Experimentsteuerung

Hardware und Software für PC und Peripherie

Klaus Betzler

Universität Osnabrück

Sommersemester 1990

Inhaltsverzeichnis

1 PC-Hardware

Abb. 1 zeigt die sehr vereinfacht dargestellte Blockstruktur eines Mikrorechners. Die Zentraleinheit (CPU) ist durch Adress-, Daten- und Steuerleitungen (Bus) einerseits mit dem Programm- und Datenspeicher verbunden (RAM, ROM), andererseits mit einer ganzen Anzahl von Ein/Ausgabe-Bausteinen. Außer dem DMA-Prozessor, der Speicherzugriffe unter Umgehung der CPU ermöglicht (Transfer von/zu Floppy oder Harddisk), können noch weitere Prozessoren integriert sein (math. Koprozessor, Graphikprozessor).

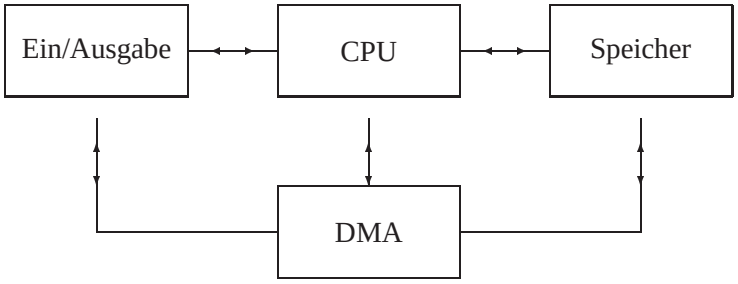


Abbildung 1: Grundstruktur eines Mikrocomputers (CPU: Central Processing Unit, DMA: Direct Memory Access)

Innerhalb dieser Vorlesung werden wir uns im wesentlichen mit den Hard- und Softwareaspekten des Ein/Ausgabebereichs im PC beschäftigen, das Schwergewicht wird in dem Teilbereich liegen, der für Steuerung und Datenerfassung nutzbar ist.

1.1 Die CPU Intel 8086

Der Prozessor i8086 ist der Archetyp einer ganzen Reihe von CPUs (8088, 80186, 80188, 80286, 80386, 80486...) mit unterschiedlicher Leistungsfähigkeit. Typische Kenndaten sind Taktfrequenzen von 4.77 (8088) bis 33 MHz (80386), interne Wortbreiten von 16 (8086) oder 32 Bit (80386), adressierbare Speicherbereiche von 1 MByte (8086), 16 MByte (80286) oder 4 GByte (80386). Alle leistungsfähigeren Prozessoren dieser Reihe können den Grundtyp 8086 vollständig simulieren. Unter dem Betriebssystem PCDOS/MSDOS wird nur dieser 8086-Modus genutzt, daher wird im folgenden nur die 8086-Hardware berücksichtigt.

Es ist üblich, 2 Funktionsblöcke zu unterscheiden, zum einen die Ausführungseinheit (Execution Unit) mit der eigentlichen Recheneinheit (ALU, arithmetic/logic unit), dem Befehlsdekodierer, dem Block der allgemeinen Register und weiteren Registern, zum anderen das Bus-Interface (Bus Interface Unit), das unter anderem die nur für die Adressierung benutzten Register enthält (s. Abb. 2).

1.1.1 Die Register des 8086

Alle Register des 8086 sind an die Wortbreite des Prozessors angepaßt, d. h. alle sind 16 Bit breit. Die allgemeinen Register AX, BX, CX, DX können in 2 mal 8 Bit aufgeteilt werden, ihre höherwertige und niederwertige Hälfte kann jeweils isoliert angesprochen und in Prozessorbefehlen benutzt werden (AH, AL — high, low). Der Registersatz des 8086 ist nicht symmetrisch, auch die allgemeinen Register sind nicht gleichwertig, können nicht für alle Prozessorbefehle gleichartig benutzt werden.

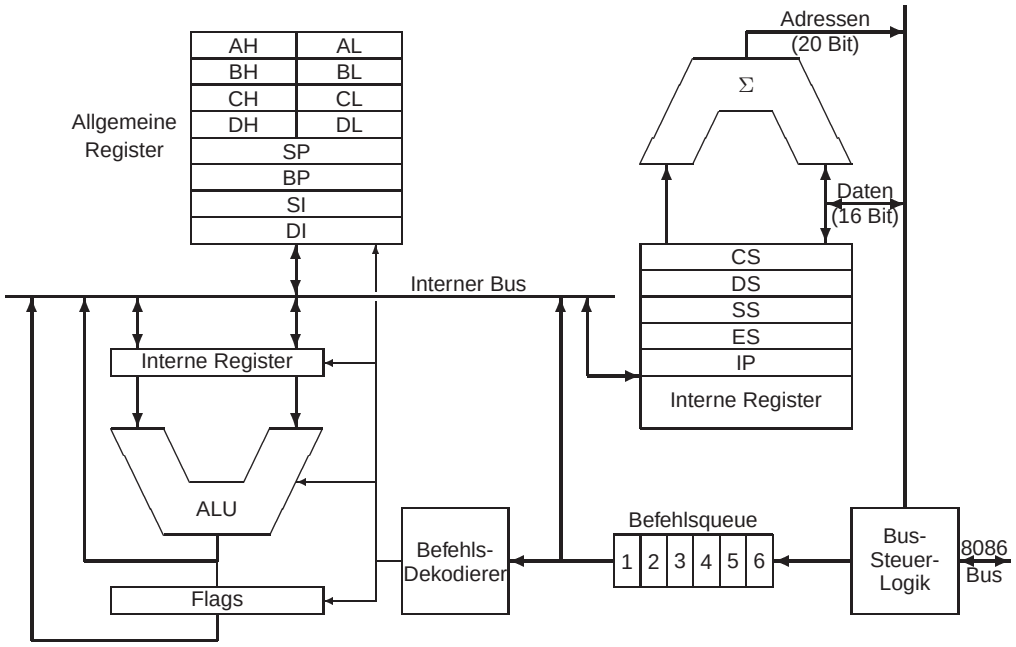


Abbildung 2: Funktionsdiagramm des Mikroprozessors Intel 8086

Verwendung der allgemeinen Register:

AX : Hauptrechenregister (Accumulator), Ein/Ausgabedaten
BX : Basisregister für Array- und Tabellenadressierung
CX : Schleifenzähler (**C**ounter), CL = Zähler bei Verschiebeoperationen
DX : Ein/Ausgabeadressen, Arithmetik
SP : Stack Operationen (**S**tack **P**ointer)
BP : Adressierung von Daten im Stack Segment (**B**ase **P**ointer)
SI : Quellindex (**S**ource) bei Blockoperationen
DI : Zielindex (**D**estination) bei Blockoperationen

Die Segmentregister halten die Segmentadressen für Code-(CS), Daten-(DS), Stack-(SS) und Extrasegment(ES), IP (**I**nstruction **P**ointer) adressiert den aktuellen Programmbefehl. Im Flagregister werden pauschalisierte Rechenergebnisse abgespeichert (Überlauf, Null, Negativ usw.), ein wichtiges Bit dort legt fest, ob ein Interrupt zugelassen wird (IE — **I**nterrupt **E**nable).

1.1.2 Speicheradressierung

Der 8086 besitzt 20 Adressleitungen, der adressierbare Speicherbereich (byteweise Adressierung) beträgt somit 1 MByte. Da kein entsprechendes Register vorhanden ist, wird ‘segmentiert’ adressiert, d. h. die Adresse wird in einem 20-Bit-Addierer durch Addition der mit 16 multiplizierten (um 4 Bit verschobenen) Segmentadresse und der Offsetadresse gebildet. Schreibweise: Seg:Ofs ($0 \leq$ Seg,Ofs $\leq$ 0FFFFH oder Seg = Segmentregister, Ofs = Kombination aus Register(n) und Zahl). Wichtige Segment-Offset-

Kombinationen sind CS:IP (Programmzeiger), DS:[*Variablenname*], DS:BX, DS:[BX][*Zahl*] für Programmdaten, DS:SI, ES:DI zur Adressierung von Datenblöcken, SS:SP (Stack) und SS:BP, SS:[BP][*Zahl*] zur Adressierung von im Stack übergebenen Daten (bei Prozedur- und Funktionsaufrufen). Die Segmentregister brauchen im allgemeinen nicht explizit angegeben werden, da der Prozessor immer eine bestimmte Zuordnung voraussetzt, oft auch nur eine einzige Zuordnung möglich ist.

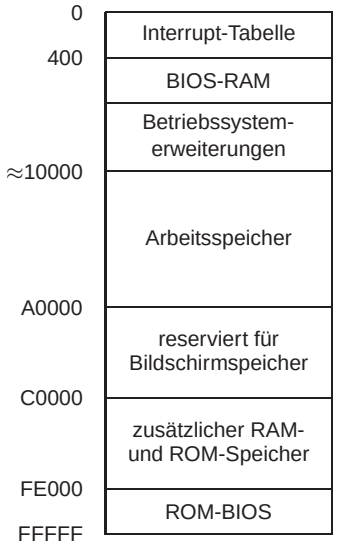
Die Adressierung von Speicherzellen erfolgt grundsätzlich byteweise, jedes Speicherbyte kann isoliert angesprochen werden. Größere Objekte (16-Bit-Worte, 32-Bit-Adressen) werden in aufeinanderfolgenden Bytes gespeichert, die Reihenfolge entspricht der Bytewertigkeit — das niedrigstwertige Byte auf der niedrigsten, das höchstwertige auf der höchsten Adresse des Objekts. Speicherbereiche, die überwiegend 16-Bit-Objekte enthalten (Daten, Stack), sollten aus Geschwindigkeitgründen bei geraden Adressen beginnen (WORD-aligned).

1.1.3 Ein/Ausgabeadressierung

Zur Adressierung von Ein/Ausgabe-Bausteinen werden nur die niederwertigen 16 Adressleitungen benutzt, mithin sind 2^{16} unterschiedliche Ein- und Ausgabebausteine grundsätzlich betreibbar. Die Adressierung erfolgt entweder indirekt über Register DX oder unmittelbar (immediate) durch Angabe der Adresse des E/A-Bausteins als Teil des Ein/Ausgabebefehls (nur für die Adressen 0..255 möglich).

1.2 Prinzipieller Aufbau eines PC

1.2.1 Speicheraufteilung



Nebenstehende Abbildung zeigt die Standardaufteilung des PC-Speichers (640 k). Die Interrupt-Tabelle enthält die Liste aller (Unter-)Programmadressen, die bei Hard- oder Softwareinterrupts aktiviert werden (s. Kap. ‘Interrupts’), das BIOS-RAM speichert wichtige Betriebssystemparameter. Die Belegung des Bildschirmspeichers hängt von der verwendeten Graphikkarte ab (A0000..AFFFF: EGA/VGA, B0000..B7FFF: MDA/Hercules, B8000..BFFFF: CGA). Ein/Ausgabe-Karten mit ‘On-Board’-Speicher (ROM oder RAM) benutzen im allgemeinen den Bereich ab C0000.

Abbildung 3: Speicheraufteilung im PC (Adressen hexadezimal)

1.2.2 Ein/Ausgabe-Hardware

Für den Betrieb des PC sind eine ganze Anzahl von Zusatzbausteinen notwendig, die über Ein/Ausgabeadressen angesprochen werden. Einen groben Überblick gibt Abb. 4. Der systemnahe Teil dieser E/A-Hardware ist mit auf der Hauptplatine integriert, klassisch in isolierten Bausteinen, heute meist als hochintegrierte 'Chip-Sets' mit identischen Funktionen. Für diesen Bereich sind die E/A-Adressen 0 – 0FFH reserviert ('unmittelbare' Adressierung durch das Betriebssystem möglich). Diese Adressen gelangen üblicherweise nicht auf den Peripheriebus, sodaß Adresskonflikte mit diesen 'internen' Adressen weitgehend ausgeschlossen sind. Für Peripheriekarten stehen die Adressen ab 100H zur Verfügung. Meist ist aber folgende Einschränkung zu beachten: Aus Kostengründen werden von den meisten Herstellern nur 8 Bit der Adresse dekodiert (gängige Größe von DIP-Schaltern und Dekoder-ICs), im allgemeinen Bit 2 bis 9 (niederwertigstes ist Bit 0). Daher ist eine Beschränkung auf die Adressen bis 3FFH sinnvoll. Bei der genannten Dekodierung kann jede Ein/Ausgabekarte unter 64 verschiedenen Adressen angesprochen werden (Spiegeladressen), die durch Translation um ganzzahlige Vielfache von 400H aus der Grundadresse gebildet werden.

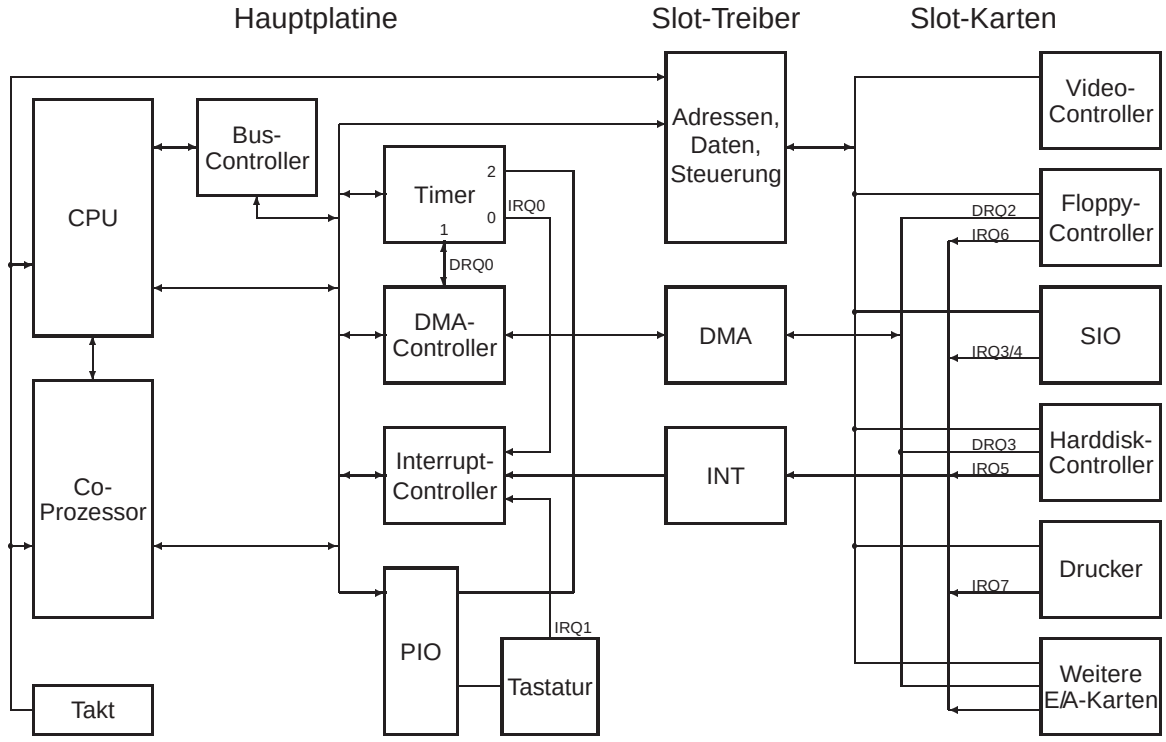


Abbildung 4: Prinzipschema des Ein/Ausgabe-Bereichs des PC (XT) mit den wichtigsten logischen Blöcken. Die Interrupt- und DMA-Kanäle sind jeweils angegeben (IRQ, DRQ), beim AT wird vom Harddisk-Controller der Interrupt 14 benutzt (2. Interrupt-Controller).

Eine Auflistung der gebräuchlichen Ein/Ausgabeadressen für die Bausteine der Hauptplatine und Standardkarten gibt Tabelle 1. Für zusätzliche Peripheriekarten sind trotz der genannten Einschränkung

Portadresse (hex)	Ein/Ausgabe-Baustein
intern (Hauptplatine)	
00 – 1F	DMA-Controller 8237
20 – 3F	Interrupt-Controller 1 : 8259A
40 – 43	Timer 8253 oder Timer 8254
60	Tastatur-Datenport
61	Systemstatus-Bits, Lautsprecher
64	Tastatur-Kommandoport
70 – 71	Echtzeituhr, CMOS-RAM (AT)
80 – 87	DMA-Seiten-Register, RAM-Refresh
A0 – A1	AT: Interrupt-Controller 2 : 8259A
F0 – FF	Coprozessor
extern (Peripheriekarten)	
1F0 – 1FF	Harddisk (AT)
278 – 27F	Drucker 2
2E8 – 2EF	Serielle Schnittstelle 4
2F8 – 2FF	Serielle Schnittstelle 2
320 – 32F	Harddisk (PC)
378 – 37F	Drucker 1
380 – 38F	Serielle Synchronschnittstelle 2
3A0 – 3AF	Serielle Synchronschnittstelle 1
3B0 – 3BB	Monochromadapter (MDA), Hercules
3BC – 3BE	Druckerschnittstelle auf MDA-Karte
3D0 – 3DF	Coloradapter (CGA), EGA, VGA usw.
3E8 – 3EF	Serielle Schnittstelle 3
3F0 – 3F7	Floppy-Controller 765
3F8 – 3FF	Serielle Schnittstelle 1

Tabelle 1: Ein/Ausgabe-Adressen im PC — Hauptplatine und Standardperipherie

genügend Lücken im Adressbereich vorhanden. Von den PC-Herstellern werden die Adressen 300H – 31FH für Zusatzkarten empfohlen und daher auf jeden Fall freigehalten (führt dazu, daß viele Hersteller die Adressvoreinstellung ihrer Karten in diesen Bereich legen — ein Umstand, der bei der Benutzung von mehreren Zusatzkarten im PC zu beachten ist).

