auch schon in einem bestimmten Intervall um die Soll-Position herum abzuschalten, damit er nicht über den Zielpunkt hinausläuft. Insgesamt ergibt sich für unser Programm der Zusammenhang zwischen der Regelabweichung (Differenz zwischen Sollwert und Istwert) und dem Tastverhältnis, das die Motorgeschwindigkeit steuert. Der Regelkreis kann durch Ändern einiger Parameter im Programm an die Eigenschaften des Motors und der Mechanik angepaßt werden, und zwar durch geeignete Wahl der Werte für das Brems-Intervall und für die Steilheit der Regelkurve.

Steuerung von Servos

Wieder bedingt durch die Arbeitsgeschwindigkeit des BASIC-Interpreters und der V24-Datenübertragung zu BERT wird unser Regelkreis vielleicht nicht so prompt reagieren, wie wir es uns wünschen. Auch hier läßt sich eine Verbesserung finden; diesmal investieren wir in die Hardware. Die sogenannten Modellbau-Servos beinhalten einen Stellmotor, ein Potentiometer zum Erfassen des Stellwinkels und einen speziellen Servoverstärker. Dieser ist genau auf die Eigenschaften des Stellmotors abgestimmt und realisiert bestimmte Werte für Stell-Geschwindigkeit und Stell-Moment. Das Servo erhält die Servo-Information über ein pulsbreiten-moduliertes Signal. Das Bild zeigt den Zusammenhang zwischen

der Dauer des H-Pegels und dem Stellwinkel des Servos, wobei die vorherige Position keine Rolle spielt.

Die meisten Servos deuten eine Impulslänge von 1,5 ms als Signal zur Mitten- oder Neutralstellung. Manchmal werden auch 1,3 ms als Neutralsignal angewendet. Wird der Impuls länger, so verstellt das Servo die Servoscheibe gegen den Uhrzeigersinn, und zwar um so weiter, je länger der Impuls ist. Bei einer Impulsdauer von 2,0 ms ist der Maximalausschlag erreicht, meist 45 Grad gegenüber der Neutralstellung. Ist der Impuls dagegen kürzer als 1,5 ms, dann dreht sich die Servoscheibe mit dem Uhrzeigersinn. Der entgegengesetzte Maximalausschlag wird bei 1,0 ms Impulsdauer erreicht. Die Impulse sollten am besten etwa alle 20 ms wiederholt werden. Bei den meisten Servos schadet es jedoch nichts, wenn der Impulsabstand größer ausfällt, da sie die zuletzt angesteuerte Stellung beibehalten.

Solche Servos können wir mit dem BERT ohne weiteren Aufwand auch ansteuern. Wir müssen lediglich einen Impuls mit einstellbarer Breite zwischen 1,0 und 2,0 Millisekunden erzeugen, am besten direkt mit dem BERT-Kommando I=expr. Das Ausbleiben von Impulsen bei der langsamen Abfrage und Bearbeitung eines Tastendrucks stört hier nicht, da das Servo beim Wegfall des Signals seine Lage beibehält. Lediglich ein Kondensator ist zur Ankoppelung der Impulsleitung notwendig, da einige Servos sonst bei einem Reset von BERT durch ein ständiges H-Signal an den Endanschlag laufen. Die Schaltung gilt für Servos mit Impulseingang in positiver Logik. Bei negativer Logik ist ein Inverter zwischen Anschluß B6 und Impulseingang vorzusehen.

Modellbauservos sind in verschiedenen Größen und Stellmomenten erhältlich; daher lassen sich mit ihnen leicht kleine Roboter aufbauen. Zum Ansteuern mehrer Servos mit BERT wird eine Auswahllogik mit UND-Gattern aufgebaut. Das Auswahlsignal kann z.B. an Port A oder D erzeugt werden, während der Impuls nach wie vor am Zeitgeberausgang B6 erzeugt wird.

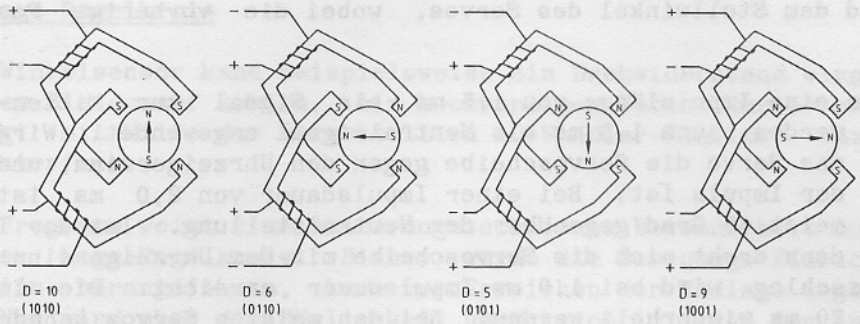
Der Schrittmotor

Schrittmotoren werden in computergesteuerten Postitionieraufgaben häufig eingesetzt. Für die Spur-Positionierung im Diskettenlaufwerk sowie für die Druckkopf-Bewegung und den Papiertransport im Drucker. Schrittmotoren sind deswegen so beliebt, weil sie Antrieb und Positionierung in sich vereinen. Ein Schrittmotor besitzt dazu in der einfachsten denkbaren Ausführung zwei Spulensysteme. Wer einen kleinen Kompaß besitzt, kann die hier beschriebenen Experimente nachvollziehen, nämlich den Kompaß in einen Schrittmotor umbauen. Dazu wird der Kompaß in zwei zueinander senkrechten Richtungen mit jeweils 100 Windungen Kupfer-Lackdraht (CuL) umwickelt.

In Reihe zu diesen beiden Elektromageten wird je ein 100-Ohm-Leistungswiderstand geschaltet, damit die Kompaßnadel nicht durch ein zu starkes Magnetfeld in wilde Schwingungen gerät. Die zwei Spulen mit den Vorwiderständen werden mit den beiden Ausgängen (OUT1-OUT2 und OUT3-OUT4) der Motorsteuerung verbunden. Schalten wir einen Ausgang ein, so wird durch die Spule ein Strom fließen, der ein Magnetfeld erzeugt. Die Kompaßnadel, die bislang nach Norden zeigte, wird sich nach diesem stärkeren Magnetfeld ausrichten:

Polen wir den Strom um, so dreht sich die Kompaßnadel natürlich in die entgegengesetzte Richtung. Es ist aber mehr oder weniger zufällig, ob sie dazu eine Rechts- oder eine Links-Drehung vollführt. Um zu einer kontrollierten Drehung zu gelangen (wir wollen ja einen Motor bauen!), bringen wir den zweiten Elektromagneten ins Spiel.

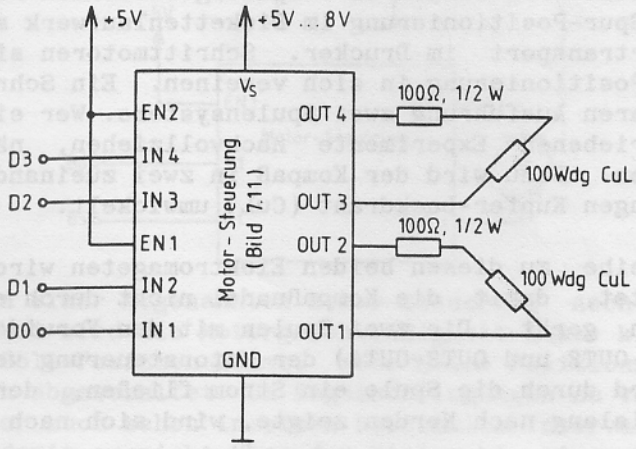
Wir haben in den Zeilen 280 bis 310 vier Bitmuster festgelegt. Man spricht hier auch von sogenannten "Phasenmustern" für die beiden Spulen. Beim Drücken einer Taste bzw. nach einer Verzögerungs-Routine im Programm wird jeweils das nächste Phasenmuster ausgegeben. So werden nacheinander die Magnetfeld-Anordnungen erzeugt.



Wie in den Abbildungen zu erkennen ist, orientiert sich die Kompaßnadel jetzt nicht mehr am Feld eines Elektromagneten allein, sondern stellt sich auf die Winkelhalbierende zwischen den beiden Spulen ein. Nach den vier Phasenschritten hat die Kompaßnadel genau eine Umdrehung zurückgelegt. Es kann sein, daß in Ihrem Aufbau eine andere Drehrichtung zu beobachten ist; dann müssen Sie eine der beiden Spulen umpolen. Wir sehen also, wie wir durch programmierten Wechsel der Phasenlagen an den beiden Spulensystemen zu einer fortschreitenden Drehung der Kompaßnadel bzw. der Schrittmotor-Welle gelangen. Durchschreiten wir die Phasenmuster in umgekehrter Reihenfolge, so läuft der Motor natürlich in der anderen Drehrichtung.

Das eben besprochene Programm können wir ebensogut zum Ansteuern eines "richtigen" Schrittmotors verwenden. Insbesondere die Schrittmotoren im Plotter-Bausatz von Fischer-Technik können wir an unsere Motorsteuer-Platine anschließen. Der Vorschaltwiderstand von 100 Ohm entfällt dabei. Es wird Sie vielleicht überraschen, daß dieser Motor wesentlich kleinere Winkelschritte durchführt als die Kompaßnadel: Erst nach 48 Schritten gelangt er wieder in die Ausgangslage zurück, d.h. die Phasenmuster werden für eine Umdrehung insgesamt 12mal durchlaufen. Der Grund hierfür liegt im komplizierteren Aufbau des Magnetsystems, das diesem Motor neben den kleineren Winkelschritten auch ein besseres Drehmoment verleiht. Das Prinzip ist aber genau das gleiche wie imn Kompaßmodell.

```
100 REM SCHRITTMOTORSTEUERUNG.B
102 REM ======================
110 REM STEUERUNG EINES UNIPOLAREN
120 REM SCHRITTMOTORS DURCH DIE BERT-
130 REM KOMMANDOS SO=... UND SP=...
140 REM DIE FREQUENZ STEIGT BIS ZUR
150 REM GRENZFREQUENZ, WENN MOEGLICH.
160 REM ANSCHLUSS DER SCHRITTMOTORPLAT.
170 REM D0,D2,D4,D6 JEWEILS TAKT
180 REM D1,D3,D5,D7 JEWEILS RICHTUNG
190 REM JE NACH WAHL DER MOTORNUMMER.
200 REM FREQUENZ -> PULSLAENGE
210 DEF FNF(F)=INT(1/(F*227.76E-6))
220 PRINT"SCHRITTMOTOR-STEUERUNG"
230 PRINT
240 INPUT"WELCHER MOTOR (0-3)";N
250 IF N<0 OR N>3 THEN GOTO 240
260 INPUT"WIEVIELE SCHRITTE  ";Z
270 INPUT"START-STOP-FREQUENZ";F1
280 INPUT"GRENZFREQUENZ      ";F2
290 F1=FNF(F1)
300 F2=FNF(F2)
```



Der hier verwendete Schrittmotor wird auch als bipolarer Schrittmotor bezeichnet. Bipolar ("zweipolig") deswegen, weil zum Betrieb die Polarität der Spulenströme gewechselt wird. Eine Alternative sind unipolare ("einpolige) Schrittmotoren. Bei diesen wird an einem zusätzlichen Anschluß in jeder Spulenmitte die Versorgungs-Spannung des Motors angelegt. Je nach Schalterstellung wird nun bei jeder Spule ein Ende auf Masse gelegt und damit eine der beiden Spulenhälften von Strom durchflossen. Entsprechend ergibt sich die Orientierung des Magnetfeldes. Unipolare Schrittmotoren haben den Vorteil, daß zu Ihrem Betrieb vier einfache Transistorschalter genügen, während beim bipolaren Schrittmotor zwei Brückenschaltung erforderlich sind. Da jedoch von einer Spule immer nur eine Hälfte stromdurchflossen ist, liefert der unipolare Schrittmotor bei gleicher Baugröße eine geringere Leistung als ein bipolarer.

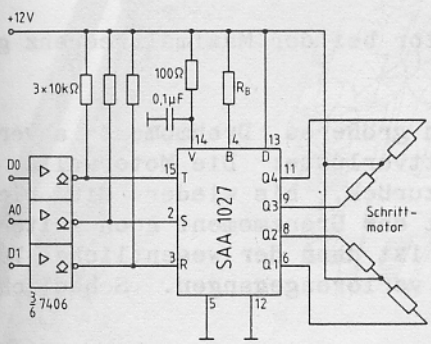
Auch zum Ansteuern unipolarer Schrittmotoren gibt es komplette integrierte Schaltkreise. Der Baustein SAA1027 ist recht komfortabel, da er die erforderlichen Phasenmuster

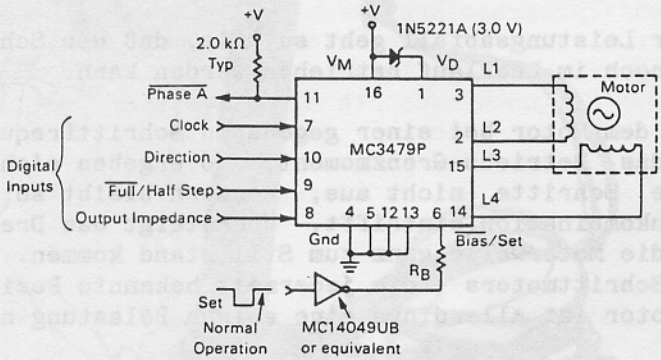
selbsttätig erzeugt. Zum Betrieb genügen zwei Signalleitungen: Takt und Richtung. Mit jeder 0-1-Flanke am Takt-Eingang T wird der nächste Schritt bewirkt. Die Drehrichtung hängt dabei vom Pegel am Richtungs-Eingang R ab. Zusätzlich kann mit einem L-Pegel am Setz-Eingang S ein vom Hersteller festgelegtes Phasenmuster (Q1 = L, Q2 = H, Q3 = L, Q4 = H) an die Ausgänge des Schaltkreises angelegt werden. Dies wird man in erster Linie zu Beginn des Betriebes tun, um wohldefinierte Anfangsverhältnisse einzustellen. Mit Hilfe des Bausteins SAA1027 ist die Programmierung des Motors erheblich einfacher geworden.

Auch einen unipolaren Schrittmotor wollen wir als Antrieb und Positionierung an unserer Drehscheibe einsetzen. Da sich die Motorwelle des Schrittmotors in Schritten von 7,5 Grad einstellen läßt, ist ein Untersetzungs-Getriebe wie bei Gleichstrom-Motoren nicht notwendig. Auch dies ist ein Vorteil des Schrittmotors. Seine Welle wird direkt mit dem Schneckenrad verbunden, allerdings elastisch; denn mit einer starren Kupplung würde das ruckartige Fortschreiten der Schrittmotorwelle nicht gedämpft und könnte Beschädigungen der Mechanik verursachen. Außerdem können sich Eigenschwingungen der Mechanik und des Motors nicht so leicht gegenseitig aufschaukeln und dadurch Schrittverluste verursachen. Letztlich wird auch das Betriebsgeräusch durch die elastische Kopplung deutlich vermindert.

Der Baustein SAA1027 kann an jedem seiner Ausgänge Q1 bis Q4 maximal 350mA schalten. Bei dem oben genannten Motor fließen jedoch nur 175mA im geschalteten Spulenstrang, so daß noch genügend Reserven verbleiben. Der Widerstand RB muß in diesem Fall mit 180 Ohm und 2/3 Watt dimensioniert sein.

Das Programm zum Betrieb des bipolaren Schrittmotors können wir jetzt vereinfachen. Den Setzeingang S steuern wir durch den Anschluß A0 an. Zu Beginn des eigentlichen Steuerungs-Programms wird der Schrittmotor initialisiert. Danach müssen an D0 nur noch die Taktsignale erzeugt werden.





Die BERT-Kommandos für den Schrittmotor-Betrieb

Betrieb und Programmierung unipolarer Schrittmotoren können wir noch komfortabler gestalten. Der besondere Vorteil ist, daß Sie sich um die Erzeugung der Schritte überhaupt nicht kümmern müssen: Sie geben nur die Anzahl der Schritte und ihre Dauer vor, und alles andere erledigt BERT. Er erzeugt im vorgegebenen Zeittakt die notwendigen Ausgangssignale und zählt die Schrittzahlregister herunter. Beim Zählerstand 0 werden keine weiteren Impulse mehr erzeugt; der Motor bleibt somit stehen. Während all dieser Aktivitäten von BERT kann der PC andere Berechnungen vornehmen und gelegentlich den aktuellen Stand des Schrittzahlregisters in BERT abfragen. Ist dieser bei Null angelangt, so kann der nächste Auftrag abgesetzt werden.

Im BERT-Kommando SP = expr gibt expr die Zeit an, während der das Phasenmuster an den Ausgängen stehenbleibt, bis das nächste ausgegeben wird. Die Schrittdauer wird hierbei in Zeitgeber-Zyklen zu 277,76 Mikrosekunden angegeben (256*1,085 µs). Der Wertebereich reicht von 1 bis 64. Will man die Frequenz des Schrittmotors wissen, so muß man das Reziproke der Pulsdauer berechnen.

Dabei gibt n die Nummer des Motors an. Es wird also in der Beispiel-Zeile das Schrittzahl-Register 0 auf 230 Schritte gesetzt. Aus dem Kommando geht für BERT auch hervor, daß unipolare Schrittmotoren über die zwei Signale Takt und Richtung, z.B. über den Schaltkreis SAA1027, angesteuert werden sollen. Insgesamt vier Motoren können gesteuert, denen jeweils zwei Anschlüsse zugeordnet sind: D0 und D1 für Richtung und Takt des Motors 0, entsprechend D2 und D3 für Motor 1, D4 und D5 für Motor 2, D6 und D7 für Motor 3. Ist expr negativ, so erscheint ein H-Pegel am Richtungsausgang (im Beispiel an D0). Das Absetzen des Kommandos an BERT startet gleichzeitig den Ablauf des Taktsignals. Zuvor mußte natürlich mit SP = expr die Taktdauer festgelegt werden.

Der Setzeingang S des Schrittmotor-Bausteins wird durch die BERT-Kommandos nicht bedient; er läßt sich mit Port A steuern. Da in aller Regel auch nur einmal zum Programmstart initialisiert werden muß, stellt dies auch kein zeitkritisches Unterfangen dar.

Schrittzahl-Register abfragen:
Während die Schrittmotoren laufen, kann man mit diesem Kommando den Stand eines beliebigen Schrittzahl-Registers jederzeit abfragen, ohne den Betrieb in irgendeiner Weise zu beeinflussen. Ist es bei Null angelangt, so ist der Bewegungsablauf zu Ende. Außerdem muß beachtet werden, daß BERT, nachdem er über das Vorzeichen von expr den Drehsinn des Motors festgelegt hat, den Absolutbetrag der Schrittzahl berechnet. Daher wird das Abfrageergebnis immer positiv (oder Null) sein, auch wenn eine negative Schrittzahl vorgegeben wurde.

Das Frequenz-Verhalten von Schrittmotoren
Mit den BERT-Kommandos können wir die Schrittmotoren in einem weiten Drehzahlbereich steuern; insbesondere ist je eine recht schnelle Schrittfolge möglich, weil wir ja den Timer im Z8 benutzen. Aus den technischen Gegebenheiten der Schrittmotoren ergeben sich dabei jedoch bestimmte Grenzen, die unbedingt einzuhalten sind. Mit zunehmender Schrittfrequenz sinkt das Drehmoment, das der Schrittmotor abgeben kann.

Dieser Leistungsabfall geht so weit, daß der Schrittmotor bei der Maximalfrequenz gerade eben noch im Leerlauf betrieben werden kann.

Wird dem Motor bei einer gegebenen Schrittfrequenz ein größeres Drehmoment abverlangt als das Betriebs-Grenzmoment, so ergeben sich Schrittverluste: Die Motorwelle führt einige Schritte nicht aus, sondern bleibt so lange zurück, bis wieder die richtige Phasenkombination eintrifft. Übersteigt das Drehmoment das Grenzmoment noch weiter, so wird die Motorwelle ganz zum Stillstand kommen. Damit ist dann der wesentliche Vorteil des Schrittmotors - die jederzeit bekannte Position - verlorengegangen. Schädlich für den Motor ist allerdings eine solche Belastung nicht.

Will man den Betriebsbereich von Schrittmotoren ausnutzen, so darf die Schrittfrequenz nur langsam von kleinen Werten aus hochgefahren werden. Rechtzeitig vor dem Anhalten muß sie dann wieder reduziert werden, um das Überspringen zu verhindern.

Das Programm zeigt ein Beispiel dafür. Mit Programmen wie diesem können Sie den von Ihnen verwendeten Schrittmotor ausgiebig testen und den Steuerungs-Ablauf ganz an die mechanischen Gegebenheiten der gesamten Apparatur anpassen.

```
100 REM SCHRITTMOTORSTEUERUNG.B
102 REM ======================
110 REM STEUERUNG EINES UNIPOLAREN
120 REM SCHRITTMOTORS DURCH DIE BERT-
130 REM KOMMANDOS SO=... UND SP=...
140 REM DIE FREQUENZ STEIGT BIS ZUR
150 REM GRENZFREQUENZ, WENN MOEGLICH.
160 REM ANSCHLUSS DER SCHRITTMOTORPLAT.
170 REM D0,D2,D4,D6 JEWEILS TAKT
180 REM D1,D3,D5,D7 JEWEILS RICHTUNG
190 REM JE NACH WAHL DER MOTORNUMMER.
200 REM FREQUENZ -> PULSLAENGE
210 DEF FNF(F)=INT(1/(F*227.76E-6))
220 PRINT"SCHRITTMOTOR-STEUERUNG"
230 PRINT
240 INPUT"WELCHER MOTOR (0-3)";N
250 IF N<0 OR N>3 THEN GOTO 240
260 INPUT"WIEVIELE SCHRITTE  ";Z
270 INPUT"START-STOP-FREQUENZ";F1
280 INPUT"GRENZFREQUENZ      ";F2
290 F1=FNF(F1)
300 F2=FNF(F2)
310 IF F1>64 THEN F1=64
320 IF F2>64 THEN F2=64
330 INPUT"BESCHLEUNIGUNG (Z.B.5)";FI
340 F=F1
350 PRINT#2,"SP=";F
355 FORI=1 TO 100:NEXT
360 PRINT#2,"S";N;"=";Z
365 FORI=1 TO 100:NEXT
370 PRINT"HOCHLAUF"
380 PRINT#2,"S";N;"?":INPUT#2,ZA
390 REM HOECHSTENS BIS GRENZFREQUENZ
400 REM ODER HALBE SCHRITTZAHL
410 IF(F<F2+FI)OR(ZA<0.5*Z) THENGOTO460
420 F=F-FI
430 PRINT#2,"SP=";F
440 PRINT"FREQU.=";FNF(F),"SCHRITT=";ZA
450 GOTO 380
460 ZE=Z-ZA
470 PRINT"MAX. GESCHWINDIGKEIT"
480 PRINT#2,"S";N;"?":INPUT#2,ZA
490 PRINT"FREQU.=";FNF(F),"SCHRITT=";ZA
500 IF ZA>ZE THEN GOTO 480
510 PRINT"ABBREMSEN"
520 PRINT#2,"S";N;"?":INPUT#2,ZA
530 F=F+FI
540 IF F>F1 THEN F=F1
550 PRINT#2,"SP=";F
560 PRINT"FREQU.=";FNF(F),"SCHRITT=";ZA
570 IF ZA>0 THEN GOTO 520
580 PRINT"ANGEKOMMEN!"
```

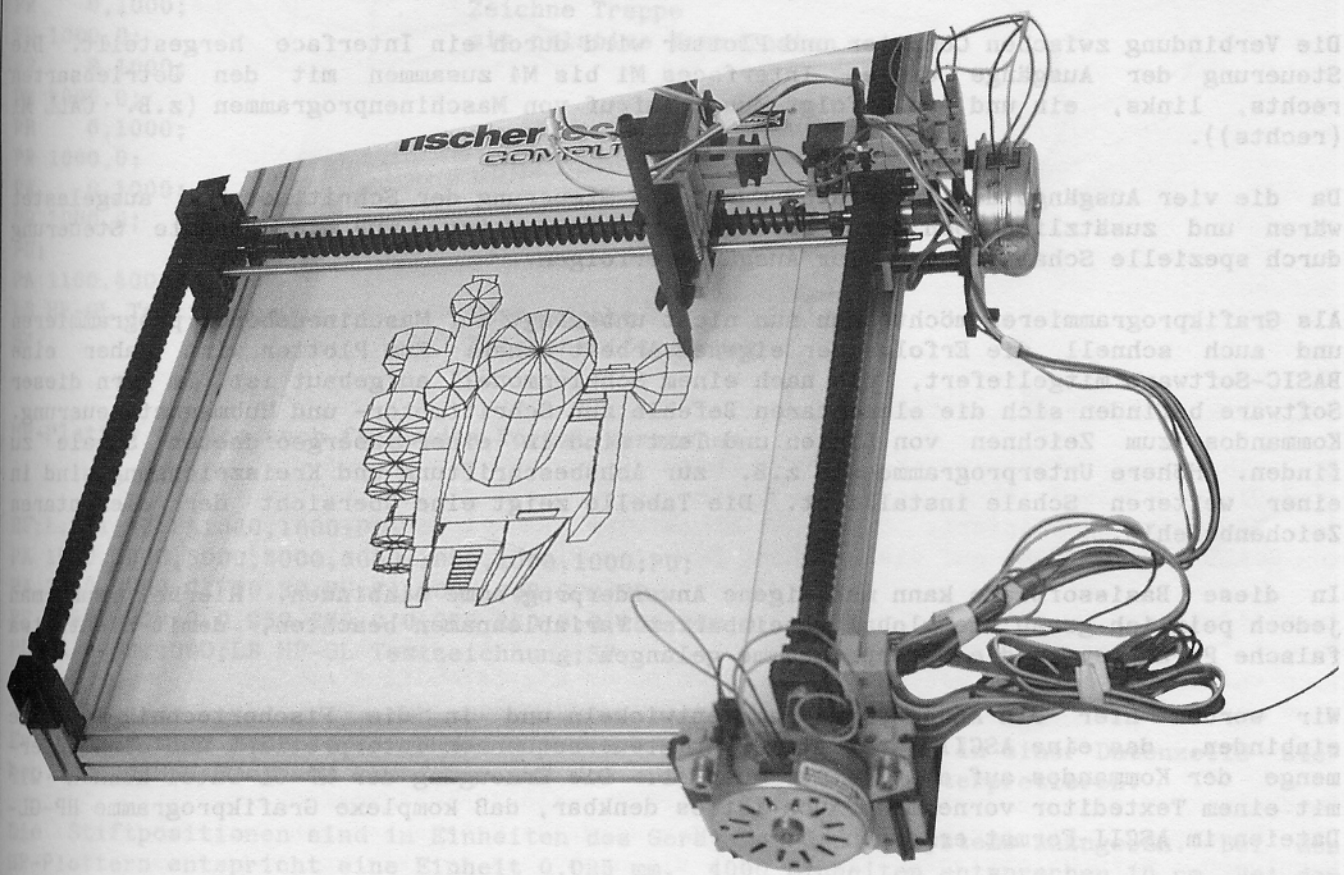
Gerätesimulation mit dem Plotter

Mit dem Fischertechnik-Plotter ist ein didaktisch wertvoller Modellbausatz auf dem Markt, der den Computerfreunden die Prinzipien der Steuerung professioneller Geräte vermittelt.

