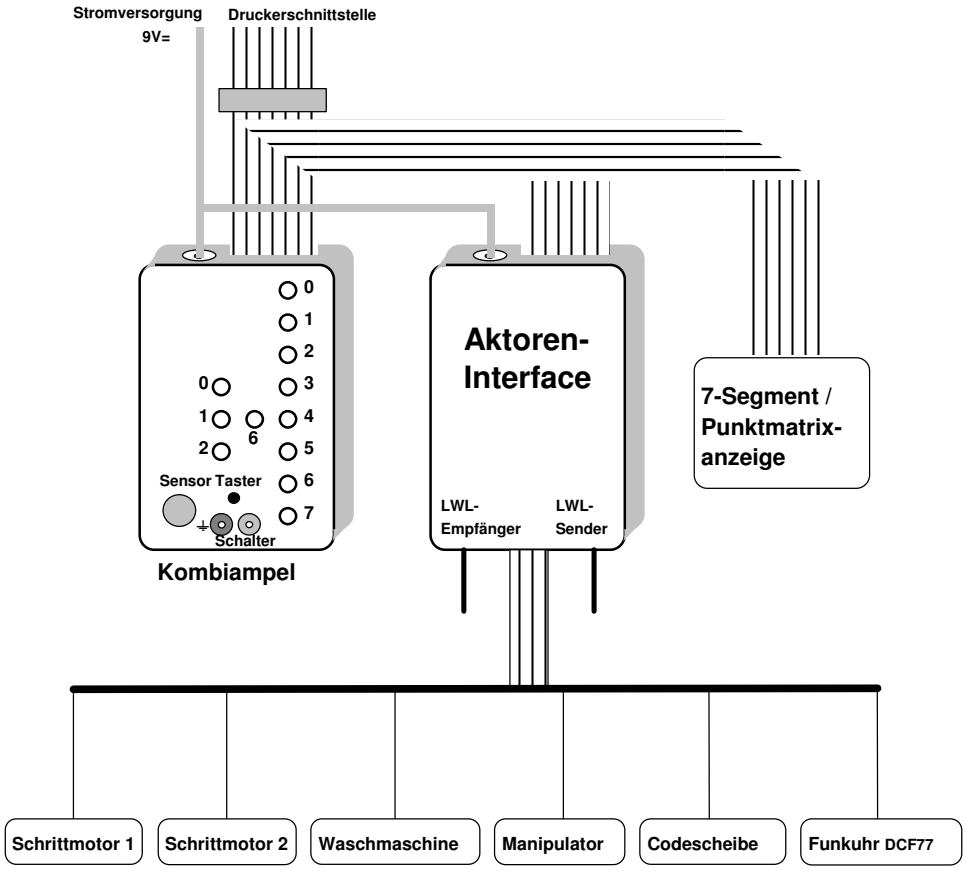
Aktoreninterface

für die

Kombiampel



Inhaltsverzeichnis

Abbildungen

Vorwort

Bei der Umsetzung des Themas Messen, Steuern, Regeln (MSR) im Unterricht spielen Aktoren eine wesentliche Rolle. Solche Aktoren sind z.B. Lampen oder Motoren.
Das Aktoreninterface-System besteht aus einem Interface für den Druckerport des PCs und einer Reihe von Modellen zu diesem Thema, einer Waschmaschine, einem Manipulator, einem Codescheibenleser, einer Schlitzscheibensteuerung und einem Aufzug. Es enthält einen Lichtwellensender und -empfänger und ist in der Lage, eine Punktmatrixanzeige zu betreiben.
Das Interface ist ein Bindeglied zwischen Rechner und Aktoren bzw. Sensoren. Es steuert die Schrittmotoren der Modelle, den Elektromagneten für den Manipulator und es schaltet die Leuchtdioden für den Lichtwellensender oder die Punktmatrixanzeige.
Mit dem Aktoreninterface wurde ein System konzipiert, das, wie bei der Kombiampel, das Prinzip von Prozeßdatenverarbeitung (PDV) mit kleinem Budget pflegt. So ist es finanziell realistisch, es im Rahmen von Partnerarbeit allen Schülern in einem Rechnerraum einer Schule zur Verfügung zu stellen. Es ermöglicht so eine problemorientierte Lernhandlung. Dies wird besonders deutlich bei den Schrittmotorprojekten. Das Aktoreninterface-System wird mit dem Prozeß-PASCAL der Kombiampel programmiert. Die Sprache Logo ist für Aufgaben, bei denen Sensoren in Echtzeit ausgewertet werden müssen ungeeignet.
Bei dem Aktorensystem wird die Kombiampel grundsätzlich als Anzeigegerät für den Zustand der Aktoren mitbenutzt. Inhaltlich werden somit die mit der Kombiampel gemachten Erfahrungen weiter vertieft. Um diese Unterrichtsreihe auch ohne Kombiampel durchführen zu können, wurde das System um eine LED-Option erweitert, welche die Anzeigeaufgaben übernimmt.
Lernende, die z.B. im Umgang mit der Kombiampel wichtige Elemente der PDV erfahren haben, können mit dem hier vorgestellten Gerät Datenfernübertragung mit einem Lichtwellenleiter kennenlernen und betreiben. Durch den Gebrauch von Aktoren z.B. für Motorsteuerungen können weitere einfache Aufgaben der PDV problemorientiert gelöst werden. Durch die Verknüpfung von Sensoren und Aktoren können auch komplexere Aufgaben der PDV in Gang gesetzt werden.
Zusätzlich zu den digital arbeitenden Aktoren und Sensoren ist die Einbeziehung analoger Meßdaten sinnvoll, die dann Einfluß auf diese Aktoren nehmen können. Ein 8 kanaliger 12Bit-AD-Wandler 0-5V kann optional eingesetzt werden.

V. Ludwig, B. John

1 Betrieb des Aktoreninterfaces

Es empfiehlt sich folgender Aufbau: An das Flachbandkabel der Kombiampel wird ein 25 poliger Sub-D-Stecker (female) angepreßt (vergl. Aufbauanleitung im Anhang). Daran wird nun der 25 polige Stecker des Flachbandkabels des Aktoreninterfaces angeschlossen. Beide Geräte erhalten eine eigene Stromversorgung aus einem eigenen Netzteil[1]. Die Kombiampel wird dazu geschaltet, weil bei den Projekten stets auf die Anzeige der Kombiampel aus Gründen der Anschaulichkeit zurückgegriffen wird.

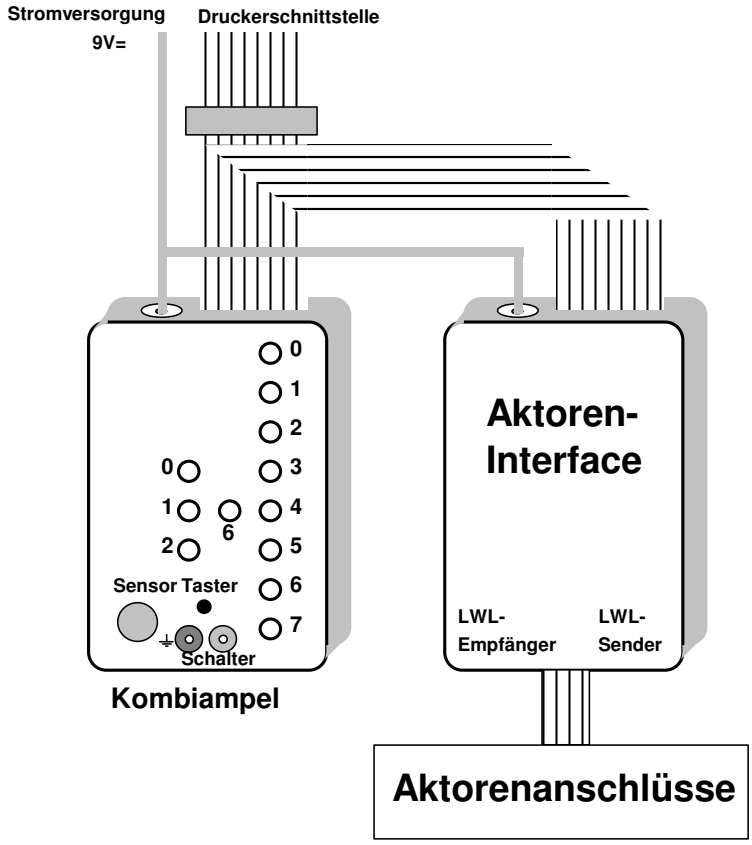


Abbildung 1-1: Betrieb mit Kombiampel

1 Wer ein starkes Netzteil etwa 9 V 1,5 A verwendet kann sich eine Kabelverzweigung herstellen, und beide Geräte damit betreiben.

2 Kurzanleitung für Pascal-Einsteiger

2.1 Bedienung des Pascal-Systems:

1. Turbo-Pascal starten

2. Zunächst wird ein neuer Programmtext angelegt.
   Dazu mit der Taste F10 in die obere Menüzeile wechseln.
   Mit der Taste D in das Dateimenü gehen und mit der Taste N eine neue Datei erzeugen.

3. Es erscheint ein leeres Textfenster. Hier wird der Programmtext eingegeben.

4. Mit der Taste F2 wird der Text gespeichert.
   Im Eingabefenster, bei dem blinkenden Cursor, den Namen des Programms eingeben und mit der Eingabe- bzw. Returntaste bestätigen.