Aus dem Bereich der Standardperipherie werden wir uns Interrupt-Controller und Timer, sowie SIO- und Drucker-Schnittstelle etwas genauer ansehen.

2 Software — Betriebssystem und Werkzeuge

2.1 Das Betriebssystem des PC

Das Betriebssystem (MSDOS, PCDOS, verschiedene Versionen — 2.xx, 3.xx, 4.xx) umfaßt die Software, die dem Anwender die Standardperipherie des PC 'verfügbar' macht. Ausführliche Beschreibungen finden sich im zum Rechner gehörenden DOS-Handbuch. In der Reihenfolge ihres Auftretens sind die folgenden Bereiche im Betriebssystem enthalten:

- Das **ROM-BIOS** (**B**asic **I**nput **O**utput **S**ystem) ist in speziellen Speicherbausteinen (**R**ead **O**nly **M**emories) fest im PC eingebaut. Es enthält Programme zum Test der Rechnerhardware, die grundlegenden Ein/Ausgaberoutinen, Ladeprogramme zum Laden des restlichen Betriebssystems von Diskette oder Harddisk. Beim Systemstart und -reset wird ein bestimmtes Programm des ROM-BIOS gestartet, das für ein definiertes 'Hochfahren' des Gesamtsystems sorgt. Einige Systemparameter werden vom ROM-BIOS im Speichersegment 40H, dem **BIOS-RAM**-Bereich vermerkt und sind somit dem Benutzer direkt zugänglich (s. Tabelle 2). Dies sind insbesondere auch die Adressen einiger Ein/Ausgabebausteine des PC. Beim Systemstart wird geprüft, welche dieser Bausteine unter welcher Adresse zu erreichen sind, die Adresse wird dann im BIOS-RAM abgelegt, nichtvorhandene Bausteine werden durch die Adresse 0000 gekennzeichnet.

- Im **Bootsektor** (erster physikalischer Sektor der Diskette oder Festplatte) sind Informationen über das Disk-Format, Ergänzungen des Ladeprogramms und die Dateinamen des Disk-Betriebssystems abgelegt. Diese Information wird vom ROM-BIOS benutzt, um die beiden Systemdateien in den Speicher zu laden.

- **IBMBIO.COM** ist die erste dieser Systemdateien, enthält Erweiterungen zum BIOS.

- **IBMDOS.COM**, die zweite Systemdatei, umfaßt Unterprogramme zur Handhabung des PC-Dateisystems (**D**isk **O**perating **S**ystem).

- **CONFIG.SYS**, eine spezielle Textdatei, definiert DOS-Erweiterungen, die nach den beiden Systemdateien resident (d. h. bis zum nächsten Reset) in den Speicher geladen werden.

- Der Kommando-Prozessor **COMMAND.COM** ist die Benutzerschnittstelle des Betriebssystem, interpretiert die von der Tastatur eingegebenen Kommandos, führt dann interne DOS-Befehle aus oder versucht Dateien zur Ausführung in den Speicher zu laden. COMMAND.COM kann vorübergehend aus dem Speicher entfernt werden (natürlich nur dann, wenn er nicht benötigt wird).

- Durch die Textdatei **AUTOEXEC.BAT** können zusätzliche Betriebssystemerweiterungen definiert werden (z. B. Tastaturanpassungen), die dann ebenfalls speicherresident geladen werden. Außerdem können in AUTOEXEC.BAT Werte für Systemvariable festgelegt werden (Environment), wie PATH, PROMPT usw.

- Im weiteren Sinne zum Betriebssystem gehören die **externen DOS-Befehle**, die durch das Starten der entsprechenden Programme des Betriebssystems ausgeführt werden.

Adr. (hex)	Bytes	Inhalt
40:00 – 40:06	je 2	Adressen der seriellen Ports COM1..COM4
40:08 – 40:0E	je 2	Adressen der Druckerschnittstellen LPT1..LPT4
40:13	2	Speicher-Größe in kByte
40:17	3	Tastatur-Status, 40:17 Shift-Status
40:4E	2	Anfangsadresse des Bildschirmspeichers
40:63	2	Adresse des Bildschirm-Controllers
40:6C	4	System-Uhr (Timer-Interrupts)

Tabelle 2: Einige der im BIOS-RAM-Bereich vermerkten Systemparameter

2.2 Debug

Das Programm Debug ist ein Systemhilfsprogramm, das zum Betriebssystem gehört, also in jedem PC zu Verfügung steht. Wie der Name andeutet, diente es in erster Linie zum ‘Entwanzen’, zum Entfernen von Fehlern aus Programmen. Mit Debug ist es möglich, Speicherinhalte, Programme, Register, einzelne Sektoren auf Diskette oder Harddisk sowohl zu inspizieren und als auch zu ändern, Programme zu testen (Einzelschrittbetrieb, Setzen von Haltepunkten), Ein/Ausgabebausteine direkt anzusprechen, zu ‘Assemblieren’ und zu ‘Disassemblieren’. Durch diese Möglichkeiten ist es ein wichtiges Hilfsmittel für Hardwaretests. Hier eine knappe Befehlsübersicht (Ausführlicheres im DOS-Handbuch):

Debug-Befehl	**Kurzbeschreibung**
A [Adresse]	**Assemble:** Umsetzung von Assemblercode in Maschinencode.
C Bereich Adresse	**Compare:** Vergleich von 2 Speicherbereichen.
D [Bereich]	**Dump:** Anzeige des Inhalts eines Speicherbereichs.
E Adresse [Liste]	**Enter:** Einschreiben von Byte-Werten in den Speicher.
F Bereich Liste	**Fill:** Füllen eines Speicherbereichs mit einem Byte-Muster.
G [=Adresse [Adresse..]]	**Go:** Ausführen eines Programms, Start bei =Adresse, Stop bei Adresse.
H Wert Wert	**Hex:** Summe und Differenz der beiden Hex-Zahlen.
I Wert	**Input:** Eingabe von Port ‘Wert’.
L [Adresse]	**Load:** Laden des aktuellen Programms.
M Bereich Adresse	**Move:** Speicherinhalt verschieben.
N Dateiname	**Name:** Name des aktuellen Programms.
O Wert Byte	**Output:** Ausgabe von ‘Byte’ auf Port ‘Wert’.
P [=Adresse] [Anzahl]	**Proceed:** Ausführen eines Programmteils.
Q	**Quit:** Verlassen von Debug.
R [Registername]	**Register:** Anzeige und Ändern von Registerinhalten.
S Bereich Liste	**Search:** Suche nach einem Byte-Muster.

T [=Adresse] [Anzahl]	**Trace:** Einzelschrittausführung.
U [Bereich]	**Unassemble:** Umsetzung von Maschinencode in Assemblercode.
W [Adresse]	**Write:** Speichern des aktuellen Programms.

Alle Zahleneingaben in Debug sind in hexadezimal zu machen. Als Adresse kann die volle Adresse mit Segment und Offset oder nur der Offsetwert angegeben werden, als Bereich wird Anfangs- und Endadresse oder Anfangsadresse und Länge (`l`*Zahl*) erwartet. Bei vielen Befehlen sind vernünftige Voreinstellungen der Parameter definiert.

Außer dem Betriebssystemprogramm Debug gibt es eine ganze Reihe von ähnlichen Programmen, die im allgemeinen weitergehende Fähigkeiten haben und benutzerfreundlicher in der Bedienung sind (VID für JPI-Modula-2, CodeView für Microsoft-Produkte u. a.). Dennoch ist es sinnvoll, sich mit Debug etwas vertraut zu machen, da es praktisch überall zur Verfügung steht.

2.3 Assembler, Assemblercode

Assemblercode ist die eineindeutige Übersetzung von binärkodierten Prozessorbefehlen in menschenverständliche Kürzel. Ein Assemblerbefehl besteht aus dem Operationscode und den zugehörigen Operanden. Beispielsweise entspricht der hexadezimale Maschinencode **0EEH** (binär 11101110) dem Assemblerbefehl **OUT DX,AL**. Der Operationscode (Op-Code) OUT definiert einen Ausgabebefehl, die Operanden DX und AL sind hier Registernamen. Die Reihenfolge der Operanden (bei 2 Operanden) ist immer *Ziel, Quelle*. Der obige Befehl ordnet für die CPU an, daß der Inhalt von Register AL an den Peripheriebaustein ausgegeben werden soll, dessen Adresse im Register DX gespeichert ist (was bedeutet demnach **IN AX,DX**?). Eine komplette Op-Code-Liste finden sie im Anhang ‘8086/8088 Instruction Set’.

Assembler sind Programme, die in der Lage sind, Listen von Assemblerbefehlen in eine Folge von binären Maschinenbefehlen (meist ein Programm), umzusetzen, Disassembler sind dazu reziprok. Grundlegende Assembler- und Disassemblerfähigkeiten sind in Debug eingebaut, zur Assemblierung längerer Programme ist es sinnvoll, leistungsfähigere Assembler zu benutzen (Microsoft MASM, JPI-Modula-2 Techkit-Assembler).

Das Programmieren in Assemblercode ist meist sehr viel umständlicher und vor allem fehleranfälliger als das Programmieren in einer Hochsprache, daher sollte Assembler(code) nur dort benutzt werden, wo dies notwendig ist, unter anderem

- für kurze Testprozeduren unter der Kontrolle von Debug,
- für einfache Änderungen in einem Programm (Patching),
- wenn hohe Anforderungen an die Geschwindigkeit (?) oder die Länge eines Programms gestellt werden und diese nicht in einer Hochsprache befriedigt werden können,
- um Befehle zu realisieren, die in einer Hochsprache nicht oder nur unbefriedigend vorhanden sind,

- um Software-Verbindungen (Interfaces) zwischen einer Hochsprache und vorhandener Software herzustellen (ROM auf einer Peripheriekarte etc.).

Die aufgeführten Punkte werden nach und nach an einigen Übungsbeispielen veranschaulicht werden.

Hier ein Beispiel für eine kurze Testprozedur, die unter Debug-Kontrolle geschrieben und ausgeführt wurde:
Die Problemstellung bestand darin, ein fehlerhaft arbeitendes 16-Bit-Eingabeport zu analysieren. Dazu sollten Daten in möglichst schneller Abfolge vom Port gelesen werden und in einem Datenfeld abgelegt werden. Das Assemblerprogramm, das dies erledigt, sieht so aus:

```
MOV   AX,CS     ; Mit den ersten drei Befehlen wird DS initialisiert,
ADD   AX,20     ; das Datensegment wird ausreichend weit oberhalb des
MOV   DS,AX     ; Programms angelegt.
MOV   DX,300    ; Adresse des Eingabeports,
MOV   CX,8000   ; Initialisierung des Schleifenzählers,
MOV   SI,0      ; Zeiger auf das Datenfeld,
IN    AX,DX     ; Schleifenanfang, Lesen des Ports,
MOV   [SI],AX   ; Abspeichern im Datenfeld,
INC   SI        ; Feldindex erhöhen,
INC   SI        ; um 2, da Cardinal-Daten,
LOOP  110       ; Sprung zum Schleifenanfang, 110 ist dessen Adresse,
INT   20        ; Programmende.
```

Mit AX, DX, SI sind in obigem Programm Register(inhalte) gemeint, [SI] adressiert einen Speicherplatz, MOV [SI],AX transferiert somit Daten aus dem Register AX zur Speicheradresse DS:SI. Debug kann keine Variablennamen benutzen, alle Konstanten, Sprungadressen u. ä. müssen daher als (Hex-)Zahlen eingegeben werden. Das Programm kann nach dem Kommando `a` im Debug eingegeben werden (Ende durch Leereingabe), mit `g` wird es gestartet. Die Daten werden mit der (Debug-)Befehlsfolge `n`*Datei*, `r cx`, `0`, `r bx`, `1`, `w 200` in *Datei* gespeichert. Zur Analyse setzt man sinnvollerweise eine höhere Programmiersprache ein, in JPI-Modula-2 stünden die Daten etwa nach

CONST imax = 32000; bytes = 2*imax; .....

VAR x : ARRAY [1..imax] OF CARDINAL; infile : FIO.File; .....

infile := FIO.Open('*Datei*');

IF bytes # FIO.RdBin(infile, x, bytes) THEN (* error action *) END; .....

im Datenfeld x zur Verfügung.

2.4 Hochsprache und Assembler

Aus den im vorangehenden Kapitel geschilderten Gründen ist es bisweilen notwendig, einzelne Programmsequenzen in Assembler zu codieren und dann mit einem Hochsprachenprogramm zu verknüpfen.

Prinzipiell geht man so vor, daß man aus den einzelnen Programmteilen linkfähige Module erstellt (.OBJ-Dateien), die dann vom Hochsprachenlinker zu einem lauffähigen Programm zusammengebunden werden können. Um Assembler-Routinen in eine Hochsprachenumgebung einzubinden (und umgekehrt), ist eine genaue Kenntnis der Hochsprachenschnittstellen notwendig. Diese Schnittstellendefinitionen müssen exakt eingehalten werden, um die meist notwendige Kommunikation zwischen Assemblerteil und Hochsprachenteil zu gewährleisten. Am Beispiel von JPI-Modula-2 sollen hier die zu beachtenden Punkte im einzelnen aufgeführt werden:

- Registerhandhabung:
  JPI-Modula-2 geht davon aus, daß vom aufgerufenen Unterprogramm das Register BP immer gerettet wird, die Register AX..DX üblicherweise nicht. Bei zeitkritischen Problemen sollte dies berücksichtigt werden, ansonsten ist es guter Stil, alle benutzten Register zu retten und vor dem Rücksprung wiederherzustellen.

- Übergabe von Parametern an die aufgerufene Prozedur:
  Parameter werden im Stack übergeben, sie werden in der Reihenfolge ihres Auftretens im Prozeduraufruf ‘gepusht’, bei Werte-Parametern wird der Wert gepusht, bei VAR-Parametern die volle Adresse (4 Byte), bei offenen Arrays wird zunächst die aktuelle Länge (2 Byte), dann die volle Adresse im Stack abgelegt (man beachte, daß der Stack ‘nach unten’ wächst, später gepushte Objekte also bei niedrigeren Adressen liegen). Es wird immer eine gerade Anzahl Bytes im Stack abgelegt (auch bei 1-Byte-Objekten).

- Rückgabe von Funktionsparametern:
  Das Ergebnis eines Funktionsaufrufs wird abhängig von der Objektgröße behandelt, 2-Byte-Objekte werden im Register AX, 4-Byte-Objekte im Registerpaar DX:AX, größere Objekte im Stack zurückgegeben.

- Prozedur- und Variablennamen:
  Da viele Linker einen Punkt nicht in Namen zulassen, wird der Punkt in der internen Handhabung durch $ bei Prozedurnamen bzw. @ bei Variablennamen ersetzt. Es muß immer der volle Name angegeben werden, da auf Linkerebene nur dieser bekannt ist, also *Modulname*`$`*Prozedurname* oder *Modulname*`@`*Variablenname*.

- Speichersegmente für Daten und Code:
  Der Programmcode eines Moduls wird im Segment `C_`*Modulname*, die zugehörigen Daten im Segment `D_`*Modulname*, die Initialisierungsroutinen im Segment `INITCODE` abgelegt. Diese Segmentsteuerung sollte auch bei Assemblerprogrammen eingehalten werden, die an Modula-Moduln angelinkt werden.

Die obigen Definitionen gelten dann, wenn die Standardeinstellungen von Modula-2 nicht verändert wurden, bei speziellen Optionen (z. B. NEAR-Prozeduren) sind sie teilweise geändert.

Sind die aufgeführten Punkte beim Schreiben eines Assemblerunterprogramms berücksichtigt, kann dieses beispielsweise mit dem Microsoft Assembler MASM mit der Option `/ml` (Unterscheidung von Groß- und Kleinschreibung) zu einer .OBJ-Datei assembliert werden. Zusätzlich muß ein Definitionsmodul erstellt werden, der keine Besonderheiten gegenüber ‘normalen’ Definitionsmoduln aufweist. Ein Anwendungsbeispiel ist der Modul WordIO (16-Bit-Ein/Ausgabe) im Archiv EXAMPLES.