Darüberhinaus ist das Modell voll funktionsfähig und kann durch die mitgelieferte BASIC-Software vom Benutzer mit eigenen Programmen gesteuert werden.

Anspruchsvolle Grafikprogrammierer können das Modell zur Simulation anderer Geräte nutzen und so ihre Programme ohne Verfügbarkeit aufwendiger Hardware testen.

Der vorliegende Beitrag beschreibt ein Verfahren, mit dem die professionellen Hewlett-Packard-Plotter auf dem Fischertechnik Plotter simuliert werden. Das auf Diskette erhältliche BASIC-Programm enthält die Einbindung einer Untermenge der HP-GL-Plotterbefehle in die Anwenderschale der Fischertechnik-Software. ASCII-Dateien mit HP-GL-Befehlssequenzen werden in grafischer Form auf dem Plotter zu Papier gebracht.



Was der Plotter kann

Der Zeichenstift eines Plotters kann in Einzelschritten in sechs Richtungen bewegt werden. Das sind die positiven und negativen Fahrbewegungen entlang der Plotterachsen sowie Heben und Senken des Zeichenstiftes. Durch programmgesteuerte Synchronisation dieser elementaren Bewegungen entstehen die grafischen Ausgaben komplexer Computerzeichnungen.

Die Verwendung unterschiedlicher Zeichenstifte kann mit dem Fischertechnik-Plotter durch stiftweise Abarbeitung der Plotterausgaben softwaregesteuert realisiert werden.

Zum Antrieb der Achsen finden Schrittmotoren Verwendung. Die Hebebewegung des Zeichenstifts erfolgt durch einen Elektromagneten. Ein Schrittmotor hat zwei unabhängig gesteuerte um 90 Grad versetzte Magnetsysteme. Zwischen den Magnetpolen befindet sich ein Dauermagnet, der mit der Motorwelle verbunden ist.

Durch die Ströme von M1 und M2 wird in den Schrittmotoren ein Magnetfeld erzeugt, bei dem die Nordpole auf der unteren und die Südpole auf der oberen Seite liegen. Die Motorwelle stellt sich entsprechend dem Magnetfeld an beiden Spulen ein.

Zur Motorbewegung ist das Muster der durch M1 und M2 erzeugten Magnetfelder zu verändern. Mit jedem Polrichtungswechsel dreht sich die Welle ein Stückchen weiter.

Die Magnetsysteme des Schrittmotors beim Plotter sind mit einer größeren Polzahl versehen, so daß der Schrittmotor Schritte von 7,5 grad ausführt und eine volle Umdrehung erst nach 48 Schritten erreicht.

Führt man diese Bewegungen rechnergesteuert durch, dann ist jederzeit die Position der angetriebenen Mechanik bekannt und es sind relative Fahrbewegungen mit Schrittmotoren ohne zusätzliche Positionsmelder durchführbar.

Die Verbindung zwischen Computer und Plotter wird durch ein Interface hergestellt. Die Steuerung der Ausgänge dieses Interfaces M1 bis M4 zusammen mit den Betriebsarten rechts, links, ein und aus erfolgt durch Aufruf von Maschinenprogrammen (z.B. CALL M1 (rechts)).

Da die vier Ausgänge des Interfaces durch die Steuerung der Schrittmotoren ausgelastet wären und zusätzlich noch der Hebevorgang zu steuern ist, muß die gesamte Steuerung durch spezielle Schaltung der vier Ausgänge erfolgen.

Als Grafikprogrammierer möchte man nun nicht unbedingt auf Maschinenebene programmieren und auch schnell die Erfolge der eigenen Arbeit sehen. Zum Plotter wird daher eine BASIC-Software mitgeliefert, die nach einem Schalenmodell aufgebaut ist. Im Kern dieser Software befinden sich die elementaren Befehle zur Schrittmotor- und Hubmagnetsteuerung. Kommandos zum Zeichnen von Linien und Text sind in einer übergeordneten Schale zu finden. Höhere Unterprogramme wie z.B. zur Achsbeschriftung und Kreiszeichnung sind in einer weiteren Schale installiert. Die Tabelle zeigt eine Übersicht der elementaren Zeichenbefehle.

In diese Basissoftware kann man eigene Anwenderprogramme einbinden. Hierbei muß man jedoch peinlich genau die global vereinbarten Variablennamen beachten, damit nicht etwa falsche Parameter in die Unterprogramme gelangen.

Wir werden hier ein Anwenderprogramm entwickeln und in die Fischertechnik-Software einbinden, das eine ASCII-Datei mit HP-GL Steuersequenzen interpretiert und eine Untermenge der Kommandos auf dem Plotter ausgibt. Die Erzeugung der HP-GL-Datei können wir mit einem Texteditor vornehmen. Auch ist es denkbar, daß komplexe Grafikprogramme HP-GL-Dateien im ASCII-Format erzeugen.

HP-GL eine grafische Kommandosprache

Hewlett-Packards Graphic-Language (HP-GL) ist eine Grafiksprache bestehend aus einem mnemotechnischen Zweibuchstabencode und den dazugehörigen Parametern. Ein Komma dient als Trennzeichen zwischen den Parametern, ein Semikolon kennzeichnet einen Befehlsabschluß. Heute hat sich HP-GL als Quasistandard durchgesetzt. Fast alle gängigen Plottermodelle im Lowcost-Bereich verfügen ebenfalls über den HP-GL-Sprachumfang, sind also zu HP-Plottern kompatibel.

Die Datei zur Zeichnung eines Testbildes enthält folgende Befehle :

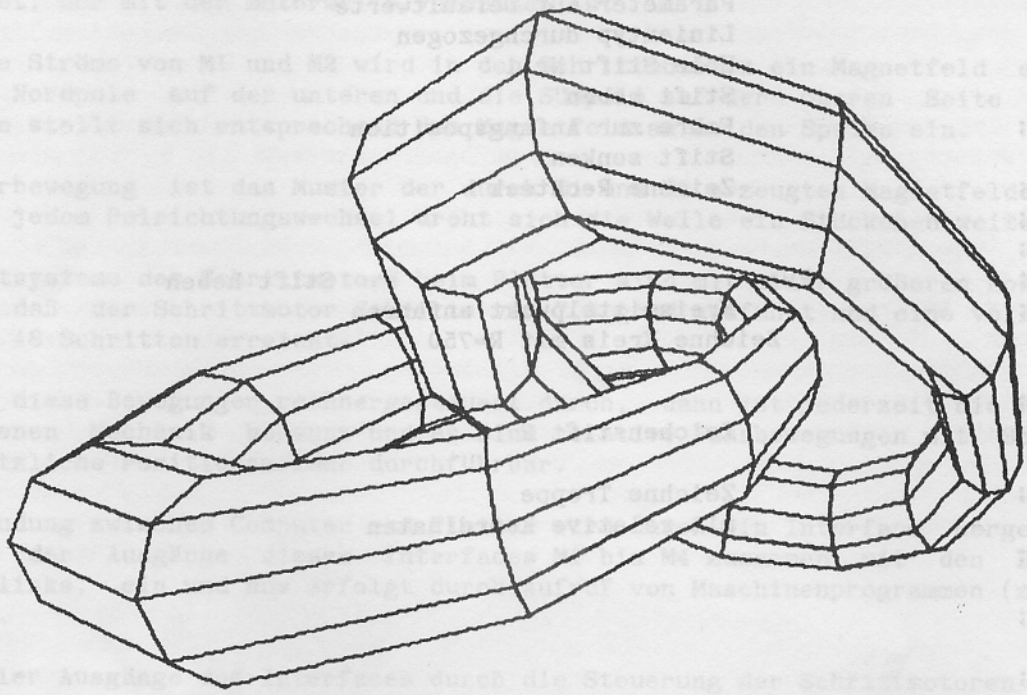
```
DF;                          Parameter auf Defaultwerte
LT;                          Linientyp durchgezogen
SP1;                         Hole Stift Nr.1
PU;                          Stift heben
PA 1000,1000;                Fahre zur Anfangsposition
PD;                          Stift senken
PA 1000,5000;                Zeichne Rechteck
PA 5000,5000;
PA 5000,1000;
PA 1000,1000;          PU;                        Stift heben
PA 3000,3000;                Kreismittelpunkt anfahren
CI 750,30;                 Zeichne Kreis mit R=750
PU;
PA 1000,1000;
SP2;                         Zeichenstift 2
PD;
PR    0,1000;                Zeichne Treppe
PR 1000,0;                   mit relative Koordinaten
PR    0,1000;
PR 1000,0;
PR    0,1000;
PR 1000,0;
PR    0,1000;
PR 1000,0;
PU;
PA 1100,4000;
LB HP-GL Testzeichnung       Textausgabe
SP;                          Zeichenstift zurück
```

HP-Plotter würden auch folgendes Format verstehen :

```
DF;L;SP1;PU;PA1000,1000;PD;
PA 1000,5000,5000,5000,5000,1000,1000,1000;PU;
PA 3000,3000;CI750,30;PU;PA1000,1000;SP2;PD;
PR 0,250,250,0,0,250,250,0,0,250,250,0,0,250,250,0;
PU;PA 1100,4000;LB HP-GL Testzeichnung;SP;
```

Es können also mehrere Kommandos mit mehreren Parameterfolgen in einer Datenzeile stehen. Das BASIC-Programm kann aber nur das erste Datenformat interpretieren.

Die Stiftpositionen sind in Einheiten des Gerätekoordinatensystems anzugeben. Bei den HP-Plottern entspricht eine Einheit 0,025 mm. 4000 Einheiten entsprechen 10 cm. Bei den A3-Plottern ist also ein Bereich von etwa 16000 x 12000 Plottereinheiten verfügbar. Der Fischertechnik-Plotter hat eine maximale Zeichenfläche von 680 x 500 Einheiten. Hier beträgt die Auflösung etwa 0,5 mm. Die unterschiedlichen Zeichenbereiche müssen durch eine Skalierung berücksichtigt werden.



Das BASIC-Programm zur Interpretation von HP-GL-Dateien

Ab Zeile 1000 der Fischertechnik-BASIC-Software beginnt das Anwenderprogramm. Zunächst wird die maximale Zeichenfläche, der linke untere und der rechte obere Begrenzungspunkt, des HP-Plotters definiert. Dieses Rechteck wird dann linear durch Unterprogrammaufruf von SCALE (GOSUB 4300) auf die Zeichenfläche des Fischertechnik abgebildet.

Zusätzlich zur Plotterdarstellung ist noch die Abbildung der HP-GL-Datei auf dem IBM Farbgrafikmonitor vorgesehen. Das Anwenderprogramm ist demnach auch als Plottersimulation auf dem Bildschirm geeignet. Im Anschluß an die Definition einiger Statusvariablen wird der Name der HP-GL-Datei eingegeben.

Abhängig vom eingelesenen HP-GL-Kommando erfolgt die Verzweigung zur weiteren Eingabe der Parameter. Bei den Plotbefehlen werden unnötige Hebebewegungen des Zeichenstiftes vermieden. Berücksichtigung findet auch die nach oben negativ zeigende y-Achsrichtung des Monitors.

Das abgebildete BASIC-Listing ist reichlich mit Kommentaren versehen, so daß es sehr leicht möglich ist, die Interpretation weiterer HP-GL-Steuersequenzen in das Programm aufzunehmen. Der Benutzer eines monochromen Monitors muß die Farbgrafikbefehle modifizieren.

Das Bild zeigt die Ausgabe der obigen Datei. Wer diese Testzeichnung z.B. mit dem COPY-Befehl auf einem HP-Plotter ausgeben möchte, muß nach dem LB-Kommando auf den entsprechenden Terminator (chr$(3)) achten, sonst werden die nachfolgenden HP-Kommandos nicht mehr ausgeführt, sondern als Text interpretiert.

Günter Pomaska

Fischertechnik BASIC Plottersoftware
Elementare Zeichenbefehle

Unterprogramm Einsprung-zeile	Funktion globale Parameter	HP-GL Syntax
HOME 40000	Fährt den gehobenen Zeichenstift in den physikalischen Nullpunkt des Plotters keine Parameter	IN; DF;
LINE 44000	Zeichnet eine Linie von der aktuellen Plotterposition zum Zielpunkt Parameter : X,Y ... Zielpunktkoordinaten absolut	PD x,y; PA x,y;
RLINE 46000	Zeichnet eine Linie von der aktuellen Plotterposition zum Zielpunkt Parameter : X,Y ... Zielpunktkoordinaten relativ zur vorherigen Stiftposition	PR x,y;
MOVE 45000	Bewegt den Zeichenstift in gehobener Position zum Zielpunkt Parameter : X,Y ... Zielpunktkoordinaten	PU x,y;
RMOVE 47000	Bewegt den Zeichenstift in gehobener Position zum Zielpunkt Parameter : X,Y ... Zielpunktkoordinaten relativ zur vorherigen Stiftposition	PU; PR x,y;
SET ORIGIN 42000	Nullpunkt des Plotterkoordinatensystems neu definieren Parameter : X,Y ... neuer Nullpunkt	
ORIGIN 41000	Fahrbewegung mit gehobenem Zeichenstift zum Nullpunkt keine Parameter	PU 0,0;
CHAR 48000	Zeichnet die angegebene Characterfolge Parameter : X,Y ... Position des Textes linksbündig A$ .... auszugebende Zeichenkette R ..... Beschriftungsrichtung G ..... Texthöhe	PU x,y; DI a,b; SI w,h; LB text;

```
RCHAR           wie CHAR jedoch mit relativer Positions-
49000           angabe zur vorherigen Stiftposition


SCALE           Skalierungsfaktoren für die Koordinaten-        SC x1,y1,
43000           achsen                                             x2,y2;

                Parameter : X1 ... rechter Rand
                            X2 ... linker Rand
                            Y1 ... unterer Rand
                            Y2 ... oberer Rand
```