5. Das Programm wird mit der Taste F9 gestartet.

6. Falls das Programm Fehler enthalten sollte, zeigt der Rechner mit dem Cursor die fehlerhafte Stelle auf dem Schirm an und gibt eine Fehlermeldung aus.
   Der Fehler wird nun korrigiert und das Programm erneut gestartet.

2.2 Programmstruktur:

Folgende beiden Zeilen bilden den Programmkopf:
Jedes PASCAL-Programm beginnt mit dem Schlüsselwort `PROGRAM`, dahinter folgt der Name des Programms.
`USES Komampel;` bindet einfache Befehle zur Steuerung der Kombiampel und des Aktoreninterfaces ein.

Mögliche Variable müssen unter dem Programmkopf definiert werden. Dazu gehört das Schlüsselwort VAR. (Beispiel: VAR Zahl : INTEGER;).

Ein Programmblock fängt mit dem Schlüsselwort `BEGIN` an und endet mit dem Schlüsselwort `END`.

Jetzt können bei Bedarf Prozeduren definiert werden, die durch Ausführen einzelner Anweisungen verschiedene Teilprobleme lösen.

Im Hauptprogramm werden die Prozeduren aufgerufen.

Das Zeichen "." beendet das PASCAL-Programm.

Die Anweisung `Lauflicht_ein;` stellt die Kombiampel auf Lauflichtbetrieb.

Jede Anweisung wird mit einem Semikolon abgeschlossen.

```
Program Beispiel;             (* Programm- *)
uses komampel;                (* kopf      *)

VAR
  Definition von Variablen;

procedure Prozedurname;       (* Prozeduren *)
  begin
    Anweisungen der Prozedur;
  end;

weitere Prozeduren...

BEGIN                         (* Hauptprogramm *)
  Lauflicht_ein;
  Anweisungen des Hauptprogramms;
END.
```

3 Vorübungen zur Steuerung[2]

Zur Ausgabe von Werten aus dem Rechner zur Kombiampel oder zum Modell gibt es zwei Arten von Anweisungen, "ausgeben (<Wert>)" und "binaus (<Bitmuster>);". Je nach Problemstellung ist es sinnvoll, die eine oder die andere Art auszuwählen.

3.1 Steuerung mit "ausgeben (<Wert>);"

Die Anweisung `Ausgeben(1);` schaltet eine Leuchtdiode ein.

```
PROGRAM Leuchtdiode;
USES Komampel;
BEGIN
      Lauflicht_ein;
      ausgeben (1);
END.
```

Aufgabe 1: Gib das Programm ein, speichere es unter dem Namen LEUCHT.PAS und starte es mit dem Menüpunkt RUN bzw. START.

Aufgabe 2: Versuche andere Leuchtdioden einzuschalten indem du die Zahl 1 im Programm durch eine andere Zahl ersetzt. Notiere den und die Wirkung.

Aufgabe 3: Schreibe 8 Programme, die jeweils die Leuchtdioden 0 bis 7 einzeln einschalten und speichere sie unter den Namen LED0.PAS bis LED7.PAS ab.

Zuordnung der Kanalnummern der Kombiampel zu den Zahlenwerten in den Anweisungen Ausg**eben (<Wert>):**

Kanalnummer (k)	7	6	5	4	3	2	1	0
2^k	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Wert von 2^k	128	64	32	16	8	4	2	1

Beispiele:

Ausgeben (5)	schaltet die Leuchtdioden von Kanal 0 und 2 gleichzeitig an.
Ausgeben (255)	schaltet alle Leuchtdioden an.
Ausgeben (0)	schaltet alle Leuchtdioden aus.

3.2 Steuerung mit "binaus (<Bitmuster>);"

Mit der Anweisung "binaus (<Bitmuster>);" aus der UNIT Komampel kann man die einzelnen Leuchtdioden ohne zu rechnen ein- und ausschalten, vergleiche Beispiel.

```
PROGRAM Binaer_Ausgabe;
USES Komampel;
BEGIN
      Lauflicht_ein;
      Binaus ('00000001');
END.
```

Das Bitmuster '00000001' schaltet den Kanal 0 ein. Speichere das Programm unter dem Namen BINAER0.PAS ab.

Beispiel für die Zuordnung der Kanalnummern der Kombiampel zu den Stellen im Bitmuster bei der Anweisung "binaus ('XXXXXXXX')". X kann den Wert 0 oder 1 annehmen.

[2] Wer mit den Schülern schon die Kombiampel behandelt hat, kann dieses Kapitel überspringen.

Kanalnummer (k)	7	6	5	4	3	2	1	0
Beispiel: ('10010101')	1	0	0	1	0	1	0	1

Aufgabe 1: Schalte so auch die LEDs der anderen Kanäle ein. Speichere die Programme als BINAER1.PAS ... BINAER7.PAS.
Aufgabe 3: Schalte alle LEDs aus.

3.3 Blinker mit ausgeben (<Wert>)

Die Anweisung `Warte(10);` erzeugt eine Pause von ca. 10 Sekunden[3].
Beispiel:

```
PROGRAM Warten;
USES Komampel;
BEGIN
  Lauflicht_ein;
  Ausgeben (1);
  Warte (10);
  Ausgeben (0);
END.
```

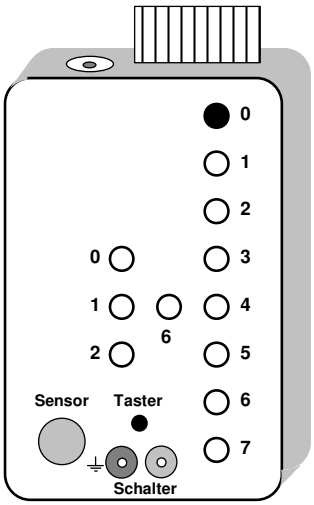


Abbildung 3-1: Kombiampel, LED 0 eingeschaltet

Aufgabe: Lasse die LED von Kanal 7 ca. 30 Sekunden leuchten.
Zusatz: Die LEDs von Kanal 3 und Kanal 5 sollen nach etwa 20 Sekunden angehen.

Leider blinkt es nur einmal. Das liegt daran, daß die Anweisung zum Blinken nur einmal gegeben wird.
Das folgende Programm sorgt für Abhilfe. Speichere es unter dem Namen BLINKAUS.PAS.

```
PROGRAM Blinkaus;
USES Komampel;
BEGIN
  Lauflicht_ein;
  REPEAT
    Ausgeben (1);
    Warte (0.5);
    Ausgeben (0);
    Warte (0.5);
  UNTIL Tastatur;
END.
```

Erklärung:

Wiederhole

die Blinkanweisungen

bis eine Taste auf der Tastatur gedrückt wird.

[3] Die Zeitauflösung der WARTE-Prozedur ist etwa 0,001 s.

3.4 Blinker mit der Anweisung "binaus (<Bitmuster>)"

Beispiel:

```
PROGRAM Warten;
USES Komampel;
BEGIN
      Lauflicht_ein;
      Binaus ('00000001');
      Warte (10);
      Binaus ('00000000');
END.
```

Aufgabe: Lasse die LED von Kanal 7 ca. 30 Sekunden leuchten.
Zusatz: Die LEDs von Kanal 3 und Kanal 5 sollen nach etwa 20 Sekunden angehen.

```
PROGRAM Blinken;
USES Komampel;
BEGIN
      Lauflicht_ein;
      REPEAT
            Binaus ('00000001');
            Warte (0.5);
            Binaus ('00000000');
            Warte (0.5);
      UNTIL Tastatur;
END.
```

Speichere es unter dem Namen BLINKBIN.PAS.

4 Datenfernübertragung mit Lichtwellenleitern

4.1 Senden von Lichtsignalen

Das Aktoreninterface besitzt als Aktor einen Sender für Lichtwellenleiter, der durch den Ausgang A0 angesteuert wird. Der Lichtwellenleiter wird in die linke Öffnung des Gerätes geschoben.

Aufgabe 1: Schreibe ein Programm LWLSENDE.PAS, das den Sender im Sekundentakt ein- und ausschaltet. Überprüfe, ob das andere Ende des LWL rot strahlt und ob das Licht im Sekundentakt an- und ausgeht.

Aufgabe 2: Ändere die Übertragungsgeschwindigkeit des obigen Programms.

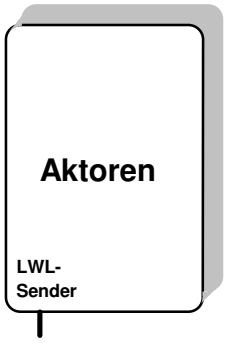


Abbildung 4-1: LWL-Sender und Empfänger gekoppelt

4.2 Gerätetest mit Lichtwellenleiter

Es sollen zunächst Lichtsignale von einem Rechner ausgeschickt, mit Hilfe eines Lichtwellenleiters übertragen und von dem gleichen Rechner zurück gelesen werden. Der Lichtwellenleiter wird in die dafür vorgesehene linke Öffnung des Senders des Aktoreninterfaces geschoben. Das andere Ende des Lichtwellenleiters wird in die rechte Öffnung des Empfängers des gleichen Aktoreninterfaces geschoben. Der LWL-Empfänger wird durch das Signal Schalter der Kombiampel abgefragt.