3 Interrupts

Üblicherweise dienen Interrupts (Unterbrechungen) in einem Rechner dazu, den determinierten Ablauf eines Programms zu unterbrechen, um zwischendurch einzelne Geräte zu bedienen. Solche Programmunterbrechungen — z. B. durch einen eingebauten Zeitgeber (Timer, Real Time Clock) ausgelöst — aktualisieren die Systemzeit im Rechner, schalten in Multiuser-Systemen zwischen den einzelnen Benutzern um, aktivieren in regelmäßigen Zeitabständen Hintergrundprogramme für das Ausdrucken von größeren Dateien. Im PC wird von diesen Interrupts konventioneller Art nach wie vor Gebrauch gemacht (Hardware-Interrupts), jedoch können beim 8086 Interrupts auch durch Programmbefehle angeordnet werden. Diese Software-Interrupts sind dann natürlich keine 'Unterbrechungen' im Wortsinn, sondern laufen ähnlich ab wie der Aufruf eines Unterprogramms, ändern also nichts am festgelegten Programmablauf. Das Betriebssystem des PC benutzt das Interrupt-System überwiegend in diesem Sinn zur Kommunikation zu und zwischen speicherresident geladenen Programm-Moduln (BIOS, DOS, DOS-Erweiterungen, residente Benutzerprogramme) und zum Aufruf von darin enthaltenen Unterprogrammen.

3.1 Hardware-Interrupts

Am 8086 sind zur Verarbeitung von Prozessor-Unterbrechungen, die durch externe Bausteine ausgelöst werden, zwei Anschlüsse vorhanden: INTR (Interrupt Request) ist ein Eingangssignal, durch das ein Interrupt angefordert wird, INTA (Interrupt Acknowledge) ein Ausgangssignal, mit dem der Prozessor die angeforderte Unterbrechung bestätigt und nähere Information — die Interruptnummer — anfordert. Ein bestimmtes Bit im Flag-Register (Interrupt-Flag, IF) legt fest, ob der Prozessor auf eine Hardware-Interrupt-Anforderung über INTR reagieren darf. Ist dieses Bit gesetzt (Assemblerbefehl `STI`, JPI-Modula-2 `SYSTEM.EI`), kann ein Interrupt ausgelöst werden, wenn nicht (`CLI`, `SYSTEM.DI`), dann werden keine Interrupts akzeptiert.

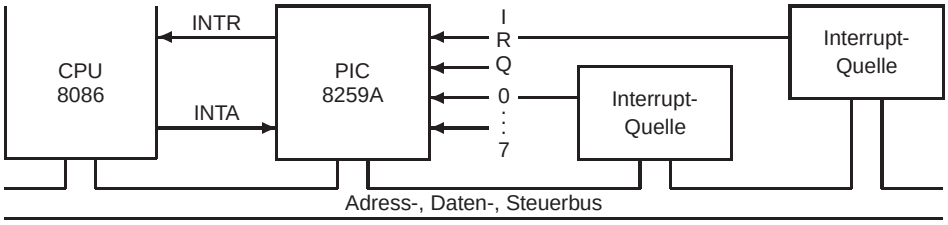


Abbildung 5: Schaltung des programmierbaren Interrupt-Controllers 8259A im PC.

An die beiden Interrupt-Leitungen ist im PC ein spezieller Baustein angeschlossen, der programmierbare Interrupt-Controller (PIC) 8259A, der seinerseits die Anforderungen externer Bausteine annimmt (Abb. 5). Dies ist notwendig, um Anforderungen zwischenzuspeichern, zwischen mehreren Interruptquellen zu unterscheiden und interruptauslösende Geräte einzeln zulassen und sperren zu können. Ein PIC kann bis zu 8 externe Interruptleitungen (Interrupt ReQuests IRQ x) überwachen, durch Kaskadierung mehrerer 8259A bis zu 64. Im PC-XT ist ein PIC eingebaut, im AT zwei, dadurch sind beim XT bis zu 8 unterschiedliche Hardware-Interrupts konfigurierbar, beim AT bis zu 16.

Der gesamte Ablauf eines Hardware-Interrupt hat etwa diese Abfolge:

1. Eine Interruptquelle (Timer, externes Gerät) betätigt die IRQx-Leitung am PIC,
2. der PIC prüft, ob dieser Interrupt zugelassen ist,
3. falls kein Interrupt mit höherer Priorität in Bearbeitung ist, wird die INTR-Leitung zum Prozessor betätigt,
4. die CPU reagiert (falls IF=1), nachdem der gerade ablaufende Befehl zu Ende geführt ist und schickt
5. ein INTA-Signal zum PIC, um anzuzeigen, daß der Interrupt akzeptiert ist,
6. ein zweites INTA-Signal zum PIC, um von diesem die Interruptnummer anzufordern,
7. der PIC legt die Interruptnummer auf den Datenbus, dies ist die Summe aus der Nummer der Interruptleitung ($0 \cdots 7$) und einer im PIC programmierbaren Konstante (8 beim PC),
8. die CPU rettet das Flag-, CS- und IP-Register, sperrt weitere Hardware-Interrupts durch Rücksetzen von IF und startet die zur Interruptnummer gehörende Bearbeitungsroutine. Die Startadresse der Interrupt-Routine wird dabei der Interrupt-Tabelle im Speicher entnommen (0:$0 \cdots$0:3FFH), dort sind die Anfangsadressen sämtlicher Interrupt-Routinen vermerkt (jeweils 4 Byte, für Interrupt n Offset bei 0:4*n, Segment bei 0:4*n+2).
9. Die Interrupt-Routine wird durch einen IRET-Befehl beendet, dieser restauriert IP-, CS- und Flag-Register, setzt damit das unterbrochene Programm fort.

Der PIC muß über einige Kommandoregister initialisiert werden, das erledigt das BIOS beim Systemstart. Bei dieser Initialisierung wird unter anderem festgelegt, auf welche Interruptnummern die 8 Interrupt-Leitungen des PIC abgebildet werden. Im laufenden Betrieb müssen vom Benutzer 2 Register des PIC beachtet werden:

- Durch Einschreiben von 20H in das Kontrollregister wird angezeigt, daß die Bearbeitung des Interrupts abgeschlossen ist. Erst danach läßt der PIC weitere Interrupts zu. Dieses Kontrollregister ist unter der Basisadresse des PIC zugänglich (20H beim PC). Jede (Hardware-)Interrupt-Bearbeitungsroutine muß somit die Befehlsfolge `MOV AL,20H; OUT 20H,AL` (Modula-2: `SYSTEM.Out(20H,20H)`) an ihrem Ende enthalten.
- Im Interrupt-Maskierungs-Register — les- und schreibbar unter Basisadresse+1 (21H im PC) — wird festgelegt, welche Interrupt-Leitungen zugelassen werden. Eine 1 im entsprechenden Bit maskiert (sperrt) den zugehörigen Interrupt, eine 0 läßt ihn zu. In diesem Register sollten nur einzelne Bits verändert werden: der alte Inhalt des Registers wird gelesen, ein Bit durch OR-Verknüpfung mit einem geeigneten Muster gesetzt oder durch AND-Verknüpfung mit dem inversen Muster zurückgesetzt, der neue Inhalt dann ins Register eingeschrieben. Interrupt 3 beispielsweise wird mit der Assemblersequenz `IN AL,21H; OR AL,08H; OUT 21H,AL` gesperrt, mit `AND AL,0F7H` anstelle des OR-Befehls zugelassen. In Modula-2 ist das Interrupt-Mask-Register über zwei Prozeduren zugänglich: `CurrentPriority()` liest das Register,

`NewPriority(mask)` setzt den Inhalt neu fest. Beim AT wird durch diese Prozeduren auch der zweite PIC mit den Interruptleitungen 8–15 berücksichtigt. Obiges Beispiel als Modula-2-Zeile:

```
SYSTEM.NewPriority(CARDINAL(BITSET(SYSTEM.CurrentPriority()) ± {3}));
```

.

3.2 Software-Interrupts

werden durch den Prozessor-Befehl `INT n` (n=0···0FFH) in einem Programm angeordnet. Der weitere Ablauf ist dann wie beim Hardware-Interrupt (8. und 9.). Während jedoch beim Hardware-Interrupt keine Parameter direkt an die Bearbeitungsroutine übergeben werden (warum?), werden beim Software-Interrupt die CPU-Register zur Parameterübergabe benutzt (zumindest bei den im Betriebssystem enthaltenen Interrupts).

3.3 Die Interrupt-Tabelle des PC

enthält die Adressen aller Interrupt-Bearbeitungs-Routinen der Rechnersoftware. Alle Arten von Interrupts im PC werden über diese Adressenliste ausgeführt, dies ist durch den internen Aufbau des 8086 festgelegt. Die Adresse der Bearbeitungsroutine für Interruptnummer n ist in den 4 Speicherbytes ab 0:4*n zu finden, die gesamte Tabelle hat somit einen Umfang von 400H Bytes.

Ein großer Teil der Interruptnummern ist in ihrer Funktion festgelegt, bestimmte Bereiche sind freigegeben und können vom Benutzer definiert werden. Hier ein Überblick über einige Funktionen:

INT 0: Division durch Null,

INT 1: Einzelschritt (für Debugging),

INT 2: Paritätsfehler (Speicherfehler),

INT 3: Haltepunkt- (Breakpoint-) Interrupt,

INT 4: Arithmetik-Überlauf,

INT 5: Print Screen, Bildschirmausdruck, wird angesprungen beim Drücken der PrtScr-Taste (Druck), aktiviert eine BIOS-Routine, die den Bildschirminhalt ausdruckt,

INT 08H: IRQ 0 (Interrupt-Leitung 0), Timer-Hardware-Interrupt,

INT 09H: IRQ 1, Tastatur-Interrupt, die Interrupt-Routine setzt den Tastatur-Scan-Code in das entsprechende ASCII-Zeichen um und schreibt beides in den Tastaturpuffer,

INT 0AH: IRQ 2, beim AT greift der zweite PIC hierauf zu, beim XT frei,

INT 0BH: IRQ 3, zweite serielle Schnittstelle COM 2 (vom BIOS nicht benutzt),

INT 0CH: IRQ 4, erste serielle Schnittstelle COM 1 (vom BIOS nicht benutzt),

INT 0DH: IRQ 5, zweite Druckerschnittstelle LPT 2 (vom BIOS nicht benutzt),

INT 0EH: IRQ 6, Disk-Controller

INT 0FH: IRQ 7, erste Druckerschnittstelle LPT 1 (vom BIOS nicht benutzt),

INT 10H: Bildschirm, alle BIOS-Routinen zum Schreiben und Lesen des Bildschirms sind über diesen Interrupt erreichbar,

INT 11H: Hardware-Konfiguration, Information in den Bits von Register AX:

0	1: Disk-Laufwerk(e) vorhanden,
2+3	Speicherbänke auf der Hauptplatine,
4+5	voreingestellter Video-Modus: 1 = CGA 40*25, 2 = CGA 80*25, 3 = Monochrom-Modus 80*25,
6+7	Zahl der Disketten-Laufwerke (falls Bit0 = 1),
9-11	Zahl der seriellen Schnittstellen,
14+15	Zahl der Drucker-Schnittstellen

INT 12H: Speichergröße, Größe des Speichers in kByte in Register AX,

INT 13H: Floppy/Harddisk, u. a. können einzelne oder eine Folge von Sektoren gelesen oder geschrieben, einzelne Spuren formatiert werden.

INT 14H: Serielle Schnittstellen, dieser vom Betriebssystem zur Bedienung vom COM1 – COM4 vorgesehene Interrupt ist nur mit Einschränkungen nutzbar, da kein Puffer vorgehalten wird.

INT 16H: Tastaturabfrage mit den Funktionen:

AH=0: Wartet auf Tastendruck, zurück mit AL=ASCII-Wert, AH=Scan-Code,

AH=1: prüft, ob eine Taste gedrückt wurde, zurück mit Z-Flag=1: keine Taste gedrückt, Z-Flag=0: Taste gedrückt, AL, AH wie oben,

AH=2: fragt den Status von Sondertasten ab, gesetzte Bits in AL zeigen an, ob die entsprechende Taste gedrückt bzw. aktiv ist, Bit 0 = SHIFT rechts, 1 = SHIFT links, 2 = CTRL (Strg), 3 = ALT gedrückt, 4 = Scroll Lock, 5 = Num Lock, 6 = Caps Lock, 7 = Insert aktiv.

INT 17H: Druckerausgabe, mit AH=0 wird das Zeichen in AL auf den durch DX adressierten Drucker ausgegeben (DX=0: LPT1), der Status des Druckers wird in AH zurückgemeldet mit Bit 0 = timeout, 3 = Fehler, 4 = Drucker selektiert, 5 = kein Papier, 6 = Acknowledge, 7 = nicht busy, mit AH=2 wird nur der Status festgestellt.

INT 19H: Bootstrap, das Betriebssystem wird neu geladen.

INT 1AH: Systemzeit, die Funktionen werden durch Register AH definiert:

AH=0: Lesen der Systemzeit (Timer-Interrupts 18.3 mal pro Sekunde), CX: obere 2 Byte, DX: untere zwei Byte des Zählers, AL>0: neuer Tag seit dem letzten Lesen.

AH=1: Setzen der Systemzeit über CX und DX.

Die weiteren Funktionen nur beim AT:

AH=2: Lesen der batteriegepufferten Echtzeituhr, die Uhrzeit wird in den Registern CH (Stunden), CL (Minuten) und DH (Sekunden) BCD-codiert übergeben.

AH=3: Setzen der Echtzeituhr, Register wie beim Lesen,

AH=4: Lesen des Datums von der Echtzeituhr, Jahr in CX, Monat und Tag in DH und DL wieder BCD-codiert,

AH=5: Setzen des Datums,

AH=6: Setzen der Alarmzeit in der Echtzeituhr (INT 70H),

AH=7: Rücksetzen des Alarms.

INT 1BH: CTRL-Break, hierher springt das Tastatur-Interrupt-Programm, wenn die Tastenkombination CTRL-Break (Strg-Untbr) gedrückt wurde.

INT 1CH: Timer-Interrupt, hierher springt die Bearbeitungs-Routine des Timer-Hardware-Interrupts, die hier vermerkte Adresse wird 18.3 mal pro Sekunde angesprungen.

INT 20H – 3FH: DOS-Aufrufe, alle DOS-Routinen werden über diesen Interruptbereich aktiviert, die überwiegende Anzahl über INT 21H.

INT 60H – 6FH: Dieser Bereich wird für Anwenderprogramme freigehalten, sind gleichzeitig mehrere Programme aktiv, können hier Konflikte auftreten.

INT 70H – 77H werden beim AT vom zweiten Interrupt-Controller verwendet, die Eingänge 0–7 im zweiten PIC (Interrupt-Leitungen 8–15 im AT) werden auf die Interruptnummern 70H–77H abgebildet.

INT 70H (IRQ 8) wird vom Echtzeituhr-Alarm benutzt,

INT 75H (IRQ 13) vom Coprozessor und

INT 76H (IRQ 14) vom Festplattencontroller.

Die übrigen Interrupt-Leitungen des zweiten PIC sind für den Anwender verfügbar.

3.4 Interrupt-Software

Die Grundregeln noch einmal zusammengefaßt:

- Interrupt-Routinen werden wie ‘normale’ Subroutinen (Prozeduren) geschrieben, statt `RET` muß am Ende `IRET` stehen.
- Beim Aufruf von Software-Interrupts können Parameter nur in den Registern, nicht im Stack übergeben werden. Bei Hardware-Interrupts ist keine direkte Parameterübergabe möglich, Variable werden auf vereinbarten Speicherplätzen abgelegt.
- Bei Hardware-Interrupts wird in einem Initialisierungsteil der zugehörige Anschluß im PIC entsperrt und das Interrupt-Flag im CPU-Flag-Register gesetzt, am Ende der Interrupt-Routine muß der PIC quittiert werden.

- Die Adresse der Interrupt-Routine wird in der Interrupt-Tabelle vermerkt, dies kann direkt oder über einen DOS-Aufruf gemacht werden.
- Der alte Eintrag in der Interrupt-Tabelle kann dazu benutzt werden, schon vorhandene Interrupt-Routinen mit einzubinden (Sprung an die alte Adresse statt `IRET`).

Das folgende Beispiel ist für die Benutzung im Debug geschrieben, es installiert eine Routine für den CTRL-Break-Interrupt (INT 1BH), die benutzt wird, um eine Endlosschleife abzubrechen (Testschleife für das Druckerport).

```
Adresse  Befehl

0100     SUB  AX,AX       ; Das DS-Register wird auf 0 gesetzt, um das Speichersegment
0102     MOV  DS,AX       ; der Interrupt-Tabelle zu adressieren,
0104     MOV  [6E],CS     ; 6EH = 4*1BH+2, CS = Segmentadresse der Int-Routine,
0108     MOV  AX,110      ; Offsetadresse der Int-Routine,
010B     MOV  [6C],AX     ; 6CH = 4*1BH
010E     JMP  117         ; Sprung zum eigentlichen Programm,

0110     SUB  AL,AL       ; die Interrupt-Routine ändert die
0112     CS:              ; Sprungweite am Ende der Endlosschleife des
0113     MOV  [124],AL    ; Programms auf 0,
0116     IRET             ; Ende der Interrupt-Routine,

0117     MOV  AX,40       ; die Adresse des Druckerports wird aus
011A     MOV  DS,AX       ; dem BIOS-RAM geholt und
011C     MOV  DX,[8]      ; in das Register DX geladen,
0120     INC  AL          ; alle Bitmuster werden nacheinander möglichst schnell an
0122     OUT  DX,AL       ; das Druckerport ausgegeben,
0123     JMP  120         ; unbedingter Sprung zum Schleifenanfang,
0125     INT  20          ; DOS-Interrupt zur Programmbeendigung.
```

Ein etwas längeres Assemblerbeispiel — eine speicherresidente Interrupt-Routine für den Timer-Interrupt 1CH — ist in CLOCK.ASM ausgeführt. Der Zugang zur Interrupt-Tabelle ist dort über DOS-Aufrufe realisiert.