Tabelle 1 : Elementare Zeichenprogramme der Fischertechnik Plottersoftware mit den äquivalenten Befehlen von HP - GL

```
1000 REM
1010 REM ************************************************
1020 REM ***          Beginn des Benutzerprogramms       ***
1021 REM ***                                             ***
1030 REM *** HP-GL Simulation mit Fischertechnik Plotter ***
1031 REM ***    mit Bildschirmgrafik fuer IBM-PC / CGA   ***
1032 REM ***                                             ***
1033 REM ************************************************
1034 REM
1040 REM Skalierung Plotter  Hewlett-Packard / Fischer
1041 X1=0
1042 X2=680
1043 Y1=0
1044 Y2=500
1045 GOSUB 43000
1049 REM Umrechnungsfaktoren IBM-Bildschirm / Plotter
1050 FX=320/X2
1060 FY=200/Y2
1061 REM XPOS, YPOS ist die aktuelle Stiftposition
1062 XPOS = 0
1063 YPOS = 0
1064 REM XA,YA ist der Anfangspunkt einer LINE
1065 REM XE,YE ist der Endpunkt einer LINE
1066 XA = 0
1067 YA = 0
1068 XE = 0
1069 YE = 0
1070 REM IPST ist der Stiftstatus, 0 -> gehoben
1071 REM                          1 -> gesenkt
1072 IPST = 0
1073 REM XALT,YALT ist der zuletzt mit gesenktem Stift
1074 REM angefahrene Punkt
1075 XALT = 0
1076 YALT = 0
1077 COLOR 14,1
1078 LOCATE 1,1
1079 CLS
1080 PRINT "          Guenter Pomaska"
1081 PRINT " "
1082 PRINT "     HP-GL Simulation mit dem"
1083 PRINT "      Fischertechnik Plotter"
1110 FILES "*.pic"
1120 LOCATE 20,1
1130 INPUT "Plotte Bilddatei ?",P$
1140 OPEN "i",#1,P$
```

```
1150 KEY OFF
1160 SCREEN 1
1170 CLS
1180 IFARBE = 1
1190 COLOR 8,2
1215 REM Eingabe der HP-GL Steuersequenz
1220 INPUT #1,A$
1230 IF EOF(1) THEN GOTO 1310
1240 KOM$=LEFT$(A$,2)
1241 REM ************************************************
1242 REM ***                                          ***
1243 REM ***  Interpretiert werden die folgenden      ***
1244 REM ***  HP-GL Befehle :                         ***
1250         IF KOM$=      "PA"          THEN GOTO 1350
1255         IF KOM$=      "PR"          THEN GOTO 1350
1260         IF KOM$=      "PU"          THEN GOTO 1730
1270         IF KOM$=      "PD"          THEN GOTO 1780
1280         IF KOM$=      "SP"          THEN GOTO 1820
1282         IF KOM$=      "CI"          THEN GOTO 1880
1285         IF KOM$=      "LB"          THEN GOTO 1855
1286 REM ***                                          ***
1287 REM ************************************************
1290 GOTO 1220
1310 CLOSE #1
1320 BEEP
1330 GOTO 1340
1335 REM Endlosschleife Abbruch mit CTRL BREAK
1340 GOTO 1330
1350 REM ************* HP-GL Befehle PA, PR ***************
1351 REM
1352 REM Die Parameter des Befehls PA werden dem Endpunkt
1353 REM einer Linie zugewiesen. Lautet der Plotbefehl PR,
1354 REM dann wird zu den relativen Koordinaten die
1355 REM aktuelle Stiftposition addiert.
1356 REM
1370 XE = VAL (MID$(A$,3,7))
1375 IF KOM$="PR" THEN XE=XE+XPOS
1376 XPOS = XE
1380 INPUT #1, A$
1390 IF EOF(1) THEN GOTO 1310
1400 YE = VAL(A$)
1410 IF KOM$="PR" THEN YE=YE+YPOS
1411 YPOS = YE
1430 IF IPST=0 THEN GOTO 1650
1432 REM Der Stift war gesenkt -> Linie zeichnen !
1433 REM Ausgabe auf dem Bildschirm
1440 LINE (XA*FX,199-YA*FY)-(XE*FX,199-YE*FY),IFARBE
1445 REM Plotterausgabe wenn Anfangspkt. nicht Endpkt.
1450 IF XA=XE AND YA=YE GOTO 1220
1480 X=XA
1490 Y=YA
1491 REM Keine Stiftbewegung zu PA wenn Linie < 0.1
1494 IF  ABS(XALT-X)<.1 AND ABS(YALT-Y)<.1 THEN GOTO 1510
1495 REM Unterprogrammaufruf MOVE
1500 GOSUB 45000
1510 X=XE
1520 Y=YE
1521 XALT = XE
1522 YALT = YE
1523 REM XALT,YALT ist der zuletzt mit gesenktem Stift
1524 REM angefahrene Punkt
1525 REM Unterprogrammaufruf LINIE
1530 GOSUB 44000
```

```
1610 YA=YE
1620 XA=XE
1630 GOTO 1220
1650 REM Penstatus war gehoben, merke Punkt
1670 XA = XE
1680 YA=YE
1710 GOTO 1220
1730 REM ***************** HP-GL Befehl  PU *****************
1750 IPST=0
1760 GOTO 1220
1780 REM ***************** HP-GL Befehl  PD *****************
1800 IPST=1
1810 GOTO 1220
1820 REM ***************** HP-GL Befehl SP *****************
1840 IFARBE=VAL (MID$(A$,3,3))
1850 GOTO 1220
1855 REM ***************** HP-GL Befehl LB *****************
1861 G=2
1862 R=0
1865 T$=A$
1866 X=XE
1867 Y=YE
1868 A$=MID$(T$,3,60)
1869 GOSUB 48000
1870 GOTO 1220
1880 REM ***************** HP-GL Befehl CI *****************
1882 R=VAL (MID$(A$,3,6))
1883 INPUT #1,A$
1884 IF EOF(1) THEN GOTO 1310
1885 WA=0
1886 WE=360
1887 X=XE
1888 Y=YE
1889 RX=R
1890 RY=R
1891 RAD=R*FX
1892 CIRCLE (X*FX,Y*FY),RAD
1893 GOSUB 21000
1894 XALT = X
1895 YALT = Y
1898 GOSUB 45000
1899 GOTO 1220
1900 REM ******************************************************


10 REM HP-GL Plottersimulation mit
20 REM Fischertechnik auf IBM-PC
30 REM (C) Dr.Guenter Pomaska 1987
40 REM Treiberprogramme der Fischer-
41 REM werke Version 2.1 werden benutzt
50 CLEAR ,&HFF00
60 DEF SEG
70 DEF USR=&HFF50
80 BLOAD"A:INTERFAC.COM",&HFF00
90 DEFINT E,R,L,A
100 LET S=7 : H=0 : R=0
110 SCREEN 0,1 : KEY OFF : WIDTH 40
120 COLOR S,H,R : CLS
420 INIT=&HFF00:M1=INIT+6:M2=M1+4:M3=M2+4:M4=M3+4
430 E1=1:E2=2:E3=4:E4=8:E5=16
440 E6=32:E7=64:E8=128
450 EX=160:EY=144
460 EIN=85:RECHTS=85:LINKS=170:AUS=255
```

```
500 CALL INIT
510 REM
520 REM fischertechnik computing
530 REM
540 REM Plot
550 REM
560 REM Copyright (c) artur fischer forschung 1985
570 REM
580 REM Plotter Tools:
590 REM
600 REM 20000 BOX
605 REM 21000 CIRCLE
610 REM 22000 AXIS
630 REM
640 REM Plotter Kommandos:
650 REM
660 REM 40000 HOME
665 REM 41000 ORIGIN
670 REM 42000 SET ORIGIN
675 REM 43000 SCALE
680 REM 44000 LINE
685 REM 45000 MOVE
690 REM 46000 RLINE
695 REM 47000 RMOVE
700 REM 48000 CHAR
705 REM 49000 RCHAR
710 REM
720 REM Plotter Bewegungen:
730 REM
740 REM 50000 +X-Schritt
750 REM 51000 -X-Schritt
760 REM 52000 +Y-Schritt
770 REM 53000 -Y-Schritt
780 REM 54000 +X/+Y-Diagonale
790 REM 55000 +X/-Y-Diagonale
800 REM 56000 -X/+Y-Diagonale
810 REM 57000 -X/-Y-Diagonale
820 REM 58000 Stift ab
830 REM 59000 Stift hoch
840 REM 60000 Endtaster ?
850 REM 61000 Bereichsüberschreitung ?
860 REM
870 CLS
875 GOSUB 19530
880 PRINT" Plotter-Initialisierung"
890 GOSUB 40000 :REM HOME
900 REM
910 CLS
915 GOSUB 19530
920 PRINT" Zeichensatz wird eingelesen "
930 DIM Z%(255,9)
940 BLOAD"IBMCHAR.CHR",VARPTR(Z%(0,0))
950 REM Alternative:
960 REM BLOAD"ASCII.CHR",....
970 GOSUB 19530
1000 REM
1010 REM ************************************************
1020 REM ***          Beginn des Benutzerprogramms        ***
1021 REM ***                                              ***
1030 REM *** HP-GL Simulation mit Fischertechnik Plotter ***
1031 REM ***    mit Bildschirmgrafik fuer IBM-PC / CGA    ***
1032 REM ***                                              ***
1033 REM ************************************************
```

```
1034 REM
1040 REM Skalierung Plotter  Hewlett-Packard / Fischer
1041 X1=0
1042 X2=680
1043 Y1=0
1044 Y2=500
1045 GOSUB 43000
1049 REM Umrechnungsfaktoren IBM-Bildschirm / Plotter
1050 FX=320/X2
1060 FY=200/Y2
1061 REM XPOS, YPOS ist die aktuelle Stiftposition
1062 XPOS = 0
1063 YPOS = 0
1064 REM XA,YA ist der Anfangspunkt einer LINE
1065 REM XE,YE ist der Endpunkt einer LINE
1066 XA = 0
1067 YA = 0
1068 XE = 0
1069 YE = 0
1070 REM IPST ist der Stiftstatus, 0 -> gehoben
1071 REM                          1 -> gesenkt
1072 IPST = 0
1073 REM XALT,YALT ist der zuletzt mit gesenktem Stift
1074 REM angefahrene Punkt
1075 XALT = 0
1076 YALT = 0
1077 COLOR 14,1
1078 LOCATE 1,1
1079 CLS
1080 PRINT "           Guenter Pomaska"
1081 PRINT " "
1082 PRINT "       HP-GL Simulation mit dem"
1083 PRINT "        Fischertechnik Plotter"
1110 FILES "*.pic"
1120 LOCATE 20,1
1130 INPUT "Plotte Bilddatei ?",P$
1140 OPEN "i",#1,P$
1150 KEY OFF
1160 SCREEN 1
1170 CLS
1180 IFARBE = 1
1190 COLOR 8,2
1215 REM Eingabe der HP-GL Steuersequenz
1220 INPUT #1,A$
1230 IF EOF(1) THEN GOTO 1310
1240 KOM$=LEFT$(A$,2)
1241 REM ************************************************
1242 REM ***                                          ***
1243 REM ***  Interpretiert werden die folgenden      ***
1244 REM ***  HP-GL Befehle :                         ***
1250       IF KOM$=      "PA"          THEN GOTO  1350
1255       IF KOM$=      "PR"          THEN GOTO  1350
1260       IF KOM$=      "PU"          THEN GOTO  1730
1270       IF KOM$=      "PD"          THEN GOTO  1780
1280       IF KOM$=      "SP"          THEN GOTO  1820
1282       IF KOM$=      "CI"          THEN GOTO  1880
1285       IF KOM$=      "LB"          THEN GOTO  1855
1286 REM ***                                          ***
1287 REM ************************************************
1290 GOTO 1220
1310 CLOSE #1
1320 BEEP
1330 GOTO 1340
```

```
1335 REM Endlosschleife Abbruch mit CTRL BREAK
1340 GOTO 1330
1350 REM ************** HP-GL Befehle PA, PR ****************
1351 REM
1352 REM Die Parameter des Befehls PA werden dem Endpunkt
1353 REM einer Linie zugewiesen. Lautet der Plotbefehl PR,
1354 REM dann wird zu den relativen Koordinaten die
1355 REM aktuelle Stiftposition addiert.
1356 REM
1370 XE = VAL (MID$(A$,3,7))
1375 IF KOM$="PR" THEN XE=XE+XPOS
1376 XPOS = XE
1380 INPUT #1, A$
1390 IF EOF(1) THEN GOTO 1310
1400 YE = VAL(A$)
1410 IF KOM$="PR" THEN YE=YE+YPOS
1411 YPOS = YE
1430 IF IPST=0 THEN GOTO 1650
1432 REM Der Stift war gesenkt -> Linie zeichnen !
1433 REM Ausgabe auf dem Bildschirm
1440 LINE (XA*FX,199-YA*FY)-(XE*FX,199-YE*FY),IFARBE
1445 REM Plotterausgabe wenn Anfangspkt. nicht Endpkt.
1450 IF XA=XE AND YA=YE GOTO 1220
1480 X=XA
1490 Y=YA
1491 REM Keine Stiftbewegung zu PA wenn Linie < 0.1
1494 IF  ABS(XALT-X)<.1 AND ABS(YALT-Y)<.1 THEN GOTO 1510
1495 REM Unterprogrammaufruf MOVE
1500 GOSUB 45000
1510 X=XE
1520 Y=YE
1521 XALT = XE
1522 YALT = YE
1523 REM XALT,YALT ist der zuletzt mit gesenktem Stift
1524 REM angefahrene Punkt
1525 REM Unterprogrammaufruf LINIE
1530 GOSUB 44000
1610 YA=YE
1620 XA=XE
1630 GOTO 1220
1650 REM Penstatus war gehoben, merke Punkt
1670 XA = XE
1680 YA=YE
1710 GOTO 1220

1730 REM ************** HP-GL Befehl  PU ****************
1750 IPST=0
1760 GOTO 1220
1780 REM ************** HP-GL Befehl  PD ****************
1800 IPST=1
1810 GOTO 1220
1820 REM ************** HP-GL Befehl SP ****************
1840 IFARBE=VAL (MID$(A$,3,3))
1850 GOTO 1220
1855 REM ************** HP-GL Befehl LB ****************
1861 G=2
1862 R=0
1865 T$=A$
1866 X=XE
1867 Y=YE
1868 A$=MID$(T$,3,60)
1869 GOSUB 48000
1870 GOTO 1220
```

```
1880 REM ****************** HP-GL Befehl CI ******************
1882 R=VAL (MID$(A$,3,6))
1883 INPUT #1,A$
1884 IF EOF(1) THEN GOTO 1310
1885 WA=0
1886 WE=360
1887 X=XE
1888 Y=YE
1889 RX=R
1890 RY=R
1891 RAD=R*FX
1892 CIRCLE (X*FX,Y*FY),RAD
1893 GOSUB 21000
1894 XALT = X
1895 YALT = Y
1898 GOSUB 45000
1899 GOTO 1220
1900 REM ******************************************************
18960 REM
18970 REM ***********************************
18980 REM *** Ende des Benutzerprogramms ***
18990 REM ***********************************
19000 END
19500 REM *************************
19510 REM *** Bildschirmmeldung ***
19520 REM *************************
19530 CLS
19540 PRINT
19550 PRINT" fischertechnik"
19560 PRINT" computing"
19570 PRINT
19580 PRINT" HP-GL Plottersimulation"
19590 PRINT" (c) Dr. Guenter Pomaska 1987 "
19630 PRINT
19640 RETURN
19800 REM
19810 REM Wichtige globale Variablen:
19820 REM
19830 REM XM,YM: Breite, Höhe der Plotterfläche
19835 REM        in Plotterschritten
19836 REM        (680,500) = rechte, obere Ecke
19837 REM        (0,0) = linke, untere Ecke
19840 REM XJ,YJ: ABSOLUTE Koordinaten des
19845 REM        aktuellen Punktes
19860 REM XH,YH: virtueller (logischer) Nullpunkt
19865 REM        in absoluten Koordinaten
19866 REM        (0,0) = am Anfang
19870 REM XS,YS: Skalierung, Anzahl Plotterschritte
19875 REM        pro logischer Einheit
19876 REM        1 Einheit = 1 Plotterschritt
19880 REM XO,YO: (=xout,yout) Flaggen,
19890 REM        werden 1, wenn XJ,YJ außerhalb der
19895 REM        Plotterfläche 0...XM,0...YM
19896 REM Festlegung der Variablen im HOME-Unterprogramm
19897 REM        MG$="aus" Stift abgehoben
19900 REM
19910 REM *******************************
19920 REM *** P l o t t e r t o o l s ***
19930 REM *******************************
19940 REM
19950 REM In dieser Schale benutzte Variablen:
19960 REM
19970 REM G,R,RX,RY,S,SK,W,WA,WE,WS,A$,X$,Y$
```

```
19980 REM X,X0,X1,X2,X3,XA,XE,XH,XS,Y
19985 REM Y0,Y1,Y2,Y3,YA,YD,YE,YH,YS
19990 REM
20000 REM *** BOX ***
20010 REM
20020 IF XA>XE THEN LET X=XE:LET XE=XA:LET XA=X
20030 IF YA>YE THEN Y=YE:YE=YA:YA=Y
20040 LET X=XA:LET Y=YA:GOSUB 45000
20050 LET X=XA:LET Y=YE:GOSUB 44000
20060 LET X=XE:LET Y=YE:GOSUB 44000
20070 LET X=XE:LET Y=YA:GOSUB 44000
20080 LET X=XA:LET Y=YA:GOSUB 44000
20090 IF S=0 THEN RETURN
20100 REM Schraffieren
20110 FOR XD=0 TO (XE-XA)+(YE-YA) STEP ABS(S)
20120 IF XD<(YE-YA) THEN LET X=XA:Y=YE-XD:GOTO 20140
20130 LET X=XA+XD+YA-YE:Y=YA
20140 IF S<0 THEN LET Y=YE+YA-Y
20150 GOSUB 45000
20160 IF XD>(XE-XA) THEN LET X=XE:Y=YE+XE-XA-XD:GOTO 20180
20170 LET X=XA+XD:Y=YE
20180 IF S<0 THEN LET Y=YE+YA-Y
20190 GOSUB 44000
20200 NEXT XD
20210 RETURN
20220 REM
21000 REM *** CIRCLE ***
21010 REM
21020 LET RX=ABS(RX):LET RY=ABS(RY)
21030 LET XA=X:LET YA=Y:REM Mittelpunkt merken
21040 LET X=XA+RX*COS(ATN(1)*WA/45)
21050 LET Y=YA+RY*SIN(ATN(1)*WA/45)
21060 GOSUB 45000
21070 REM Winkelschrittweite
21080 LET WS=SGN(WE-WA)*800/(RX*XS+RY*YS+40)
21090 FOR W=WA TO WE STEP WS
21100 LET X=XA+RX*COS(ATN(1)*W/45)
21110 LET Y=YA+RY*SIN(ATN(1)*W/45)
21120 GOSUB 44000
21130 NEXT W
21140 LET X=XA+RX*COS(ATN(1)*WE/45)
21150 LET Y=YA+RY*SIN(ATN(1)*WE/45)
21160 GOSUB 44000:REM Kreisbogen schlieaen
21170 RETURN
21180 REM
22000 REM *** AXIS ***
22010 REM
22020 REM XA,XE,YA,YE geben die Lage der Achsen
22030 REM in der bisherigen Skalierung an.
22040 REM X1,X2,Y1,Y2 sind die Koordinaten-Werte
22050 REM an den Achsenendpunkten.
22060 REM
22070 REM Skalierung der Plotterfläche
22080 REM entspr. Koordinatenachsen
22090 REM
22100 REM X0,X3,Y0,Y3: neue Koordinaten
22110 REM am Rande der Plotterfläche
22120 REM
22130 LET X0=X1-(X2-X1)*(XA+XH/XS)/(XE-XA)
22140 LET X3=X2+(X2-X1)*((XM-XH)/XS-XE)/(XE-XA)
22150 LET Y0=Y1-(Y2-Y1)*(YA+YH/YS)/(YE-YA)
22160 LET Y3=Y2+(Y2-Y1)*((YM-YH)/YS-YE)/(YE-YA)
22170 LET XS=XM/(X3-X0)
```

```
22180 LET YS=YM/(Y3-Y0)
22190 LET XH=-X0*XS
22200 LET YH=-Y0*YS
22210 REM Achsen einzeichnen
22220 IF Y1>0 OR Y2<0 THEN 22490
22230 REM X-Achse
22240 LET X=X1:LET Y=0:GOSUB 45000
22250 LET X=X2:LET Y=0:GOSUB 44000
22260 LET X=-10/XS:LET Y=5/YS:GOSUB 46000
22270 LET X=10/XS:LET Y=-5/YS:GOSUB 47000
22280 LET X=-10/XS:LET Y=-5/YS:GOSUB 46000
22290 REM Skalierung und Beschriftung
22300 IF SK=0 THEN GOTO 22490
22310 REM Schrittweite für Skalierung
22320 LET XE=INT(LOG(X2-X1)/LOG(10)-.3)
22323 LET XD=1
22326 IF XE>0 THEN FOR I=1 TO XE:XD=XD*10:NEXT
22329 IF XE<0 THEN FOR I=XE TO -1:XD=XD/10:NEXT
22330 IF (LEN(STR$(XD))+1)*10>XD*XS THEN XD=XD*5:GOTO 22350
22340 IF (LEN(STR$(XD))+1)*40<XD*XS THEN XD=XD/2
22350 LET X0=0
22360 IF X0+1.2*XD<X2 THEN X0=X0+XD:GOTO 22360
22370 REM keine Beschriftung am Nullpunkt
22380 IF ABS(X0)<XD/2 THEN GOTO 22450
22390 LET X=X0:LET Y=3/YS:GOSUB 45000
22400 LET X=X0:LET Y=-3/YS:GOSUB 44000
22410 LET A$=STR$(X0):LET R=0:LET G=2
22420 IF X0>0 THEN LET A$="+"+RIGHT$(A$,LEN(A$)-1)
22430 LET X=X0-5*LEN(A$)/XS:LET Y=-18/YS
22440 GOSUB 48000
22450 IF X0-1.2*XD>X1 THEN LET X0=X0-XD:GOTO 22380
22460 LET A$=X$:LET R=0:LET G=2
22470 LET X=X2-10*LEN(A$)/XS:LET Y=12/YS
22480 GOSUB 48000
22490 IF X1>0 OR X2<0 THEN RETURN
22500 REM Y-Achse
22510 LET X=0:LET Y=Y1:GOSUB 45000
22520 LET X=0:LET Y=Y2:GOSUB 44000
22530 LET X=-5/XS:LET Y=-10/YS:GOSUB 46000
22540 LET X=5/XS:LET Y=10/YS:GOSUB 47000
22550 LET X=5/XS:LET Y=-10/YS:GOSUB 46000
22560 REM Skalierung und Beschriftung
22570 IF SK=0 THEN RETURN
22580 REM Schrittweite für Skalierung
22590 LET YE=INT(LOG(Y2-Y1)/LOG(10)-.3)
22593 LET YD=1
22596 IF YE>0 THEN FOR I=1 TO YE:YD=YD*10:NEXT
22599 IF YE<0 THEN FOR I=YE TO -1:YD=YD/10:NEXT
22600 IF YD*YS<20 THEN YD=YD*100:GOTO 22620
22610 IF YD*YS>60 THEN YD=YD/2
22620 LET Y0=0
22630 IF Y0+1.2*YD<Y2 THEN LET Y0=Y0+YD:GOTO 22630
22640 REM keine Beschriftung am Nullpunkt
22650 IF ABS(Y0)<YD/2 THEN 22720
22660 LET Y=Y0:LET X=-3/XS:GOSUB 45000
22670 LET Y=Y0:LET X=3/XS:GOSUB 44000
22680 LET A$=STR$(Y0):LET R=0:LET G=2
22690 IF Y0>0 THEN LET A$="+"+RIGHT$(A$,LEN(A$)-1)
22700 LET X=8/XS:LET Y=Y0-6/YS
22710 GOSUB 48000
22720 IF Y0-1.2*YD>Y1 THEN Y0=Y0-YD:GOTO 22650
22730 LET A$=Y$:LET R=1:LET G=2
22740 LET X=-12/XS:LET Y=Y2-10*LEN(A$)/YS
```

```
22750 GOSUB 48000
22760 RETURN
22770 REM
39900 REM ******************************************
39910 REM *** E I N F A C H E  k O M M A N D O S ***
39920 REM ******************************************
39930 REM
39940 REM In dieser Schale benutzte Variablen:
39950 REM
39960 REM A$,E7,E8,I0,J0,J1,D,DX,DY,NX,NY,PE
39970 REM R,XA%,YA%,XN%,YN%,Z%,Z%(,)
39980 REM X,Y,X1,Y1,X2,Y2,XH,YH,XJ,YJ
39985 REM XM,YM,X0,Y0,XS,YS,XY,H
39990 REM
40000 REM *** HOME ***
40010 REM
40020 GOSUB 59030:REM Stift hoch
40030 LET XJ=0:LET YJ=0
40040 IF USR(E7)=1 AND USR(E8)=1 THEN GOSUB 57060:GOTO 40040
40050 IF USR(E7)=1 THEN GOSUB 51050:GOTO 40050
40060 IF USR(E7)=0 THEN GOSUB 50050:GOTO 40060
40070 IF USR(E8)=1 THEN GOSUB 53050:GOTO 40070
40080 IF USR(E8)=0 THEN GOSUB 52050:GOTO 40080
40090 GOSUB 54060:GOSUB 54060
40100 LET XM=680:YM=500
40110 LET XH=0:YH=0:XS=1:YS=1
40120 LET XOUT=0:YOUT=0
40125 LET R=0: LET G=2
40130 RETURN
40140 REM
41000 REM *** ORIGIN ***
41010 GOSUB 59000:REM Stift hoch
41020 LET X=0:LET Y=0
41030 GOSUB 45000
41040 RETURN
41050 REM
42000 REM *** SET ORIGIN ***
42010 LET XH=XH+X*XS
42020 LET YH=YH+Y*YS
42030 RETURN
42040 REM
43000 REM *** SCALE ***
43010 REM
43020 IF NOT(X1<X2) OR NOT(Y1<Y2) THEN RETURN
43030 LET XS=XM/(X2-X1)
43040 LET YS=YM/(Y2-Y1)
43050 LET XH=-X1*XS
43060 LET YH=-Y1*YS
43070 RETURN
43080 REM
44000 REM *** LINE ***
44010 REM
44020 GOSUB 58000:REM Stift ab
44030 LET X=INT(XH+XS*X-XJ+.5)
44040 LET Y=INT(YH+YS*Y-YJ+.5)
44050 REM X,Y jetzt rel. Entfernung
44060 REM in Plotterschritten
44070 REM Anzahl der Plotterschritte zum Zielpunkt
44080 LET DX=ABS(X):LET DY=ABS(Y)
44090 LET NX=0:LET NY=0:REM Schrittzähler
44100 LET D=0:REM Abstand zur optimalen Geraden
44110 IF NX=DX AND NY=DY THEN RETURN
44120 IF ABS(D+DY)<ABS(D+DY-DX) THEN 44190
```

```
44130 IF ABS(D-DX)<ABS(D+DY-DX) THEN 44240
44140 REM Diagonalschritt
44150 ON 1-2*(X>0)-(Y>0) GOSUB 57000,56000,55000,54000
44160 LET NX=NX+1:LET NY=NY+1
44170 LET D=D+DY-DX
44180 GOTO 44110
44190 REM X-Schritt
44200 ON -(X>0)+1 GOSUB 51000,50000
44210 LET NX=NX+1
44220 LET D=D+DY
44230 GOTO 44110
44240 REM Y-Schritt
44250 ON -(Y>0)+1 GOSUB 53000,52000
44260 LET NY=NY+1
44270 LET D=D-DX
44280 GOTO 44110
44290 REM
45000 REM *** MOVE ***
45010 REM
45020 GOSUB 59000:REM Stift hoch
45030 GOTO 44030
45040 REM
46000 REM *** RLINE ***
46010 REM
46020 GOSUB 58000:REM Stift ab
46030 LET X=INT(XS*X+.5)
46040 LET Y=INT(YS*Y+.5)
46050 GOTO 44050
46060 REM
47000 REM *** RMOVE ***
47010 REM
47020 GOSUB 59000:REM Stift hoch
47030 GOTO 46030
47040 REM
48000 REM *** CHAR ***
48010 REM
48020 GOSUB 45000:REM MOVE nach X,Y
48030 IF A$="" THEN RETURN
48040 FOR I0=1 TO LEN(A$)
48050 LET J0=0
48060 LET XALT%=0:LET YALT%=3:REM Startpunkt
48070 IF MID$(A$,I0,1)=CHR$(8) THEN XNEU%=-5:YNEU%=3:PENS=0:GOTO 48220
48080 LET Z%=Z%(ASC(MID$(A$,I0,1)),J0):LET J1=0
48090 IF Z%=0 THEN 48200
48100 LET XNEU%=INT(Z%/1000):LET YNEU%=INT(Z%/100)-10*XNEU%
48110 LET Z%=100*(Z%-1000*XNEU%-100*YNEU%)
48120 LET PENS=0
48130 IF XNEU%>4 THEN LET XNEU%=XNEU%-5:LET PENS=-1
48140 GOSUB 48250
48150 LET XALT%=XNEU%:LET YALT%=YNEU%
48160 IF Z%<>0 THEN LET J1=1:GOTO 48100
48170 IF J0=9 THEN GOTO 48200
48180 IF J1=1 THEN LET J0=J0+1:GOTO 48080
48190 IF Z%(ASC(MID$(A$,I0,1)),J0+1)<>0 THEN LET J1=1:GOTO 48100
48200 REM Buchstabe fertig, zum Endpunkt rechts unten
48210 LET XNEU%=5:LET YNEU%=3:LET PENS=0
48220 GOSUB 48250
48230 NEXT I0
48240 RETURN
48250 REM Vektor drehen, strecken, zeichnen
48260 LET X=(XNEU%-XALT%)*G
48270 LET Y=(YNEU%-YALT%)*G
48280 IF R=1 THEN XY=X:X=-Y:Y=XY
```

```
48290 IF R=2 THEN X=-X:Y=-Y
48300 IF R=3 THEN XY=X:X=Y:Y=-XY
48310 LET X=X/XS:LET Y=Y/YS
48320 IF PENS THEN GOSUB 46000:REM RLINE
48330 IF NOT PENS THEN GOSUB 47000:REM RMOVE
48340 RETURN
48350 REM
49000 REM *** RCHAR ***
49010 REM
49020 GOSUB 47000:REM RMOVE X,Y
49030 GOTO 48040
49040 REM
49920 REM ************************************************
49930 REM *** e l e m e n t a r e  B e w e g u n g e n ***
49940 REM ************************************************
49960 REM In dieser Schale benutzte Variablen
49970 REM
49980 REM M1...M4,E7,E8,RECHTS,LINKS,EIN,AUS,INIT,IT,MG$
49985 REM XJ,YJ,XM,YM,X0,Y0
49990 REM
50000 REM *** +x-Bewegung ***
50010 REM
50020 LET XJ=XJ+1
50030 GOSUB 61000: REM *** OUT? ***
50040 IF XOUT THEN RETURN
50050 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(RECHTS)
50060 GOSUB 60000:REM *** ET ??? ***
50070 CALL M1(LINKS):CALL M2(LINKS):CALL M3(RECHTS)
50080 GOSUB 60000:REM *** ET ??? ***
50090 CALL M1(RECHTS):CALL M2(LINKS):CALL M3(RECHTS)
50100 GOSUB 60000:REM *** ET ??? ***
50110 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
50120 GOSUB 60000:REM *** ET ??? ***
50130 RETURN
50140 REM
51000 REM *** -x-Bewegung ***
51010 REM
51020 LET XJ=XJ-1
51030 GOSUB 61000: REM *** OUT? ***
51040 IF XOUT THEN RETURN
51050 CALL M1(RECHTS):CALL M2(LINKS):CALL M3(RECHTS)
51060 GOSUB 60000:REM *** ET ??? ***
51070 CALL M1(LINKS):CALL M2(LINKS):CALL M3(RECHTS)
51080 GOSUB 60000:REM *** ET ??? ***
51090 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(RECHTS)
51100 GOSUB 60000:REM *** ET ??? ***
51110 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
51120 GOSUB 60000:REM *** ET ??? ***
51130 RETURN
51140 REM
52000 REM *** +y-Bewegung ***
52010 REM
52020 LET YJ=YJ+1
52030 GOSUB 61000: REM *** OUT? ***
52040 IF YOUT THEN RETURN
52050 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(RECHTS)
52060 GOSUB 60000:REM *** ET ??? ***
52070 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(LINKS)
52080 GOSUB 60000:REM *** ET ??? ***
52090 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(LINKS)
52100 GOSUB 60000:REM *** ET ??? ***
52110 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
52120 GOSUB 60000:REM *** ET ??? ***
```

```
52130 RETURN
52140 REM
53000 REM *** -y-Bewegung ***
53010 REM
53020 LET YJ=YJ-1
53030 GOSUB 61000: REM *** OUT? ***
53040 IF YOUT THEN RETURN
53050 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(LINKS)
53060 GOSUB 60000:REM *** ET ??? ***
53070 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(LINKS)
53080 GOSUB 60000:REM *** ET ??? ***
53090 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(RECHTS)
53100 GOSUB 60000:REM *** ET ??? ***
53110 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
53120 GOSUB 60000:REM *** ET ??? ***
53130 RETURN
53140 REM
54000 REM *** +x/+y-Diagonale ***
54010 REM
54020 LET XJ=XJ+1:LET YJ=YJ+1
54030 GOSUB 61000: REM *** OUT? ***
54040 IF XOUT THEN 52040
54050 IF YOUT THEN 50040
54060 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(RECHTS)
54070 GOSUB 60000:REM *** ET ??? ***
54080 CALL M1(LINKS):CALL M2(LINKS):CALL M3(LINKS)
54090 GOSUB 60000:REM *** ET ??? ***
54100 CALL M1(RECHTS):CALL M2(LINKS):CALL M3(LINKS)
54110 GOSUB 60000:REM *** ET ??? ***
54120 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
54130 GOSUB 60000:REM *** ET ??? ***
54140 RETURN
54150 REM
55000 REM *** +x/-y-Diagonale ***
55010 REM
55020 LET XJ=XJ+1:LET YJ=YJ-1
55030 GOSUB 61000: REM *** OUT? ***
55040 IF XOUT THEN 53040
55050 IF YOUT THEN 50040
55060 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(LINKS)
55070 GOSUB 60000:REM *** ET ??? ***
55080 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(LINKS)
55090 GOSUB 60000:REM *** ET ??? ***
55100 CALL M1(LINKS):CALL M2(LINKS):CALL M3(RECHTS)
55110 GOSUB 60000:REM *** ET ??? ***
55120 CALL M1(RECHTS):CALL M2(LINKS):CALL M3(RECHTS)
55130 GOSUB 60000:REM *** ET ??? ***
55140 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
55150 GOSUB 60000:REM *** ET ??? ***
55160 RETURN
55170 REM
56000 REM *** -x/+y-Diagonale ***
56010 REM
56020 LET XJ=XJ-1:LET YJ=YJ+1
56030 GOSUB 61000: REM *** OUT? ***
56040 IF XOUT THEN GOTO 52040
56050 IF YOUT THEN GOTO 51040
56060 CALL M1(RECHTS):CALL M2(LINKS):CALL M3(RECHTS)
56070 GOSUB 60000:REM *** ET ??? ***
56080 CALL M1(LINKS):CALL M2(LINKS):CALL M3(RECHTS)
56090 GOSUB 60000:REM *** ET ??? ***
56100 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(LINKS)
56110 GOSUB 60000:REM *** ET ??? ***
```

```
56120 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(LINKS)
56130 GOSUB 60000:REM *** ET ??? ***
56140 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
56150 GOSUB 60000:REM *** ET ??? ***
56160 RETURN
56170 REM
57000 REM *** -x/-y-Diagonale ***
57010 REM
57020 LET XJ=XJ-1:LET YJ=YJ-1
57030 GOSUB 61000: REM *** OUT? ***
57040 IF XOUT THEN GOTO 53040
57050 IF YOUT THEN GOTO 51040
57060 CALL M1(RECHTS):CALL M2(LINKS):CALL M3(LINKS)
57070 GOSUB 60000:REM *** ET ??? ***
57080 CALL M1(LINKS):CALL M2(LINKS):CALL M3(LINKS)
57090 GOSUB 60000:REM *** ET ??? ***
57100 CALL M1(LINKS):CALL M2(RECHTS):CALL M3(RECHTS)
57110 GOSUB 60000:REM *** ET ??? ***
57120 CALL M1(RECHTS):CALL M2(RECHTS):CALL M3(RECHTS)
57130 GOSUB 60000:REM *** ET ??? ***
57140 RETURN
57150 REM
58000 REM *** MG-ein ***
58010 REM
58020 IF MG$="ein" THEN RETURN
58030 IF XOUT OR YOUT THEN RETURN
58040 CALL INIT
58050 FOR Z=1 TO 400:NEXT
58060 CALL M4(EIN)
58070 FOR Z=1 TO 100:CALL M4(EIN):NEXT
58080 LET MG$="ein"
58090 RETURN
58100 REM
59000 REM *** MG-aus ***
59010 REM
59020 IF MG$="aus" THEN RETURN
59030 CALL M4(LINKS)
59040 CALL M4(AUS)
59050 FOR Z=1 TO 100:NEXT
59060 LET MG$="aus"
59070 RETURN
59080 REM
60000 REM *** ET-gedrueckt ***
60010 REM
60020 IF (USR(E7)=0 AND XJ<>0) OR (USR(E8)=0 AND YJ<>0) THEN GOTO 60040
60030 RETURN
60040 CLS:LOCATE 2,1:COLOR 0,7:PRINT"Achtung Dejustierung!":COLOR 7,0
60050 STOP
60060 REM
61000 REM *** OUT of Range ? ***
61010 REM
61020 IF XOUT OR YOUT THEN GOTO 61090
61030 REM letzte Position war innerhalb
61040 LET XOUT=XJ<0 OR XJ>XM
61050 LET YOUT=YJ<0 OR YJ>YM
61060 IF XOUT OR YOUT THEN GOSUB 59030
61080 RETURN
61090 REM letzte Position war ausserhalb
61100 LET XOUT=XJ<0 OR XJ>XM
61110 LET YOUT=YJ<0 OR YJ>YM
61120 IF XOUT OR YOUT THEN RETURN
61130 IF MG$="ein" THEN GOSUB 58000
61150 RETURN
```

Roboter-Steuerung

Das Include-File "INTERFAC.INC" erlaubt die Ansteuerung des Fischertechnik-Roboters von Turbo-Pascal aus, allerdings nur für einen IBM-PC oder AT (oder -kompatiblen). In INTERFAC.INC wurden alle Prozeduren/Funktionen zusammengefaßt, die allein das Steuern und Abfragen der Ein- und Ausgabeleitungen des Fischertechnik-Interfaces betreffen. Man muß nicht notwendigerweise einen Roboter an das Interface anschließen, sondern man kann mit INTERFACE.INC die E/A-Funktionen beliebig steuern.