Aufgabe: Ergänze das Programm LWLSEND.PAS, das im Sekundentakt Licht auf den Lichtwellenleiter bringt so, daß die empfangenen Lichtsignale auf den Ausgang A1 der Kombiampel gebracht werden. Speichere das Programm unter LWLTEST.PAS ab.

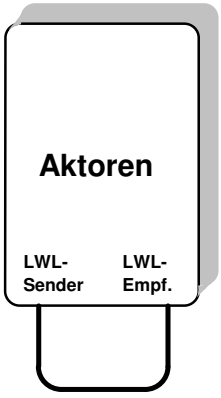


Abbildung 4-2: Der LWL-Sender

4.3 Senden und Empfangen von Lichtsignalen mit zwei Rechnern

Es sollen Daten in eine Richtung (simplex) von einem Senderechner mit Hilfe eines Lichtwellenleiters in einen Empfangsrechner transportiert werden. Der Lichtwellenleiter wird in die dafür vorgesehene linke Öffnung des Senders des Aktoreninterfaces geschoben. Das andere Ende des Lichtwellenleiters wird in die rechte Öffnung des Empfängers des zweiten Aktoreninterfaces geschoben.

Aufgabe 1: Sprich mit Deiner Nachbargruppe ab, wer sendet und wer empfängt und führe eine Signalübertragung durch. Speichere das Überprüfungsprogramm unter LWLBITSE.PAS bzw. LWLBITEM.PAS ab.

Aufgabe 2: Tausche die Rollen der beiden Rechner.

Das Empfangssystem kann zusätzlich auch als Sender arbeiten, wenn das empfangene Signal wieder auf dem Ausgang 0 ausgegeben wird.

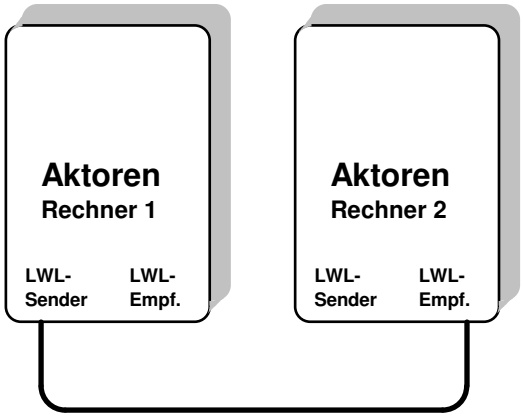


Abbildung 4-3: Sende- Empfangsbetrieb, simplex

Aufgabe 3: Stelle mit Hilfe eines weiteren Lichtleiters eine Verbindung über drei Rechner her und Überprüfe die empfangenen Lichtsignale. Speichere das Programm für den mittleren Rechner unter dem Namen SENDEMPF.PAS.

Aufgabe 4: Verbinde so viele Rechner wie möglich durch Lichtwellenleiter miteinander.

4.4 Übertragungsgeschwindigkeit messen

Das nachfolgende Programm `LWLGESCH.PAS` zählt bei angeschlossenem Lichtwellenleiterkabel die mit maximaler Frequenz ausgeschickten Impulse und registriert die empfangenen Impulse. Wenn ein Impuls gelesen wird, leuchtet Ausgang 2.

```
program LWLgeschwindigkeit;
uses komampel;
VAR z1, z2 : integer;
begin
  z1:=0;
  z2:=0;
  lauflicht_ein;
  repeat
    warte(0.002);         (*Uebertragungsgeschwindigkeit festlegen*)
    z1:=z1+1;             (*gesendetes Lichtsignal zaehlen*)
    if schalter then      (*Lichtsignal empfangen*)
      begin
        ausgeben (2);     (*Empfangenes Signal ausgeben*)
        z2:=z2+1;         (*Empfangenens Signal zählen*)
      end;
  until tastatur;
  writeln (z1,' ',z2);    (*Zaehlergebnisse anzeigen*)
  ausgeben (0);
end.
```

Aufgabe: Stelle fest, bis zu welcher Zeit die Anzahl der gesendeten Signale gleich der Anzahl der empfangenen Signale ist.

4.5 Reichweitenermittlung

Bei der Übertragung durch einen Lichtwellenleiter geht Energie verloren (Dämpfung 0,3 dB/m bei dem hier vorgeschlagenen LWL). Das bedeutet, daß nach etwa 10m Länge das Licht so stark abgeschwächt ist, daß es den Empfänger nicht mehr sicher steuern kann. Auch wenn dieser Versuch wegen der benötigten ca.12 m Lichtleiter teuer ist, kann er lohnend sein, erklärt er doch mit Hilfe die Notwendigkeit von Verstärkung bei längeren Übertragungstecken.
Das Prinzip der Verstärkung von Signalen wird mit Hilfe aller zur Verfügung stehenden Aktoreninterfaces gezeigt. Dafür werden die Lichtwellensender jeweils mit den Lichtwellenempfängern der nachfolgenden Geräte verbunden und die Programme LWLVERST.PAS gestartet.

```
program LWLverstaerker;
uses komampel;
VAR z1, z2, z3 : integer;
begin
  lauflicht_ein;
  repeat
    if schalter then
      begin
        ausgeben (1);       (* empfangenes Signal *)
                            (* auf den LWL-Sender geben*)
      end
  until tastatur;
  ausgeben (0);
end.
```

4.6 Das Prinzip der seriellen Datenübertragung[4]

Sollen mehr als ein Bit über eine Leitung übertragen werden, so kann das nur zeitlich nacheinander geschehen. Soll z.B. das parallele Bitmuster "10101010" seriell übertragen werden, so muß zunächst festgelegt werden, wann die Übertragung beginnt. Wenn keine Datenübertragung stattfindet, hat die Leitung den Wert 0. Um anzuzeigen , daß eine Datenübertragung stattfinden soll, wird als erstes ein "1"-Signal als Startbit gesendet. Danach folgen in festem Zeitabstand die Zustände der einzelnen Datenbits, beginnend mit dem niederwertigen Bit 0. Den Abschluß bildet das Stopbit mit dem Wert "0". Danach ist die Leitung wieder frei für die nächste Übertragung. Für jedes Bit steht eine bestimmte Zeit, die Bitzeit, zur Verfügung.

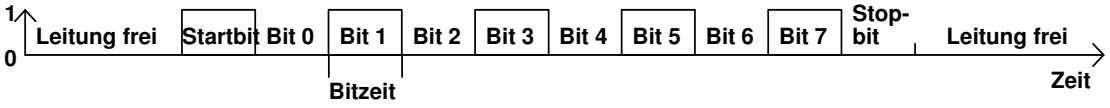


Abbildung 4-4: Zeitablauf beim Senden

Der Empfänger muß die seriell übertragenen Bits wieder in das richtige parallele Bitmuster umwandeln. Der Empfänger wartet zunächst, bis der Startimpuls mit dem Wert "1" kommt und beginnt dann mit dem Einlesen des darauf folgenden Bitmusters. Das Einlesen muß nun im gleichen Takt der Bitzeit geschehen, wie beim Senden. Um die Werte der einzelnen Bits zuverlässig erkennen zu können, wird immer in der Mitte der zugehörigen Bitzeit nachgesehen, welchen Wert das gerade empfangene Datenbit hat. Um nach dem Startimpuls die Mitte des Bits 0 zu erreichen, muß der Empfänger zunächst die 1,5 fache Bitzeit abwarten, bis er das Bit liest. Danach wartet er 7 mal die Bitzeit ab und liest dann jeweils die Datenbits 1 bis 7 ein. Den Abschluß bildet noch einmal eine Bitzeit, um in das Stopbit zu kommen. Das Bit wird aber nicht mehr ausgewertet. Während des Einlesens ordnet der Empfänger die empfangenen Bits wieder als korrektes paralleles Bitmuster an.

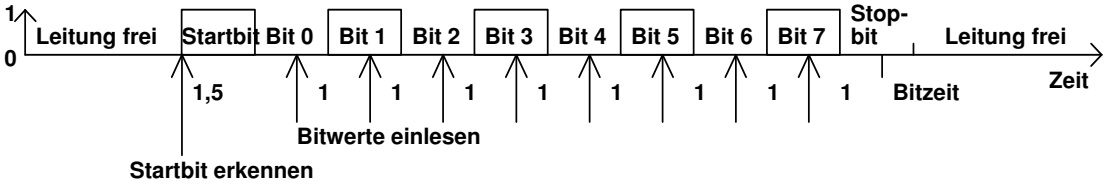


Abbildung 4-5: Zeitablauf bei Empfang

4.7 Die serielle Datenübertragung mit Lichtleitern

Bei der seriellen Datenübertragung gibt es als einfache Möglichkeit in Form von Simplex-Betrieb den Datentransports von z. B. Rechner 1 als Sender zu Rechner 2 als Empfänger oder umgekehrt. Hierbei muß die Richtung der Datenübertragung vereinbart werden. Beim Duplex-Betrieb ist die Datenübertragung zwischen zwei Rechner ohne Absprache möglich.