JPI-Modula-2 steuert die Einhaltung der genannten Bedingungen durch Compiler-Anweisungen:

$J+ legt fest, daß Prozeduren mit `IRET` enden, $J- macht dies wieder rückgängig.

$C FF ordnet an, daß beim Prozedureintritt alle Register gerettet werden, Standard ist $C F0, nur ES, DS, SI und DI werden gerettet.

$W+ verbietet die Verwendung von Registervariablen. Wenn die Interrupt-Routine Zugang zu (globalen) Variablen haben soll, müssen diese im Speicher gehalten werden.

Im Modul RS232.MOD kann die Verwendung dieser Compileranweisungen verfolgt werden. Dieser Modul enthält eine Interrupt-Routine zur Bedienung der seriellen Schnittstelle im PC.

4 Zeit im PC

Nahezu überall im Experimentalbereich spielt die Zeit als Parameter eine gewisse Rolle: als reproduzierbare Zählzeit eines Ereigniszählers, als Ansteuerfrequenz eines Schrittmotors, als Wartezeit, die bei einem Analog-Digital-Wandler eingehalten werden muß, bis ein gültiger Wert vorliegt, als Koordinate, von der andere Parameter — Temperatur, Druck, usw. — funktional abhängen sollen.

Der PC ist von seinem Konzept her ("Personal"-Computer) nicht für diesen Anwendungsbereich gebaut, ein exakter Echtzeitbetrieb ist nicht oder nur in begrenztem Umfang möglich. Dies hängt in erster Linie damit zusammen, daß die Latenzzeit für Interrupts (Zeit zwischen Anforderung und Sprung zur Bearbeitungsroutine) in gewissen Grenzen schwankt. Nur dort, wo solche Ungenauigkeiten keine Rolle spielen (z. B. bei der Taktfrequenz eines Schrittmotors), können Zeiten vom PC vorgegeben werden, wo es auf hohe Genauigkeit und Reproduzierbarkeit ankommt (z. B. bei der Meßzeit eines Ereigniszählers), sollte der Zeittakt mit im Peripheriegerät generiert werden.

Im PC stehen verschiedene "Uhren" für den Anwender zur Verfügung:
Kurze und mittlere Wartezeiten (10 μsec ... $\approx$10 sec) realisiert man am einfachsten durch "Abzählen" des CPU-Taktes mit geeigneten FOR-Schleifen. Diese sollten vor der Benutzung geeicht werden, entweder am Timer-Takt oder an der Echtzeituhr des PC. Ein Beispiel im Modul SM; die Delay-Prozedur von Modula-2 arbeitet ähnlich, die Zeiteinheit ist dort 1 millisec. Die Methode hat den offensichtlichen Nachteil, daß der Rechner während der Wartezeit für andere Aktionen ausfällt. Oft kann dies aber in Kauf genommen werden (kurze Laufzeit eines Schrittmotors — lange Meßzeit).
Längere Wartezeiten sind einfacher durch Abfrage der System- oder Echtzeituhr oder über den Timer-Interrupt zu generieren. Daneben besteht natürlich auch die Möglichkeit, Zeiten von außen her vorzugeben (rechtzeitiger Tastendruck des Benutzers, Hardware-Interrupt von einem Gerät — gut geeignet ist IRQ 7).

4.1 Zeit-Interrupts

Außer dem schon besprochenen BIOS-Interrupt 1AH zur Abfrage der Echtzeituhr und des Timer-Zählers gibt es zwei DOS-Interrupts, mit denen man Zeit und Datum der PC-Systemuhr erfragen kann:

- Über INT 21H mit AH=2CH erfährt man die Systemzeit in folgenden Registern:
  CH=Stunden, CL=Minuten, DH=Sekunden, DL=1/100 Sekunden,
- über INT 21H mit AH=2AH das Datum mit CX=Jahr, DH=Monat, DL=Tag.

Ein Anwendungsbeispiel ist die Prozedur GetTime, die einen String mit dem Aufbau JJJJMMTThhmm konstruiert.

Neben diesen Interrupts zur Zeitabfrage können die beiden Timer-Takt-Interrupts (INT 8, INT 1CH) benutzt werden, wenn eine vorgegebene Zeitabfolge einzuhalten ist. INT 8 wird bei jedem Timer-Interrupt aktiviert, über eine BIOS-Routine wird die Systemzeit (Timer-Zähler im BIOS-RAM) inkrementiert und unter anderem der INT 1CH aktiviert. Dieser ist für Benutzeranwendungen vorgesehen, die dort angehängte Interrupt-Routine wird ca. 18 mal pro Sekunde aufgerufen (Beispiel in CLOCK.ASM). Der Ablauf ist wie für einen Software-Interrupt typisch, abgesehen davon, daß jede Sperre des Hardware-Interrupts INT 8 (durch DI oder Maskierung) auch den INT 1CH betrifft.

In Sonderfällen ist es auch möglich, INT 8 direkt zu benutzen. Das kann dann sinnvoll sein, wenn durch eine Änderung der Timer-Zeitkonstante (s. nächster Abschnitt) für Steuerungsanwendungen ein schnellerer Zeittakt im PC erzeugt werden soll. Die direkte Benutzung von INT 8 hat den Vorteil, daß zur Bearbeitung des Timer-Interrupts sehr kurze Routinen benutzt werden können, dadurch auch ein relativ schneller Zeittakt möglich wird. Wird von der angehängten Interrupt-Routine nicht zur BIOS-Routine weiterverkettet — was aus Zeitgründen geboten sein kann —, muß in der Interrupt-Routine der PIC quittiert werden, da sonst keine weiteren Interrupts zugelassen werden. Außerdem ist darauf zu achten, daß die Benutzung des INT 8 und die geänderte Zeitkonstante zu einer undefinierten oder falschen Systemzeit führen, diese sollte danach wieder richtiggestellt werden.

4.2 Der Timerbaustein 8253/8254

Der Zeittakt für die Systemuhr des PC wird durch einen Zeitgeberbaustein (Timer) 8253 bzw. den funktionsähnlichen 8254 erzeugt. IC-Bausteine dieses Typs werden meist auch auf Peripheriekarten verwendet, wenn dort ein programmierbarer Zeittakt benötigt wird. In dem Timerbaustein sind 3 unabhängige 16-Bit-Zähler integriert, die Zählkapazität pro Kanal beträgt mithin 2^{16}, die Zählweise ist abwärts, pro Kanal gibt es einen Zähleingang (Clock) und einen Ausgang, der den Nulldurchgang des jeweiligen Zählers anzeigt. Die einzelnen Zähler des 8253 können von ihrem Aufbau her auch als Ereigniszähler benutzt werden, diese Verwendung ist aber aus verschiedenen Gründen problematisch, im allgemeinen auch von der Beschaltung nicht vorgesehen. Bei der Verwendung als Taktgenerator (Timer) werden die Zähler auf bestimmte Anfangswerte programmiert (Preset), die beim Nulldurchgang jeweils wieder restauriert werden. Dadurch sind 2^{16} unterschiedliche Zeitkonstanten programmierbar. Vom Timerbaustein werden 4 aufeinanderfolgende Adressen im Ein/Ausgabeadressbereich belegt: Timerkanal0 ist über die Basisadresse zugänglich (im PC 40H), Timer1 über Basis+1, Timer2 über Basis+2, das Steuerregister für alle Timer über Basis+3. Die 3 Timerkanäle im PC werden für die folgenden Funktionen genutzt:

Timer0 veranlaßt bei jedem Nulldurchgang einen Timer-Interrupt, der Ausgang ist an die IRQ0-Leitung angeschlossen.

Timer1 steuert das Auffrischen der dynamischen RAM-Bausteine. Zu diesem Zweck werden im Zeitabstand von 15 μsec DMA-Zyklen im DMA-Controller angestoßen.

Timer2 liefert ein Rechtecksignal zur Ansteuerung des eingebauten Lautsprechers.

Die Eingangsfrequenz für alle Timerkanäle liefert ein quarzstabilisierter Frequenzgenerator. Dessen Frequenz ist so gewählt, daß 2^{16} Timer-Interrupts genau einer Stunde entsprechen. Timer0 ist auf 2^{16} programmiert, die Quarzfrequenz ist somit genau $2^{16} * 2^{16} / 3600$=1193046 Hz. IBM verwendet 1193180 Hz, Olivetti beim M 24 aus Kostengründen[1] 1228800 Hz. Aus der obigen Bedingung (2^{16} Timer-Interrupts pro Stunde) erklärt sich auch der etwas willkürlich erscheinende Timertakt von ca. 18 Hz.

[1] Beim M 24 werden der Takt für den Timer und der für die serielle Schnittstelle (1843200 Hz) aus dem gleichen Quarz durch Drei- bzw. Zweiteilung erzeugt.

Die Programmierung des Timers 8253

Jeder Timerkanal n des 8253 wird durch Einschreiben eines Steuerbytes in das Steuerregister (Basisadresse+3) und anschließendes Einschreiben des Preset-Wertes auf die Timeradresse n (Basis+n) programmiert.

Aufbau des Steuerbytes für den Timer 8253:

BIT	7	6	5	4	3	2	1	0
	S1	S0	RL1	RL0	M2	M1	M0	BCD

S1, S0 selektieren den Timerkanal (00, 01, 10 für Timer0, 1, 2).

RL1, RL0 legen fest, wie der Timerkanal beschrieben (Preset) und gelesen werden werden soll (10: nur höherwertiges Byte, 01: nur niederwertiges Byte, 11: zuerst niederwertiges, dann höherwertiges Byte). 00 auf diesen Bits ordnet an, daß der aktuelle Zählerstand zum Lesen zwischengespeichert werden soll.

M2, M1, M0 legen die Betriebsart des Zählers fest, in Betriebsart 2 (010) wird bei jedem Nulldurchgang ein kurzer Impuls am Ausgang erzeugt, der zum Aktivieren von Interruptleitungen und ähnlichem benutzt werden kann, in Betriebsart 3 (011) arbeitet der Zähler als Rechteckgenerator.

BCD legt fest, ob dezimal (1) oder binär (0) gezählt werden soll. Dezimal beträgt die Zählkapazität 10^4 (4 Stellen), binär 2^{16}.

Nach dem Einschreiben des Steuerbytes sollte — wie in den RL-Bits festgelegt — der Preset-Wert für den jeweiligen Zählerkanal eingetragen werden. Das Lesen des aktuellen Zählerstands kann jederzeit (nach Zwischenspeicherung (Latching)) durch Einlesen der durch die Programmierung festgelegten Byte-Anzahl erfolgen. Der Preset-Wert läßt sich nach der Anfangsprogrammierung auch ohne neues Steuerbyte ändern.

Eine Modula-2-Prozedur für die Initialisierung könnte ungefähr so aussehen:

```
CONST
  tbase = ...;                  (* timer base address               *)
  rl = 3;                       (* read/load low byte, then high byte *)

PROCEDURE TimerInit(tnr, mode : SHORTCARD;     (* tnr=0..2, mode=0..5 *)
                    preset : CARDINAL);        (* time constant       *)
BEGIN
  SYSTEM.Out(tbase+3, (tnr<<6)+(rl<<4)+(mode<<1));       (* command   *)
  SYSTEM.Out(tbase+tnr, SHORTCARD(preset));              (* low byte  *)
  SYSTEM.Out(tbase+tnr, SHORTCARD(preset>>8));           (* high byte *)
END TimerInit;
```

5 Parallele Schnittstellen

werden benutzt, um Peripheriegeräte über mehrere Leitungen gleichzeitig anzusprechen und um Daten schnell byte- oder wortweise einzulesen oder auszugeben. Ein Standardgerät, das im allgemeinen auf diese Weise angeschlossen ist, ist der Drucker. Die Druckerschnittstelle arbeitet mit 8 Datenleitungen (ein Byte wird jeweils komplett zum Drucker übertragen), einer Leitung, die anzeigt, daß die Daten gültig sind und einer Leitung, die meldet, ob der Drucker beschäftigt ist. Weitere Leitungen sind für Statusmeldungen zuständig.

Bei konfigurierbaren Parallel-Schnittstellen (die also nicht ausschließlich für einen Verwendungszweck vorgesehen sind wie die Druckerschnittstelle) wird überwiegend der Baustein 8255 verwendet, für festgelegte Spezialanwendungen sind oft Sonderanfertigungen notwendig (schnelle Kopplung von externen Geräten an den PC).

5.1 Die Druckerschnittstelle des PC

wird, da nur für einen speziellen Zweck — die Ansteuerung des Druckers — vorgesehen, über festverdrahtete, nicht weiter konfigurierbare Ausgabe- und Eingabe-Register betrieben. Einen schematischen Überblick über die Schaltung und die Zuordnung der Register-Bits zur 25-poligen Buchse am PC gibt Abb. 6.

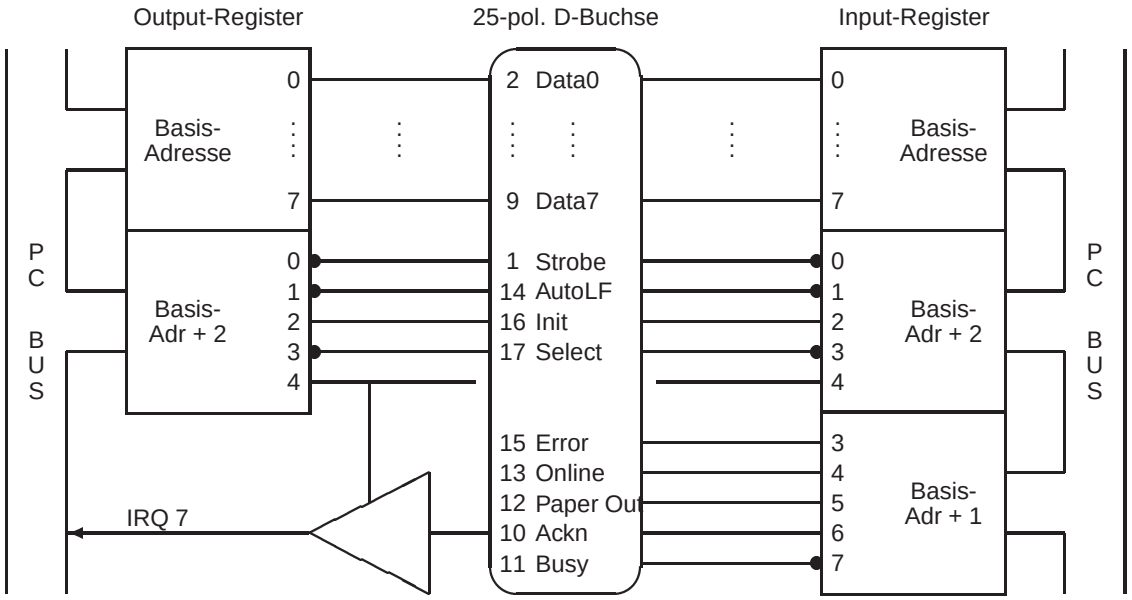


Abbildung 6: Schaltung des Druckeranschlusses im PC, Pins 18–25 der 25-poligen D-Buchse sind mit Masse verbunden. • : Invertierung des Signals.

Über die Basisadresse der Druckerschnittstelle (aus dem BIOS-RAM bei 40H:8) werden die 8 Datenleitungen zum Drucker angesprochen, ihr Status kann über diese Adresse festgelegt und gelesen werden.

Das Ausgaberegister mit Basisadresse+2 bedient mit den Bits 0–3 die zum Drucker gehenden Kommandoleitungen. Bit 4 dieses Registers legt fest, ob die vom Drucker kommende Acknowledge-Leitung auf die IRQ7-Leitung des PC zugreifen und damit einen Interrupt auslösen kann. Auch der Status dieser Leitungen kann — über Basisadresse+2 — eingelesen werden. Die vom Drucker kommenden Statusleitungen sind mit den Bits 3–7 eines Eingaberegister verbunden, das unter Basisadresse+1 gelesen werden kann.