Acht- oder 16-Bit-Interface

Mit der Konstanten BITCOUNT, die auf 8 oder 16 gesetzt werden kann, wird den Prozeduren mitgeteilt, ob die einfache oder die doppelte Ausbaustufe des Interfaces vorliegt. In jedem Fall bleiben die Eingabeleitungen E1-E8 erhalten, wenn aber ein erweitertes Interface benützt wird, werden zusätzlich die Leitungen E9-E16 verfügbar. Bei den Ausgabe-Leitungen werden dann statt vier Motoren acht Motoren ansteuerbar.

Druckerport

Mit der Konstanten DRUCKERPORT, die normalerweise auf 1 gesetzt sein wird, was das Interface an LPT1 voraussetzt, kann man den verwendeten Druckeranschluß auswählen. Man muß sich nicht um die Portadressen Gedanken machen. Den Port bestimmt die Intialisierungsprozedur von selbst!

Initialisierung

Als erste Prozedur aus INTERFACE.INC muß unbedingt INIT aufgerufen werden, dadurch werden einige Initialisierungen vorgenommen und die Motoren werden ausgeschaltet.

Potentiometer

Mit den Funktionen POTI(EX) oder POTI(EY) werden die Potentiometer-Werte der X- bzw. Y-Potentiometer ausgelesen. Die Werte gehen von etwa 10 = sehr geringer Widerstand bis ca. 1000 = sehr hoher Widerstand. Dabei braucht man sich nicht um die Taktgeschwindigkeit des verwendeten Rechners zu kümmern, weil für die Zeitbestimmung der Interrupt-Timer benützt wird, der in jedem Fall mit der selben Geschwindigkeit arbeitet.

Schalter

Die Schalter-Eingänge können mit der bool'schen Funktion LESE(E1) bis LESE(E8) bzw. E16 ausgelesen werden. Dabei liefert LESE den Wert TRUE, wenn der Schalter offen oder nicht angeschlossen ist (+5 Volt), und FALSE, wenn der Schalter geschlossen wurde (0 Volt).

Motoren

Die Motoren können mit der Prozedur SCHALTE gesteuert werden. Um z.B. den Motor Nr. 2 nach LINKS einzuschalten (Vorausgesetzt, man hat die Stromzufuhr zu dem Motor richtig gepolt), braucht man nur SCHALTE(M2,LINKS) auszuführen. Um den Motor dann wieder auszuschalten, verwendet man SCHALTE(M2,AUS). Man kann aber auch direkt in den Gegenlauf schalten (mit SCHALTE(M2,RECHTS)), ohne den Motor vorher zu stoppen.

Dabei muß man natürlich darauf achten, daß das Interface den Motor von selbst ausschaltet, wenn über mehr als zwei Sekunden kein Signal vom Computer geschickt wird. Das ist so eingerichtet, damit der Modell-Roboter nicht beschädigt wird, wenn das Steuerprogramm einmal abstürzen sollte. Daher muß man in regelmäßigen Abständen einen Schalter auslesen oder den letzten SCHALTE-Aufruf wiederholen.

Noch Maschinen-Näher?

Wer die E/A-Möglichkeiten noch direkter nützen will, kann dies auch tun. In dem Fall werden statt den vier bzw. acht Motoren acht bzw. 16 TTL-Ausgabeleitungen unterstützt. Die Prozedur PUT_WORD(X) setzt die Ausgänge entsprechend den Bits von X.

Die Eingänge werden mit GET_WORD simultan eingelesen, was dann von Vorteil sein kann, wenn mehrere Eingabeleitungen gleichzeitig geprüft werden sollen.

Lager-Kran-Steuerung

Das Modul LAGER.INC ist dazu gedacht, das Modell eines Lagerkrans zu steuern. Dabei sind für die X- und Y- Koordinaten die Motoren M1 und M2 zuständig. Es werden die Schalter E1 und E3 so angebracht, daß sie gedrückt werden, wenn der Kran am Anschlag ankommt. E1 wird gedrückt, wenn der X-Anschlag erreicht ist, und E3, beim Y-Anschlag.

Die Schalter E2 und E4 sind mit Lichtschranken verbunden. Mit deren Impulsen wird die Bewegung in X und Y-Richtung gemessen. Der Kran wird mit dem Motor M3 nach oben und unten bewegt. Dabei sind die Schalter E5 und E6 mit dem oberen und unteren Anschlag verbunden. Das Zufassen erledigt der Motor M4. Der Schalter E7 wird betätigt, wenn der Greifer maximal geöffnet ist. Das Zupacken wird über eine Zeitschleife geregelt.

Konstanten

Am Anfang des Moduls werden einige Konstanten definiert. Das sind 'Greif_Zeit', die die Zeit in Millisekunden angibt, nach der der Greifer fest geschlossen ist. 'Nachlauf' gibt an, wieviele Koordinateneinheiten die Motoren vor dem eigentlichen Ziel gestoppt werden sollen. Die Motoren laufen nämlich gewöhnlich nach dem Ausschalten noch etwas nach. Der Nachlauf kann auch null sein. Mit 'Gegenlauf' wird die Zeit in Millisekunden angegeben, wie lange der Motor in die Gegenrichtung gepolt wird, um den Nachlauf zu verringern. Der Gegenlauf muß mindestens eins betragen. 'Warte_Phase' gibt die Zahl der Millisekunden an, nach der die Motoren sicher stehen. Während dieser Zeit werden die Eingänge weiterhin abgefragt. Warte_Phase muß mindestens eins betragen. 'Halte_Zeit' hat die selbe Bedeutung wir Warte_Phase.

Die Initialisierung

Um die Variablen zu initialisieren und das Modell in eine definierte Ausgangslage zu versetzen, muß am Anfang und danach in regelmäßigen Abständen die Prozedur HOME aufgerufen werden.

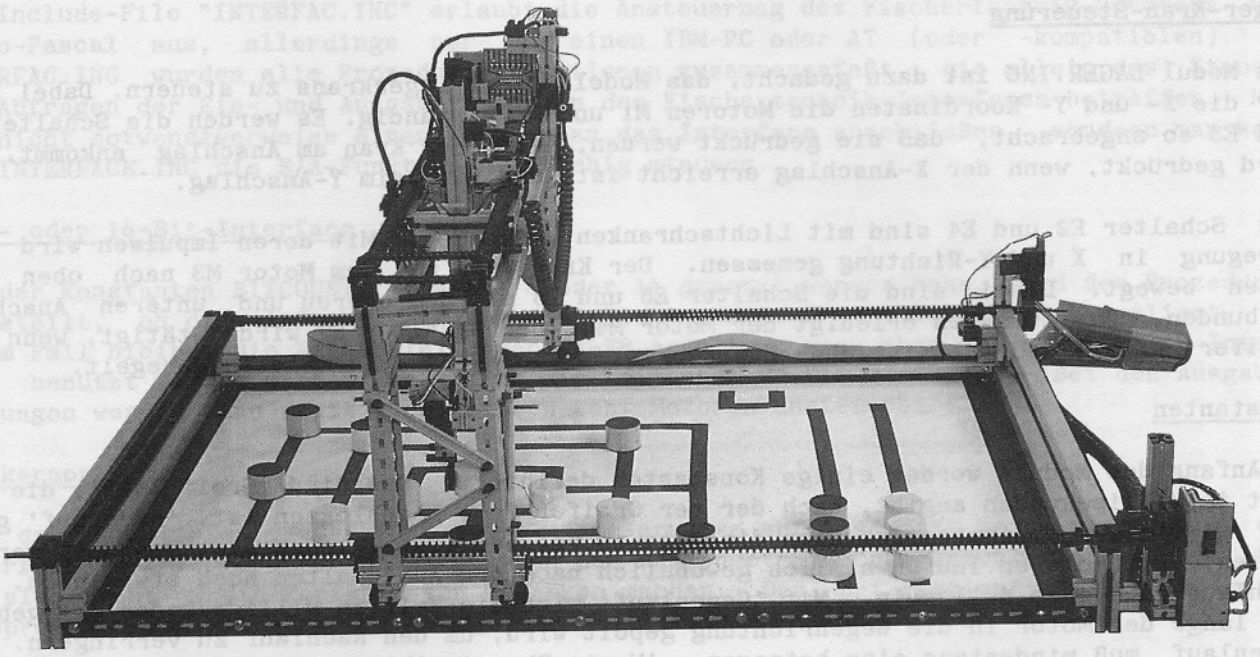
Lagesteuerung

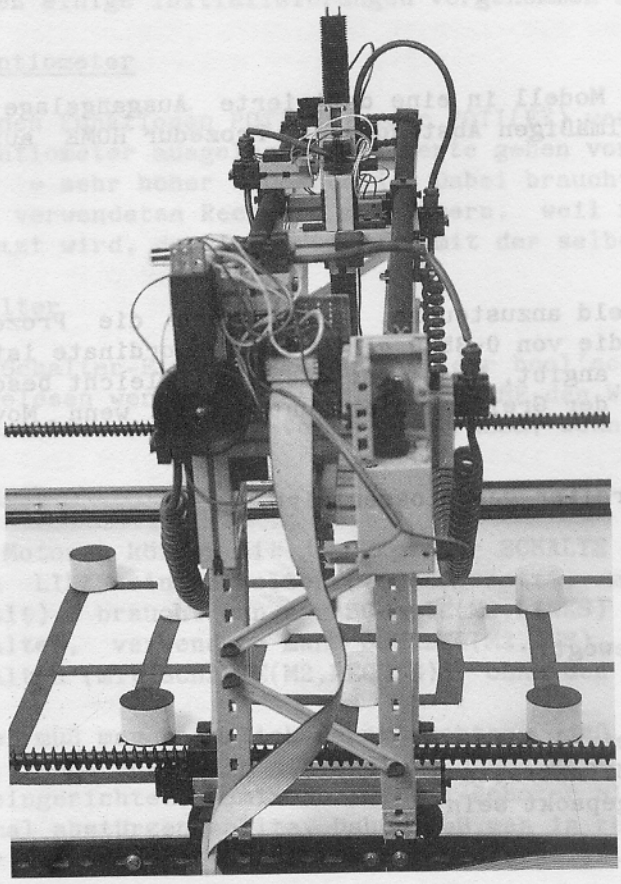
Um eine bestimmte Position auf dem Spielfeld anzusteuern, benützt man die Prozedur MoveTo(X,Y). X ist dabei die X-Koordinate, die von 0-3500 geht, die Y-Koordinate ist im Bereich 0-2300. Wenn man hier zu große Werte angibt, kann dies das Modell leicht beschädigen! Außerdem sollte man aufpassen, daß der Greifarm nicht unten ist, wenn MoveTo aufgerufen wird.

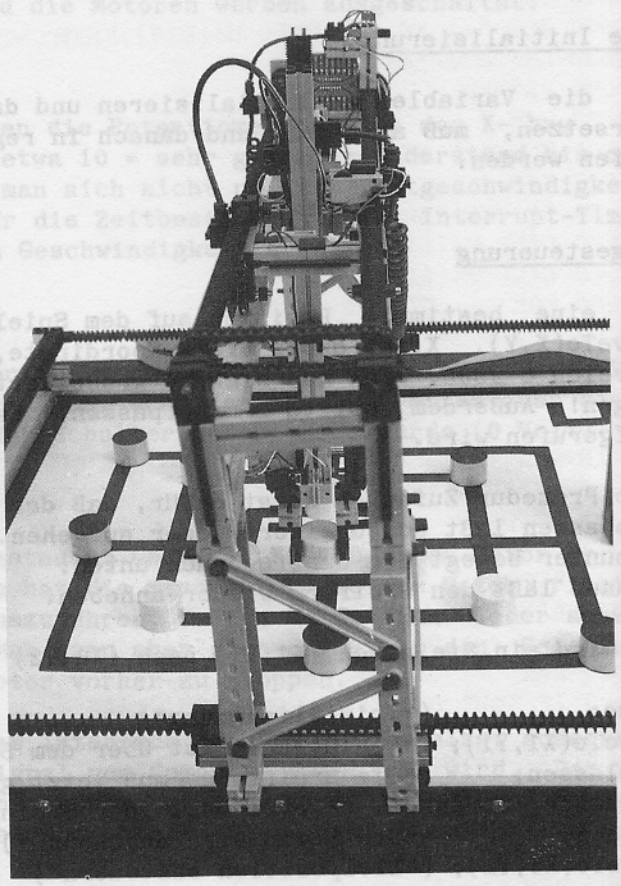
Die Prozedur Zufassen sorgt dafür, daß der Greifer geschlossen wird.
Loslassen läßt den Greifer wieder aufgehen.
Hinunter bewegt den Greifer nach unten.
Hinauf läßt den Greifarm wieder anheben.

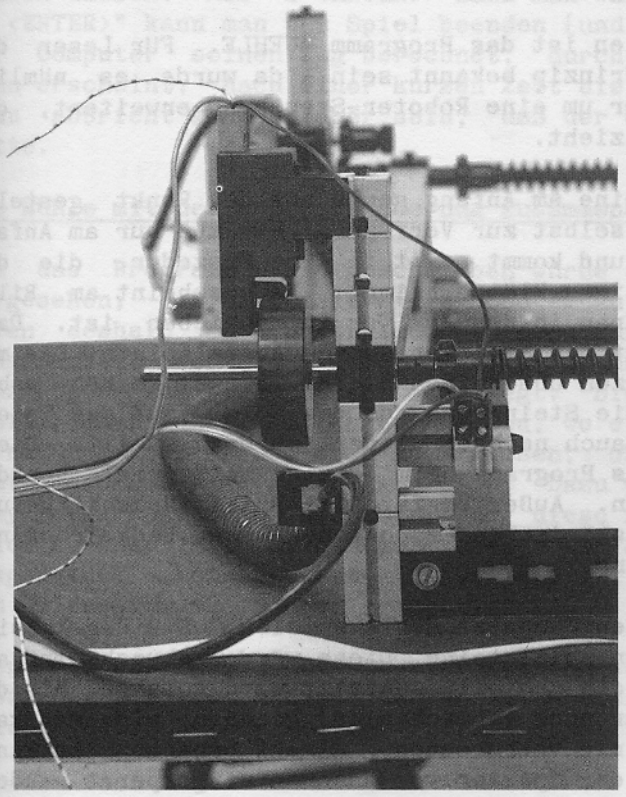
So wird ein Stein von (X1,Y1) nach (X2,Y2) bewegt:

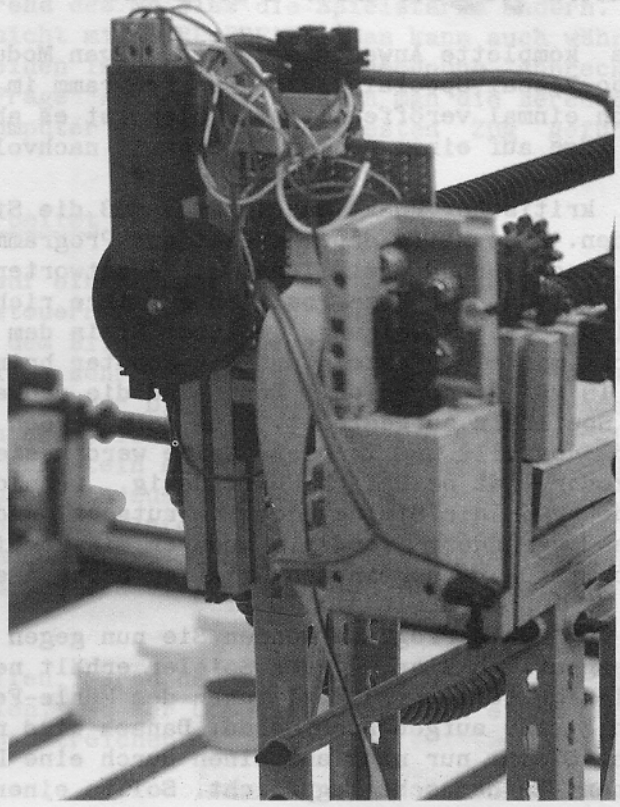
```
Home;           (*Initialisierung*)
MoveTo(X1,Y1); (*Der Geifarm ist über dem Stein*)
Loslassen;      (*Die Greifzange muß jetzt geöffnet sein*)
Hinunter; Zufassen; (*Jetzt müßte der Stein gepackt sein*)
Hinauf;         (*Der Stein wird angehoben*)
MoveTo(X2,Y2); (*Zielposition ansteuern*)
Hinunter; Loslassen; (*Der Stein wird abgesetzt*)
Hinauf;         (*Der Greifarm wird wieder angehoben*)
Home;           (*An den Start zurück*)
```

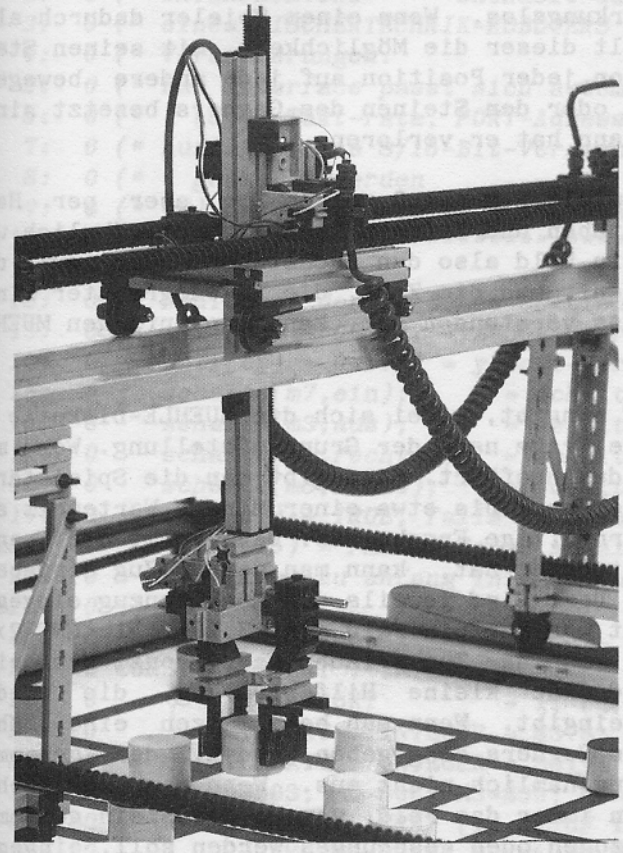


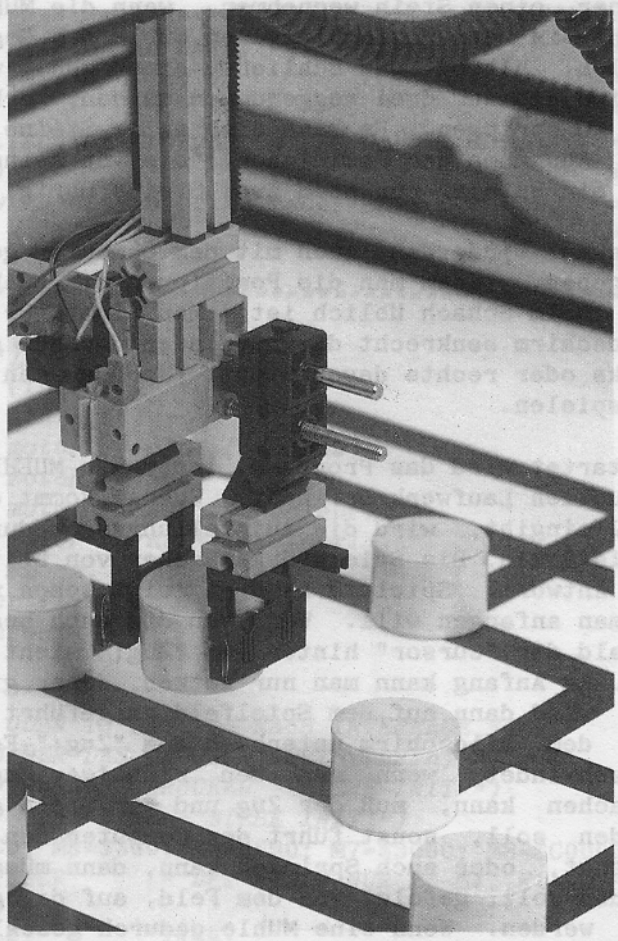












Mühle gegen den Computer

Eine komplette Anwendung zu den obigen Modulen ist das Programm MUEHLE. Für Leser der Turbo-Pascal-Specials wird das Programm im Prinzip bekannt sein - da wurde es nämlich schon einmal veröffentlicht. Hier ist es aber um eine Roboter-Steuerung erweitert, die die Züge auf einem realen Spielbrett nachvollzieht.

Der kritische Punkt dabei ist, daß die Steine am Anfang genau auf den Punkt gestellt werden. Eine Hilfe dabei stellt das Programm selbst zur Verfügung: Man muß nur am Anfang die Frage "Grundaufstellung" mit J antworten und kommt somit in eine Prozedur, die die Steine von der Home-Position auf ihre richtigen Plätze bringt. Es erscheint am Bildschirm eine Anweisung, welcher Stein dem Kran in den Greifer zu schieben ist. Dann drückt man eine Taste, und der Roboter bringt den Stein an seinen Abstellplatz. Das muß man 18 mal wiederholen. Dann sind die schwarzen und weißen Steine in zwei Reihen neben dem Speilfeld aufgestellt. Von dort werden die Steine während des Spiels auf das Spielfeld gestellt und in diese Reihe werden sie auch nachher wieder zurückgestellt. Diese Prozedur ist natürlich überflüssig, wenn das Programm beim letzten mal normal beendet wurde und die Steine nicht verrutscht wurden. Außerdem ist die Spielstärkensteuerung etwas verändert worden, wodurch die kombinatorischen Explosionen des Back-Tracking-Algorithmus besser in den Griff zu bringen sind.

Mit dem Mühle-Programm können Sie nun gegen den Computer Mühle spielen. Die Regeln sind ja eigentlich klar. Jeder Spieler erhält neun Spielsteine. Diese werden abwechselnd auf beliebige (erlaubte) Positionen des Mühle-Feldes gestellt. Gesetzt wird solange, bis die neun Steine aufgebraucht sind. Danach wird nur noch (abwechselnd) gezogen, also man kann seine Steine nur noch auf einen durch eine Linie verbundenen, direkt benachbarten Punkt ziehen. Es herrscht Zugpflicht. Sollte einer der Spieler vom anderen eingesperrt werden, so daß dieser nicht mehr ziehen kann, so hat er verloren. Wenn man aber einmal drei Positionen in einer Linie besetzt hat, so nennt man das eine Mühle und man muß dem Gegner einen Stein wegnehmen, wenn die Mühle geschlossen wird. Dieser Stein darf aber nicht in einer geschlossenen Mühle des Gegners sein. Hat der Gegner alle Steine in Mühlen, bleibt das Schließen einer Mühle wirkungslos. Wenn einem Spieler dadurch alle Steine bis auf drei weggenommen wurden, erhält dieser die Möglichkeit, mit seinen Steinen zu springen. Er kann also seine Steine von jeder Position auf jede andere bewegen, mit Ausnahme der Positionen, die von seinen oder den Steinen des Gegners besetzt sind. Hat ein Spieler nur noch zwei Steine übrig, dann hat er verloren.