4.7.1 Simplex-Betrieb

Der Lichtwellenleiter wird in die dafür vorgesehene linke Öffnung des Senders der Aktoren für die Kombiampel geschoben. Das andere Ende des Lichtwellenleiters wird in die rechte Öffnung des Empfängers der anderen Aktoren für die Kombiampel geschoben. Mit Hilfe des zweiten Lichtwellenleiters werden der noch offene Lichtwellensender und der Lichtwellenempfänger beider Aktoren für die Kombiampel verbunden.

[4] Die Ausführungen zur seriellen Datenübertragung sind aus dem Begleitmaterialien Messen, Steuern und Regeln mit der Kombiampel in wesentlichen Teilen übernommen und ergänzt worden.

Für eine saubere Übertragung müssen die Bitzeiten auf dem Sende- und Empfangsrechner übereinstimmen.
Bei modernen Rechnern ab 486 kann die Wartezeit der PASCAL-Prozedur DELAY, die der Anweisung WARTE zugrunde liegt, zu kurze Laufzeiten haben. Der Quelltext KOMAMPEL.PAS der Unit KOMAMPEL.TPU enthält eine Konstante "Zeitkorrekturfaktor", mit der diese unterschiedlichen Laufzeiten ausgeglichen werden können. Das Programm ZEITTEST.PAS (siehe Diskette) ermittelt diesen Korrekturfaktor. Die Zeitmessung muss für beide beteiligte Rechner durchgeführt werden.
Die Unit KOMAMPEL muß nach der Änderung auf beiden Rechnern neu übersetzt werden.
Der Faktor läßt sich auch mit der Stoppuhr finden.
Die Programme dürfen nicht unter Windows laufen, denn Windows verbraucht unberechenbar viel Zeit für die eigene Systemverwaltung, womit die Zeiteinteilung des Pascalprogramms vollständig gestört ist.

4.7.1.1 Senden eines Bytes

Das erste Sendeprogramm BYTESEND.PAS für ein Byte setzt den oben geschilderten Zeitablauf unmittelbar in Software um.

```
program byte_senden;
uses komampel;
begin
  lauflicht_ein;
               (* Ausgabe des Bitmuster 01010101 auf Bit 0 *)
  ausgeben(0); (*  Startimpuls senden *)
  warte(1);
  ausgeben(0); (*  Bit 0 senden *)
  warte(1);
  ausgeben(1); (*  Bit 1 senden *)
  warte(1);
  ausgeben(0); (*  Bit 2 senden *)
  warte(1);
  ausgeben(1); (*  Bit 3 senden *)
  warte(1);
  ausgeben(0); (*  Bit 4 senden *)
  warte(1);
  ausgeben(1); (*  Bit 5 senden *)
  warte(1);
  ausgeben(0); (*  Bit 6 senden *)
  warte(1);
  ausgeben(1); (*  Bit 7 senden *)
  warte(1);
  ausgeben(0); (*  Stopbit senden*)
  warte(1);
end.
```

Mit dem Programm OSZI.EXE auf der Diskette kann der Impulsverlauf des obigen Programms als Oszillogramm, wie in der Grafik unter 3.6.1 dargestellt, auf dem Bildschirm des Empfangsrechners verfolgt werden. Im selben Verzeichnis wie das Programm OSZI.EXE muss auch der passende Grafiktreiber vorhanden sein (z.B. EGAVGA.BGI). Die Treiber sind alle auf der Diskette zu finden.

4.7.1.2 Empfangen eines Bytes

Das erste Empfangsprogramm BYTEEMPF.PAS für ein Byte bildet den Zeitablauf auf die gleiche Weise ab.

```
program byte_empfaenger;
uses komampel;
var  empfangsbyte : integer;

begin
  lauflicht_ein;
  empfangsbyte := 0;
  repeat until schalter;(*warten, bis der Startimpuls kommt*)
  warte(1.5);     (*die Mitte des ersten Datenbits abwarten*)
  if schalter then      (* Bit 0 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;[5]
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;[6]
  if schalter then       (* Bit 1 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;
  if schalter then        (* Bit 2 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;
  if schalter then        (* Bit 3 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;
  if schalter then        (* Bit 4 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;
  if schalter then        (* Bit 5 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;
  if schalter then        (* Bit 6 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 128;
```

[5] Der empfangene Wert wird auf Bit 7 des Empfangsbytes geschrieben. Ist der Wert "1", wird 128 addiert. Ist der Wert "0", müßte 0 addiert werden, diese Addition wird aber nicht durchgeführt, weil sich der Wert des Empfangsbytes dabei nicht ändert.

[6] Die Division durch 2 schiebt das bisher empfangene Bitmuster um eine Binärstelle nach rechts und sorgt so für die richtige Anordnung der empfangenen Bits.

```
    end;
  ausgeben(empfangsbyte);
  warte(1);
  empfangsbyte := empfangsbyte DIV 2;
  if schalter then        (* Bit 7 einlesen *)
    begin
      empfangsbyte := empfangsbyte + 1;
    end;
  ausgeben(empfangsbyte);
  warte(1);               (* Stopbit abwarten *)
end.
```

4.7.1.3 Senden eines Bytes auf elegantere Art

Das folgende Sendeprogramm BYTESENS.PAS erlaubt es, eine beliebige Zahl von 0 bis 255, die auf dem Senderechner eingegeben wird, auf den Empfangsrechner zu übertragen. Dabei ist die Sendung der Datenbits in eine Schleife gekleidet.

```
program byte_senden;
uses komampel;

VAR
  ausgabezahl : integer;
  bitzaehler : integer;
begin
  lauflicht_ein;
  write('Welche Zahl soll übertragen werden ? ');
  readln(ausgabezahl);
  ausgeben(1);                        (* Startimpuls senden *)
  warte(1);
  bitzaehler := 8;
  repeat                              (* Datenbits senden *)
    ausgeben(ausgabezahl);
    ausgabezahl := ausgabezahl DIV 2;
    warte(1);
    bitzaehler := bitzaehler - 1;
  until bitzaehler = 0;
  ausgeben(0);                        (* Stopbit senden *)
  warte(1);
end.
```

4.7.1.4 Empfangen eines Bytes auf elegantere Art

Das Empfangsprogramm BYTEEMPS.PAS empfängt das gesendete Byte und schreibt die empfangene Zahl sowie das zugehörige ASCII-Zeichen auf den Bildschirm. Auch hier ist das Einlesen der Datenbits in eine Schleife gekleidet.

```
program byte_empfaenger;
uses komampel;

var
  empfangsbyte : integer;
  bitzaehler : integer;

begin
  lauflicht_ein;
  empfangsbyte := 0;
  bitzaehler := 8;
  repeat until schalter;
    (* warten, bis der Startimpuls kommt *)
  warte(1.5);
    (* die Mitte des ersten Datenbits abwarten *)
  repeat
    empfangsbyte := empfangsbyte div 2;
    ( *Empfangene Bits um eine Stelle nach unten schieben *)
```

```
    if schalter then  (*Datenbit einlesen*)
      begin
        empfangsbyte := empfangsbyte + 128;
      end;
    ausgeben(empfangsbyte);
    warte(1);
    bitzaehler := bitzaehler - 1;
  until bitzaehler = 0;
  writeln('Empfangener Zahlenwert: ',empfangsbyte);
  writeln('Empfangenes Zeichen : ',chr(empfangsbyte));
  repeat until tastatur;
end.
```

4.7.1.5 Übertragen von einzelnen Zeichen

Zur Vorbereitung auf die angestrebte Textübertragung hier ein Programm zur Übertragung eines Zeichens von der Tastatur des einen Rechners auf den Bildschirm des anderen Rechners. Als Empfänger dient wieder das Programm BYTEEMPS.PAS, das ja auch das Zeichen ausgibt.

ZEICHSEN.PAS:

```
program byte_senden;
uses komampel, crt;      (* Das Einbinden der Unit CRT ist fuer den *)
                         (* Befehl readkey notwendig. *)
VAR
  ausgabezahl : integer;
  bitzaehler : integer;
  zeichen : char;

begin
  lauflicht_ein;
  write('Druecke eine Zeichentaste auf der Tastatur ');
  zeichen := readkey;
  ausgabezahl := ord(zeichen);
  writeln;
  writeln(ausgabezahl);
  ausgeben(1);        (* Startimpuls *)
  warte(1);
  bitzaehler := 8;
  repeat
    ausgeben(ausgabezahl);
    ausgabezahl := ausgabezahl div 2;
    warte(1);
    bitzaehler := bitzaehler - 1;
  until bitzaehler = 0;
  ausgeben(0);         (* Stopbit *)
  warte(1);
end.
```

4.7.1.6 Übertragen von Text

Das Programmpaar TEXTSEND.PAS und TEXTEMPF.PAS bildet, aufbauend auf BYTESENS.PAS und BYTEEMPS.PAS, ein serielles Kommunikationssystem zwischen zwei Rechnern, bei dem sich die beteiligten Schüler beliebige Texte von der Tastatur des Senderechners auf den Bildschirm des Empfängers übertragen können.