Insgesamt sind somit durch die Druckerschnittstelle in jedem PC 12 Ausgabeleitungen und 5 Eingabeleitungen vorhanden, eine Eingabeleitung kann als Interruptleitung benutzt werden. Die Signale sind TTL-kompatibel, d. h. logisch 1 ist nominell +5 V, logisch 0 nominell 0 V. Zumindest die Datenleitungen und die Strobeleitung lassen relativ niederohmigen Betrieb zu ($> 150\,\Omega$). Verwendet man paarweise verdrillte Kabel oder Flachbandkabel zum Anschluß, so ist durch die Anordnung der Masseleitungen ein sehr störsicherer Betrieb bis zu Frequenzen von mehr als 100 kHz möglich. Mithin ist die Druckerschnittstelle auch sehr gut für Steuerungszwecke zu gebrauchen. Eine Anwendung, die Schrittmotorsteuerung und die Abfrage der Endschalter an einem Monochromator, zeigt der Modul SM.

Die IRQ7-Leitung wird vom Betriebssystem nicht genutzt (wie auch die IRQ5-Leitung einer eventuell vorhandenen zweiten Druckerschnittstelle), daher bietet sie eine — sehr einfach zu handhabende — Möglichkeit für Anwender-Interrupts. An den Acknowledge-Anschluß (Pin 10) wird das Gerät angeschlossen, das den Interrupt auslösen soll. In Frage kommen z. B. PC-Einsteckkarten, die nach einer Initialisierung durch den Rechner im wesentlichen selbständig arbeiten (Bildaufnahme, Multichannelscaling, Impulshöhenanalyse) und von Zeit zu Zeit die gesammelten Daten an den PC übergeben, oder externe Datenerfassungssysteme mit ähnlicher Arbeitsweise. Der PC braucht dann nicht ständig “auf der Lauer” zu liegen, sondern wird zu gegebener Zeit zur Ausführung der passenden Interrupt-Routine veranlaßt. Die Acknowledge-Leitung muß — wie aus der Schnittstellenschaltung ersichtlich — durch das Setzen von Bit 4 im Ausgaberegister Basisadresse+2 mit der IRQ7-Leitung verbunden werden, der Interrupt kann dann mit einem kurzen Impuls (einige Mikrosekunden) ausgelöst werden. Für Interrupt-Routine und PIC gelten die schon besprochenen Regeln. Das Modula-2-Programm IRQ7 verdeutlicht diese nochmals an einem einfachen Beispiel, als Interruptquelle dient ein an die Druckerschnittstelle angeschlossener Schalter.

5.2 Der programmierbare Parallel-E/A-Baustein 8255 A

Auf fast allen Steckkarten, die parallele Ein/Ausgabeleitungen zur Verfügung stellen und/oder über solche statischen Leitungen andere Bausteine betreiben (D/A-, A/D-Wandler, Zähler, Relais), wird zu diesem Zweck der Intel-Schaltkreis 8255 A eingesetzt. Er besitzt 24 programmierbare E/A-Leitungen, die in drei Gruppen zu je 8 unterteilt sind (Ports A, B, C). Port C läßt sich weiter in 2 Gruppen zu je 4 aufteilen (CH, CL). Für jede der Gruppen kann (muß) die Betriebsart (Eingabe oder Ausgabe) definiert werden, so daß je nach Programmierung $0, 4, 8 \cdots 24$ Eingabe- und entsprechend $24, 20, 16 \cdots 0$ Ausgabeleitungen vorhanden sind. Der Baustein belegt 4 aufeinanderfolgende Adressen im E/A-Adressraum, die Basisadresse ist Port A zugeteilt, Basis+1 Port B, Basis+2 Port C, auf Basisadresse+3 wird das Steuerbyte geschrieben, das entweder Betriebsart und Betriebsmodus festlegt oder ein Einzelbitkommando enthält. Die genaue Beschaltung der E/A-Leitungen (Steckerart und -belegung etc.) variiert von Karte zu Karte und muß im Einzelfall der Beschreibung entnommen werden.

Aufbau des Steuerbytes für den Parallelbaustein 8255 A (Betriebsart und -modus):

BIT	7	6	5	4	3	2	1	0
	1	M2	M1	A	CH	M0	B	CL

BIT 7 = 1 legt fest, daß mit diesem Kommandobyte Betriebsart und Betriebsmodus festgelegt werden sollen.

M2, M1 definieren den Betriebsmodus für Port A und CH,

M0 den für Port B und CL, diese 3 Bits sollten auf 0 gesetzt werden, dies bedeutet normale Ein- oder Ausgabe. Die anderen möglichen Betriebsmodi (automatischer Quittungs- oder Interruptbetrieb) sind hardwaremäßig meist nicht vorgesehen.

A, CH, B, CL definieren die Betriebsart für die einzelnen Gruppen, ein auf 0 gesetztes Bit bedeutet Ausgabe, ein auf 1 gesetztes Eingabe.

Das Steuerbyte 83H würde mithin die Gruppen A und CH als Ausgabe-, die Gruppen B und CL als Eingabeleitungen festlegen.

Wie beim Druckerport kann auch beim 8255 der aktuelle Status der Ausgabeleitungen durch einen Input-Befehl gelesen werden. Dies ist wichtig, wenn es darauf ankommt, nur einzelne Leitungen (Bits) anzusprechen, ohne die übrigen eines Ports oder einer Gruppe zu verändern.

Für Port C gibt es beim 8255 eine zusätzliche Möglichkeit, einzelne Bits gezielt anzusprechen, das Einzelbitkommando, bei dem das Steuerbyte (Basisadresse+3) wie folgt aufgebaut sein muß:

BIT	7	6	5	4	3	2	1	0
	0				N2	N1	N0	W

BIT 7 = 0 ist das Zeichen für ein Einzelbitkommando.

N2, N1, N0 adressieren das Bit in Port C, das auf den Wert von

W gesetzt wird.

01H würde Bit 0 auf 1 setzen, 06H Bit 3 auf 0, 0FH Bit 7 auf 1.

Beispiele für das Arbeiten mit dem Baustein 8255 gibt der Modul Interface, der Prozeduren zum Betrieb mehrerer externer Geräte (Zähler, Relais, Digital-Analog-Wandler) beinhaltet, die über zwei 8255 an den PC angeschlossen sind.

6 D/A- und A/D-Wandler

Experimentelle Steuergrößen (Heizstrom, elektrisches Feld, ...), die vom Rechner vorgegeben werden sollen, müssen dazu meist zuerst in Analogwerte umgesetzt werden. Diese Digital/Analog-Wandlung erfolgt in digital ansteuerbaren Peripheriegeräten oder auf PC-Karten durch spezielle Bausteine — D/A-Wandler. Experimentelle Meßwerte andererseits (Spannung, Widerstand, Temperatur, Lichtintensität, Druck, ...) liegen am Experiment im allgemeinen analog vor und müssen — gegebenenfalls nach Umwandlung in elektrische Größen — zur Bearbeitung durch den Rechner in Digitalwerte umgeformt werden. Auch diese Analog/Digital-Wandlung erfolgt — in externen Meßgeräten wie Digitalvoltmetern oder auf PC-Karten — durch spezielle Bausteine, A/D-Wandler. Im folgenden sollen die Verfahren, nach denen die Wandlerbausteine arbeiten, etwas näher betrachtet werden.

6.1 Digital/Analog-Wandler

Das praktisch ausschließlich verwendete technische Konzept für Digital/Analog(D/A)-Wandler ist das der Stromsummation: Am Eingang eines Operationsverstärkers werden Ströme aufsummiert, deren Größe der Wertigkeit der einzelnen Bits in dem umzuwandelnden Digitalwert entspricht. Die idealisierte Schaltung zeigt Abb. 7.

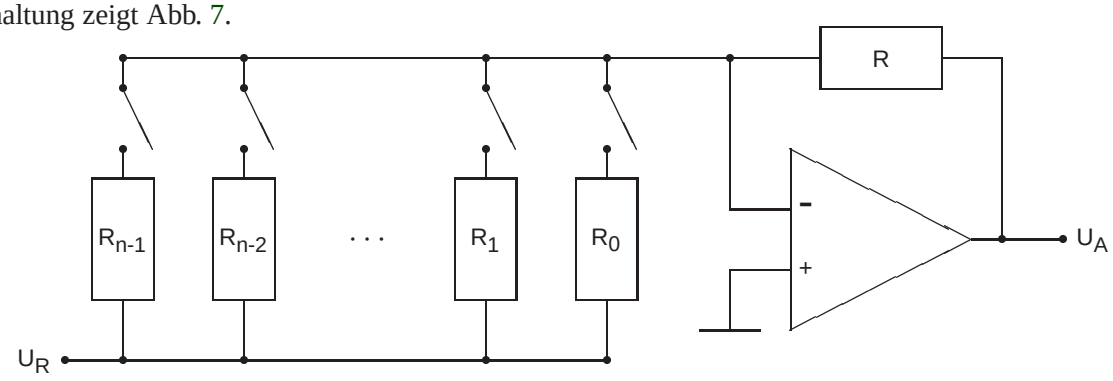


Abbildung 7: Prinzipschaltung eines D/A-Wandlers.

Beim idealen Operationsverstärker sind Differenzeingangsspannung und Eingangsströme gleich null, daraus ergibt sich für die Ausgangsspannung U_A:

$$U_A = -U_R \cdot R \cdot \sum_{i=0}^{n-1} S_i/R_i$$

$S_i = 1$, wenn der entsprechende Schalter geschlossen, $S_i = 0$, wenn der entsprechende Schalter geöffnet ist. Die Schalter (meist als Feldeffekttransistoren realisiert) werden durch die einzelnen Bits des umzusetzenden Digitalwerts angesteuert, die Widerstände R_i haben die Widerstandswerte $R_i = R_0 \cdot 2^{-i}$. Damit erhält man eine Ausgangsspannung, die proportional zum Digitalwert ist.

Die Schaltung Abb. 7 hat den großen Nachteil, daß man sehr unterschiedliche Widerstandswerte benötigt. In der Praxis verwendet man daher eine modifizierte Schaltung, das R/2R-Netzwerk. Das Funktionsprinzip wird aus Abb. 8 deutlich, wenn man sich klarmacht, daß an den Punkten i=0,1,· · ·,n-1 jeweils die Spannung $U_R \cdot 2^{-i}$ anliegt.

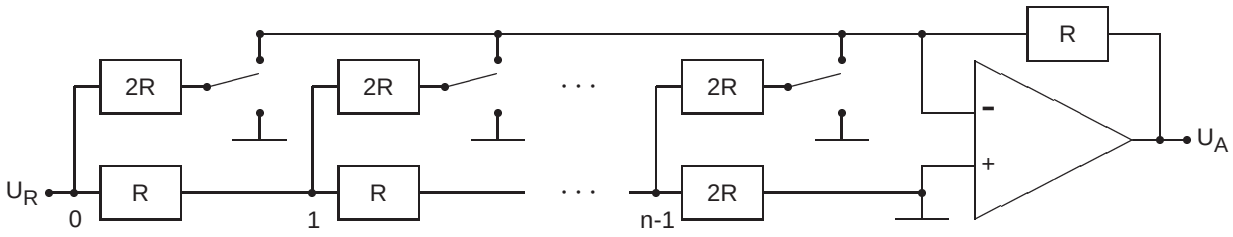


Abbildung 8: Digital/Analog-Wandler mit R/2R-Netzwerk

Die Güte eines D/A-Wandlers und damit seine Verwendungsmöglichkeit für eine bestimmte Steuerungsaufgabe läßt sich durch einige Kenngrößen charakterisieren:

- Die **Auflösung** gibt die Bitbreite des gewandelten Digitalwertes an. Sie liegt im allgemeinen zwischen 8 und 16 Bit, durch die neueren Entwicklungen im Audiobereich (CD-Player) sind zur Zeit auch D/A-Wandler mit hoher Auflösung (14 Bit, 16 Bit) relativ preisgünstig. Die Auflösung bestimmt die minimale Schrittweite, mit der eine Steuergröße vom Rechner vorgegeben werden kann.
- **Genauigkeit** und **Linearität** sowie deren Temperaturabhängigkeit geben an, wie gut die vorhandene (digitale) Auflösung in die Realität der analogen Welt umgesetzt wird. Diese Größen sind von den Toleranzen und dem Temperatur- und Alterungsverhalten des internen Widerstandsnetzwerks abhängig. Hohe Anforderungen hier bedingen hohen Herstellungsaufwand.
- Die **Geschwindigkeit**, oft angegeben als Anstiegszeit (10%–90%) für einen Ausgangsspannungssprung von der minimalen zur maximalen Ausgangsspannung, kann als Kriterium im allgemeinen außer acht gelassen werden, da die Datenausgabe vom PC fast immer langsamer ist und da die typischen Zeitkonstanten der zu steuernden Geräte (Heizung etc.) meist sehr viel größer sind.

Die Ansteuerung von D/A-Wandlern durch den Rechner hängt davon ab, wie der Digitalteil des Wandlers ausgeführt ist. Im einfachsten Fall sind die “Schalter” durch TTL-kompatible Logiksignale direkt zugänglich, dann muß ein Baustein vorgeschaltet werden, der den Digitalwert zwischenspeichert und auch nach dem Ausgabebefehl statisch am D/A-Wandler anliegen läßt (z. B. ein paralleler E/A-Baustein 8255).

Ist ein internes Speicherregister im D/A-Wandler vorhanden, wird der Digitalwert entweder parallel oder getaktet seriell übergeben. Parallele D/A-Wandler sind ähnlich wie parallele E/A-Bausteine zu sehen: Der Digitalwert wird — in Bytes aufgeteilt — durch Ausgabebefehle übergeben, ein Trigger-Befehl veranlaßt die interne Übernahme des kompletten Werts.

Bei seriellen D/A-Wandlern wird der Digitalwert durch einen Daten- und einen Takteingang seriell (bitweise) in ein Schieberegister eingetaktet, die interne Übernahme wird durch einen weiteren Eingang getriggert. Drei Leitungen reichen also zum Anschluß — z. B. an ein Parallelport im PC — aus. Ein Ansteuerprogramm für diese Art von D/A-Wandlern muß den umzuwandelnden Digitalwert in eine Bitfolge umsetzen und diese zusammen mit richtig gesetztem Taktbit und Übernahmebit über eine parallele Schnittstelle seriell ausgeben. Eine Prozedur, die diese Art der Ansteuerung leistet, ist im Modul Interface enthalten (DAC).

6.2 Analog/Digital-Wandler

Im Gegensatz zu den D/A-Wandlern werden unterschiedliche technische Konzepte verwendet, die sich in Auflösung, Geschwindigkeit und Aufwand zum Teil beträchtlich unterscheiden.

6.2.1 Parallel-A/D-Wandler

auch als Flash-A/D-Wandler bezeichnet, vergleichen die anliegende Analogspannung in einer Reihe von 2^n Komparatoren[2] mit 2^n Vergleichsspannungen (n = Bitbreite des Digitalwertes).

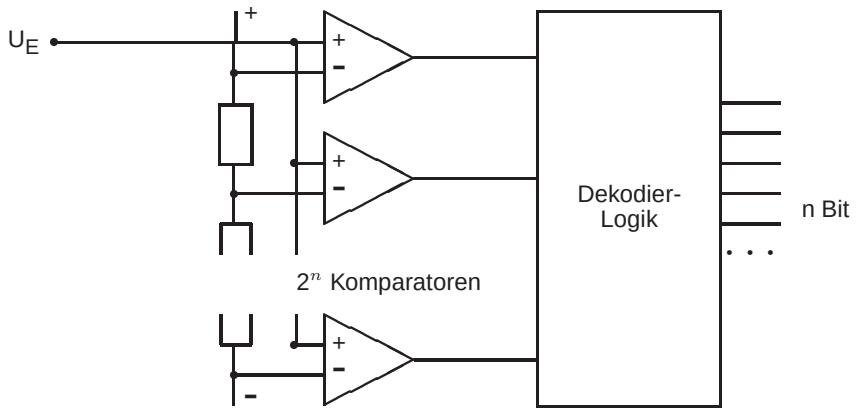


Abbildung 9: Parallel-Analog/Digital-Wandler

Das Prinzip ist sehr aufwendig (für einen 8-Bit-Wandler benötigt man 256 Komparatoren), allerdings auch sehr schnell. Die schnellsten Flash-Wandler erreichen Wandlungsraten von etwa 1 Gigahertz. Eingesetzt werden solche Wandler in Digitalisierern, mit denen sehr schnelle Vorgänge aufgezeichnet werden sollen (Transientenrecorder). Diese Digitalisierer speichern die Daten zunächst intern in einem entsprechend schnellen Speicher, aus dem sie dann langsamer abgerufen werden können. Ein direkter Anschluß von Flash-Wandlern an Rechner ist aus naheliegenden Gründen meist nicht sinnvoll.

6.2.2 Integrations- und Zählverfahren

Bei diesem Verfahren wird durch die Meßspannung der Strom einer Kondensatoraufladung gesteuert (Operationsverstärker als Integrator geschaltet). Die Zeit, die benötigt wird, um eine bestimmte Aufladespannung zu erreichen, ist umgekehrt proportional zum Ladestrom und damit zur Meßgröße. Durch Abzählen der Taktimpulse eines hochgenauen Taktgenerators (quarzstabilisiert) kann diese Zeit und damit die Meßgröße sehr genau bestimmt werden.