Genauso wird das auch mit dem Computer gespielt. Man muß seinen Zug aber per Hand eingeben, indem man die Positionen mit Buchstaben und Ziffern kennzeichnet, ähnlich wie das beim Schach üblich ist. Man nimmt für ein Feld also den Buchstaben, der auf dem Bildschirm senkrecht darunter oder darüber steht, und die Zahl, die in waagrechter Linie links oder rechts davon steht. Nachdem man das verstanden hat, kann man anfangen MUEHLE zu spielen.

Gestartet wird das Programm, indem man MUEHLE eingibt, wobei sich die MUEHLE-Diskette im aktuellen Laufwerk befindet. Zuerst kommt die Frage nach der Grundaufstellung. Wenn man da J eingibt, wird die Aufstellungsprozedur durchgeführt. Dann gibt man die Spielstärke 1 bis 9 ein. Die Spielstärken gehen von blitzschnell bis etwa einer Minute Wartezeit auf die Antwort. Spielstärke drei zeigt schon vernünftige Ergebnisse. Dann kommt die Frage, ob man anfangen will. Wenn man die auch beantwortet hat, kann man seinen Zug eingeben, sobald der "Cursor" hinter dem "Zug:" steht. Dort wird jeweils der Menschenzug eingegeben. Am Anfang kann man nur setzen, daher gibt man nur ein Feld ein. Z.B.: "A1 <ENTER>". Das wird dann auf dem Spielfeld ausgeführt, wenn der Zug erlaubt war, sonst erscheint auf dem Bildschirm unterhalb dem "Zug:"-Feld eine kleine Hilfestellung, die wieder verschwindet, wenn man den richtigen Zug eingibt. Wenn man beim Setzen eine Mühle zumachen kann, muß der Zug und der Stein des Gegners eingegeben werden, der genommen werden soll, sonst führt der Computer den Zug nämlich nicht aus. Wenn man zu Ziehen anfängt, oder auch Springen kann, dann müssen immer das Feld, von wo der Stein genommen werden soll, gefolgt von dem Feld, auf das gezogen oder gesprungen werden soll, eingegeben werden. Wenn eine Mühle dadurch geschlossen wird, ist als dritte Angabe das Feld einzugeben, auf dem der Stein geschlagen wird. Beispiel: "A1 D1 E4" bedeutet Ziehe von A1 nach D1 und nimm E4.

Falls man keine Idee mehr hat, kann man mit "V <ENTER>" den Computer einen Zugvorschlag machen lassen. Mit "S <ENTER>" kann man während des Spieles die Spielstärke ändern. Mit "H <ENTER>" kann man das Spiel beenden (und nicht mit CTRL-BREAK). Man kann auch während der Computer seinen Zug berechnet, durch einen Tastendruck die Berechnung abbrechen. Dann erscheint, nach einer kurzen Zeit die Frage "Abbrechen?". Wenn man die Berechnung dann abbricht kann es aber sein, daß der Computer noch nicht den besten Zug gefunden hatte.

Wie Mühle mit der Robotersteuerung zusammenarbeitet

Als das Programm Mühle geschrieben wurde (auf einem Apple II) war es natürlich nicht vorgesehen, daß es jemals einen Roboter zu steuern hätte. Entsprechende Schwierigkeiten waren deshalb auch unvermeidbar, denn auf dem Bildschirm können Mühlesteine einfach hingezaubert werden, was dem Roboter natürlich sehr schwer fällt. Kurz: Die gesamte Konzeption war nicht darauf ausgelegt. Die Prozedur Display_Move hatte einen Parameter weniger. Man mußte nämlich nur wissen, wo ein Leerzeichen, ein X oder ein O zu schreiben wäre. Der Roboter muß aber auch wissen, ob der Stein aufgehoben, abgesetzt, vom Standplatz genommen oder geschlagen wird. Dazu wurde ein zusätzlicher Parameter in der Display_Move-Prozedur aufgenommen, der diese fehlenden Informationen übermittelt. Wenn das Spiel zu Ende ist, sorgt die Prozedur Aufräumen dafür, daß die Steine wieder auf ihren Ausgangsort gestellt werden. Auch das ist natürlich viel einfacher, wenn man dazu nur den Bildschirm zu löschen braucht.

Wenn Sie aber Mühle ohne Roboter spielen wollen, brauchen Sie nur die beiden Include-Anweisungen zu entfernen, und an allen Stellen, wo der Compiler daraufhin einen Fehler meldet, die Roboter-Prozedur-Aufrufe ersatzlos streichen.

Listing of **INTERFAC.INC**

```
  1:  0 (*************************************************************************)
  2:  0 (* INTERFACE.INC ---- enthaelt das allgemeine HARDWARE-HANDLING         *)
  3:  0 (* eines FISCHERTECHNIK-ROBOTERS  (IBM-PC/AT)                           *)
  4:  0 (* Verbesserungen:                                                      *)
  5:  0 (* Das Interface passt sich automatisch an die vorhandene Hardware an  *)
  6:  0 (*   (d.h. Takt-rate, PORT-Adresse)                                     *)
  7:  0 (* Ausnahme: die 8/16-Bit-Version wird unterstuetzt, aber muss von Hand *)
  8:  0 (*   gewaehlt werden                                                    *)
  9:  0 (* (C) der Ueberarbeitung bei B.E.                                      *)
 10:  0 (*************************************************************************)
 11:  0
 12:  0 (* Bedienung der Hardware-Sektion *)
 13:  0 (* poti(ex) = 0..??? = x-poti-wert (integer)
 14:  0    poti(ey) = 0..??? = y-poti-wert (integer)
 15:  0    schalte(m7,ein);       = schalte motor #7 ein! (=rechts)
 16:  0    schalte(m3,aus);       = schalte motor #3 aus!
 17:  0    schalte(m1,rechts);    = schalte motor #1 auf rechtslauf!
 18:  0    schalte(m8,links);     = schalte motor #8 auf linkslauf!
 19:  0    lese(e1) = TRUE, falls schalter #1 an.
 20:  0    lese(e16) = FALSE, falls schalter #16 aus.
 21:  0    und immer zu anfang init ausfuehren!
 22:  0 *)
 23:  0
 24:  0 CONST BITCOUNT : INTEGER = 8; (* 8 ODER 16 BIT, JE NACH INTERFACE *)
 25:  0       DRUCKERPORT         = 1; (* NUMMER DES DRUCKER-PORTS (LPTn) *)
 26:  0       DATAPORT : INTEGER = $378; (* ADRESSE DRUCKER - SIEHE INIT *)
 27:  0       CTLPORT  : INTEGER = $379; (* BUSY-LINE - SIEHE INIT *)
 28:  0       M1=$03; M2=$0C; M3=$30; M4=$C0; M5=$300; M6=$C00; M7=$3000; M8=$C000;
 29:  0       MOTOREN : ARRAY [1..8] OF INTEGER     (* BEI 8 BIT NUR M1-M4, E1-E8 *)
 30:  0         = (M1,M2,M3,M4,M5,M6,M7,M8);
 31:  0       E1=1;        E2=2;        E3=4;        E4=8;
 32:  0       E5=16;       E6=32;       E7=64;       E8=128;
 33:  0       E9=256;      E10=512;     E11=1024;    E12=2048;
```

```
34:  0       E13=4096;    E14=8192;    E15=16384;    E16=$8000;
35:  0       SCHALTER : ARRAY [1..16] OF INTEGER
36:  0         = (E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15,E16);
37:  0       EX=0;          EY=1;
38:  0       EIN=$5555;     AUS=$FFFF;
39:  0       RECHTS=$5555;  LINKS=$AAAA;
40:  0
41:  0 VAR   last_word : integer;
42:  0
43:  0 PROCEDURE put_word(wert:integer);
44:  0   BEGIN INLINE (
45:  1                          (*        .8086                    *)
46:  1                          (*WERT     EXT                      *)
47:  1                          (*BITCOUNT EXT                      *)
48:  1                          (*DATAPORT EXT                      *)
49:  1 $8B/$9E/WERT             (*        MOV BX,WERT[BP]          *)
50:  1 /$2E/$8B/$0E/BITCOUNT    (*        MOV CX,CS:BITCOUNT       *)
51:  1 /$83/$F9/$08             (*        CMP CX,8                 *)
52:  1 /$77/$04                 (*        JA L0  ; MEHR ALS 8 BITS?*)
53:  1 /$8A/$FB                 (*        MOV BH,BL  ; 8 BITS: ANPASSUNG*)
54:  1 /$32/$DB                 (*        XOR BL,BL  ; DES WERTES  *)
55:  1 /$2E/$8B/$16/DATAPORT    (*L0       MOV DX,CS:DATAPORT       *)
56:  1 /$FA                     (*        CLI                      *)
57:  1 /$B0/$30                 (*L1       MOV AL,30H               *)
58:  1 /$D1/$C3                 (*        ROL BX,1                 *)
59:  1 /$73/$02                 (*        JNC L2                   *)
60:  1 /$0C/$04                 (*        OR AL,04H                *)
61:  1 /$EE                     (*L2       OUT DX,AL                *)
62:  1 /$0C/$08                 (*        OR AL,08H                *)
63:  1 /$EB/$00                 (*        JMP SHORT $+2            *)
64:  1 /$EE                     (*        OUT DX,AL                *)
65:  1 /$E2/$F0                 (*        LOOP L1                  *)
66:  1 /$B0/$39                 (*        MOV AL,39H               *)
67:  1 /$EB/$00                 (*        JMP SHORT $+2            *)
68:  1 /$EE                     (*        OUT DX,AL                *)
69:  1 /$FB                     (*        STI                      *)
70:  1                          (*        END                      *)
71:  1 )
72:  1 END;
73:  0
74:  0 FUNCTION get_word:integer;
75:  0   BEGIN INLINE (
76:  1                          (*        .8086                    *)
77:  1                          (*BITCOUNT EXT                      *)
78:  1                          (*DATAPORT EXT                      *)
79:  1                          (*CTLPORT  EXT                      *)
80:  1 $33/$DB                  (*        XOR BX,BX                *)
81:  1 /$2E/$8B/$0E/BITCOUNT    (*        MOV CX,CS:BITCOUNT       *)
82:  1 /$2E/$8B/$16/DATAPORT    (*        MOV DX,CS:DATAPORT       *)
83:  1 /$FA                     (*        CLI                      *)
84:  1 /$B0/$32                 (*        MOV AL,32H               *)
85:  1 /$EE                     (*        OUT DX,AL                *)
86:  1 /$0C/$08                 (*        OR AL,08H                *)
87:  1 /$EB/$00                 (*        JMP SHORT $+2            *)
88:  1 /$EE                     (*        OUT DX,AL                *)
89:  1 /$2E/$8B/$16/CTLPORT     (*L1       MOV DX,CS:CTLPORT        *)
90:  1 /$EC                     (*        IN AL,DX                 *)
91:  1 /$D0/$C0                 (*        ROL AL,1                 *)
92:  1 /$D1/$D3                 (*        RCL BX,1                 *)
93:  1 /$2E/$8B/$16/DATAPORT    (*        MOV DX,CS:DATAPORT       *)
94:  1 /$B0/$30                 (*        MOV AL,30H               *)
95:  1 /$EE                     (*        OUT DX,AL                *)
96:  1 /$B0/$38                 (*        MOV AL,38H               *)
```

```
 97:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
 98:  1 /$EE                      (*        OUT DX,AL                        *)
 99:  1 /$E2/$E7                  (*        LOOP L1                          *)
100:  1 /$FB                      (*        STI                              *)
101:  1 /$8B/$C3                  (*        MOV AX,BX                        *)
102:  1 /$2E/$83/$3E/BITCOUNT/$08
103:  1                           (*        CMP W CS:BITCOUNT,8              *)
104:  1 /$76/$02                  (*        JBE L2  ; 16 BITS?               *)
105:  1 /$86/$C4                  (*        XCHG AL,AH  ; JA: AL/AH VERTAUSCHEN *)
106:  1 /$8B/$E5                  (*L2       MOV SP,BP                        *)
107:  1 /$5D                      (*        POP BP                           *)
108:  1 /$C2/$02/$00              (*        RET 2                            *)
109:  1                           (*        END                              *)
110:  1 )
111:  1 END;
112:  0
113:  0 FUNCTION poti(potinr:integer):integer;
114:  0   BEGIN INLINE (
115:  1                           (*        .8086                            *)
116:  1                           (*POTINR   EXT                              *)
117:  1                           (*DATAPORT EXT                              *)
118:  1                           (*CTLPORT  EXT                              *)
119:  1                           (*TIMER    EQU 40H                         *)
120:  1 $8A/$86/POTINR              (*        MOV AL,POTINR[BP]                *)
121:  1 /$2E/$8B/$16/DATAPORT      (*        MOV DX,CS:DATAPORT               *)
122:  1 /$24/$01                  (*        AND AL,1                         *)
123:  1 /$B1/$04                  (*        MOV CL,4                         *)
124:  1 /$D2/$E0                  (*        SHL AL,CL                        *)
125:  1 /$34/$10                  (*        XOR AL,10H                       *)
126:  1 /$04/$90                  (*        ADD AL,90H                       *)
127:  1 /$FA                      (*L1       CLI                              *)
128:  1 /$B9/$FF/$07              (*        MOV CX,07FFH                     *)
129:  1 /$EE                      (*        OUT DX,AL                        *)
130:  1 /$B0/$38                  (*        MOV AL,38H                       *)
131:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
132:  1 /$EE                      (*        OUT DX,AL                        *)
133:  1 /$B0/$00                  (*        MOV AL,00H  ; READ TIMER #0      *)
134:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
135:  1 /$E6/$43                  (*        OUT TIMER+3,AL  ; TIMER-CONTROL  *)
136:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
137:  1 /$E4/$40                  (*        IN AL,TIMER+0                    *)
138:  1 /$8A/$D8                  (*        MOV BL,AL                        *)
139:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
140:  1 /$E4/$40                  (*        IN AL,TIMER+0                    *)
141:  1 /$8A/$F8                  (*        MOV BH,AL                        *)
142:  1 /$8B/$F3                  (*        MOV SI,BX  ; SI=TIMER-WERT       *)
143:  1 /$2E/$8B/$16/CTLPORT      (*        MOV DX,CS:CTLPORT                *)
144:  1 /$EC                      (*L2       IN AL,DX                         *)
145:  1 /$D0/$D0                  (*        RCL AL,1                         *)
146:  1 /$73/$08                  (*        JNC L3                           *)
147:  1 /$E2/$F9                  (*        LOOP L2                          *)
148:  1 /$B8/$FF/$0F              (*        MOV AX,0FFFH                     *)
149:  1 /$E9/$18/$00              (*        JMP L4                           *)
150:  1 /$B0/$00                  (*L3       MOV AL,00H  ; READ TIMER #0      *)
151:  1 /$E6/$43                  (*        OUT TIMER+3,AL  ; TIMER-CONTROL  *)
152:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
153:  1 /$E4/$40                  (*        IN AL,TIMER+0                    *)
154:  1 /$8A/$D8                  (*        MOV BL,AL                        *)
155:  1 /$EB/$00                  (*        JMP SHORT $+2                    *)
156:  1 /$E4/$40                  (*        IN AL,TIMER+0                    *)
157:  1 /$8A/$F8                  (*        MOV BH,AL  ; BX=NEUER TIMER-WERT *)
158:  1 /$2B/$F3                  (*        SUB SI,BX  ; SI=ZEIT IN 0.4 MIC.SEC'*)
159:  1 /$B1/$04                  (*        MOV CL,4                         *)
```

```
  160:  1 /$D3/$EE                (*        SHR SI,CL                    *)
  161:  1 /$8B/$C6                (*        MOV AX,SI                    *)
  162:  1 /$FB                    (*L4      STI                          *)
  163:  1 /$8B/$E5                (*        MOV SP,BP                    *)
  164:  1 /$5D                    (*        POP BP                       *)
  165:  1 /$C2/$04/$00            (*        RET 2+2                      *)
  166:  1                         (*        END                          *)
  167:  1 )
  168:  1 END;
  169:  0
  170:  0 PROCEDURE init;  (* initialisierung der hardware-sektion *)
  171:  0 VAR ports : ARRAY [1..3] OF integer ABSOLUTE $40:8;
  172:  0   BEGIN
  173:  1     IF ports[DRUCKERPORT]=0 THEN
  174:  1       BEGIN
  175:  2         writeln;
  176:  2         writeln('Fehler: "LPT',DRUCKERPORT,'" nicht vorhanden!');
  177:  2         halt
  178:  2       END;
  179:  1     dataport:=ports[DRUCKERPORT];      (* LPTn *)
  180:  1     ctlport:=succ(dataport);
  181:  1     put_word(0);
  182:  1     last_word:=0
  183:  1   END;
  184:  0
  185:  0 PROCEDURE schalte (motor,aktion : integer);
  186:  0   BEGIN
  187:  1     last_word:=(last_word OR motor) XOR (aktion AND motor);
  188:  1     put_word(last_word)
  189:  1   END;
  190:  0
  191:  0 FUNCTION lese (eingang : integer) : boolean;
  192:  0   BEGIN
  193:  1     lese:=(get_word AND eingang) <> 0
  194:  1   END;
  195:  0
  196:  0 (* ende hardware-sektion *)
```