TEXTSEND.PAS:

```
program text_senden;
uses komampel,crt;

const
  pause = 0.1;
VAR
  ausgabezeichen : char;

procedure senden(ausgabezahl:integer);
  var  bitzaehler : integer;
  begin
    ausgeben(1); (* Startimpuls *)
    warte(pause);
    bitzaehler := 8;
    repeat
      ausgeben(ausgabezahl);
      ausgabezahl := ausgabezahl div 2;
      warte(pause);
      bitzaehler := bitzaehler - 1;
    until bitzaehler = 0;
    ausgeben(0); (* Stopbit *)
    warte(pause);
  end;

begin
  lauflicht_ein;
  writeln('Gib eine Textzeile ein : ');
  repeat
    ausgabezeichen := readkey;
    write(ausgabezeichen);
    senden(ord(ausgabezeichen));
  until ausgabezeichen < #31;
end.
```

TEXTEMPF.PAS

```
program text_empfangen;
uses komampel;

const  pause = 0.1;
var  empfangszeichen : integer;

procedure empfaenger(var empfangsbyte : integer);
  var  bitzaehler : integer;
  begin
    empfangsbyte := 0;
    bitzaehler := 8;
    repeat until schalter or tastatur;
                    (* warten auf den Startimpuls oder *)
                    (* Abbruch ueber die Tastatur *)
    warte(1.5*pause);
    (* Mitte des ersten Datenbits abwarten *)
    repeat
      empfangsbyte := empfangsbyte div 2;
     (* Empfangene Bits um eine Stelle nach unten schieben *)
      if schalter or tastatur then  (* Datenbit einlesen *)
        begin
          empfangsbyte := empfangsbyte + 128;
        end;
      ausgeben(empfangsbyte);
      warte(pause);
      bitzaehler := bitzaehler - 1;
      if tastatur then   (* Abbruch ueber die Tastatur *)
        begin
```

```
          empfangsbyte := 0;
          bitzaehler := 0;
        end;
    until bitzaehler = 0;
  end;

begin
  lauflicht_ein;
  writeln('Abbruch mit <ESC>');
  repeat
    empfaenger(empfangszeichen);
    write(chr(empfangszeichen));
  until (empfangszeichen < 32) or tastatur;
  repeat until tastatur;
end.
```

4.7.2 Halbduplex-Betrieb

Bislang wurden die Daten nur in eine Richtung übertragen. Bei der realen Datenfernübertragung ist aber eine Verbindung in beide Richtungen sinnvoll. Dazu wird der freie Sender bzw. Empfänger der beiden Aktoreninterfaces zusätzlich mit einem zweiten Lichtwellenleiter verbunden.

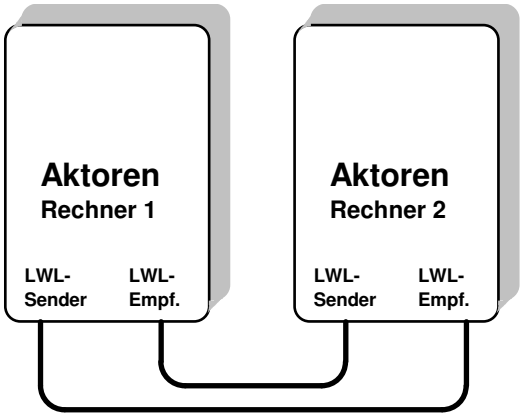


Abbildung 4-6: Halbduplexbetrieb

Das Program LWLHDUP.PAS

Bei der Erstellung des Programms LWLDUPLE.PAS wurden die Programme TEXTSEND.PAS und TEXTEMPF.PAS zusammengefaßt. Die jeweiligen Hauptprogramme wurden in Prozeduren umgewandelt und es wurde ein neues Hauptprogramm erstellt, das dafür sorgt, daß die Rechner nach Betätigen der entsprechenden Tasten auf Empfang oder Senden geschaltet werden. Damit ein Ausstieg aus dem Programm möglich ist, wurde eine Tasterabfrage der Kombiampel eingefügt. Stelle eine zweiseitige Datenverbindung her.

```
program text_duplex_empfangen_und_senden;
uses komampel, crt;

const
  pause = 0.01;
var
  auswahl : char;

procedure empfangen (var empfangsbyte : integer);
  var
    bitzaehler : integer;
  begin
    empfangsbyte := 0;
    bitzaehler := 8;
    repeat until schalter or tastatur;
(* warten, bis der Startimpuls kommt. *)
    warte(1.5*pause);
(* die Mitte des ersten Datenbits abwarten *)
    repeat
      empfangsbyte := empfangsbyte div 2;
```

```
      (* Empfangene Bits um eine Stelle nach oben schieben *)
      if schalter or tastatur then
        (* Datenbit einlesen und Taster abfragen *)
        begin
          empfangsbyte := empfangsbyte + 128;
        end;
      ausgeben(empfangsbyte);
      warte(pause);
      bitzaehler := bitzaehler - 1;
      if tastatur then
        begin
          empfangsbyte := 0;
          bitzaehler := 0;
        end;
    until bitzaehler = 0;
  end;

procedure senden(ausgabezahl : integer);
  var
    bitzaehler : integer;
  begin
    ausgabezahl := ord(zeichen);
    ausgeben(1);        (* Startimpuls *)
    warte(pause);
    bitzaehler := 8;
    repeat
      ausgeben(ausgabezahl);
      ausgabezahl := ausgabezahl div 2;
      warte(pause);
      bitzaehler := bitzaehler - 1;
    until bitzaehler = 0;
    ausgeben(0);         (* Stopbit *)
    warte(pause);
  end;

procedure textsenden;
var ausgabezeichen : char;
begin
  writeln('Gib eine Textzeile ein : ');
  repeat
    ausgabezeichen := readkey;
    write(ausgabezeichen);
    senden(ord(ausgabezeichen));
  until ausgabezeichen < #31;
end;

procedure textempfangen;
var
  empfangszeichen : integer;
begin
  writeln;
  writeln ('Empfangener Text (Abbruch mit <ESC>):');
  repeat
      empfangen(empfangszeichen);
      write(chr(empfangszeichen));
  until (empfangszeichen < 32);
end;

begin
  lauflicht_ein;
  repeat
    writeln ('Senden: "s", Empfangen: "e", Beenden: "q"');
    readln(auswahl);
    if auswahl = 's' then
      begin
        textsenden;
      end
```

```
    else
      begin
        if auswahl = 'e' then
          begin
            textempfangen;
          end;
      end;
  until auswahl = 'q';
  ausgeben (0);
  repeat until tastatur;
end.
```

5 Steuerung von Schrittmotoren

5.1 Aufbau eines unipolaren Schrittmotors

Damit Lernende zunächst eine originale Begegnung mit einem Schrittmotor erfahren, wird ein realer unipolarer Schrittmotor vorgeführt. Das Datenblatt, wenn vorhanden, gibt z.B. folgende technischen Daten an:

- Unipolarer 4-Strang-Typ
- Schritte pro Umdrehung.
- Schrittwinkel 1,8 Grad.
- Strangwiderstand 28 Ohm
- Strangstrom 0,17 bis 0,4 A
- Betriebsspannung 5V bis 12 V

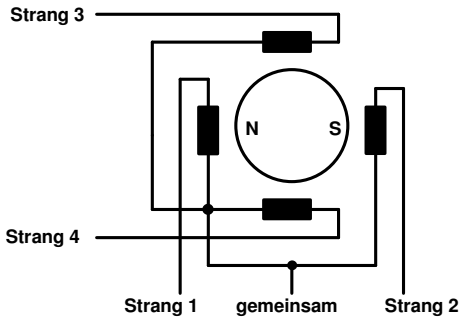


Abbildung 5-1: Schaltung eines unipolaren Schrittmotors mit 5 Anschlüssen

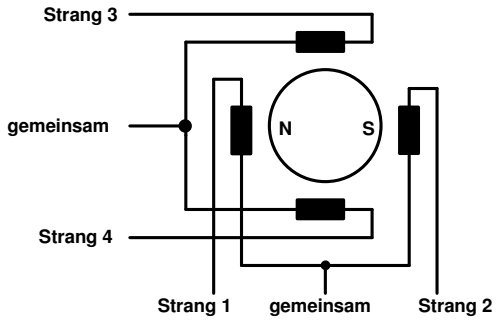


Abbildung 5-2: Schaltung eines unipolaren Schrittmotors mit 6 Anschlüssen

Wenn kein Datenblatt vorliegt, beginnt man mit der Klassifizierung bipolar oder unipolar. Einen bipolaren Schrittmotor erkennt man sofort an seinen 4 Anschlußleitungen. Dieser Motortyp scheidet für das Aktoreninterface aus[7]. Der unipolare hat 5 oder 6 Anschlußdrähte. Bei einem Motor mit 5 Anschlußleitungen sind die beiden gemeinsamen Leitungen der Spulenpaare zusammengeschaltet. Mit einem Ohmmeter werden die Anschlüsse ermittelt.
Beispiele für 6 Anschlüsse:
Möglichkeit a: Ein Anschluß zeigt gegenüber den anderen 5 Anschlüssen zweimal den gleichen Widerstand und dreimal einen unendlichen Widerstand. Dieser Anschluß ist eine der beiden gemeinsamen Stromversorgungen. Die beiden anderen Anschlüsse mit dem gleichen Widerstandswert sind die zugehörigen Stranganschlüsse. Die drei anderen gehören zum zweiten Strang.
Möglichkeit b: Ein Anschluß zeigt gegenüber drei anderen Anschlüssen unendlichen Widerstand, bei den beiden anderen zwei verschiedene Werte, wobei einer der beiden Werte doppelt so groß ist wie der andere. Dieser Anschluß ist ein Stranganschluß. Die Leitung mit dem größeren Widerstand ist der zweite Stranganschluß, die andere ist die gemeinsame Leitung.