Eine Verbesserung dieses ‘Single-Slope’-Verfahrens ist das ‘Dual-Slope’-Verfahren, dessen Funktionsweise in Abb. 10 dargestellt ist. Die Integration über die Meßspannung erfolgt hier für eine fest vorgegebene Zeit $t_1 - t_0$, die durch Abzählen von Z_0 Taktimpulsen gemessen wird. Die nach dieser Zeit

[2] Ein Komparator kann als speziell beschalteter Operationsverstärker angesehen werden, der die beiden Eingangsspannungen vergleicht und das Vergleichsresultat in ein Logiksignal umsetzt. $U_+ > U_- \Rightarrow 1$, $U_+ < U_- \Rightarrow 0$.

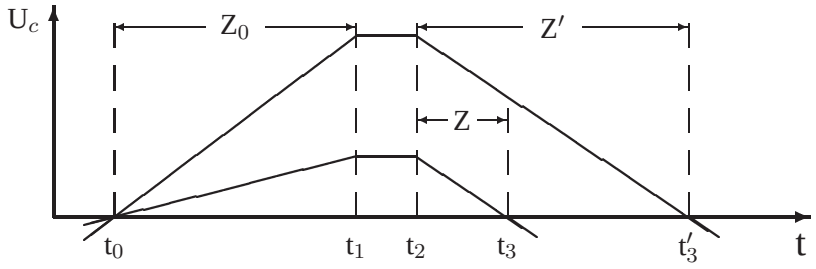


Abbildung 10: Dual-Slope-Verfahren: Zeitlicher Spannungsverlauf an der Integratorkapazität für zwei verschiedene Meßspannungen.

erreichte Ladespannung ist proportional zur Meßgröße. Zum Zeitpunkt t_1 wird die Meßspannung abgeschaltet, zum Zeitpunkt t_2 stattdessen eine hochgenaue Referenzspannung umgekehrten Vorzeichens am Integrator angelegt. Die Zeit bis zur Entladung des Kondensators ($t_3 - t_2$) ist nun — da die steuernde Referenzspannung konstant ist und damit die Steigung der Entladegeraden fest — proportional zur Ladespannung bei t_2. Auch diese Zeit wird durch Zählen von Taktimpulsen gemessen ($Z,\ Z'$). Die Meßspannung kann dann auf sehr einfache Weise berechnet werden:

$$U_{mess} = -U_{ref} \cdot Z/Z_0$$

Zweckmäßigerweise werden $-U_{ref}$ und Z_0 so gewählt, daß ihr Quotient der geforderten Auflösung entsprechen (z. B. $-U_{ref}/Z_0 \ = \ 1\ \mu V$), dann ist Z ohne weitere Umrechnung der Zahlenwert der Meßspannung.

Der große Vorteil des Dual-Slope-Verfahrens besteht darin, daß Ungenauigkeiten des Taktgebers und des Integrators nur eine geringe Rolle spielen, da sie in Auflade- und Entladevorgang in gleicher Weise eingehen und damit das Meßergebnis in erster Näherung nicht verfälschen. Zur Unterdrückung von Störspannungen kann die Aufladezeit $t_1 - t_0$ so festgesetzt werden, daß sie einem ganzzahligen Vielfachen der Periode der Störspannung entspricht. Üblich sind Vielfache von 20 millisec, um 50 Hz-Störungen auszuschalten.

Integrierende A/D-Wandler sind die weitaus genauesten, das Verfahren wird insbesondere bei Digitalmultimetern eingesetzt. Ein kleiner Nachteil sind die relativ langen Integrationszeiten, die keine allzu schnelle Meßfolge zulassen.

Eine etwas modifiziertes Verfahren wird bei der Impulshöhenanalyse in Vielkanalanalysatoren benutzt. Eine Kapazität wird auf eine Spannung aufgeladen, die dem Spitzenwert des zu analysierenden Impulses entspricht, die Entladung erfolgt dann mit konstantem Strom. Die Entladezeit wird wie beim Dual-Slope-Verfahren gemessen, der Zahlenwert ist proportional zur Impulshöhe und dient zur Adressierung des Vielkanalanalysatorspeichers. Hierbei sind Taktfrequenzen von einigen 100 MHz Stand der Technik, die Entladezeit und damit der notwendige Zeitabstand zum nächsten Impuls beträgt etwa 10 μsec.

6.2.3 Vergleichsverfahren

Beim Vergleichs- oder Kompensationsverfahren wird die Meßspannung in einem Komparator mit einer Testspannung verglichen, die durch einen Digital/Analog-Wandler erzeugt wird. Die vereinfachte

Schaltung zeigt Abb. 11.

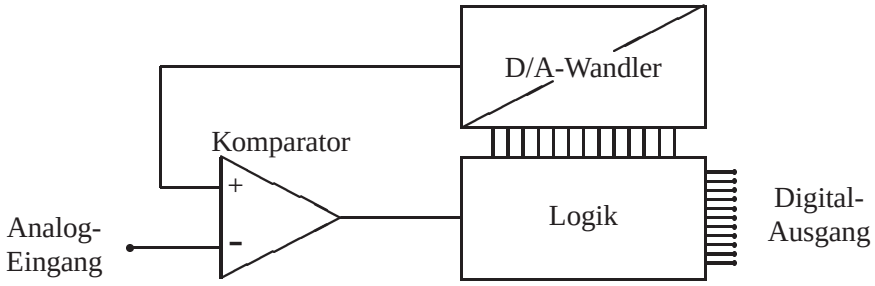


Abbildung 11: Grundschaltung von Analog/Digital-Wandlern, die nach dem Vergleichsprinzip arbeiten.

Im Logikteil wird der Digitalwert der Testspannung generiert und der Ausgang des Komparators beobachtet. Dieser Teil kann durch Hardware auf dem Baustein oder durch Rechnersoftware realisiert sein. Zur Generierung der Testspannung sind mehrere Strategien möglich:

- Bei der einfachen Abzählstrategie wird der Digitalwert, bei 0 beginnend, solange hochgezählt, bis der Komparatorausgang von logisch 0 auf 1 wechselt. Diese Strategie ist die langsamste.
- Beim Nachlaufverfahren wird — abhängig vom Komparatorausgang — vorwärts oder rückwärts gezählt, bis der richtige Wert erreicht ist. Dies Verfahren ist sehr gut geeignet bei langsam veränderlicher Meßgröße, in diesem Fall liegt praktisch immer der richtige Digitalwert am Ausgang an.
- Beim Intervallschachtelungsverfahren (sukzessive Approximation) werden, beim höchstwertigen beginnend, alle Bits nacheinander abgetestet und je nach Komparatorreaktion im Digitalwert auf 0 oder 1 gesetzt. Der Meßwert wird durch die fortgesetzte Halbierung des Intervalls sehr schnell erreicht. Voraussetzung ist, daß die Meßspannung während der Intervallschachtelung einigermaßen konstant bleibt. Dies wird im allgemeinen durch eine vorgesetzte ‘Sample-and-Hold’-Schaltung erreicht, die auf ein Startsignal hin den Meßwert abtastet (sample) und dann festhält (hold).

Die meisten der auf PC-Karten verwendeten Analog/Digital-Wandler arbeiten nach diesem letztgenannten Verfahren. Ein Ansteuerprogramm muß zunächst eine A/D-Wandlung starten, sodann eine festgelegte Mindestzeit (Wandlungszeit) warten, dann den Digitalwert einlesen. Die Wandlungszeit liegt, abhängig vom A/D-Wandler, zwischen 1 und 100 μsec. Häufig ist dem A/D-Wandler ein Analogmultiplexer vorgesetzt, der zwischen mehreren Meßstellen umschalten kann, dann muß das Programm zuallererst den richtigen Kanal anwählen. Eine Prozedur (AD) für einen A/D-Wandler, der nach dem Prinzip der sukzessiven Approximation arbeitet, findet sich im Modul SM.

Die folgenden Beispielprozeduren sollen die drei beschriebenen Strategien noch etwas näher erläutern. Die Prozeduren arbeiten mit einem A/D-Wandler zusammen, bei dem der Logikteil durch Rechnersoftware realisiert wird.

Zunächst die allgemeinen Deklarationen und die D/A-Wandler-Prozedur — der 12-Bit-D/A-Wandler wird mit 2 aufeinanderfolgenden Adressen (LowByte⇒daAdr, HighByte⇒daAdr+1) angesprochen:

```
FROM SYSTEM IMPORT In, Out;

CONST
  base = 390H;          (* base address of card               *)
  adMux = base + 8;     (* multiplexer address                *)
  adComp = base + 9;    (* comparator address                 *)
  gt = 0;               (* bit which indicates ‘greater than’ *)

PROCEDURE DA(daChannel, daValue : CARDINAL);
  VAR
    daAdr : CARDINAL;
  BEGIN
    daAdr := base + daChannel*2;
    Out(daAdr, SHORTCARD(daValue MOD 256));
    Out(daAdr + 1, SHORTCARD(daValue DIV 256));
  END DA;
```

Die erste Prozedur AD inkrementiert den Digitalwert, bis der Komparator Erfolg meldet oder das Ende des 12-Bit-Bereichs erreicht ist (einfaches Abzählprinzip):

```
PROCEDURE AD(adChannel : SHORTCARD) : CARDINAL;
  VAR
    adValue : CARDINAL;
  BEGIN
    Out(adMux, adChannel);     (* select multiplexer channel *)
    adValue:=0;
    REPEAT
      DA(0, adValue);
      INC(adValue);
    UNTIL (gt IN BITSET(In(adComp))) OR (adValue=4095);
    RETURN(adValue);
  END AD;
```

Die zweite Variante verwendet das Nachlaufverfahren, es wird — abhängig vom Komparatorausgang — dekrementiert und/oder inkrementiert. Der gemessene Digitalwert muß beim nächsten Aufruf wieder zur Verfügung stehen, daher als statische Variable außerhalb der Prozedur deklariert werden. Bei der Initialisierung wird er zweckmäßigerweise mit einem Schätzwert vorbesetzt.

```
VAR
  adValue : CARDINAL;

PROCEDURE AD(adChannel : SHORTCARD) : CARDINAL;
  BEGIN
    Out(adMux, adChannel);     (* select multiplexer channel *)
```

```
    WHILE (gt IN BITSET(In(adComp))) AND (adValue>1) DO
      DEC(adValue);            (* decrement while greater    *)
      DA(0, adValue);
    END;
    WHILE ~(gt IN BITSET(In(adComp))) AND (adValue<4095) DO
      INC(adValue);            (* increment while below      *)
      DA(0, adValue);
    END;
    RETURN(adValue);
  END AD;
```

Beim Verfahren der sukzessiven Approximation werden in einer `FOR`-Schleife nacheinander die einzelnen Bits abgetestet. Man beginnt beim höchstwertigen, durch `INCL` wird ein Bit auf 1 gesetzt, falls der Digitalwert zu groß geworden ist, wird es mit `EXCL` wieder auf 0 zurückgesetzt.

```
PROCEDURE AD(adChannel : SHORTCARD) : CARDINAL;
  VAR
    adValue, i : CARDINAL;
  BEGIN
    Out(adMux, adChannel);     (* select multiplexer channel *)
    adValue:=0;
    FOR i:=11 TO 0 BY -1 DO
      INCL(BITSET(adValue, i));
      DA(0, adValue);
      IF (gt IN BITSET(In(adComp))) THEN EXCL(BITSET(adValue), i) END;
    END;
    RETURN(adValue);
  END AD;
```

In den obigen Formulierungen liefern die beiden ersten Prozeduren einen Digitalwert, der den Analogwert von oben her nähert, die dritte Prozedur dagegen die Näherung von unten her. Wo nötig, läßt sich dies leicht ändern.

6.2.4 Spannungs-Frequenz-Wandlung

Bei diesem Verfahren wird die Meßspannung in eine Wechselspannung oder eine Impulsfolge umgesetzt, deren Frequenz zur Meßgröße proportional ist. Dies leisten spezielle ICs, Spannungs/Frequenz-Wandler. Die Spannungsmessung ist damit in eine Frequenzmessung transformiert. Die Frequenz wird mit einem Zähler gemessen, der auf eine feste Zählzeit eingestellt wird, fortlaufend mißt und die jeweiligen Meßwerte zum Auslesen in einem ‘Latch’ zwischenspeichert (Ratemeter). Prozedurbeispiele für die Initialisierung und das Auslesen eines so betriebenen Zählerbausteins enthält der Modul Interface.

Der Vorteil des Verfahrens liegt u. a. darin, daß es integrierend ist, damit sehr rauschunempfindlich und wenig störanfällig. Allerdings zählt es zu den langsameren A/D-Wandlungsprinzipien, Zählzeiten von

0.01 – 1 sec sind gebräuchlich. Ein weiterer Vorteil — oft wesentlich — ist die einfach durchzuführende Potentialtrennung zwischen Meßstelle und Rechner (nächster Abschnitt).

6.3 Potentialtrennung

Das Arbeiten mit analogen Größen erfordert ein stabiles und vor allem einheitliches Referenzpotential im gesamten Meßsystem (“Meßerde”). Die Problematik wird deutlich, wenn man sich klarmacht, daß bei 12 Bit Auflösung und einem Spannungsbereich $0\cdots5$ V einem Bit eine Spannung von etwa 1 mV entspricht. Störspannungen in dieser Größenordnung lassen sich in einem größeren System oft nicht verhindern. Ein Ausweg, der Störungen meist abhilft, ist die elektrische Trennung von Meß- bzw. Steuerungssystem und Datenverarbeitung. Einheitliche Bezugspotentiale brauchen dann nur noch in den Teilsystemen vorhanden zu sein.

Informationen zwischen den Teilsystemen werden am einfachsten auf optischem Wege übertragen. Dies kann durch Optokoppler[3] oder — bei höheren Anforderungen, z. B. in einer Hochspannungsumgebung — mit Lichtleiterverbindungen realisiert werden. Für die optische Ankopplung sind insbesondere Bausteine und Geräte geeignet, die über wenige Leitungen angebunden werden können, da ansonsten der Aufwand hoch wird. Man verwendet daher für die analoge Ankopplung in einem gestörten Bereich Spannungs/Frequenz-Wandler, die sehr nahe an der Meßstelle angeordnet sind und ihre Meßfrequenz optisch an einen Zähler im Rechner weitergeben bzw. seriell ansteuerbare D/A-Wandler, die vom Rechner aus optisch angesteuert werden.

6.4 Digitale Regelung

Ein wichtiges Anwendungsgebiet für A/D- und D/A-Wandler ist der Bereich der Regelungstechnik. Das grundlegende Prinzip, nach dem alle Regler — analoge wie digitale — arbeiten, ist das des geschlossenen Regelkreises (Abb. 12).

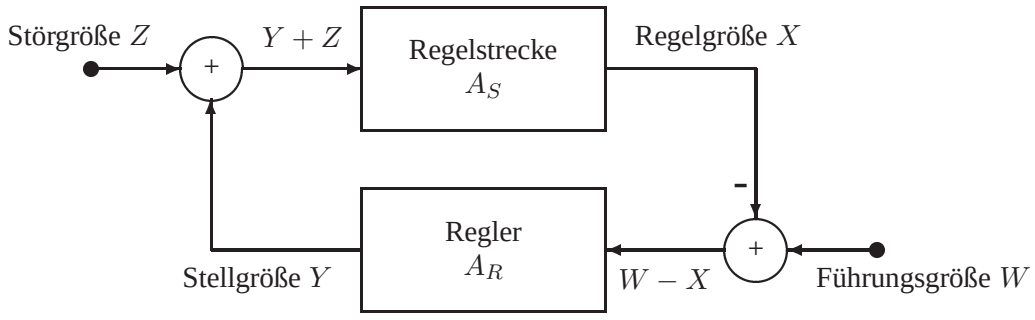


Abbildung 12: Regelkreis

Die **Regelstrecke** ist das Gerät, das geregelt werden soll, z. B. ein Ofen oder Kryostat. Sie wird be-

[3] Optokoppler sind Schaltkreise, die eine Lumineszenzdiode und eine Photodiode — elektrisch getrennt, optisch gekoppelt — enthalten. Die Isolationsspannungen liegen im Bereich von einigen hundert bis etwa 2000 Volt.

schrieben durch ihre Übertragungsfunktion A_S, die im Idealfall eine Konstante ist (lineare Regelstrecke).

Die **Regelgröße** ist der physikalische Parameter, der geregelt werden soll, z. B. die aktuelle Temperatur (Ist-Temperatur, Ist-Wert).

Die **Führungsgröße** ist der Vorgabewert für die Regelgröße (Soll-Temperatur, Soll-Wert).

Der **Regler** ist ein Verstärker im ganz allgemeinen Sinne mit bestimmten Eigenschaften, die durch die Übertragungsfunktion A_R beschrieben werden. Beim **Zwei-Punkt-Regler** ist die Übertragungsfunktion eine Stufenfunktion mit Hysterese (Bimetallthermostat o. ä.), beim **Proportionalregler** eine Konstante, die Verstärkung des Reglers.

Die **Stellgröße** ist die physikalische Größe, mit der die Strecke betrieben wird (Heizleistung, -strom, -spannung).