Listing of **LAGER.INC**

```
    1:  0 (****************************** LAGER.INC ***********************************)
    2:  0 (* Ab hier beginnt die spezielle Steuerung                                  *)
    3:  0 (* eines Lager-Kranes                                                       *)
    4:  0 (****************************************************************************)
    5:  0
    6:  0 CONST Greif_Zeit  =1000; (* Millisekunden, fuer Handschliessen  *)
    7:  0       Greifer     =  M4; (* Hand-motor                          *)
    8:  0       Greifer_Auf =  E7; (* Schalter fuer Hand offen            *)
    9:  0
   10:  0       Heber       =  M3; (* Z-Koordinate                        *)
   11:  0       Heber_Oben  =  E5; (* oberer Anschlag erreicht            *)
   12:  0       Heber_Unten =  E6; (* unterer Anschlag erreicht           *)
   13:  0
   14:  0       Nachlauf    =   3; (* Ausschalten, bevor die Position erreicht *)
   15:  0       Gegenlauf   =   5; (* Warteschleife fuer Gegenlauf (ß ms)  *)
   16:  0       Warte_Phase = 250; (* Warteschleife fuer Motor-Nachlauf (ß ms) *)
   17:  0       Halte_Zeit  = 250; (* Millisekunden fuer Halten (bei home) *)
   18:  0
   19:  0 VAR Position    : ARRAY [1..2] OF integer;
   20:  0     Eingang     : ARRAY [1..2] OF boolean;
   21:  0     Ausgelassen,
   22:  0     Aufgehoben  : boolean;
```

```
23:  0
24:  0 PROCEDURE Zufassen;
25:  0 VAR i : integer;
26:  0   BEGIN
27:  1     IF Ausgelassen THEN
28:  1       FOR i:=1 TO Greif_Zeit DIV 10 DO
29:  1         BEGIN
30:  2           schalte(Greifer,LINKS);
31:  2           delay(10)
32:  2         END;
33:  1     schalte(Greifer,AUS);
34:  1     Ausgelassen:=FALSE
35:  1   END;
36:  0
37:  0 PROCEDURE Loslassen;
38:  0   BEGIN
39:  1     IF NOT Ausgelassen THEN
40:  1        BEGIN
41:  2          schalte(Greifer,RECHTS);
42:  2          WHILE lese(Greifer_Auf) DO
43:  2        END;
44:  1     schalte(Greifer,AUS);
45:  1     Ausgelassen:=TRUE
46:  1   END;
47:  0
48:  0 PROCEDURE Hinunter;
49:  0   BEGIN
50:  1     IF Aufgehoben THEN
51:  1       BEGIN
52:  2         schalte(Heber,LINKS);
53:  2         WHILE lese(Heber_Unten) DO
54:  2       END;
55:  1     schalte(Heber,AUS);
56:  1     Aufgehoben:=FALSE
57:  1   END;
58:  0
59:  0 PROCEDURE Hinauf;
60:  0   BEGIN
61:  1     IF NOT Aufgehoben THEN
62:  1        BEGIN
63:  2          schalte(Heber,RECHTS);
64:  2          WHILE lese(Heber_Oben) DO
65:  2        END;
66:  1     schalte(Heber,AUS);
67:  1     Aufgehoben:=TRUE
68:  1   END;
69:  0
70:  0 PROCEDURE MoveTo(X,Y : integer);
71:  0 CONST Nach : ARRAY [-1..+1] OF integer = (RECHTS,AUS,LINKS);
72:  0 VAR k,W : integer;
73:  0     Ziele,
74:  0     Richtungen  : ARRAY [1..2] OF integer;
75:  0     Wait        : ARRAY [1..2] OF integer;
76:  0     Done        : ARRAY [1..2] OF boolean;
77:  0
78:  0   PROCEDURE SetzeRichtungen(i,Z : integer);
79:  0     BEGIN
80:  1       IF abs(Position[i]-Z)>Nachlauf THEN
81:  1         IF Position[i]>=Z
82:  1           THEN Richtungen[i]:=-1
83:  1           ELSE Richtungen[i]:=+1
84:  1         ELSE Richtungen[i]:=0;
85:  1       Ziele[i]:=Z-Richtungen[i]*Nachlauf
```

```
 86:  1    END;
 87:  0
 88:  0  BEGIN
 89:  1    IF (X<0) OR (Y<0)
 90:  1    THEN
 91:  1      BEGIN
 92:  2        Writeln;
 93:  2        Writeln('Fehler: (',X,',',Y,') nicht im Steuerbereich!');
 94:  2        halt
 95:  2      END;
 96:  1    SetzeRichtungen(1,X);
 97:  1    SetzeRichtungen(2,Y);
 98:  1    FOR k:=1 TO 2 DO BEGIN Wait[k]:=0; Done[k]:=FALSE END;
 99:  1    FOR k:=1 TO 2 DO schalte(MOTOREN[k],Nach[Richtungen[k]]);
100:  1    REPEAT
101:  2      W:=get_word;
102:  2      FOR k:=1 TO 2 DO
103:  2        BEGIN
104:  3          IF Richtungen[k]<>0 THEN
105:  3            IF (W AND SCHALTER[2*k]<>0) <> Eingang[k] THEN
106:  3              BEGIN
107:  4                Eingang[k]:=NOT Eingang[k];
108:  4                IF Eingang[k] THEN
109:  4                  BEGIN
110:  5                    Position[k]:=Position[k]+Richtungen[k];
111:  5                    IF Position[k]=Ziele[k] THEN
112:  5                      BEGIN
113:  6                        schalte(MOTOREN[k],Nach[-Richtungen[k]]);
114:  6                        wait[k]:=Gegenlauf
115:  6                      END
116:  5                  END
117:  4              END;
118:  3          IF Richtungen[k]<0 THEN
119:  3            IF W AND SCHALTER[pred(2*k)]=0 THEN
120:  3              BEGIN
121:  4                schalte(MOTOREN[k],AUS);
122:  4                Position[k]:=0;
123:  4                Richtungen[k]:=0
124:  4              END;
125:  3          IF Wait[k]<>0 THEN
126:  3            BEGIN
127:  4              Wait[k]:=pred(Wait[k]);
128:  4              IF Wait[k]=0 THEN
129:  4              IF Done[k] THEN Richtungen[k]:=0
130:  4                ELSE
131:  4                  BEGIN
132:  5                    Schalte(Motoren[k],AUS);
133:  5                    Wait[k]:=Warte_Phase;
134:  5                    Done[k]:=TRUE
135:  5                  END
136:  4            END
137:  3        END
138:  2    UNTIL (Richtungen[1]=0) AND (Richtungen[2]=0)
139:  1  END;
140:  0
141:  0 PROCEDURE Home;
142:  0   VAR h1,h2 : boolean;
143:  0       x1,x2 : integer;
144:  0       i     : integer;
145:  0   BEGIN
146:  1     init;
147:  1     FOR i:=1 TO 2 DO Position[i]:=0;
148:  1     Aufgehoben:=TRUE;
```

```
  149:  1     Ausgelassen:=TRUE;
  150:  1     WHILE lese(Greifer_Auf) DO schalte(Greifer,RECHTS);
  151:  1     schalte(Greifer,AUS);
  152:  1     WHILE lese(Heber_Oben) DO schalte(Heber,RECHTS);
  153:  1     schalte(Heber,AUS);
  154:  1     h1:=true; h2:=true;
  155:  1     REPEAT
  156:  2       IF h1 THEN
  157:  2         IF lese(E1)
  158:  2           THEN schalte(M1,RECHTS)
  159:  2           ELSE BEGIN schalte(M1,AUS); h1:=false END;
  160:  2       IF h2 THEN
  161:  2         IF lese(E3)
  162:  2           THEN schalte(M2,RECHTS)
  163:  2           ELSE BEGIN schalte(M2,AUS); h2:=false END
  164:  2     UNTIL NOT (h1 OR h2);
  165:  1     x1:=0; x2:=0;
  166:  1     schalte(M1+M2,LINKS);
  167:  1     REPEAT
  168:  2       CASE x1 OF
  169:  3         0 : IF lese(E1)     THEN x1:=succ(x1);
  170:  3         1 : IF NOT lese(E2) THEN x1:=succ(x1);
  171:  3         2 : IF lese(E2)     THEN x1:=succ(x1);
  172:  3         3 : IF NOT lese(E2) THEN x1:=succ(x1);
  173:  3         4 : IF lese(E2)     THEN BEGIN schalte(M1,AUS); x1:=succ(x1) END;
  174:  3         5 :
  175:  3       END;
  176:  2       CASE x2 OF
  177:  3         0 : IF lese(E3)     THEN x2:=succ(x2);
  178:  3         1 : IF NOT lese(E4) THEN x2:=succ(x2);
  179:  3         2 : IF lese(E4)     THEN x2:=succ(x2);
  180:  3         3 : IF NOT lese(E4) THEN x2:=succ(x2);
  181:  3         4 : IF lese(E4)     THEN BEGIN schalte(M2,AUS); x2:=succ(x2) END;
  182:  3         5 :
  183:  3       END
  184:  2     UNTIL (x1=5) AND (x2=5);
  185:  1     delay(Halte_Zeit);
  186:  1     FOR i:=1 TO 2 DO Eingang[i]:=lese(Schalter[2*i])
  187:  1   END;


Listing of MUEHLE.PAS

    1:  0 PROGRAM Muehle (input,output);  (*$C-*)
    2:  0
    3:  0 {$IINTERFACE.INC}
    4:  0 {$ILAGER.INC}
    5:  0
    6:  0 (*
    7:  0       Verwendet Shannon A - Strategie und den
    8:  0       Alpha - Beta - Algorithmus
    9:  0 *)
   10:  0 CONST zeigen  = true;
   11:  0       MaxSp   = 9;       (* Maximale Spielstaerke *)
   12:  0
   13:  0 TYPE tstein    = (leer,Spieler,Rechner);
   14:  0      tstr      = STRING[30];
   15:  0      tzug      = ARRAY[0..3] OF integer;
   16:  0      tStat     = ARRAY[Spieler..Rechner] OF RECORD
   17:  1                              Modus : (Setzen,Ziehen,Springen);
   18:  1                              steine,zaehler : integer;
   19:  1                            END;
   20:  0      tbrett = RECORD
```

```
21:  1                Anzahl : 0..1;
22:  1                k,L    : integer;
23:  1                Brett  : ARRAY[0..23] OF tstein;
24:  1                Status : tStat;
25:  1                CASE integer OF
26:  1                  0 : (Zug  : tzug);
27:  1                  1 : (von,              (* von Feldposition *)
28:  1                       zu,               (* zu  Feldposition *)
29:  1                       Schl_Nr:integer; (* Nr. des schlagbaren Steins *)
30:  1                       Muehle : boolean)
31:  1              END;
32:  0
33:  0 CONST steine:ARRAY[leer..Rechner] OF STRING[5]=(' ','X','O');
34:  0       geg : ARRAY[Spieler..Rechner] OF tstein=(Rechner,Spieler);
35:  0
36:  0   (* Tabelle mit den Nachbarpositionen, mit denen eine
37:  0      Muehle geschlossen werden kann *)
38:  0
39:  0       Muehlen_Nachb:ARRAY[0..23,0..3] OF byte=
40:  0              ((1,2,9,21),(0,2,4,7),(0,1,14,23),(4,5,10,18),
41:  0               (3,5,1,7),(3,4,13,20),(7,8,11,15),(6,8,1,4),
42:  0               (6,7,12,17),(10,11,0,21),(9,11,3,18),(9,10,6,15),
43:  0               (13,14,8,17),(5,20,12,14),(2,23,12,13),(6,11,16,17),
44:  0               (15,17,19,22),(15,16,8,12),(19,20,3,10),(18,20,16,22),
45:  0               (18,19,5,13),(22,23,0,9),(21,23,16,19),(21,22,2,14));
46:  0
47:  0   (* Tabelle mit den max. vier moeglichen Nachbarpositionen
48:  0      eines Steines. 30 gibt eine nicht vorh. Position an *)
49:  0
50:  0       Nachbar:ARRAY[0..23,0..3] OF byte=
51:  0              ((1,9,30,30),(2,4,0,30),(30,14,1,30),(4,10,30,30),
52:  0               (5,7,3,1),(30,13,4,30),(7,11,30,30),(8,30,6,4),
53:  0               (30,12,7,30),(10,21,30,0),(11,18,9,3),(30,15,10,6),
54:  0               (13,17,30,8),(14,20,12,5),(30,23,13,2),(16,30,30,11),
55:  0               (17,19,15,30),(30,30,16,12),(19,30,30,10),(20,22,18,16),
56:  0               (30,30,19,13),(22,30,30,9),(23,30,21,19),(30,30,22,14));
57:  0
58:  0 (* Tabelle mit den 16 moeglichen Muehlen *)
59:  0
60:  0       Muehlen:ARRAY[0..15,0..2] OF byte=
61:  0              ((0,1,2),(3,4,5),(6,7,8),(9,10,11),(12,13,14),
62:  0               (15,16,17),(18,19,20),(21,22,23),(0,9,21),
63:  0               (3,10,18),(6,11,15),(1,4,7),(16,19,22),
64:  0               (8,12,17),(5,13,20),(2,14,23));
65:  0
66:  0 (* Liste mit Feldbezeichnungen der Positionen auf dem Spielbrett *)
67:  0
68:  0       posit :ARRAY[0..23] OF STRING[2]=
69:  0             ('A1','D1','G1','B2','D2','F2','C3','D3',
70:  0              'E3','A4','B4','C4','E4','F4','G4','C5',
71:  0              'D5','E5','B6','D6','F6','A7','D7','G7');
72:  0
73:  0 VAR  Feld          : tbrett;          (* Spielbrett *)
74:  0       Spiel_ist_aus : boolean;
75:  0       Spielstaerke  : integer;
76:  0       Muehlen_menge : ARRAY[0..15] OF SET OF 0..23;
77:  0       (* Wird auf die selben Werte, wie die Konstante Muehlen gesetzt
78:  0          ==> schnellere Abfrage, ob in Muehle                    *)
79:  0      MinMax         : integer;
80:  0      BestZug        : tzug;
81:  0
82:  0 TYPE ListenInhalt= RECORD
83:  1                      value : integer;
```

```
 84:  1                    Zug    : tzug;
 85:  1                  END;
 86:  0
 87:  0 FUNCTION kleiner(x,y : ListenInhalt):boolean;
 88:  0   BEGIN kleiner:=x.value>y.value END; (* Absteigende Liste *)
 89:  0
 90:  0 (*************************************************************************)
 91:  0 (*  Bibliotheks-Modul LISTE.BIB                                          *)
 92:  0 (*************************************************************************)
 93:  0
 94:  0 TYPE   Liste         = ^ListenEintrag;
 95:  0      ListenEintrag = RECORD
 96:  1                        Eintrag   : ListenInhalt;
 97:  1                        Naechster : Liste
 98:  1                      END;
 99:  0
100:  0 PROCEDURE Einfuegen(VAR L : Liste; E : ListenInhalt);
101:  0   VAR p,q : Liste;
102:  0   BEGIN
103:  1     q:=NIL; p:=L;
104:  1     WHILE p<>NIL DO
105:  1       WITH p^ DO
106:  1         IF kleiner(Eintrag,E) THEN BEGIN q:=p; p:=naechster END ELSE p:=NIL;
107:  1     new(p); p^.Eintrag:=E;
108:  1     IF q=NIL
109:  1       THEN BEGIN p^.Naechster:=L; L:=p END
110:  1       ELSE BEGIN p^.Naechster:=q^.naechster; q^.Naechster:=p END;
111:  1   END;
112:  0
113:  0 PROCEDURE initvars;
114:  0 VAR a,b:integer;
115:  0 BEGIN
116:  1   FOR a:=0 TO 15 DO
117:  1   BEGIN
118:  2     Muehlen_menge[a]:=[];
119:  2     FOR b:=0 TO 2 DO Muehlen_menge[a]:=[Muehlen[a,b]]+Muehlen_menge[a];
120:  2   END;
121:  1 END;
122:  0
123:  0 PROCEDURE zeige_Status(x : tstein);
124:  0 CONST ModusAnz:ARRAY[Setzen..Springen] OF STRING[8]=
125:  0                  ('Setzen','Ziehen','Springen');
126:  0 BEGIN
127:  1   WITH Feld DO BEGIN
128:  2     gotoxy(57,10); write('Spielersteine : ',Status[Spieler].steine);
129:  2     gotoxy(57,11); write('Rechnersteine : ',Status[Rechner].steine);
130:  2     gotoxy(57,14); write('Modus : ',ModusAnz[Status[x].Modus]);
131:  2   END;
132:  1   ClrEol;
133:  1 END;
134:  0
135:  0 PROCEDURE hole_Spielstaerke;
136:  0 VAR CH:char;
137:  0 BEGIN
138:  1   gotoxy(35,22); ClrEol;
139:  1   LowVideo; write('Spielstaerke (1..',MaxSp,') ? '); HighVideo;
140:  1   REPEAT
141:  2     read(KBD,CH); Spielstaerke:=ord(CH)-ord('0');
142:  2   UNTIL Spielstaerke IN [1..MaxSp];
143:  1   gotoxy(57,8); write('Spielstaerke ',Spielstaerke);
144:  1 END;
145:  0
146:  0 FUNCTION Antwort_Ja(Frage:tstr):boolean;
```

```
147:  0 VAR CH:char;
148:  0 BEGIN
149:  1   gotoxy(35,22); ClrEol;
150:  1   LowVideo; write(Frage,' (J/N) ? '); HighVideo;
151:  1   REPEAT
152:  2     read(KBD,CH); CH:=upcase(CH);
153:  2   UNTIL CH IN ['J','N'];
154:  1   Antwort_Ja:=CH='J';
155:  1 END;
156:  0
157:  0 PROCEDURE verloren(wer_denn:tstein);
158:  0 BEGIN
159:  1   gotoxy(1,23); LowVideo;
160:  1   IF wer_denn=Spieler THEN write('  Sie haben')
161:  1     ELSE write('  Ich habe');
162:  1   write(' verloren ! '); normvideo;
163:  1   Spiel_ist_aus:=true;
164:  1 END;
165:  0
166:  0 PROCEDURE Initial;
167:  0 (* Bildschirm aufbauen und Anfangswerte festlegen *)
168:  0 VAR k:tstein; i:integer;
169:  0 BEGIN
170:  1   ClrScr; HighVideo; writeln('CHIP-Special-Muehle':41); gotoxy(1,3);
171:  1   writeln('':19,'A   B   C   D   E   F   G');
172:  1   writeln; LowVideo;
173:  1   writeln('':19,'  ---------   ---------  ','Mensch   : ':23,steine[Spieler]);
174:  1   writeln('':19,'|         |         |','Computer : ':23,steine[Rechner]);
175:  1   writeln('':19,'|     -----   -----     |');
176:  1   writeln('':19,'|   |     |     |   |');
177:  1   writeln('':19,'|   |   --- ---   |   |');
178:  1   writeln('':19,'|   |   |       |   |   |');
179:  1   writeln('':19,' --- ---           --- --- ');
180:  1   writeln('':19,'|   |   |       |   |   |');
181:  1   writeln('':19,'|   |   --- ---   |   |');
182:  1   writeln('':19,'|   |     |     |   |');
183:  1   writeln('':19,'|     -----   -----     |');
184:  1   writeln('':19,'|         |         |');
185:  1   writeln('':19,'  ---------   ---------  ');
186:  1   writeln; HighVideo;
187:  1   writeln('':19,'A   B   C   D   E   F   G');
188:  1   FOR i:=1 TO 7 DO
189:  1   BEGIN
190:  2     gotoxy(16,2*i+3); write(i);
191:  2     gotoxy(48,2*i+3); write(i);
192:  2   END;
193:  1   gotoxy(5,22); write('Computerzug :');
194:  1   fillchar(Feld.Brett,SizeOf(Feld.Brett),leer);
195:  1   FOR k:=Spieler TO Rechner DO
196:  1     WITH Feld.Status[k] DO BEGIN
197:  2       Modus:=Setzen; steine:=0; zaehler:=9;
198:  2     END;
199:  1   Feld.Muehle:=false; Spiel_ist_aus:=false;
200:  1   hole_Spielstaerke;
201:  1 END;
202:  0
203:  0 FUNCTION Muehle_zu(ad:integer; Art:tstein):boolean;
204:  0 (* Testet, ob eine Muehle geschlossen wurde *)
205:  0 BEGIN
206:  1   IF (Feld.Brett[Muehlen_Nachb[ad,0]]=Art) AND
207:  1      (Feld.Brett[Muehlen_Nachb[ad,1]]=Art) THEN Muehle_zu:=true
208:  1   ELSE
209:  1   Muehle_zu:=(Feld.Brett[Muehlen_Nachb[ad,3]]=Art) AND
```

```
210:  1              (Feld.Brett[Muehlen_Nachb[ad,2]]=Art);
211:  1 END;
212:  0
213:  0 PROCEDURE Zug_ermitteln(Art:tstein);
214:  0 (* Liefert einen Zug zurueck; Anzahl=0, falls kein Zug moeglich *)
215:  0 LABEL exit;
216:  0 VAR j:integer;
217:  0 PROCEDURE teste_Muehle;
218:  0 (* Testen, ob eine Muehle geschlossen wurde; wenn ja dann
219:  0    schlagbaren Stein ermitteln *)
220:  0 LABEL exit;
221:  0 BEGIN
222:  1   Feld.Muehle:=false;
223:  1   IF Muehle_zu(Feld.zu,Art) THEN BEGIN
224:  2     WHILE Feld.Schl_Nr<23 DO BEGIN
225:  3       Feld.Schl_Nr:=succ(Feld.Schl_Nr);
226:  3       IF Feld.Brett[Feld.Schl_Nr]=geg[Art] THEN
227:  3         IF NOT Muehle_zu(Feld.Schl_Nr,geg[Art]) THEN BEGIN
228:  4           Feld.Anzahl:=1; Feld.Muehle:=true; GOTO EXIT;
229:  4         END;
230:  3     END;
231:  2   END;
232:  1   Feld.Schl_Nr:=-1;
233:  1   exit:
234:  1 END;
235:  0
236:  0 BEGIN  (* Zug_ermitteln *)
237:  1 WITH Feld DO
238:  1 BEGIN
239:  2   Anzahl:=0;
240:  2   WITH Status[Art] DO IF steine+zaehler<3 THEN GOTO exit;
241:  2   IF Schl_Nr>-1 THEN teste_Muehle;
242:  2   WHILE ((k<23) OR (L>-1)) AND (Anzahl=0) DO BEGIN
243:  3     IF L=-1 THEN BEGIN  (* Naechsten Stein behandeln *)
244:  4                 k:=succ(k); Schl_Nr:=-1;
245:  4                 END;
246:  3     CASE Status[Art].Modus OF
247:  4       Setzen : IF Brett[k]=leer THEN BEGIN
248:  5                  Anzahl:=1; zu:=k;
249:  5                  teste_Muehle; GOTO exit;
250:  5                END;
251:  4       Ziehen : IF Brett[k]=Art THEN BEGIN
252:  5                  WHILE L<3 DO BEGIN
253:  6                    L:=succ(L);
254:  6                    j:=Nachbar[k,L];
255:  6                    IF j<>30 THEN
256:  6                      IF Brett[j]=leer THEN BEGIN
257:  7                        Anzahl:=1; von:=k;
258:  7                        zu:=j; Brett[k]:=leer;
259:  7                        teste_Muehle; Brett[k]:=Art; GOTO exit;
260:  7                      END;
261:  6                  END; (* while *)
262:  5                  L:=-1; (* alle Richtungen durch *)
263:  5                END;
264:  4       Springen: IF Brett[k]=Art THEN BEGIN
265:  5                  WHILE L<23 DO BEGIN
266:  6                    L:=succ(L);
267:  6                    IF Brett[L]=leer THEN BEGIN
268:  7                      Anzahl:=1; von:=k;
269:  7                      zu:=L; Brett[k]:=leer;
270:  7                      teste_Muehle; Brett[k]:=Art; GOTO exit;
271:  7                    END;
272:  6                  END;
```

```
273:  5                  L:=-1;
274:  5                END;
275:  4      END; (* case *)
276:  3    END; (* while *)
277:  2  END; (* with *)
278:  1  exit:
279:  1 END;
280:  0
281:  0 CONST Warten:integer=0;
282:  0 PROCEDURE home1;
283:  0   BEGIN
284:  1   Warten:=Warten+1;
285:  1   IF Warten=10 THEN
286:  1     BEGIN
287:  2       home;
288:  2       Warten:=0
289:  2     END
290:  1   END;
291:  0
292:  0 PROCEDURE Display_Move(m,ad:integer; Stein:tstein);
293:  0 CONST Positionen:ARRAY[0..23,1..2] OF integer=
294:  0   ((1150,2300),(2350,2300),(3500,2300), (1550,2000),(2350,2000),(3100,2000),
295:  0    (1950,1700),(2350,1700),(2750,1700), (1150,1400),(1550,1400),(1950,1400),
296:  0    (2750,1400),(3100,1400),(3500,1400), (1950,1100),(2350,1100),(2750,1100),
297:  0    (1550, 800),(2350, 800),(3100, 800), (1150, 500),(2350, 500),(3500, 500));
298:  0 VAR X,Y : integer;
299:  0 BEGIN
300:  1   X:=ord(posit[ad][1])-ord('A'); Y:=ord(posit[ad][2])-ord('1');
301:  1   gotoxy(20+X*4,5+Y*2); write(steine[Stein]);
302:  1   gotoxy(20+X*4,5+Y*2);
303:  1   CASE m OF
304:  2   3 : BEGIN (* Stein wird geschlagen *)
305:  3         MoveTo(Positionen[ad,1],Positionen[ad,2]);
306:  3         Hinunter; Zufassen; Hinauf;
307:  3         WITH Feld.Status[Feld.Brett[ad]] DO
308:  3           BEGIN
309:  4             X:=50+400*ord(Feld.Brett[ad]=Rechner);
310:  4             Y:=200+300*(8-(+zaehler +steine))
311:  4           END;
312:  3         MoveTo(X,Y); Hinunter; Loslassen; Hinauf;
313:  3         home1
314:  3       END;
315:  2   2 : BEGIN (* Stein wird neu in's Spiel gebracht *)
316:  3         WITH Feld.Status[Stein] DO
317:  3           BEGIN
318:  4             X:=50+400*ord(Stein=Rechner);
319:  4             Y:=200+300*(8-zaehler)
320:  4           END;
321:  3         MoveTo(X,Y); Hinunter; Zufassen; Hinauf;
322:  3         MoveTo(Positionen[ad,1],Positionen[ad,2]);
323:  3         Hinunter; Loslassen; Hinauf;
324:  3         home1
325:  3       END;
326:  2   0 : BEGIN (* Stein wird aufgehoben *)
327:  3         MoveTo(Positionen[ad,1],Positionen[ad,2]);
328:  3         Hinunter; Zufassen; Hinauf;
329:  3       END;
330:  2   1 : BEGIN (* Stein, der sich in der Klaue befindet wird gesetzt *)
331:  3         MoveTo(Positionen[ad,1],Positionen[ad,2]);
332:  3         Hinunter; Loslassen; Hinauf;
333:  3         home1
334:  3       END
335:  2     END;
```

```
336:  1   END;
337:  0
338:  0 PROCEDURE mache_Zug(Stein:tstein; show:boolean);
339:  0 (* Zug ausfuehren und gegebenenfalls auf dem Spielbrett anzeigen *)
340:  0
341:  0 PROCEDURE setze_Stein(m,ad:integer; Stein:tstein);
342:  0 BEGIN
343:  1   IF show THEN Display_Move(m,ad,Stein);
344:  1   Feld.Brett[ad]:=Stein;
345:  1 END;
346:  0
347:  0 BEGIN (* mache_Zug *)
348:  1   WITH Feld DO
349:  1   BEGIN
350:  2     CASE Status[Stein].Modus OF
351:  3       Setzen : BEGIN
352:  4                  WITH Status[Stein] DO BEGIN
353:  5                    steine:=succ(steine); zaehler:=pred(zaehler);
354:  5                    IF zaehler=0 THEN Modus:=Ziehen;
355:  5                    setze_Stein(2,zu,Stein)
356:  5                  END;
357:  4                END;
358:  3       Springen,
359:  3       Ziehen : BEGIN
360:  4                  setze_Stein(0,von,leer);
361:  4                  setze_Stein(1,zu,Stein)
362:  4                END
363:  3     END;
364:  2     IF Muehle THEN WITH Status[geg[Stein]] DO
365:  2     BEGIN
366:  3       steine:=pred(steine);
367:  3       IF (steine=3) AND (zaehler=0) THEN Modus:=Springen;
368:  3       setze_Stein(3,Schl_Nr,leer);
369:  3       IF show THEN IF steine+zaehler=2 THEN verloren(geg[Stein]);
370:  3     END;
371:  2   END;
372:  1 END;
373:  0
374:  0 FUNCTION Bewertung(AmZug:tstein):integer;
375:  0 CONST maxval = maxint;
376:  0 VAR Summe,
377:  0     spst,
378:  0     rest  : integer;
379:  0     x     : Spieler..Rechner;
380:  0     om,gm : ARRAY[Spieler..Rechner] OF integer;
381:  0
382:  0 PROCEDURE offen_zu_Muehlen(k:tstein; VAR om,gm :integer);
383:  0 (* Anzahl der offenen und geschlossenen Muehlen ermitteln *)
384:  0 VAR i,j,Z,L,f,anz,n,korr,halb:integer; m:tstein;
385:  0 BEGIN
386:  1   om:=0; gm:=0; halb:=0; korr:=0;
387:  1   (* Korrektur der offenen Muehlen ueber Kreuz *)
388:  1   FOR i:=0 TO 15 DO BEGIN
389:  2     Z:=0;
390:  2     FOR j:=0 TO 2 DO
391:  2       BEGIN
392:  3         m:=Feld.Brett[Muehlen[i,j]];
393:  3         IF m=k THEN Z:=succ(Z) ELSE IF m<>leer THEN Z:=4
394:  3                                ELSE f:=Muehlen[i,j];
395:  3       END;
396:  2     CASE Z OF
397:  3       2 : BEGIN
398:  4             IF Feld.Brett[Muehlen_Nachb[f,0]]=k THEN
```

```
399:  4              IF Feld.Brett[Muehlen_Nachb[f,1]]=k THEN
400:  4               IF Feld.Brett[Muehlen_Nachb[f,2]]=k THEN
401:  4                IF Feld.Brett[Muehlen_Nachb[f,3]]=k THEN
402:  4                  korr:=succ(korr); (* Zahl der ueberkreuz-Muehlen *)
403:  4             CASE Feld.Status[k].Modus OF
404:  5               Setzen,
405:  5               Springen : om:=succ(om);
406:  5               Ziehen   : BEGIN anz:=0;
407:  6                            FOR j:=0 TO 3 DO
408:  6                              BEGIN n:=Nachbar[f,j];
409:  7                              IF n<>30 THEN
410:  7                              IF Feld.Brett[n]=k THEN BEGIN
411:  8                                  IF NOT(n IN Muehlen_menge[i]) THEN
412:  8                                     anz:=succ(anz) END
413:  7                                ELSE IF Feld.Brett[n]<>leer THEN halb:=halb OR 1;
414:  7                                     (* Muehle verstellt? *)
415:  7                              END;
416:  6                            IF anz <> 0 THEN
417:  6                              BEGIN om:=succ(om);
418:  7                                IF odd(halb) THEN halb:=2;
419:  7                              END ELSE halb:=halb AND 2;
420:  6                          END;
421:  5              END; (* case Modus *)
422:  4            END;
423:  3       3 : gm:=succ(gm);
424:  3     END;  (* case Z of *)
425:  2   END;  (* for i *)
426:  1   om := om - korr DIV 2;
427:  1   IF k<>AmZug THEN
428:  1   IF Feld.Status[geg[k]].Modus=Ziehen
429:  1     THEN BEGIN IF halb AND 2<>0 THEN om := pred(om) END
430:  1     ELSE IF om<>0 THEN om:=pred(om);
431:  1 END;
432:  0
433:  0 FUNCTION SteinBewertung(k:tstein):integer;
434:  0 (* Bewertet werden die Bewegungsfreiheiten der steine und
435:  0    ob als naechster Zug eine Muehle geschlossen werden kann *)
436:  0 VAR i,j,N,Wert : integer;
437:  0 BEGIN
438:  1   Wert:=0;
439:  1   FOR i:=0 TO 23 DO
440:  1     IF Feld.Brett[i]=k THEN BEGIN
441:  2       FOR j:=0 TO 3 DO
442:  2         BEGIN
443:  3         N:=Nachbar[i,j];
444:  3         IF N<>30 THEN IF Feld.Brett[N]=leer THEN Wert:=succ(Wert);
445:  3         END;
446:  2     END;
447:  1     SteinBewertung:=Wert
448:  1 END;
449:  0
450:  0 BEGIN (* Bewertung *)
451:  1   offen_zu_Muehlen(Spieler,om[Spieler],gm[Spieler]);
452:  1   offen_zu_Muehlen(Rechner,om[Rechner],gm[Rechner]);
453:  1   spst:=SteinBewertung(Spieler);
454:  1   rest:=SteinBewertung(Rechner);
455:  1   IF om[AmZug]<>0 THEN
456:  1      BEGIN Summe:=5*(2*ord(AmZug)-3);
457:  2            IF om[geg[AmZug]]<>0 THEN om[geg[AmZug]]:=pred(om[geg[AmZug]])
458:  2      END
459:  1   ELSE IF om[geg[AmZug]]<>0 THEN Summe:=-5*(2*ord(AmZug)-3)
460:  1           ELSE Summe:=0;
461:  1   Summe:=Summe-40*om[Spieler]-30*gm[Spieler]-spst
```

```
462:  1                +40*om[Rechner]+30*gm[Rechner]+rest;
463:  1   WITH Feld.Status[Rechner] DO Summe:=Summe+20*(steine+zaehler);
464:  1   WITH Feld.Status[Spieler] DO Summe:=Summe-20*(steine+zaehler);
465:  1   IF Feld.Status[AmZug].Modus <> Setzen THEN
466:  1     BEGIN
467:  2       FOR x:=Spieler TO Rechner DO
468:  2         WITH Feld.Status[x] DO
469:  2           IF (steine=3) AND (om[geg[x]]<>0) THEN
470:  2              Summe:=Summe+80*(3-2*ord(x));
471:  2         IF (spst<=3) AND (Feld.Status[Spieler].Modus=Ziehen) THEN
472:  2            BEGIN Summe:=Summe+4*(4-spst);
473:  3            IF spst=0 THEN IF AmZug=Spieler THEN
474:  3               Summe:=maxval ELSE Summe:=Summe+80;
475:  3               END;
476:  2         IF (rest<=3) AND (Feld.Status[Rechner].Modus=Ziehen) THEN
477:  2            BEGIN Summe:=Summe-4*(4-rest);
478:  3            IF rest=0 THEN IF AmZug=Rechner THEN
479:  3               Summe:=-maxval ELSE Summe:=Summe-80;
480:  3               END;
481:  2         IF Feld.Status[Rechner].steine<3 THEN Summe:=-maxval
482:  2            ELSE IF Feld.Status[Spieler].steine<3 THEN Summe:=maxval;
483:  2     END ELSE
484:  1     IF Feld.Status[AmZug].zaehler < 5 THEN
485:  1        BEGIN
486:  2        IF (spst<=4) THEN Summe:=Summe+7*(5-spst);
487:  2        IF (rest<=4) THEN Summe:=Summe-7*(5-rest);
488:  2        END;
489:  1   Bewertung:=Summe;
490:  1 END;
491:  0
492:  0
493:  0 PROCEDURE MachZugListe(VAR ZugListe : Liste; AmZug : tstein;
494:  0                         vz : integer; VAR Anzahl : integer);
495:  0 VAR HilfsBrett : tbrett;
496:  0     tmp       : ListenInhalt;
497:  0 BEGIN Feld.k:=-1; Feld.Schl_Nr:=-1; Feld.L:=-1;
498:  1       ZugListe:=NIL; Anzahl:=0;
499:  1       REPEAT
500:  2         Zug_ermitteln(AmZug);
501:  2         IF Feld.Anzahl=1 THEN
502:  2         BEGIN Anzahl:=succ(Anzahl);
503:  3           HilfsBrett:=Feld; mache_Zug(AmZug,NOT zeigen);
504:  3           tmp.value:=vz*Bewertung(geg[AmZug]); tmp.Zug:=Feld.Zug;
505:  3           einfuegen(ZugListe,tmp); Feld:=HilfsBrett;
506:  3         END;
507:  2       UNTIL Feld.Anzahl=0;
508:  1 END;
509:  0
510:  0 (*$A-*)
511:  0 FUNCTION Schachtlung (AmZug : tstein; i,min,max,
512:  0                       zuege,A1,A2 : integer):integer;
513:  0 LABEL exit;
514:  0 VAR HilfsBrett : tbrett;
515:  0     ZugListe   : Liste;
516:  0     Wert,Anz   : integer;
517:  0     MinMax     : integer;
518:  0     Heap       : ^integer;
519:  0 BEGIN Mark(Heap);
520:  1   ZugListe:=NIL;
521:  1   IF odd(i) THEN minmax:=min ELSE minmax:=max;
522:  1   IF abs(minmax)>maxint-i THEN
523:  1     BEGIN
524:  2       IF minmax>0 THEN minmax:=maxint-i ELSE minmax:=i-maxint;
```

```
525:  2        IF odd(i) AND (minmax>=max)
526:  2        OR NOT odd(i) AND (minmax<=min)
527:  2          THEN GOTO exit;
528:  2      END;
529:  1    IF zuege<=A1 THEN
530:  1      BEGIN
531:  2        Feld.k:=-1; Feld.Schl_Nr:=-1; Feld.L:=-1;
532:  2        REPEAT
533:  3        Zug_ermitteln(AmZug);
534:  3        IF Feld.Anzahl=1 THEN
535:  3        BEGIN
536:  4          HilfsBrett:=Feld; mache_Zug(AmZug,NOT zeigen);
537:  4          Wert:=Bewertung(geg[AmZug] (* am Zug *)); Feld:=HilfsBrett;
538:  4          IF odd(i) THEN
539:  4          BEGIN
540:  5            IF Wert=maxint THEN Wert:=maxint-succ(i);
541:  5            IF Wert>minmax THEN minmax:=Wert;
542:  5            IF minmax>=max THEN GOTO exit;
543:  5          END
544:  4          ELSE BEGIN
545:  5            IF Wert=-maxint THEN Wert:=succ(i)-maxint;
546:  5            IF Wert<minmax THEN minmax:=Wert;
547:  5            IF minmax<=min THEN GOTO exit;
548:  5          END;
549:  4        END;
550:  3        UNTIL Feld.Anzahl=0;
551:  2      END
552:  1    ELSE
553:  1      BEGIN MachZugListe(ZugListe,AmZug,pred(2*ord(odd(i))),Anz);
554:  2        HilfsBrett:=Feld; IF Anz<>0 THEN zuege:=zuege DIV Anz;
555:  2        WHILE ZugListe<>NIL DO
556:  2        BEGIN
557:  3          Feld.Zug:=ZugListe^.Eintrag.Zug; mache_Zug(AmZug,NOT zeigen);
558:  3          IF odd(i) THEN Wert:=Schachtlung(geg[AmZug],succ(i),
559:  3                                           minmax,max,zuege,A2,Anz)
560:  3                    ELSE Wert:=Schachtlung(geg[AmZug],succ(i),
561:  3                                           min,minmax,zuege,A2,Anz);
562:  3          Feld:=HilfsBrett;
563:  3          ZugListe:=ZugListe^.naechster;
564:  3          IF odd(i) THEN
565:  3          BEGIN
566:  4            IF Wert>minmax THEN minmax:=Wert;
567:  4            IF minmax>=max THEN GOTO exit;
568:  4          END
569:  3          ELSE BEGIN
570:  4            IF Wert<minmax THEN minmax:=Wert;
571:  4            IF minmax<=min THEN GOTO exit;
572:  4          END;
573:  3        END;
574:  2      END;
575:  1    exit: Schachtlung:=minmax;
576:  1          Release(Heap);
577:  1  END;
578:  0  (*$A+*)
579:  0
580:  0  PROCEDURE writemove;
581:  0    VAR Z:integer;
582:  0    BEGIN
583:  1      WITH Feld DO
584:  1        FOR Z:=ord(Status[Rechner].Modus=Setzen) TO 1+ord(Muehle) DO
585:  1          write(posit[Zug[Z]],' ');
586:  1    END;
587:  0
```

```
588:  0 PROCEDURE Schachtlung_Rechner;
589:  0 LABEL exit;
590:  0 CONST zuege : ARRAY [1..9] OF integer=(16,50,160,300,600,
591:  0                                        1000,3200,10000,32000);
592:  0 VAR HilfsBrett : tbrett;
593:  0     ZugListe   : Liste;
594:  0     Wert,
595:  0     Anz,SpSt   : integer;
596:  0     Heap       : ^integer;
597:  0   BEGIN Mark(Heap);
598:  1         SpSt:=zuege[SpielStaerke];
599:  1         MachZugListe(ZugListe,Rechner,1,Anz);
600:  1         Minmax:=1-Maxint;
601:  1         IF ZugListe<>NIL THEN
602:  1           IF Anz>=SpSt THEN
603:  1             BEGIN wert:=ZugListe^.Eintrag.Value;
604:  2                   IF Wert=maxint THEN Wert:=maxint-2;
605:  2                   Minmax:=Wert; BestZug:=ZugListe^.Eintrag.Zug;
606:  2             END
607:  1           ELSE BEGIN HilfsBrett:=Feld; SpSt:=SpSt DIV Anz;
608:  2             REPEAT
609:  3             Feld.Zug:=ZugListe^.Eintrag.Zug;
610:  3             gotoxy(57,16); writemove; ClrEol; gotoxy(19,22);
611:  3             mache_Zug(Rechner,NOT zeigen);
612:  3             Wert:=Schachtlung(Spieler,2,pred(Minmax),Maxint,SpSt,Anz,Anz);
613:  3             IF (Wert>Minmax) OR
614:  3             ((Wert=Minmax) AND (random(2)=0)) THEN
615:  3                BEGIN BestZug:=ZugListe^.Eintrag.Zug;
616:  4                      Minmax:=Wert; Feld:=HilfsBrett;
617:  4                      Feld.Zug:=BestZug; gotoxy(57,17); WriteMove;
618:  4                      Write('= ',Wert); ClrEol; gotoxy(19,22);
619:  4                END;
620:  3             Feld:=HilfsBrett;
621:  3             IF KeyPressed THEN IF Antwort_Ja('Abbrechen') THEN GOTO exit;
622:  3             ZugListe:=Zugliste^.naechster;
623:  3             UNTIL ZugListe=NIL;
624:  2           END;
625:  1 exit: Release(Heap);
626:  1 END;
627:  0
628:  0 PROCEDURE SpielerZug;
629:  0 (* SpielerZug abfragen *)
630:  0 LABEL exit;
631:  0 VAR SpZug : tstr;
632:  0     Y,Z,offs :integer;
633:  0     num:tzug;
634:  0     mu,
635:  0     gueltig,kein_Zug_moeglich:boolean;
636:  0
637:  0   PROCEDURE invertiere;
638:  0     VAR i:0..23;
639:  0         h:tStat;
640:  0     BEGIN
641:  1       h[Spieler]:=Feld.Status[Rechner];
642:  1       h[Rechner]:=Feld.Status[Spieler];
643:  1       Feld.Status:=h;
644:  1       FOR i:=0 TO 23 DO
645:  1         IF Feld.Brett[i]=Rechner THEN Feld.Brett[i]:=Spieler
646:  1         ELSE IF Feld.Brett[i]=Spieler THEN Feld.Brett[i]:=Rechner
647:  1     END;
648:  0
649:  0 BEGIN gotoxy(27,24); write('H(alt) S(pielstaerke) V(orschlag)');
650:  1   WITH Feld DO BEGIN
```

```
651:  2     zeige_Status(Spieler);
652:  2     REPEAT
653:  3       REPEAT
654:  4         gotoxy(35,22); write('Zug : '); ClrEol;
655:  4         read(SpZug);
656:  4         FOR Z:=1 TO length(SpZug) DO SpZug[Z]:=upcase(SpZug[Z]);
657:  4         WHILE pos(' ',SpZug)>0 DO  delete(SpZug,pos(' ',SpZug),1);
658:  4         IF SpZug='S' THEN hole_Spielstaerke;
659:  4         IF SpZug='V' THEN
660:  4            BEGIN invertiere; gotoxy(57,17); ClrEol; gotoxy(57,16); ClrEol;
661:  5                  Schachtlung_Rechner; Feld.Zug:=BestZug;
662:  5                  gotoxy(57,16); write('Vorschlag: '); Writemove; ClrEol;
663:  5                  invertiere;
664:  5               END;
665:  4         IF SpZug='H' THEN BEGIN
666:  5           Spiel_ist_aus:=Antwort_Ja('Spiel beenden');
667:  5           IF Spiel_ist_aus THEN GOTO exit;
668:  5         END;
669:  4       UNTIL (SpZug<>'S') AND (SpZug<>'H') AND (SpZug<>'V');
670:  3       Schl_Nr:=-1; k:=-1; L:=-1; Zug_ermitteln(Spieler);
671:  3       kein_Zug_moeglich:=Feld.Anzahl=0;
672:  3       offs:=ord(Status[Spieler].Modus=Setzen); mu:=false;
673:  3       fillchar(num,SizeOf(num),-1);
674:  3       FOR Z:=0 TO 2-offs DO
675:  3         FOR Y:=0 TO 23 DO
676:  3           IF posit[Y]=copy(SpZug,2*Z+1,2) THEN num[Z+offs]:=Y;
677:  3       REPEAT
678:  4         gueltig:=true; mu:=mu OR Muehle;
679:  4         FOR Z:=offs TO 1+ord(Muehle) DO
680:  4           gueltig:=gueltig AND (num[Z]=Zug[Z]);
681:  4         IF NOT muehle THEN gueltig:=gueltig AND (num[2]=-1);
682:  4         IF NOT gueltig THEN Zug_ermitteln(Spieler);
683:  4       UNTIL gueltig OR (Anzahl=0);
684:  3       IF NOT gueltig AND NOT kein_Zug_moeglich THEN BEGIN
685:  4         gotoxy(27,23); LowVideo;
686:  4         write('Eingabe:  ');
687:  4         IF offs=1 THEN write('<SETZ>')
688:  4           ELSE write('<VON> <NACH>');
689:  4         IF mu THEN IF offs=1 THEN write('  oder  <SETZ> <NIMM>')
690:  4                              ELSE write('  oder  <VON> <NACH> <NIMM>');
691:  4         REPEAT UNTIL keypressed;
692:  4         gotoxy(27,23); HighVideo; ClrEol;
693:  4       END;
694:  3     UNTIL gueltig OR kein_Zug_moeglich;
695:  2     IF kein_Zug_moeglich THEN verloren(Spieler)
696:  2     ELSE mache_Zug(Spieler,zeigen);
697:  2   END;
698:  1   exit: gotoxy(1,24); ClrEol;
699:  1 END;
700:  0
701:  0
702:  0 PROCEDURE RechnerZug;
703:  0 (* Naechsten RechnerZug ermitteln *)
704:  0
705:  0 BEGIN (* RechnerZug *)
706:  1   gotoxy(19,22); write(''':8); gotoxy(57,16); ClrEol; gotoxy(57,17); ClrEol;
707:  1   zeige_Status(Rechner); gotoxy(19,22);
708:  1   Schachtlung_Rechner;
709:  1   gotoxy(57,17); ClrEol; gotoxy(57,16);
710:  1   IF abs(minmax)<32000 THEN write('Bewertung : ',minmax)
711:  1     ELSE write(steine[tstein(1+ord(minmax<0))],
712:  1                ' verliert in ',(maxint-abs(minmax)-1) DIV 2);
713:  1   ClrEol; gotoxy(19,22); write(#7);
```

```
714:  1   IF minmax=-maxint+1 THEN verloren(Rechner)
715:  1   ELSE BEGIN
716:  2     Feld.Zug:=BestZug;
717:  2     WriteMove;
718:  2     mache_Zug(Rechner,zeigen);
719:  2     IF minmax=maxint-2 THEN
720:  2       BEGIN zeige_Status(Spieler); verloren(Spieler) END;
721:  2   END;
722:  1 END;
723:  0
724:  0 PROCEDURE Aufraeumen;
725:  0 VAR z:integer;
726:  0   BEGIN
727:  1     WITH Feld DO
728:  1     FOR z:=0 TO 23
729:  1       DO IF Brett[z]<>leer THEN
730:  1         WITH Status[Brett[z]] DO
731:  1         BEGIN
732:  2           steine:=steine-1;
733:  2           Display_Move(3,z,leer);
734:  2         END
735:  1   END;
736:  0
737:  0 PROCEDURE Aufstellen;
738:  0 VAR c:char;  i,j,X,Y:integer;
739:  0   BEGIN
740:  1     write('Grundaufstellung (J/N)');
741:  1     REPEAT
742:  2       read(kbd,c);
743:  2       c:=upcase(c);
744:  2     UNTIL c IN [#13,'J','N'];
745:  1     writeln(c);
746:  1     IF c<>'J' THEN exit;
747:  1     FOR i:=0 TO 1 DO
748:  1     FOR j:=0 TO 8 DO
749:  1       BEGIN
750:  2         home; MoveTo(800,0);
751:  2         Hinunter;
752:  2           write(j+1,'. ');
753:  2           IF i=0 THEN write ('schwarzen')
754:  2                  ELSE write ('weissen');
755:  2           write (' Stein!'); ClrEol;
756:  2           read(kbd,c);
757:  2           IF c=^c THEN exit;
758:  2           write(#13);
759:  2         Zufassen; Hinauf;
760:  2           BEGIN
761:  3             X:=50+400*i;
762:  3             Y:=200+300*j
763:  3           END;
764:  2         MoveTo(X,Y);
765:  2         Hinunter; Loslassen; Hinauf;
766:  2       END;
767:  1
768:  1   END;
769:  0
770:  0 BEGIN  (* Muehle *)
771:  1   initvars; randomize;
772:  1   Aufstellen;
773:  1   home;
774:  1   REPEAT
775:  2     Initial;
776:  2     IF Antwort_Ja('Wollen Sie anfangen') THEN
```

```
 777:  2       BEGIN
 778:  3       SpielerZug;
 779:  3       IF NOT Spiel_ist_aus THEN BEGIN
 780:  4         REPEAT Feld.Zug[1]:=random(24);
 781:  5         UNTIL  Feld.brett[Feld.Zug[1]]=leer;
 782:  4         gotoxy(19,22); write(#7,posit[Feld.Zug[1]]);
 783:  4         mache_Zug(Rechner,zeigen);
 784:  4       END;
 785:  3       END
 786:  2     ELSE
 787:  2       BEGIN
 788:  3       Feld.Zug[1]:=random(24);
 789:  3       gotoxy(19,22); write(#7,posit[Feld.Zug[1]]);
 790:  3       mache_Zug(Rechner,zeigen);
 791:  3       END;
 792:  2     WHILE NOT Spiel_ist_aus DO BEGIN
 793:  3       SpielerZug;
 794:  3       IF NOT Spiel_ist_aus THEN RechnerZug;
 795:  3     END;
 796:  2     Aufraeumen
 797:  2   UNTIL NOT Antwort_Ja('Noch ein Spiel');
 798:  1   gotoxy(1,24);
 799:  1   home
 800:  1 END.
```