[7] Ein technisch unsauberer aber physikalisch möglicher Betrieb von bipolaren Schrittmotoren mit dem Aktoreninterface ist möglich, wenn jeder Strang einen Vorwiderstand erhält. Widerstandswert und Leistung der Vorwiderstände sind abhängig von den Daten des verwendeten Schrittmotors.

Beispiel für 5 Anschlüsse:
Herausfinden des Anschlusses, der gegenüber den anderen Anschlüssen den gleichen Wert zeigt. Dieser Anschluß ist für die gemeinsame Stromversorgung herausgeführt, die restlichen 4 sind die einzelnen Stranganschlüsse. Da bei einem Schrittmotor mit 5 Anschlüssen die korrespondierenden Spulenpaare nicht gemessen werden können, müssen sie später experimentell ermittelt werden.
Das Öffnen eines Schrittmotors zum Aufzeigen von Stator und Rotor mit ihren besonderen Merkmalen kann ein Startpunkt für den Umgang mit Schrittmotoren bei Lernenden sein. Das kann jedoch, je nach Schrittmotor, auch bedeuten, daß er dadurch unbrauchbar wird.
Bei einem zerlegten unipolaren Schrittmotor konnte folgendes beobachtet werden: Es sind 6 Zuleitungen vorhanden. Das Zählen der Spulen ergibt 4. Jeder Eisenkern einer Spule besitzt 4 Zähne. Der Rotor ist aus zwei zahnradähnlichen Zylindern aufgebaut. Das Zählen der beiden Reihen der Magnetzähne des Rotors ergibt jeweils die Zahl 25. Beide Zylinder sind um 1 Zahn gegeneinander versetzt. Über diese Daten (4*25*2) kann die Schrittzahl ermittelt werden: 200.

5.2 Vorbereitende Experimente

Da die Steuerung von unipolaren Schrittmotoren besser zu durchschauen ist, wenn man eine didaktische Reduktion durchführt, beginnt man zunächst mit dem Vormodell eines Schrittmotors.
Bei den folgenden Steuerungen hilft die dazu geschaltete Kombiampel, den Ablauf der einzelnen Schritte bei langsamer Abfolge des Programms darzustellen. Bei schnellerem Programmablauf hilft sie, eine Vorstellung über die Geschwindigkeit der Programmausführung zu geben.

5.2.1 Geeignete Spulen

Da der ULN2803 im Aktoreninterface max. 500 mA pro Schrittmotorstrang zur Verfügung stellt, muß eine Spule bei einer Steuerspannung von 5 V einen Widerstand von mindestens 10 Ohm haben.
Spulen mit Eisenkern sind als Schülergeräte in Physiksammlungen vorhanden. Ihr Nachteil ist, daß sie sehr niederohmig sind. Außerdem werden sie in der Physiksammlung ständig gebraucht.
Preiswerte, handliche Spulen mit Eisenkern und einem Widerstand von ca. 50 Ohm können bezogen werden, vgl. Bezugsquellen. Sie können im Informatikraum verbleiben.
Spulen mit Eisenkern erhält man auch, wenn man die Kontakte von Relais entfernt. Sie sind klein, hochohmig und können ebenso im Informatikraum bleiben.

5.2.2 Herstellung einer Kompaßnadel

Eine Kompaßnadel kann leicht aus einer halben Büroklammer, einem Druckknopf, einer Nähnadel und z.B. einer Korkscheibe, aus einem Korken geschnitten, hergestellt werden. Die metallischen Druckknöpfe sind z.B. als 6-er Briefchen im Nähzubehörhandel zu erhalten. Wenn man sie aufknöpft, erhält man 12 Druckknöpfe. Die Druckknöpfe besitzen jeweils 4 Löcher, die auch für das Durchstecken einer aufgebogenen Büroklammer geeignet sind. Eine Büroklammer besteht aus einem äußeren und einem inneren Teil. Nur die beiden Schenkel (ca. 2,5 cm) des äußeren Teils werden benötigt, der Rest wird abgetrennt. Die beiden Schenkel werden so durch zwei gegenüberliegende Löcher eines Druckknopfes gesteckt, daß die Biegung des äußeren Teils der ehemaligen Büroklammer über die äußere Wölbung des Kopfes des Druckknopfes geführt wird. Der Draht wird dann gerade gebogen und durch mehrmaliges Überstreichen mit einem Pol eines Magneten magnetisiert. Mit Nagellack oder rotem Edding wird der Nordpol, mit grünem Edding der Südpol markiert. Die so hergestellte Kompaßnadel wird mit der inneren Wölbung des Kopfes des Druckknopfes auf die Spitze einer Nähnadel gesetzt, die z.B. auf einer Korkscheibe aufgespießt ist. Die Korkscheibe sollte auf eine Unterlage geklebt sein. Mit Hilfe eines Seitenschneiders kann ein evtl. fehlendes Gleichgewicht der beiden Pole hergestellt werden.

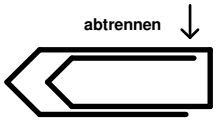


Abbildung 5-3: Bearbeitung der Büroklammer

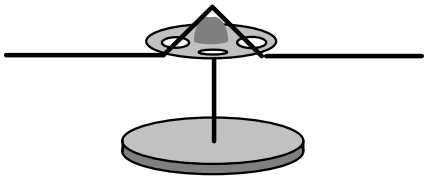


Abbildung 5-4: Der fertige Kompaß

5.2.3 Versuch mit einer Spule

Die Wicklung der Spule wird an den Schraubklemmen + und A0 des Anschlußbausteins angeschlossen, der über das 16-polige Kabel mit dem Aktoreninterface verbunden ist.
Der Versuch wird folgendermaßen aufgebaut und durchgeführt:

- Aufstellen der Kompaßnadel und auspendeln lassen.
- Hinzustellen der Spule senkrecht zur vorhandenen Ausrichtung der Kompaßnadel. Dabei sollte der Abstand so gewählt werden, daß die vorhandene Ausrichtung nicht wesentlich beeinflußt wird.
- Einschalten der Spule und Überprüfen, ob eine Ablenkung erfolgt.

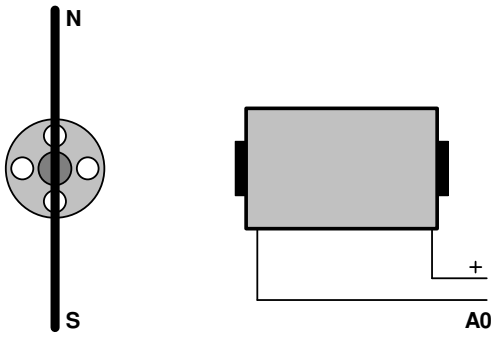


Abbildung 5-5: Ablenkung der Magnetnadel mit einer Spule

Bei den folgenden Programmen ist es sinnvoll, den Ausgabebefehl "binaus(<Bitmuster>)" zu verwenden, damit nicht gerechnet werden muss.

Aufgabe 1: Schreibe das Programm EINSPULE.PAS, das die Kompaßnadel immer dann ablenkt, wenn der Taster der Kombiampel gedrückt wird. Notiere, welcher Pol angezogen wird (rot = Nordpol der Magnetnadel). Speichere das Programm unter EINSPULE.PAS ab.
Aufgabe 2: Pole die Spule um und notiere, welcher Pol nun angezogen wird.
Aufgabe 3: Drehe die Spule um und notiere, welcher Pol angezogen wird.

5.2.4 Versuch mit zwei Spulen

Die Wicklung der zweiten Spule wird mit den Schraubklemmen + sowie A1 des Anschlußbausteins verbunden. Die Spulen werden entsprechend der Abbildung angeordnet. Der Abstand der Spulen von der Kompaßnadel soll gleich sein.

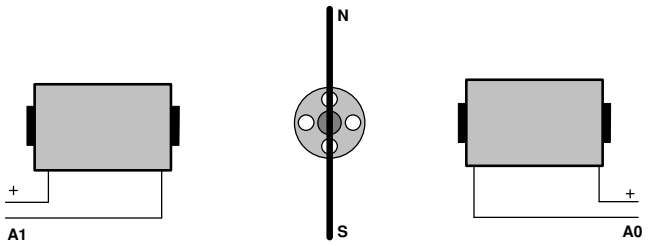


Abbildung 5-6: Ablenkung der Magnetnadel mit zwei Spulen

Aufgabe 1: Schreibe das Programm ZWESPULE.PAS, das den Nordpol der Kompaßnadel zur Spule an A0 hin ablenkt, wenn der Taster der Kombiampel betätigt wird. Wenn der Sensor betätigt wird, soll der Nordpol der Kompaßnadel zur Spule an A1 abgelenkt werden. Eventuell müssen die Spulen umgepolt werden.

Aufgabe 2: Ändere das Programm ZWESPULE.PAS. Der Nordpol der Kompaßnadel soll zuerst von der Spule am Anschluß A0 angezogen werden. Nach einer Wartezeit soll er von der Spule am Anschluß A1 angezogen werden.

Aufgabe 3: Dieser Vorgang soll sich ständig wiederholen. Versuche durch einen geeigneten Zeitablauf die Magnetnadel zu drehen. Damit hast du einen einfachen Schrittmotor konstruiert. Speichere das Programm unter ZWESPUVO.PAS ab.