In der **Störgröße** sind die von außen einwirkenden Störungen zusammengefaßt (Schwankungen der Umgebungstemperatur, zusätzliche Wärmezufuhr durch Lichteinstrahlung o. ä.).

Idealisiert gilt bei linearer Regelstrecke für einen Proportionalregler

für die Stellgröße

$$Y = A_R \cdot (W - X),$$

für die Regelgröße

$$X = A_S \cdot (Y + Z)$$

und damit für die Abhängigkeit der Regelgröße von Führungsgröße und Störung

$$X = \frac{A_R A_S}{1 + A_R A_S} \cdot W + \frac{A_S}{1 + A_R A_S} \cdot Z.$$

Um den Einfluß der Störung Z gering zu halten, muß man A_R sehr groß machen. Dies hat dann Nachteile, wenn Regelstrecke und Regler sich nicht so ideal verhalten wie angenommen. Beispielsweise führen die immer vorhandenen Verzögerungszeiten in Regler und Regelstrecke dazu, daß die Regelung beim Überschreiten einer bestimmten Gesamtverstärkung schwingt. Ein Ausweg ist, die reine Proportionalregelung dadurch zu erweitern, daß man die Vorgeschichte und den Trend mit berücksichtigt. Zum Proportionalteil $A_R \cdot (W - X)$ wird ein Integralteil

$$A_I \cdot \frac{1}{\tau_I} \cdot \int_{-\infty}^{t_0} (W(t) - X(t)) \cdot \exp \frac{t - t_0}{\tau_I} \cdot dt$$

und ein Differentialteil

$$A_D \cdot \tau_D \cdot \frac{d(W - X)}{dt}$$

hinzugenommen (**PID-Regler**).

PID-Regler sind im analogen Bereich heute Stand der Technik. Ein Analogregler kann als Verstärker im engeren Sinne angesehen werden, bei dem die Verstärkung, sowie Integral- und Differentialanteil variiert werden können.

Beim digitalen Regler wird die gemessene Regelgröße in einen Digitalwert umgesetzt (A/D-Wandler), die Führungsgröße liegt digital als Konstante, Funktion oder Tabelle vor und der eigentliche Regelalgorithmus ist durch ein Programm realisiert. Die Stellgröße wird durch einen D/A-Wandler wieder analog gemacht und — nach entsprechender Verstärkung — der Regelstrecke zugeführt. Wichtig ist eine sinnvolle Diskretisierung, der Zeittakt dafür kann intern (Timer) oder extern (Meßtakt eines Digitalmultimeters) vorgegeben werden.

Der Vorteil des Analogreglers ist sicher im geringeren Aufwand und damit auch geringeren Preis zu sehen; wo dies keine allzu große Rolle spielt, sprechen einige wesentliche Punkte für die Verwendung von digitalen Reglern (Aufzählung ohne Anspruch auf Vollständigkeit):

- Die Führungsgröße kann eine beliebige zeitliche Funktion sein. So lassen sich in Öfen (Kristallzucht) sehr komplexe Temperaturprogramme realisieren.
- Störgrößen (Umgebungstemperatur etc.) können separat gemessen und in der richtigen Weise berücksichtigt werden, bevor ihr Einfluß — zeitlich verzögert — in der Regelgröße zu sehen ist.
- Die Übertragungsfunktion darf auch sehr kompliziert sein, den Rechner stört das nicht. Insbesondere lassen sich Nichtlinearitäten der Regelstrecke berücksichtigen und Grenzbedingungen für die Stellgröße oder deren Änderung festlegen.
- Driftprobleme spielen nur noch in den Wandlern eine Rolle, nicht mehr im Reglerteil.
- Die Übertragungsfunktion kann während der Regelung geändert werden, es lassen sich Regelalgorithmen konstruieren, die sich während der Regelung an die Eigenschaften der Regelstrecke anpassen (adaptive Regler).

7 Die serielle Schnittstelle

Serielle Verbindungen sind die vom Leitungsaufwand her einfachsten genormten 2-Punkt-Verbindungen in der DV-Technik. Ursprünglich sind sie für langsame, zeichenorientierte Datenübertragung zwischen Fernschreibern oder zwischen Rechner und Terminal konzipiert. Am PC ist meist zumindest eine serielle Schnittstelle vorhanden, an die Peripheriegeräte wie Plotter, Drucker, Maus, Datenerfassungsgeräte oder auch ein anderer Rechner angeschlossen werden können.

7.1 Grundlagen und Schnittstellennorm

Damit Geräte verschiedener Hersteller ohne weitere Anpassungsarbeit miteinander kommunizieren können, müssen alle relevanten Parameter herstellerübergreifend festgelegt sein. Für serielle Verbindungen sind mehrere Normen definiert, die sich in der Hardwareauslegung unterscheiden. Je nach Norm sind unterschiedliche Maximalentfernungen und Maximalübertragungsgeschwindigkeiten möglich. In der zur Zeit gebräuchlichsten Norm (USA: RS 232 C, BRD: DIN 66020, 66022, Europa: CCITT V24)[4], die auch bei der PC-Schnittstelle verwendet wird, sind unter anderem die folgenden Parameter vereinbart:

- Der High-Pegel einer Leitung liegt zwischen +3 und +15 Volt, der Low-Pegel zwischen -3 und -15 Volt.
- Daten werden in negativer Logik (1 = Low-Pegel), Steuersignale in positiver Logik (True, aktiv = High-Pegel) übertragen.
- Der Ruhezustand der Datenleitung ist logisch 1. Bei der asynchronen Übertragung werden Zeichen einzeln übertragen, die einzelnen Bits eines Zeichens in Folge. Jedes Bit hat die gleiche zeitliche Länge. Die Übertragung wird durch ein auf logisch 0 gesetztes Bit, das Startbit, eingeleitet, dann folgen die Datenbits in steigender Wertigkeit, danach eventuell ein Paritätsbit P, am Ende mindestens 1–2 Bits logisch 1 (Stopbits, = Ruhezustand der Leitung).

  Die Zeichenlänge (Wortlänge) n liegt zwischen 5 (Fernschreiber) und 8 Bit (8 Bit ASCII[5], IBM-Zeichensatz).
- Die Übertragungsfrequenz, das ist die reziproke ‘Bit-Zeit’, muß zwischen Sender und Empfänger vereinbart werden, damit sich der Empfänger nach dem Startbit auf die Bitfolge synchronisieren kann. Üblich sind $75 \cdot 2^N$ bit/sec mit $N = 0 \cdots 9$, d. h. 75, 150, 300, ...9600, 19200, 38400 bit/sec, in Sonderfällen werden aber auch andere Übertragungsgeschwindigkeiten eingestellt[6].

[4] Die Nummern der Normblätter werden häufig auch zur Bezeichnung der Schnittstelle verwendet: RS-232-Interface, V24-Schnittstelle.

[5] ASCII = American Standard Code for Information Interchange

[6] Für bit/sec ist die Einheit baud gebräuchlich, die Übertragungsgeschwindigkeit wird oft als Baud-Rate bezeichnet.

- Neben den beiden Datenleitungen für Senden (TXD) und Empfangen (RXD) sind einige Steuerleitungen definiert, die anzeigen, ob das betreffende Gerät eingeschaltet ist (DSR, DTR), empfangsbereit ist (CTS, RTS), die Leitung in Ordnung ist (DCD) oder ein Anruf angekommen ist (RI). Ein Teil dieser Leitungen ist für den MODEM[7]-Betrieb und damit die Datenfernübertragung gedacht, im lokalen Betrieb können sie zur Datenflußsteuerung (Quittungsbetrieb) benutzt werden.

- Als Steckverbindungen werden 25-polige D-Stecker und -Buchsen benutzt, aus Platzgründen im PC vielfach auch 9-polige, auf Spezialkarten teilweise noch kleinere. Die Steckerbelegung ist davon abhängig, ob es sich um ein Datenendgerät (DTE, Data Terminal Equipment) oder um ein MODEM-artiges Gerät (DCE, Date Communication Equipment) handelt. Die Steckerbelegung für DTE-Geräte ist in Tabelle 3 angegeben. Bei DCE-Geräten ist die Belegung komplementär.

Pin 9-pol.	Pin 25-p.	Ein/ Aus	Signal
1	8	E	DCD, Data Carrier Detected, Leitung in Ordnung
2	3	E	RXD, Receive Data, Dateneingang
3	2	A	TXD, Transmit Data, Datenausgang
4	20	A	DTR, Data Terminal Ready, Terminal betriebsbereit
5	7		GND, Signal Ground, Masseanschluß
6	6	E	DSR, Data Set Ready, externes Gerät betriebsbereit
7	4	A	RTS, Request To Send, Sendeanforderung zum externen Gerät
8	5	E	CTS, Clear To Send, Sendeanforderung vom externen Gerät
9	22	E	RI, Ring Indicator, Wählsignal vom MODEM

Tabelle 3: Steckerbelegung der seriellen Schnittstelle im PC

Verbindungskabel zwischen einem DTE- und einem DCE-Gerät verbinden jeweils gleiche Pin-Nummern, bei Kabeln zwischen 2 DTE-Geräten (z. B. zwischen PC und Plotter) müssen die sich entsprechenden Pins (RXD↔TXD, CTS↔RTS) jeweils ‘gekreuzt’ verbunden werden. Oft genügen zum Anschluß die Datenleitungen und Masse, teilweise werden die Steuerleitungen zur Datenflußkontrolle benutzt.

7.2 Der serielle Portbaustein 8250

Die Umsetzung der Daten zwischen Rechner-Bus und seriellem Anschluß besorgen eigens dafür konzipierte Bausteine, im PC-XT ist das der 8250, beim AT der 16450[8]. Der Baustein enthält Sendeteil,

[7] MODEMs (Abkürzung für MOdulator/DEModulator) sind Geräte, die die seriellen Rechtecksignale auf geeignete Trägerfrequenzen umsetzen, sodaß eine Übertragung über weite Strecken im Telefonnetz möglich wird

[8] Bei neueren PCs werden oft auch höherintegrierte VLSI-Chips (Very Large Scale Integration) eingesetzt, die diese Funktionen mit beinhalten.

Empfangsteil, Steuerleitungslogik, Interruptlogik und Taktgenerator. Die Daten-, Kommando- und Statusregister sind über insgesamt 7 aufeinanderfolgende Ein/Ausgabeadressen zugänglich.

Über die Basisadresse werden die beiden Register adressiert, die Sende- und Empfangsdaten halten. Beim Lesen von dieser Adresse greift man auf das zuletzt empfangene Zeichen zu, beim Schreiben auf diese Adresse wird das nächste zu sendende Zeichen im Baustein abgelegt. Vor einem Zugriff auf die Datenregister sollte das Line Status Register geprüft werden.

Das Interrupt Enable Register (IER, Basisadresse+1) legt fest, welche Interruptbedingungen des Bausteins zugelassen werden:

BIT	7	6	5	4	3	2	1	0
IER(+1):	0	0	0	0	EMSI	ELSI	ETBEI	ERBFI

0 ERBFI, Enable Receiver Buffer Full Interrupt, Interrupt bei jedem empfangenen Zeichen.

1 ETBEI, Enable Transmitter Buffer Empty Interrupt, I. nach jedem gesendeten Zeichen.

2 ELSI, Enable Line Status Interrupt, I. bei fehlerhaften Empfangsdaten (Parität usw.).

3 EMSI, Enable Modem Status Interrupt, I. aufgrund des Status der Steuerleitungen.

Durch ein auf 1 gesetztes Bit wird der entsprechende Interrupt aktiviert, 0 deaktiviert.

Das Interrupt Identification Register (IIR, Basisadresse+2, nur lesbar) dient dazu, die Art des Interrupts festzustellen:

BIT	7	6	5	4	3	2	1	0
IIR(+2):	0	0	0	0	0	IIB1	IIB0	NPI

0 NPI, No Pending Interrupt, “1” zeigt an, daß kein Interrupt anliegt, “0”, daß eine Interruptsituation auf Bearbeitung wartet.

2,1 Diese beiden Bits identifizieren den Interrupt (Int. Identification Bits). Es bedeutet:

- 3: Line Status Interrupt, fehlerhafte Empfangsdaten,
- 2: Empfangsdaten vorhanden,
- 1: Senderegister leer,
- 0: Statusänderung der Steuerleitungen.

Über das Line Control Register (LCR, Basisadr+3) wird die Betriebsart des Bausteins festgelegt:

BIT	7	6	5	4	3	2	1	0
LCR(+3):	DLAB	Break	STP	EPS	PEN	STB	WLS1	WLS0

1,0 WLS, Word Length Select, diese beiden Bits legen die Zeichenlänge fest (Inhalt dieser beiden Bits plus 5).

2 STB, Stop Bits, Zahl der Stop Bits wird auf 1 (STB=0) oder 2 (STB=1) festgelegt.

3 PEN, Parity Enable, PEN=1 veranlaßt die Generation und die Überprüfung eines Paritätsbits.

4 EPS, Even Parity Select, ordnet gerade (EPS=1) oder ungerade Parität (EPS=0) an (wenn PEN=1).

5 STP, Stick Parity, festes Paritätsbit, wenn STP=1. Das Paritätsbit wird 1, wenn PEN=1 und EPS=0, dagegen 0, wenn PEN=1 und EPS=1.

6 Break, setzt die Sendeleitung auf logisch 0. Dies kann dazu benutzt werden, dem angeschlossenen Gerät einfache Mitteilungen zu machen, z. B. das Umschalten der Baudrate zu veranlassen.

7 DLAB, Divisor Latch Access Bit, Zugriff auf das Divisorregister (dies ist ein 16-Bit-Register) des Taktgenerators. Ist diese Bit auf 1 gesetzt, kann über die Basisadresse (Low-Byte) und Basisadresse+1 (High-Byte) das Divisorregister des internen Taktgenerators gelesen und beschrieben werden. Die Baudrate ist der Quotient aus 115200 und dem Inhalt des Divisorregisters.

Das Modem Control Register (MCR, Basisadr+4) kontrolliert die vom Baustein ausgehenden Steuerleitungen:

BIT	7	6	5	4	3	2	1	0
MCR(+4):	0	0	0	Loop	Out2	Out1	RTS	DTR

0 DTR, Data Terminal Ready, und

1 RTS, Request To Send, programmieren die vom PC ausgehenden Steuerleitungen (“1”: aktiv, “0”: inaktiv).

2 Out1, eine zusätzliche Ausgangsleitung, wird im PC nicht benutzt.

3 Out2 wird dazu benutzt, die Verbindung des 8250-Interrupt-Ausgangs mit der entsprechenden IRQ-Leitung im PC zu schalten (“0”: kein Interrupt-Zugriff, das ist die BIOS-Voreinstellung, “1”: Interrupt-Zugriff möglich).

4 Loop = 1 schließt zu Testzwecken die Ausgangssignale des Bausteins an die Eingänge an.

Das Line Status Register (LSR, Basisadresse+5, nur lesbar) gibt Auskunft über den Status der Datenregister:

BIT	7	6	5	4	3	2	1	0
LSR(+5):	0	TSRE	THRE	BI	FE	PE	OR	DR

0 DR, Data Ready, zeigt an, daß Daten zum Einlesen bereit sind. Das Bit wird auf 0 zurückgesetzt, wenn die anstehenden Daten gelesen sind.

1 OR, Overrun Error, Lesedaten wurden nicht rechtzeitig abgeholt und von neuen Daten überschrieben.

2 PE, Parity Error, fehlerhafte Parität bei den Empfangsdaten.

3 FE, Framing Error, Fehler in der Wortlänge oder in den Stopbits.

4 BI, Break Interrupt, Empfangsleitung im “Break”-Zustand.

5 THRE, Transmitter Holding Register Empty, der Sendepuffer ist leer, ein neues Zeichen kann ausgegeben werden.

6 TSRE, Transmitter Shift Register Empty, das Sende-Shift-Register ist leer, d. h. das letzte Zeichen hat den Baustein verlassen.

Das Modem Status Register (MSR, Basisadr+6, nur lesbar) zeigt den Status der Steuerleitungseingänge an:

BIT	7	6	5	4	3	2	1	0
MSR(+6):	DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

0 DCTS, Delta Clear To Send,

1 DDSR, Delta Data Set Ready,

2 TERI, Trailing Edge Ring Indicator und

3 DDCD, Delta Data Carrier Detected zeigen Änderungen der Status-Leitungen an.

4 CTS, Clear To Send,

5 DSR, Data Set Ready,

6 RI, Ring Indicator und

7 DCD, Data Carrier Detected wiederspiegeln direkt den Status der Leitungen (“1”: aktiv, “0”: inaktiv).

7.3 Die Programmierung der seriellen Schnittstelle

Die Grundprogrammierung des Bausteins 8250 wird vom BIOS vorgenommen, dabei werden alle Parameter auf Standardwerte festgelegt. Will man andere Werte verwenden, kann die Einstellung mit dem DOS-Kommando **MODE** geändert werden. Wesentlich flexibler ist es jedoch, wenn das Programm, das mit der seriellen Schnittstelle arbeitet, die Grundprogrammierung des Bausteine selbst vornimmt. Es kann dann insbesondere die DOS-Beschränkung in der Baudrate (9600 baud) umgangen werden.