Listing of **ROBOTER.INC**

```
   1:  0 (*****************************************************************************)
   2:  0 (* ROBOTER.INC ---- Software-interface fuer Robot-Kid                        *)
   3:  0 (*****************************************************************************)
   4:  0
   5:  0 CONST Zeit   = 1500; (* millisekunden, fuer hand-schliessen *)
   6:  0       Warten =  300; (* millisekunden, bis motoren stehen *)
   7:  0
   8:  0 VAR POSITIONEN : ARRAY [1..4] OF integer;
   9:  0     ZIELE      : ARRAY [1..3] OF integer;
  10:  0     RICHTUNGEN : ARRAY [1..3] OF integer;
  11:  0     LETZTER    : ARRAY [1..3] OF boolean;
  12:  0
  13:  0
  14:  0 (* HOME-Prozedur, wie sie im Fischertechnik-
  15:  0      ROBOT-Programm zu finden ist:
  16:  0 3000 REM Heimposition anfahren
  17:  0 3010 LET H1=1:H2=1:H3=1:H4=1
  18:  0 3020 IF USR(E1)=1 AND H1=1 THEN CALL M1(RECHTS)
  19:  0 3030 IF USR(E3)=1 AND H2=1 THEN CALL M2(RECHTS)
  20:  0 3040 IF USR(E5)=1 AND H3=1 THEN CALL M3(RECHTS)
  21:  0 3050 IF USR(E7)=1 AND H4=1 THEN CALL M4(RECHTS)
  22:  0 3060 IF USR(E1)=0 THEN CALL M1(LINKS):LET H1=-1
  23:  0 3070 IF USR(E3)=0 THEN CALL M2(LINKS):LET H2=-1
  24:  0 3080 IF USR(E5)=0 THEN CALL M3(LINKS):LET H3=-1
  25:  0 3090 IF USR(E7)=0 THEN CALL M4(AUS):LET H4=0
  26:  0 3100 IF USR(E1)=1 AND H1=-1 THEN CALL M1(AUS):H1=0
  27:  0 3110 IF USR(E3)=1 AND H2=-1 THEN CALL M2(AUS):H2=0
  28:  0 3120 IF USR(E5)=1 AND H3=-1 THEN CALL M3(AUS):H3=0
  29:  0 3130 IF H1<>0 OR H2<>0 OR H3<>0 OR H4<>0 THEN GOTO 3020
  30:  0 3140 CALL INIT
  31:  0 3150 RETURN
  32:  0 *)
  33:  0
  34:  0 (* hier das Pascal-Aequivalent: *)
  35:  0
```

```
36:  0 PROCEDURE HOME;
37:  0   VAR h1,h2,h3,h4:integer;
38:  0       i : integer;
39:  0
40:  0   BEGIN
41:  1     init;
42:  1     FOR i:=1 TO 3 DO
43:  1       BEGIN
44:  2         POSITIONEN[i]:=0;
45:  2         RICHTUNGEN[i]:=0;
46:  2         ZIELE[i]:=0
47:  2       END;
48:  1     POSITIONEN[4]:=1;
49:  1     h1:=1; h2:=1; h3:=1; h4:=1;
50:  1     REPEAT
51:  2       IF lese(E1) AND (H1=1) THEN schalte(M1,RECHTS);
52:  2       IF lese(E3) AND (H2=1) THEN schalte(M2,RECHTS);
53:  2       IF lese(E5) AND (H3=1) THEN schalte(M3,RECHTS);
54:  2       IF lese(E7) AND (H4=1) THEN schalte(M4,RECHTS);
55:  2       IF NOT lese(E1) THEN BEGIN schalte(M1,LINKS); H1:=-1 END;
56:  2       IF NOT lese(E3) THEN BEGIN schalte(M2,LINKS); H2:=-1 END;
57:  2       IF NOT lese(E5) THEN BEGIN schalte(M3,LINKS); H3:=-1 END;
58:  2       IF NOT lese(E7) THEN BEGIN schalte(M4,AUS); H4:=0    END;
59:  2       IF lese(E1) AND (H1=-1) THEN BEGIN schalte(M1,AUS); H1:=0  END;
60:  2       IF lese(E3) AND (H2=-1) THEN BEGIN schalte(M2,AUS); H2:=0  END;
61:  2       IF lese(E5) AND (H3=-1) THEN BEGIN schalte(M3,AUS); H3:=0  END;
62:  2     UNTIL (h1=0) AND (h2=0) AND (h3=0) AND (h4=0);
63:  1     INIT;
64:  1     delay(Warten);
65:  1     FOR i:=1 TO 3 DO LETZTER[i]:=lese(SCHALTER[2*i])
66:  1   END;
67:  0
68:  0 PROCEDURE Hand_Zu;
69:  0 VAR i : integer;
70:  0   BEGIN
71:  1     IF POSITIONEN[4]<>0 THEN
72:  1       FOR i:=1 TO zeit DIV 10 DO
73:  1         BEGIN
74:  2           schalte(M4,LINKS);
75:  2           delay(10);
76:  2         END;
77:  1     schalte(M4,AUS);
78:  1     POSITIONEN[4]:=0
79:  1   END;
80:  0
81:  0 PROCEDURE Hand_Auf;
82:  0   BEGIN
83:  1     IF POSITIONEN[4]=0 THEN
84:  1       BEGIN
85:  2         schalte(M4,RECHTS);
86:  2         WHILE lese(E7) DO
87:  2       END;
88:  1     schalte(M4,AUS);
89:  1     POSITIONEN[4]:=1
90:  1   END;
91:  0
92:  0 PROCEDURE Posit(motor,Wert:integer); (* ziel merken *)
93:  0   BEGIN
94:  1     IF Wert>=0 THEN
95:  1     IF POSITIONEN[motor]<>Wert
96:  1     THEN BEGIN
97:  2            IF POSITIONEN[motor]>=Wert
98:  2              THEN RICHTUNGEN[motor]:=-1
```

```
 99:  2            ELSE RICHTUNGEN[motor]:=+1;
100:  2          ZIELE[motor]:=Wert
101:  2        END
102:  1    ELSE RICHTUNGEN[motor]:=0
103:  1  END;
104:  0
105:  0 PROCEDURE Starte; (* motoren simultan steuern *)
106:  0 VAR i,W,Z : integer;
107:  0     bremse : integer;
108:  0   BEGIN
109:  1     Z:=0;  bremse:=e1+e3+e5;
110:  1     FOR i:=1 TO 3 DO
111:  1       IF RICHTUNGEN[i]<>0
112:  1         THEN
113:  1           BEGIN
114:  2             Z:=succ(Z);
115:  2             IF RICHTUNGEN[i]>0 THEN schalte(MOTOREN[i],LINKS)
116:  2                                ELSE schalte(MOTOREN[i],RECHTS)
117:  2           END
118:  1         ELSE schalte(MOTOREN[i],AUS);
119:  1     WHILE Z<>0 DO
120:  1       BEGIN
121:  2       W:=get_word; (*schalterchen*)
122:  2       IF W AND Bremse<>Bremse
123:  2         THEN BEGIN HOME; Exit END;
124:  2       FOR i:=1 TO 3 DO
125:  2         IF RICHTUNGEN[i]<>0 THEN
126:  2            IF LETZTER[i]<>(W AND SCHALTER[2*i]<>0)
127:  2              THEN BEGIN
128:  3                     POSITIONEN[i]:=POSITIONEN[i]+RICHTUNGEN[i];
129:  3                     LETZTER[i]:=NOT LETZTER[i];
130:  3                     IF POSITIONEN[i]=ZIELE[i]
131:  3                        THEN BEGIN
132:  4                               schalte(MOTOREN[i],AUS);
133:  4                               Z:=pred(Z)
134:  4                             END
135:  3                   END
136:  2       END;
137:  1     FOR Z:=1 TO Warten DO
138:  1       BEGIN
139:  2         W:=get_word;
140:  2         FOR i:=1 TO 3 DO
141:  2           IF RICHTUNGEN[i]<>0 THEN
142:  2             IF LETZTER[i]<>(W AND SCHALTER[2*i]<>0)
143:  2               THEN BEGIN
144:  3                      POSITIONEN[i]:=POSITIONEN[i]+RICHTUNGEN[i];
145:  3                      LETZTER[i]:=NOT LETZTER[i];
146:  3                    END
147:  2       END;
148:  1     FOR i:=1 TO 3 DO RICHTUNGEN[i]:=0
149:  1   END;
150:  0
151:  0 (* ende ROBOTER.INC *)
```

So koppeln Sie Interfaces

Der nachfolgend beschriebene Umbau zeigt, wie Sie mit einem Computer zwei Fischertechnik-Interfaces steuern. Wenn Sie die Methode weiter verfolgen, können auch noch mehrere Interfaces gesteuert werden. Die Interfaces werden in einer Reihe verkettet, d.h. das erste Interface ist direkt an den Computer angeschlossen, das zweite Interface ist an das erste angeschlossen und ggf. ein drittes Interface an das zweite und so fort. Das erste Interface muß das zu Ihrem Computer passende Interface sein, das zweite (und weitere) Interface muß ein Fischertechnik-Interface für Apple-Computer (Art. Nr. 30563) sein. Der Grund dafür: jedes Fischertechnik-Interface bildet auf seiner Platine einen (eingeschränkten) Apple-Port nach.