5.3 Vier Spulen, Schrittmotormodell im Vollschrittverfahren

Die Wicklungen der dritten und vierten Spule werden mit den Schraubklemmen + und A2 bzw. A3 des Anschlußbausteins verbunden.

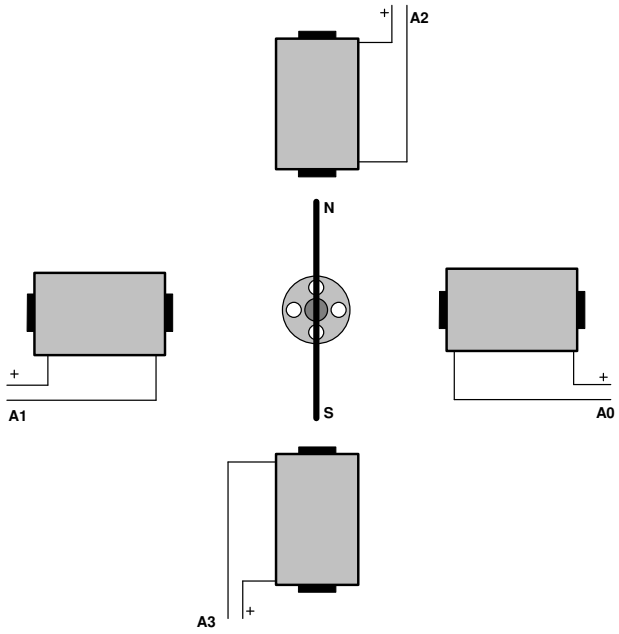


Abbildung 5-7: Ablenkung der Magnetnadel mit vier Spulen

Aufgabe 1: Schreibe ein entsprechendes Schrittmotorprogramm für die beiden neuen Spulen.

Aufgabe 2: Schreibe das Programm VOLLSCHR.PAS, das die vier Spulen nacheinander einzeln anspricht. Damit ist der Schrittmotor vollständig.

5.4 Schrittmotormodell im Halbschrittverfahren

Damit der Schrittmotor bei zukünftigen Anwendungen „runder“ läuft, soll das Halbschrittverfahren angewandt werden. Dieses Verfahren nutzt die Tatsache aus, daß zwei benachbarte Spulen, die gleichzeitig angesteuert werden, die Kompaßnadel zwischen die beiden Spulen ablenken.

Aufgabe 1: Steuere zwei benachbarte Spulen gleichzeitig an und notiere das Verhalten der Kompaßnadel.
Aufgabe 2: Steuere zuerst die linke Spule, dann beide Spulen und zuletzt die rechte Spule mit einer längeren Wartezeit an. Notiere das jeweilige Verhalten der Kompaßnadel.
Aufgabe 3: Schreibe das Programm HALBSCHR.PAS. Erweitere das Programm aus Aufgabe 2 auf alle vier Spulen. Speichere das Programm unter HALBSCHR.PAS ab.
Aufgabe 4: Erniedrige die Zeit der Warteschleifen. Welches Problem tritt auf?

5.5 Schrittmotormodell mit Ringmagnet

Einen weiteren Schritt zum Übergang zum realen Schrittmotor stellt ein Schrittmotormodell[8] her, das mit einem Rotor aus einem magnetischen Ringkern mit zwei Polen arbeitet. Hier ist die erzeugte Kraft bereits so groß, daß eine Getriebeanordnung möglich ist. Auch hier eignen sich die bereits erstellten Programme VOLLSCHR.PAS, HALBSCHR.PAS zur Ansteuerung.

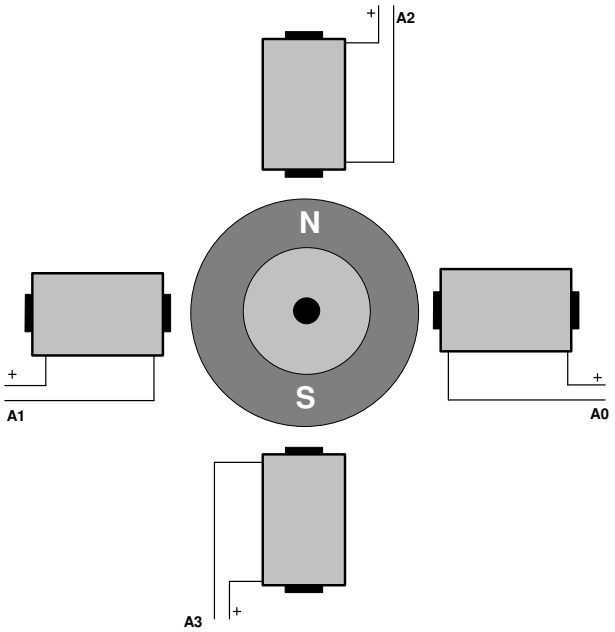


Abbildung 5-8: Schrittmotormodell mit Ringmagnet

5.6 Steuerung eines unipolaren Schrittmotors

Ein unipolarer Schrittmotor wird an die Anschlüsse + (gemeinsamer Anschluß) sowie A0 bis A3 des Aktoreninterfaces angeschlossen. Die Achse wird mit einem Montageflansch[9] versehen. Zur Verdeutlichung der Drehrichtung wird darauf eine Markierung angebracht.

Aufgabe 1: Steuere den Schrittmotor mit den Programmen VOLLSCHR..PAS bzw. HALBSCHR.PAS an.

[8] Bezugsquellen vgl. Anhang

[9] Bezugsquellen vgl. Anhang

Aufgabe2: Schreibe das Programm HALBSCHR.PAS mit Hilfe einer for zaehler := 0 to do - Anweisung so um, daß der Schrittmotor 360 Grad nach rechts dreht. Speichere das Programm unter EINEUMRE.PAS ab.

Aufgabe 3: Schreibe ein Programm EINEUMLI.PAS, das den Schrittmotor 360 Grad in die andere Richtung drehen läßt.

Aufgabe 4: Schreibe ein Programm EINELIRE.PAS, das den Schrittmotor 360 Grad in die eine, dann nach einer Pause in die andere Richtung drehen läßt.

5.7 Steuerung von zwei unipolaren Schrittmotoren

Ein zweiter unipolarer Schrittmotor wird an die Anschlüsse + sowie A4 - A7 des Anschlußbausteins des Aktoreninterfaces angeschlossen und mit einem Montageflansch versehen. Darauf wird wieder zur Verdeutlichung der Drehrichtung eine Markierung angebracht.

Aufgabe 1: Steuere den zweiten Schrittmotor mit den Programmen VOLLSCHR.PAS bzw. HALBSCHR.PAS an.

Aufgabe 2: Beide Schrittmotoren sollen sich gleichzeitig drehen. Speichere das Programm unter ZWEIMOT.PAS.

Aufgabe 3: Schreibe ein Programm SCHRLIRE.PAS, das den Schrittmotor bei Betätigung des Tasters links herum und bei Betätigung des Schalters der Kombiampel rechts herum drehen läßt.

Aufgabe 4: Lasse zwei Schrittmotoren im Halbschrittverfahren gleichzeitig gleichsinnig links herum laufen. Speichere das Programm unter ZWEIGLLI.PAS ab.

Aufgabe 5: Lasse zwei Schrittmotoren im Halbschrittverfahren gleichzeitig gleichsinnig rechts herum laufen. Speichere das Programm unter ZWEIGLRE.PAS ab.

Aufgabe 6: Lasse zwei Schrittmotoren im Halbschrittverfahren gleichzeitig gegensinnig laufen. Speichere das Programm unter ZWEIGEGE.PAS ab.

6 Steuerung einer Waschmaschine[10]

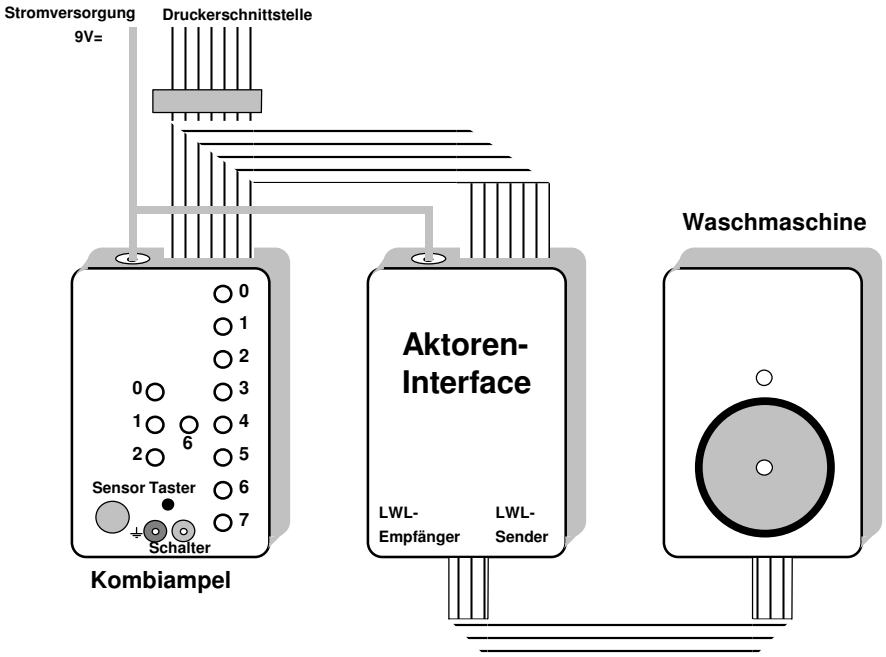


Abbildung 6-1: Anordnung der Waschmaschine

6.1 Aufbau des Waschmaschinenmodells

Das Manipulator-/Waschmaschinenmodell stellt eine Waschmaschine dar, wenn das große Zahnrad montiert wird.