Zur Programmierung der Betriebsart wird das Line Control Register entsprechend gesetzt, mit DLAB=1 kann die Baudrate eingestellt werden (anschließendes DLAB=0 nicht vergessen). Soll im Interrupt-Betrieb gearbeitet werden, müssen die gewünschten Bits im Interrupt Enable Register auf 1 gesetzt werden. Nach dem üblichen Prozedere (Interrupt-Routine, Interrupt-Tabelle, Interrupt-Controller) wird dann die Interrupt-Leitung durch das Modem Control Register freigegeben (zusammen mit dem Setzen der Steuerleitungsausgänge). Die Interrupt-Routine muß die Art des Interrupts identifizieren können (Interrupt Identification Register) und dann die passende Reaktion aktivieren. Ein Beispiel für den Interrupt-Betrieb des Bausteins ist der Modul RS232 aus dem TechKit von JPI.

Soll die Schnittstelle nicht im Interrupt-, sondern im Nachfrage-Modus (Polling) betrieben werden, dann sind individuelle Routinen für Senden und Empfangen nötig. Die Senderoutine muß solange abwarten, bis der Sendepuffer leer ist (Line Status Register), dann kann ein Zeichen ausgegeben werden (Sendedatenregister, Basisadresse). Die Empfängerroutine muß zunächst nachsehen, ob ein neues Zeichen zum Einlesen bereitsteht (Line Status Register), dann das Zeichen lesen (Empfangsdatenregister, Basisadresse).

7.4 Quittungsbetrieb

Für Empfängergeräte, die Daten über eine serielle Leitung schneller empfangen, als diese im Gerät verarbeitet werden können (Drucker, Plotter), muß eine Möglichkeit vorgesehen werden, den Datenfluß zu steuern (Quittungsbetrieb, Handshake). Zwei Arten der Datenflußsteuerung werden überwiegend verwendet:

- Beim ‘Hardware-Handshake’ wird durch den Status der Steuerleitungen mitgeteilt, ob ein Gerät empfangsbereit ist. Die meisten Geräte benutzen zu diesem Zweck die CTS- bzw. RTS-Leitung (CTS-RTS-Handshake); werden andere Steuerleitungen benutzt, kann unter Umständen eine Umkonfiguration durch ein geeignetes Verbindungskabel notwendig werden (HP-Plotter). Das Sendergerät muß vor der Ausgabe jedes einzelnen Zeichens den Steuerleitungsstatus überprüfen und den ‘Aktiv’-Zustand der Handshake-Steuerleitung abwarten.

- Beim ‘Software-Handshake’ (auch XON-XOFF-Handshake) sind zwei Zeichen vereinbart, die vom Empfängergerät zum Stop und zur Wiederaufnahme der Übertragung ans Sendergerät geschickt werden. Gestoppt wird mit ‘XOFF’ (meist CTRL-S), wiedergestartet mit ‘XON’ (meist CTRL-Q). Das Sendergerät sollte insbesondere auf XOFF prompt reagieren, d. h. die Datenausgabe anhalten.

Bei der Übertragung von größeren Datenmengen zwischen Rechnern ist es oft sinnvoll, im Blockbetrieb zu arbeiten. Dabei bildet das Sendergerät nach einem vereinbarten Algorithmus einen Block aus einer

bestimmten Anzahl von Zeichen, der zusammen mit einer Prüfsumme verschickt wird. Der Empfänger prüft den Block anhand der Prüfsumme auf Richtigkeit und quittiert positiv oder negativ. Bei negativer Quittung wird der Block nochmals geschickt. Auf diese Weise läßt sich eine hohe Datensicherheit erreichen. Die meisten Datenübertragungsprogramme arbeiten mit solchen ‘Protokollen’ (Kermit, Crosstalk, X-Modem, . . .).

7.5 Andere Übertragungsnormen

Die RS 232 C-Schnittstellen-Norm ist relativ alt und für niedrige Übertragungsraten konzipiert. Laut Spezifikation ist sie für Leitungslängen bis zu 15 m und für Übertragungsraten bis zu 20 kbit/s ausgelegt. Neben der RS 232 C existieren zwei wesentlich leistungsfähigere Normen, die allerdings noch wenig verwendet werden.

Die Norm RS 423 A definiert eine unsymmetrische Schnittstelle, bei der die Datenübertragung über ein (mit dem Wellenwiderstand) abgeschlossenes Koaxialkabel erfolgt. Die maximale Übertragungsrate ist 300 kbit/s, die maximale Leitungslänge 600 m.

Die Norm RS 422 A benutzt symmetrische Leitungstreiber und -empfänger und abgeschlossene, verdrillte Zweidrahtleitungen. Die maximale Leitungslänge ist 1200 m, die maximale Übertragungsrate 2 Mbit/s (bei dann allerdings verringerter Leitungslänge von max. 60 m).

Noch höhere Übertragungsraten sind durch Lichtleiterverbindungen, größere Übertragungsstrecken durch Modulationsverfahren (Telefon-Modem) oder über die einschlägigen Postdienste (Datex-P) realisierbar.

Für alle Normen sind Treiberbausteine verfügbar, die eine Umsetzung des vom Schnittstellenbaustein generierten TTL-Signals auf die Norm-Signale vornehmen.

8 Der IEC-Bus

Die in der vorhergehenden Kapiteln beschriebenen Schnittstellen (parallel, seriell) werden im wesentlichen für 2-Punkt-Verbindungen benutzt (PC↔Plotter, PC↔Drucker). Diese Verbindungstechnik wird sehr aufwendig, wenn eine größere Anzahl von Meßgeräten eingebunden werden soll, da der PC für jedes Meßgerät eine individuelle Schnittstelle zur Verfügung stellen muß.

Im Gegensatz zu den 2-Punkt-Verbindungen zeichnen sich Bussysteme dadurch aus, daß eine Vielzahl von Geräten an eine Schnittstelle angeschlossen werden kann, und über den Bus eine Kommunikation mit dem PC wie auch zwischen den angeschlossenen Geräten möglich ist. Das Bussystem, das sich auf dem Meßgerätesektor seit einigen Jahren weitgehend durchgesetzt hat, ist der IEC-Bus. Dieser sehr gut genormte Schnittstellenbus[9] besteht aus 8 Daten-, 3 Handshake- und 5 allgemeinen Steuerleitungen (Abb. 13).

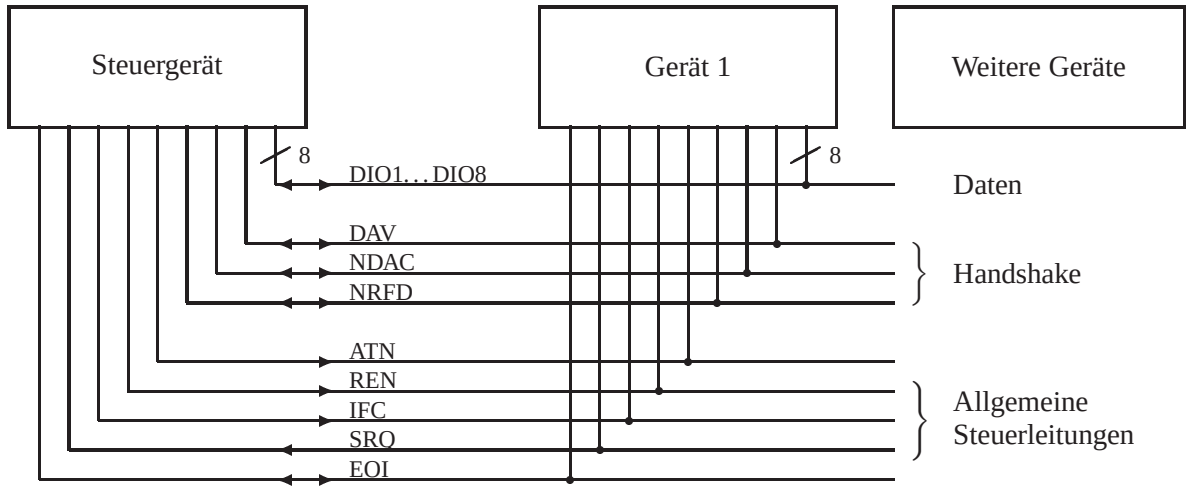


Abbildung 13: Die 16 Signalleitungen des IEC-Bus.

Der Bus wird in negativer Logik mit TTL-Spannungen betrieben, der aktive Leitungszustand bzw. logisch 1 entspricht somit ca. 0 V, inaktiv oder logisch 0 dagegen ca. 5 V. Die Signalausgänge sind als ‘offene Kollektor’ Ausgänge realisiert, durch diese ‘Wired-Or’-Verknüpfung wird sichergestellt, daß auch dann keine Buskonflikte auftreten, wenn mehrere Geräte gleichzeitig auf eine bestimmte Busleitung zugreifen. Andererseits kann ein einzelnes Gerät den Aktiv-Status einer Leitung erzwingen.

Gegenüber einem Rechnerbus weist der IEC-Bus einige Unterschiede auf:

Die Datenübertragung erfolgt asynchron, d. h. sie wird nicht — wie beim PC-Bus — durch ein zentrales Taktsignal synchronisiert, sondern durch ein Quittungsverfahren mit den 3 Handshakeleitungen. Das Signal für gültige Daten (DAV: Data Valid) wird vom jeweiligen Sprechergerät nur dann aktiviert, wenn alle Hörergeräte am Bus ihre Bereitschaft durch das deaktivierte NRFD-Signal (Not Ready For Data) anzeigen. Die Daten bleiben danach so lange gültig, bis alle Hörergeräte das NDAC-Signal (Not Data

[9] Die international gültigen Schnittstellen-Normen sind IEEE 488 (USA) und IEC 66.22 (Europa). Für den IEC-Bus sind auch die Bezeichnungen IEEE-Bus (sprich: ai-triple-i), HPIB (Hewlett Packard Interface Bus) und GPIB (General Purpose Interface Bus) gebräuchlich.

Accepted) deaktiviert und damit gemeldet haben, daß die Daten empfangen wurden. Die Datenübertragungsgeschwindigkeit richtet sich also dynamisch nach dem jeweils langsamsten aktiven Gerät am Bus. Dennoch sind — bei hinreichend schnellen Teilnehmern — maximale Übertragungsraten von ca. 1 MByte/sec möglich.

Die 8 Datenleitungen (DIO1...DIO8) werden sowohl für Daten als auch für Kommandos benutzt, die Unterscheidung erfolgt durch die Steuerleitung Attention (ATN). Nur ein Gerät am Bus — als Steuergerät (Controller) konfiguriert — darf die Attention-Leitung bedienen und somit Kommandos erteilen. Ein Teil der Kommandos dient dazu, andere Geräte am Bus als Sprecher- (Talker) oder Hörer-Geräte (Listener) zu adressieren. Durch diese Adressierkommandos wird eine Datenübertragung zwischen einem Sprecher und einem oder mehreren Hörern eingeleitet. Insgesamt sind 31 Hörer- und 31 Sprecheradressen in einem IEC-Bus-System möglich, die übrigen Kommandobytes sind für weitere Befehle reserviert. Unter anderem ist eine weitergehende Adressunterteilung über Sekundäradressen möglich.

Die Funktion der weiteren Steuerleitungen:

REN (Remote Enable) schaltet die lokale Bedienungsmöglichkeit der angeschlossenen Geräte ab.

IFC (Interface Clear) bringt das Schnittstellensystem in einen definierten Anfangszustand.

SRQ (Service Request) wird von einem angeschlossenen Gerät aktiviert, wenn es eine Bedienung wünscht (z. B. am Ende einer Messung zum Datentransfer). Die dazu nötige Unterbrechung wird vom Steuergerät veranlaßt, erfolgt daher nicht unbedingt prompt. Die Reaktion des Steuergeräts muß mit einer Umfrage (Serial Poll, Parallel Poll) beginnen, um festzustellen, welches Gerät die SRQ-Leitung aktiviert hat.

EOI (End Or Identify) hat zwei Funktionen: Zum einen zeigt der gerade aktive Sprecher damit das Ende des Datentransfers an. Zum anderen wird EOI zusammen mit ATN vom Controller benutzt, um eine Parallelabfrage (Parallel Poll) anzuordnen.

Für weitere technische Details sei an dieser Stelle auf die Literatur zum IEC-Bus verwiesen[10].

Die Datenübertragung auf dem IEC-Bus erfolgt ‘bit-parallel’ und ‘byte-seriell’. Hinsichtlich der Länge des Datenstroms und der Codierung der einzelnen Bytes bestehen keine Einschränkungen. Daher muß die Art der Datencodierung jeweils zwischen Talker- und Listener-Gerät vereinbart werden. Da aber bei praktisch allen Peripheriegeräten das Datenformat für den IEC-Bus nicht veränderbar ist, bedeutet ‘vereinbaren’, daß sich der PC als flexibelstes Gerät am Bus auf das jeweilige Datenformat einstellen muß. Zwei Codierungsarten werden überwiegend verwendet:

- Binäre Codierung dort, wo große Datenmengen möglichst schnell übertragen werden sollen. Einige Digitalspeicheroszilloskope (BBC-Metrawatt) übertragen ihre Daten auf diese Weise — bei einer Auflösung von 8 Bit wird für einen Datenpunkt nur 1 Byte benötigt.
- ASCII-Codierung dort, wo es eher auf Sicherheit und Verständlichkeit ankommt und die Geschwindigkeit nicht im Vordergrund steht. Die Daten werden als Text übertragen, somit ist eine sofortige Kontrolle möglich. Fast alle Digitalmultimeter benutzen diese Codierungsart sowohl für

[10] Eine ausführliche Darstellung findet sich in dem Buch “Grundlagen und Anwendung des IEC-Bus” von Georg Walz, erschienen im Markt & Technik Verlag.

ihre Programmierung als auch für die Meßdaten. Da jeweils nur ein Meßwert übertragen wird, werden keine hohen Geschwindigkeiten benötigt.

PC-Karten für den IEC-Bus enthalten im allgemeinen einen intelligenten Schnittstellenbaustein, der einen großen Teil des Busmanagements (insbesondere den Quittungsablauf) selbständig erledigt. Meist können die Karten sowohl als Controller (überwiegende Betriebsart im PC) wie auch als Talker/Listener konfiguriert werden.

Die Grundsoftware für die Karten ist entweder als ROM auf der Karte (Keithley) oder als resident ladbarer Softwaremodul (National Instruments und Nachbauten) vorhanden. In beiden Fällen ist es nötig, die Einbindung in eine Hochsprache über geeignete Assembler-Prozeduren vorzunehmen. Am Modul **IEC** für Keithley-Karten wird aufgezeigt, wie die Einbindung in Modula 2 im Falle einer ROM-Software erfolgen kann, der Modul **ib** für National-Instruments-Karten zeigt die Einbindung der von National Instruments gelieferten resident ladbaren Software.

Die National-Instruments-Software und der Modul **ib** sollten etwa nach dem folgenden Schema verwendet werden:

1 Zunächst muß das speicherresidente Treiberprogramm `GPIB.COM` konfiguriert werden. Mit dem Konfigurierungsprogramm `IBCONF.EXE` werden die Hardwareparameter für die über den IEC-Bus angeschlossenen Geräte festgelegt, insbesondere deren Adresse. Außerdem können Gerätenamen vergeben werden. Das Treiberprogramm wird mit einer '`DEVICE=`'-Zeile im `CONFIG.SYS` als residenter DOS-Gerätetreiber definiert. Nach jeder Änderung des Treiberprogramms muß daher auch neu gebootet werden.

2 Ein Programm, das auf ein IEC-Bus-Gerät zugreift, muß zur logischen Eröffnung der Verbindung den Gerätenamen `name` mit einer 'Handle' (`VAR`iable vom Typ `CARDINAL`) verbinden. Dazu dient die Prozedur `ib.FIND('name')`. Dies ist analog zum Öffnen einer Datei im Schreib/Lese-Zugriff. Alle weiteren Befehle für das IEC-Bus-Gerät benutzen die Handle zur logischen Adressierung. Die physikalische IEC-Bus-Adresse wird auf dieser Ebene nie verwendet.

3 Bei programmierbaren Geräten kann die Programmierung mit `ib.WRT(...)` erfolgen. Diese Prozedur adressiert das zur Handle gehörende IEC-Bus-Gerät als Listener und sendet dann den spezifizierten String-Parameter (`ARRAY OF CHAR`) als Datenfolge.

4 Die Übernahme von Meßdaten wird durch die Prozedur `ib.RD(...)` angeordnet. Das Gerät wird als Talker adressiert und sendet die Datenfolge, sobald der Controller die ATN-Leitung deaktiviert.

Einige Meßgeräte können nicht im Listener-Betrieb arbeiten, somit nicht mit `ib.WRT(...)` angesprochen werden. Um dennoch — dann natürlich sehr eingeschränkt — unterschiedliche Funktionen im Gerät aktivieren zu können, wird oft ein weiteres Kommandobyte, die Sekundäradresse benutzt. Diese wird beim Aufruf von `ib.RD(...)` automatisch nach der Talker-Adresse übertragen, wenn dies durch `ib.SAD(...)` veranlaßt wurde.