Wir weisen der Vollständigkeit halber darauf hin, daß bei den nachfolgend beschriebenen Eingriffen mit großer Sorgfalt vorgegangen werden muß, um nicht die Leiterplatte des Interfaces zu beschädigen. Insbesondere ist für die Lötarbeit ein Elektronik-Lötkolben, am besten mit Temperaturregelung, und einwandfreies Feinlot zu benutzen.

Der Umbau ist bei dem ersten Interface (genauer gesagt: bei allen bis auf das letzte Interface) notwendig. Öffnen Sie das Gehäuse. Unterhalb der Position, wo das Anschlußkabel zum Computer befestigt ist, finden Sie einen bislang unbenutzten Platz für einen 16-poligen IC-Anschluß. Dieser Anschluß ist mit "DCH" bezeichnet. Setzen Sie an dieser Stelle einen 16-poligen IC-Sockel ein und verlöten Sie ihn auf der Platinenunterseite.

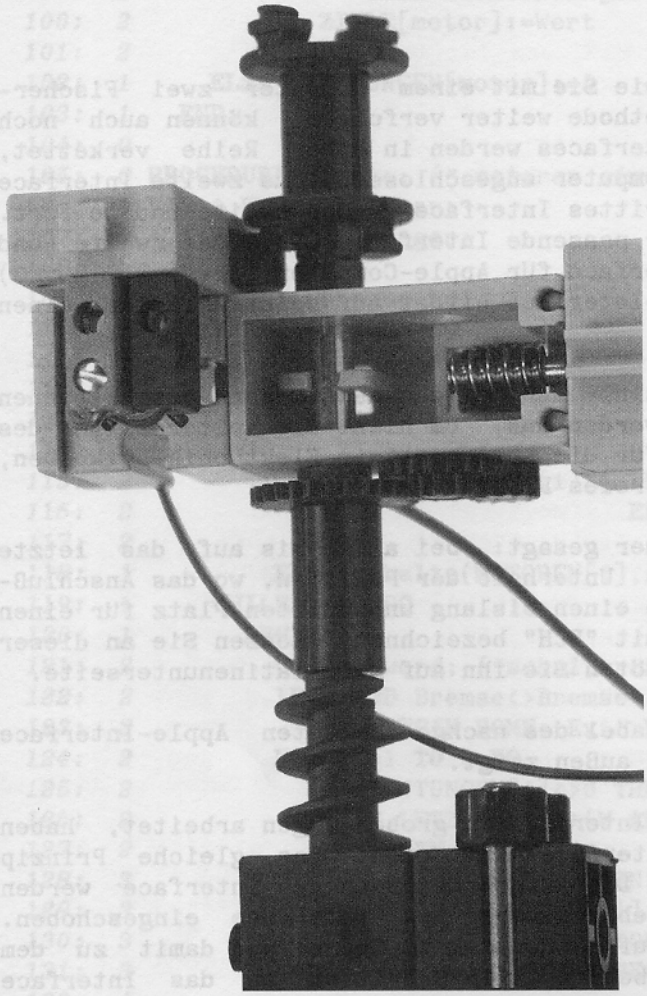
Stecken Sie nun in den IC-Sockel das Anschlußkabel des nachgeschalteten Apple-Interface ein, so daß das Kabel von der Platine weg nach außen zeigt.

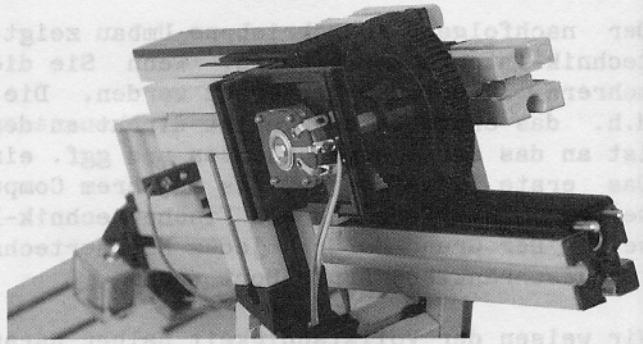
Nun zu der Software. Wie das Fischertechnik-Interface in groben Zügen arbeitet, haben Sie sicher der Anleitung Ihres ersten IBM-Interface entnommen. Das gleiche Prinzip trifft auch für alle anderen Interfaces zu. Die Ausgabedaten an das Interface werden normalerweise in acht Schritten in ein Schieberegister des Interface eingeschoben. Dieses Schieberegister besitzt einen "Überlauf", der zum IC-Sockel und damit zu dem nächsten Interface führt. Wenn nun 16 Ausgabebits in 16 Schritten an das Interface ausgegeben werden, so wandern die ersten acht Bits durch das Schieberegister des ersten Interface hindurch und landen in dem zweiten Interface. Die nachfolgenden acht Bits verbleiben dann in dem ersten Interface.

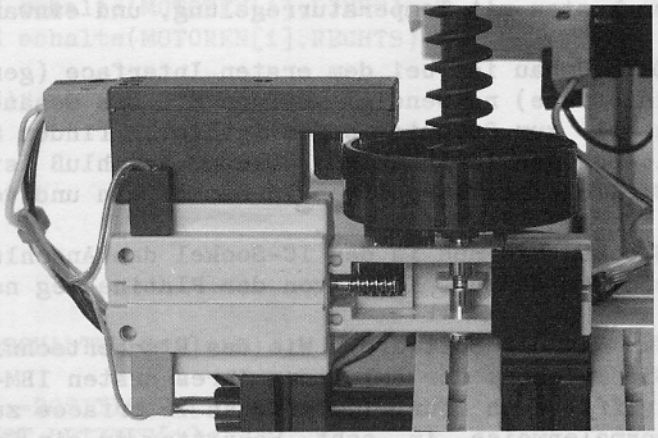
Genau umgekehrt arbeitet wiederum die Eingabe: innerhalb von acht Taktschritten erscheinen zunächst die Daten des ersten Interface. Gleichzeitig rutschen aber auch die Daten des zweiten Interface in das Schieberegister des ersten Interface nach. Mit weiteren acht Taktschritten werden dann auch diese an den Computer übergeben.

Damit dieser "verlängerte" Datentransfer auch einwandfrei abläuft, sind die Taktleitung "CLOCK" und die Übernahmesignale "LOAD OUT" und "LOAD IN" gleichermaßen an das erste und (über den IC-Sockel) an das zweite Interface gelegt.

Eine Einschränkung betrifft den Betrieb der Analogeingänge. Da diese nicht in dem Schieberegisterverfahren wie die digitalen Ein- und Ausgänge arbeiten, lassen sie sich nicht "verlängern". Nur das erste Interface kann zur Analogwerteingabe herangezogen werden.







Positionierung

Zur Positionierung von Motoren gibt es beim Fischertechnik-Computing-System drei alternative Verfahren. Bei der Serie Experimental erfolgt die Rückmeldung über Schalter. Mit dieser Methode ist eine Auflösung von zwei Impulsen je Umdehung der Meßachse zu erreichen.

Bei der Serie Computing erfolgt die Positionierung über Potentiometer. Dabei kann maximal eine Umdrehung um 270 Grad aufgelöst werden, die bei richtiger Einstellung der Software eine Auflösung auf ein Grad genau bei theoretischen Meßwerten zwischen 1 und 255 ergibt.

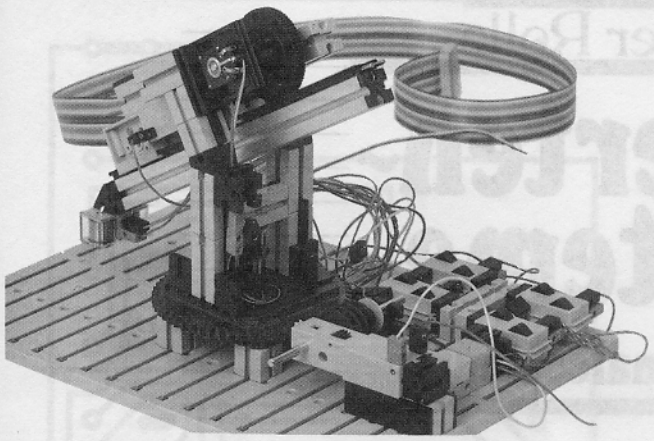
Die komfortabelste Methode bieten die Inkrementalgeber, wie sie beim Lernroboter eingesetzt werden. Sie haben eine Auflösung von 32 Impulsen je Umdrehung der Meßachse.

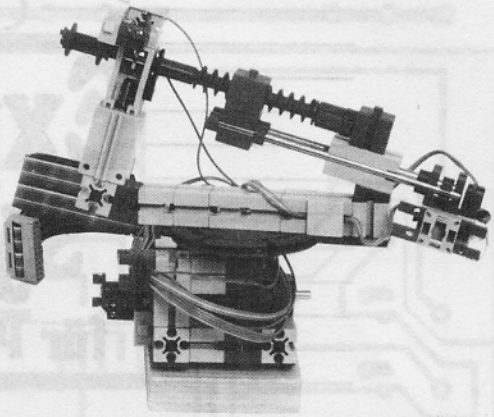
Bei Verwendung von Schrittmotoren, wie sie beim Plotter eingesetzt werden, kann der Rechner mitzählen, welche Position erreicht wurde.

Der Buggy

Der Buggy ist das anspruchsvollste Modell in der Serie Experimental von Fischer-Technik. Mit diesem Modell lassen sich nicht nur grundlegende Steuertechniken erarbeiten, sondern auch klassische Experimente nachvollziehen. Der Buggy kann seine Umgebung erkunden, Lichtsensoren und Berührungstaster liefern ihm die notwendige Informationen.

Eines der anspruchsvollsten Modelle ist der Lernroboter. Er erschließt die Welt der Roboter. Allerdings kann er sicher nicht - wie unlängst bei einem Wettbewerb - Tischtennis spielen. Wer zwei Roboter besitzt, kann jedoch Staffellauf programmieren oder mit modifizierten Greifern die Bewegung der menschlichen Hände (als Modell): zwei Hände halten einen Gegenstand nachempfingen.

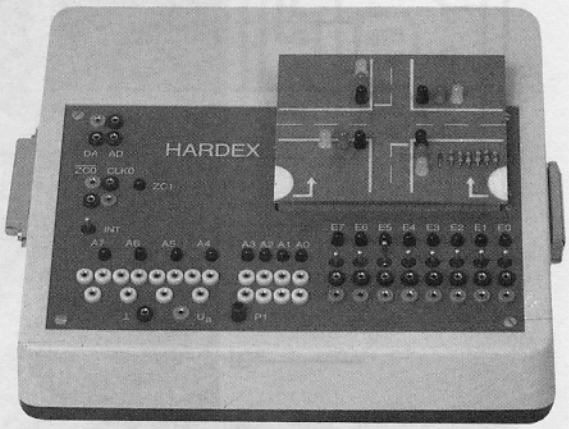


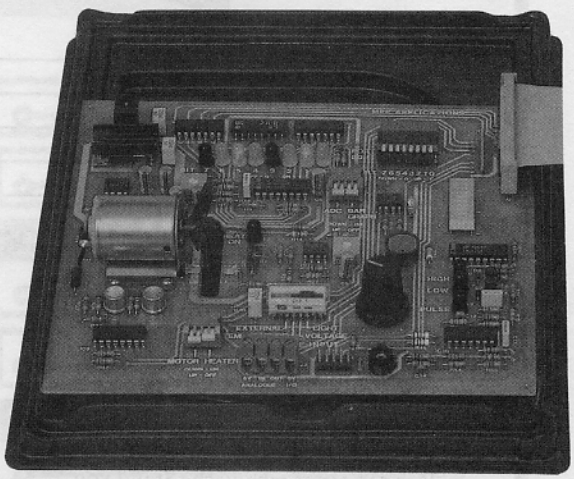


Steuerung mit Assembler

Von Bardehle-Datentechnik wird ein Experimentiersystem mit didaktischen Unterlagen für den Einstieg in die Regeltechnik angeboten. Das Spektrum reicht vom Micro-Professor, einem Z80-System, bis zu einem 68 000-Einplatinensystem. Zu den Beispielen gehören einfache Aufgaben, wie die Steuerung einer Ampelanlage, als auch komplizierte Anwendungen, wie der Betrieb eines Schrittmotors.







Bezugsquellennachweis:

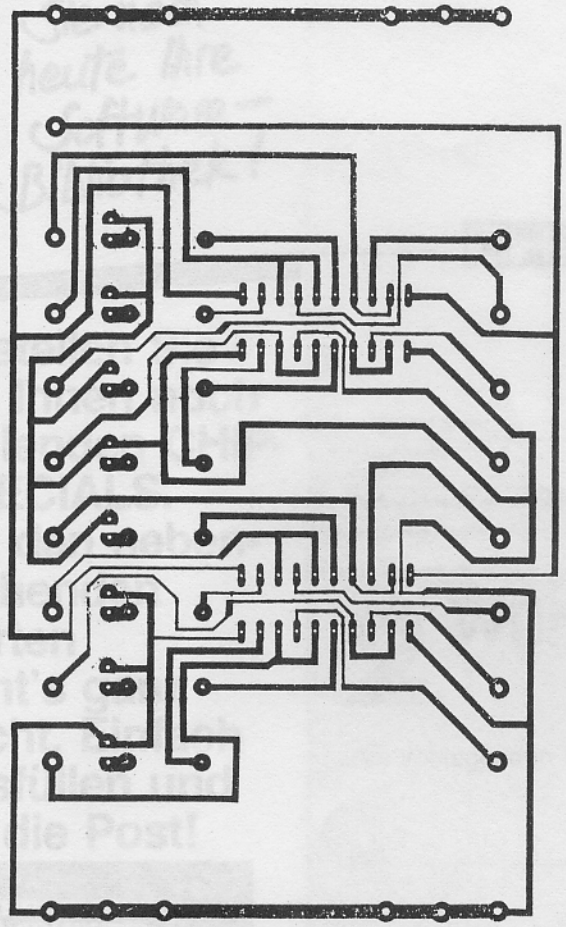
Fischer-Technik Baukästen und Ersatzteile gibt es im qualifizierten Spielwarenhandel. Ersatzteile zudem beim Service-Center in Tumlingen.

BERT ist ein Einplatinen-Steuercomputer im Vertrieb von vgs, Verlagsgesellschaft Köln. Der Rechner ist Teil eines umfangreichen Programms zur Aus- und Weiterbildung in Elektronik und Microelektronik.

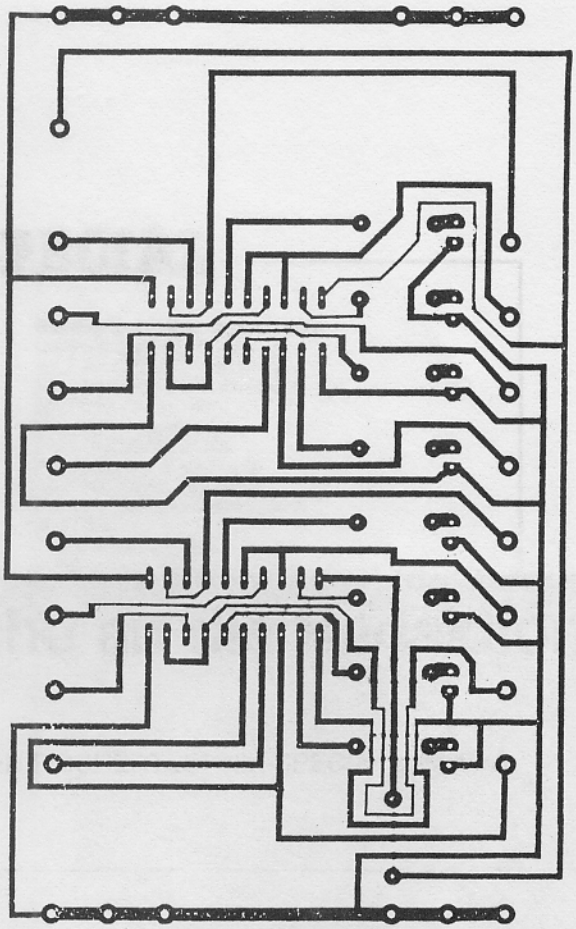
Der Microprofessor und andere Einplatinensysteme werden zusammen mit Schnittstellenkarten und didaktischen Materialien von Bardehle Datentechnik, Liboriusstraße 4, 4796 Salzkotten vertrieben.

Das Interface für den Sharp PC 1500 incl. Software wird von H. Luellmann, Lindenstraße 21, 2875 Ganderkesee, angeboten.

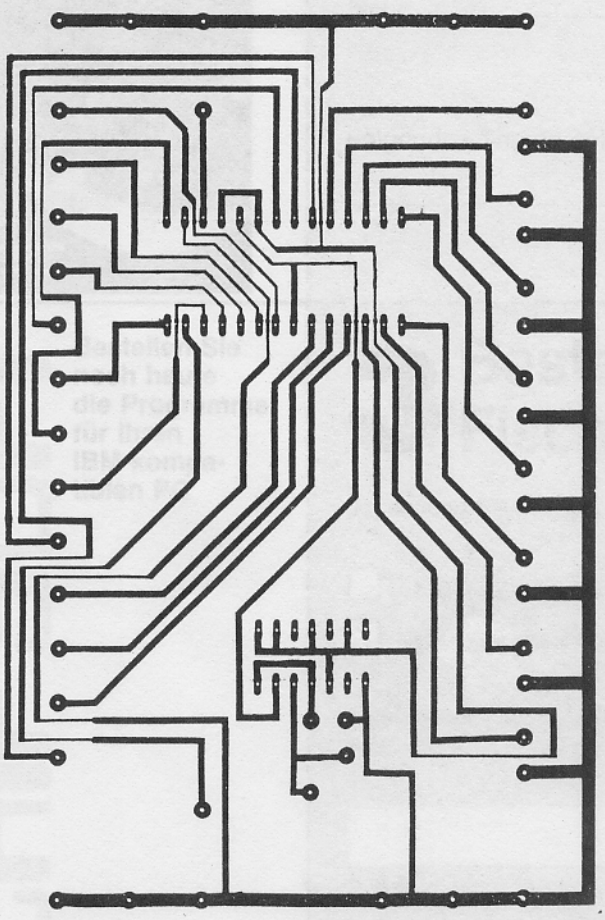
Eingabe



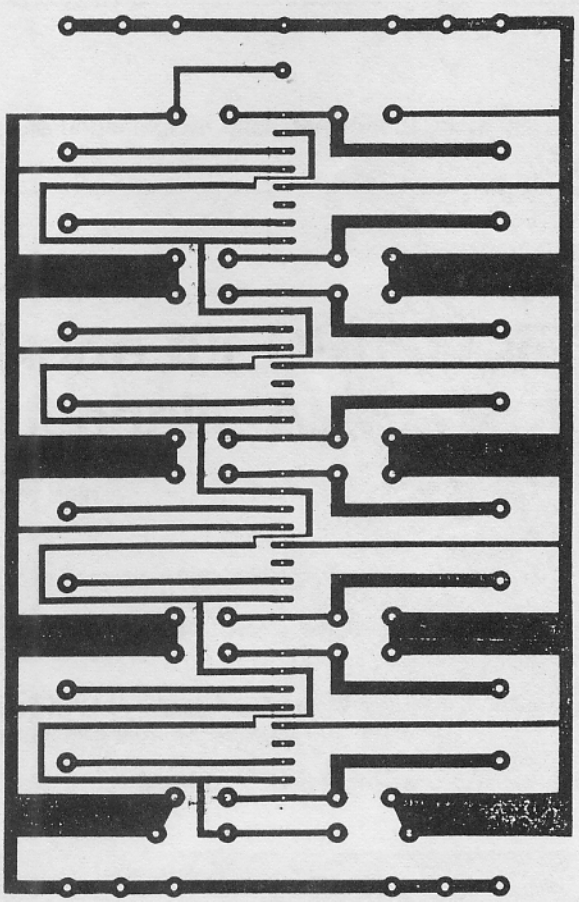
Ausgabe



AD-Wandler



Motor



CHIP SPECIAL

1. Auflage 1987

Best.-Nr. 0952

Redaktionsdirektor: Richard Kerler

Chefredakteur: Armin Schwarz
(verantwortlich für den Inhalt)

Stellvertretender Chefredakteur: Ulrich Kern

Redaktionsassistenz, Leserservice: Petra Eibicht

Layout: Karlheinz Dereser

Autoren dieser Ausgabe: Gerhard Bader, Hermann Dölker, Günter Pomaska, Bernd Edlinger

Gestaltung: Hans Kuh

Titelfoto: studio eins

Redaktion: Vogel-Verlag KG Würzburg,
Redaktion CHIP-Special, Schillerstr. 23 a,
D-8000 München 2, Telefon (089) 51 49 30,
Telekopierer 53 50 00, Telex 5 216 449

Verlag: Vogel-Verlag KG, Postfach 67 40,
D-8700 Würzburg 1, Tel. (0931) 41 02-1,
Telex 6 88 83, Telefax (0931) 41 02-5 29.
Telegramme: CHIP-Würzburg

Verlagsdirektor: Dr. Andreas Kaiser

Anzeigenleiter: Friedrich Mangold, Würzburg
(verantwortlich für Anzeigen)

Anzeigenservice: CHIP, Postfach 67 40,
8700 Würzburg 1, Tel. (0931) 41 02-1,
Telex 6 88 83, Michael Belgrad,
Durchwahl 41 02-4 33.

Vertriebsleitung: Axel Herbschleb, Würzburg

Vertrieb Handelsauflage: Vereinigte Motor-Verlage GmbH & Co. KG, Leuschnerstr. 1,
D-7000 Stuttgart 1, Tel. (07 11) 20 43-1

Bezugsmöglichkeiten: Bestellungen nehmen der Verlag und alle Buchhandlungen im In- und Ausland entgegen. Sollte die Zeitschrift aus Gründen, die nicht vom Verlag zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.

Bankverbindungen Vogel-Verlag:
Dresdner Bank AG, Würzburg
(BLZ 790 800 52) 3 14 88 90 000,
Bay. Vereinsbank AG, Würzburg
(BLZ 790 200 76) 2 50 61 73,
Kreissparkasse Würzburg
(BLZ 790 501 30) 1 74 00,
Postscheckkonto Nürnberg
(BLZ 760 100 85) 9991-853
Ausland: Postscheckkonto Zürich
80 47 064,
Niederlande 2 662 395
Banque Veuve Morin-Pons, Paris
155 410 314

Gesamtherstellung und Versand: VOGEL-DRUCK WÜRZBURG, Max-Planck-Str. 7/9,
D-8700 Würzburg

Unverlangte Manuskripte werden nur zurückgesandt, wenn Rückporto beigefügt ist. Für die mit Namen oder Signatur des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung.
Die in dieser Zeitschrift veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Nachdruck, Vervielfältigung sowie Speicherung in Datenverarbeitungsanlagen nur mit ausdrücklicher Genehmigung des Verlages.
Jede im Bereich eines gewerblichen Unternehmens hergestellte oder benutzte Kopie dient gewerblichen Zwecken gem. § 54 (2) UrhG und verpflichtet zur Gebührenzahlung an die VG Wort, Abteilung Wissenschaft, Goethestraße 49, 8000 München 2, von der die Zahlungsmodalitäten zu erfragen sind.
Die Redaktion hat die Manuskripte und Programme sorgfältig geprüft. Für Fehler im Text, in Schaltbildern, Aufbauskizzen, Listings usw. sowie deren Folgen kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes, auch werden Warennamen ohne Gewährleistung einer freien Verwendung benutzt.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Bader, Gerhard:
Fischer-Technik und Computer: Programme für Atari ST, Commodore 64/128, Schneider CPC, IBM PC u. Kompatible / [Autor dieser Ausg.: Gerhard Bader].
1. Aufl. – Würzburg: Vogel, 1987.
(Chip special)
ISBN 3-8023-0952-9
NE: HST

In dieser Ausgabe

Autor: Gerhard Bader

Schlaues Gespann	Computer lernen sehen und fühlen Schaltungen zur Aus- und Eingabe
Steuern mit Licht	Wir steuern ein Transportband
Regeln Schritt für Schritt	Grundlagen der digitalen Regelung Dreipunkt-Regelung
Wenn sich die Zustände ändern	Nachführ-Regelung Niveauregelung bei bewegten Flüssigkeiten
Bewegungsabläufe kontrollieren	Interface-System Bert Mechanische Antriebe positionieren
Tempo im Griff	Geschwindigkeits-Steuerung Positionierung mit Codierung Die Bert-Kommandos für den Schrittmotor
Grafik mit Pfiff	Gerätesimulation mit dem Plotter HP-GL-Simulation in BASIC
Turbo-Pascal steuert Interface	Roboter-Steuerung
Rekursionen und Bewegungen	Mühle gegen den Computer Interface-Steuerung unter Turbo-Pascal Mühle-Algorithmen in Turbo-Pascal Roboter-Steuerung mit Turbo-Pascal
Umbau und Alternativen	So koppeln Sie Interfaces

4 003634 009520