6.2 Ausgangs- und Eingangsbelegung

Das Waschmaschinenmodell wird an das Aktoreninterface angeschlossen. Parallel zum Aktoreninterface wird die Kombiampel geschaltet. Es werden die Ausgänge A4-A7 für die Ansteuerung des Schrittmotors und die Ausgänge A0-A3 zur Anzeige einzelner Waschvorgänge genutzt.

- A0: Waschmittel einfüllen
- A1: Heizung an
- A2: Ventil öffnen
- A3: Pumpe an

Die Sensoren der Kombiampel können für folgende Aufgaben herangezogen werden:

- Taster: Start des Waschmaschinenprogramms
- Schalter: Trommeltür zu

Aufgabe: Beobachte zu Hause einen Waschvorgang einer Waschmaschine und notiere die einzelnen Waschvorgänge.

[10] V. Ludwig, Ein Schülergerät für eine Grundbildung Informatik-Teil VI S. 4 ff PC-Profi Sparkassenverlag Stuttgart

6.3 Die Prozeduren für die Drehbewegungen der Waschtrommel

Die Waschtrommel führt eine langsame links-rechts-Bewegungen aus, hier schaukeln genannt. Sie führt eine schnelle Bewegung in einer Richtung aus, schleudern.

Aufgabe 1: Schreibe die Prozedur drehen_links, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach links mit der Pausenzeit 0.01s vollziehen läßt. Abspeichern unter WASCHMAS.PAS.

Aufgabe 2: Schreibe die Prozedur drehen_rechts, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach rechts mit der Pausenzeit 0.01s vollziehen läßt.

Aufgabe 3: Schreibe die Prozedur drehen_schnell, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach rechts mit einer möglichst kleinen Pausenzeit drehen läßt.

Aufgabe 4: Schreibe die Prozedur drehen_rechts_pumpe, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach rechts mit der Pausenzeit 0.01s vollziehen läßt und gleichzeitig den Ausgang für das Einschalten der Pumpe auf 1 setzt.

Aufgabe 5: Schreibe die Prozedur drehen_links_pumpe, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach links mit der Pausenzeit 0.01s vollziehen läßt und gleichzeitig den Ausgang für das Einschalten der Pumpe auf 1 setzt.

Aufgabe 6: Schreibe die Prozedur drehen_rechts_heizung, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach rechts mit der Pausenzeit 0.01s vollziehen läßt und gleichzeitig den Ausgang für das Einschalten der Heizung auf 1 setzt.

Aufgabe 7: Schreibe die Prozedur drehen_links_heizung, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach links mit der Pausenzeit 0.01s vollziehen läßt und gleichzeitig den Ausgang für das Einschalten der Heizung auf 1 setzt.

Aufgabe 8: Schreibe die Prozedur drehen_schnell_pumpe, die den Schrittmotor im Halbschrittverfahren eine Umdrehung nach rechts mit einer möglichst kleinen Pausenzeit drehen läßt und gleichzeitig den Ausgang für das Einschalten der Pumpe auf 1 setzt.

Aufgabe 9: Schreibe die Prozedur schaukeln, die den Schrittmotor 50 Umdrehungen nach rechts und 50 Umdrehungen nach links vollziehen läßt. Benutze dazu die Zählschleife for zaehler := 1 to 50 do. Setze im Programmkopf unter der Zeile USES KOMAMPEL; die Zeile VAR ZAEHLER:INTEGER; ein. Rufe jeweils die Prozeduren drehen_rechts bzw. drehen_links auf.

Aufgabe 10: Schreibe die Prozedur schaukeln_pumpe, die den Schrittmotor 50 Umdrehungen nach rechts und 50 Umdrehungen nach links vollziehen läßt. Benutze dazu die Zählschleife for zaehler := 1 to 50 do und rufe jeweils die Prozeduren drehen_rechts_pumpe bzw. drehen_links_pumpe auf.

Aufgabe 11: Schreibe die Prozedur schaukeln_heizung, die den Schrittmotor 50 Umdrehungen nach rechts und 50 Umdrehungen nach links vollziehen läßt. Benutze dazu die Zählschleife for zaehler := 1 to 50 do und rufe jeweils die Prozeduren drehen_rechts_heizung bzw. drehen_links_heizung auf.

Aufgabe 12: Schreibe die Prozedur vorgang_aus, die alle Ausgänge auf 0 setzt.

Aufgabe 13: Schreibe die Prozedur pumpe_ein, die den Ausgang für die Pumpe auf 1 setzt.

Aufgabe 14: Schreibe die Prozedur ventil_auf, die den Ausgang für das Wasserventil zum Einlassen des Wassers auf 1 setzt.

Aufgabe 15: Schreibe die Prozedur Heizung_an, die den Ausgang für das Einschalten der Heizung auf 1 setzt.

Aufgabe 16: Schreibe die Prozedur waschmittel_holen, die den Ausgang für das Einspülen des Waschmittels auf 1 setzt.

6.4 Die Prozeduren für die Waschgänge

6.4.1 Die Prozedur zum Einlassen des Wassers

Mit `sound (Zeit)` kann man einen Ton im Lautsprecher des Computers erzeugen, mit `no-sound` stellt man ihn wieder ab. Hier ist die Prozedur für das Geräusch des Wassereinlassens.

```
procedure wasser_einlassen;
begin
  if schalter then
    ventil_auf
      for zaehler := 0 to 30000 do
          begin
             if schalter then
                sound(random(5000)+2200);
                warte (0.001);
          end;
       nosound;
end;
```

Aufgabe: Übertrage die obige Prozedur in das Programm WASCHMAS.PAS hinter die Prozedur ventil_auf. Füge im Programmkopf hinter USES KOMAMPEL, CRT; ein. Probiere die Prozedur aus.

6.4.2 Die Prozedur Waschmittel einspülen

Diese ist genauso aufgebaut wie die Prozedur wasser_einlassen; Die Zählschleife ist aber nur 1/3 so lang.

Aufgabe: Füge die Prozedur waschmittel_einspuelen hinter die Prozedur wasser_einlassen ein.

6.4.3 Die Prozedur für die Vorwäsche

Die Prozedur berücksichtigt zunächst, daß nur dann Wasser eingelassen werden kann, wenn die Trommeltür der Waschmaschine geschlossen ist. Nun wird die Prozedur ventil_auf aufgerufen. Danach folgt der Aufruf der Prozedur wasser_einlassen. Jetzt soll 3 mal geschaukelt werden. Da in der Prozedur Schaukeln die Zählschleife die Variable Zähler benutzt, wird unter der Zeile mit dem Prozedurnamen die Variable zaehler_vorwaschen deklariert. Auch die Zählschleife berücksichtigt, daß nur dann Wasser eingelassen werden kann, wenn die Trommeltür der Waschmaschine geschlossen ist. Das Wassereinlassen wird durch Aufruf der Prozedur ausschalten beendet. Nun wird mit einer Zählschleife bis 6 die Wäsche geschaukelt. Danach wird die Pumpe eingeschaltet, damit das Schmutzwasser abgepumpt werden kann. Während diese Vorgangs soll 3 mal geschaukelt werden. Mit dem Ausschalten der Pumpe wird die Vorwäsche beendet.

Aufgabe 1: Schreibe die Prozedur Vorwaschen und rufe sie im Hauptprogramm auf, um sie zu überprüfen.
Aufgabe 2: Füge mit der Anweisung writeln (' Vorwaschen'); eine Bildschirmausgabe ein.
Aufgabe 3: Füge an den entsprechenden Stellen ebenfalls ein:
Ventil öffnen
Wassereinlassen
Ventil schließen
Pumpe einschalten
Lauge abpumpen

6.4.4 Die Prozedur Waschen

Die Prozedur Waschen enthält folgende Anweisungen bzw. Prozeduraufrufe:

- Trommeltür muß zu sein
- Ventil auf
- Schaukeln 3 mal
- Waschmittel ein
- Schaukeln 3 mal
- Waschmittel aus
- Ventil zu
- Heizung ein
- Schaukeln 10 mal
- Heizung aus
- Pumpe ein
- Schaukeln 3 mal
- Pumpe aus

Aufgabe 1: Schreibe die Prozedur waschen und überprüfe sie im Hauptprogramm, indem du den Prozeduraufruf vorwaschen durch Einklammerung als Kommentierung (vorwaschen*) unterdrückst.*
Aufgabe 2: Füge folgende Bildschirmausgaben ein:
Ventil öffnen, Wasser einlassen
Ventil schließen
Ventil öffnen, Waschmittel einspülen
Ventil schließen
Heizung einschalten, Wasser erwärmen
Heizung ausschalten
Pumpe einschalten, Lauge abpumpen

6.4.5 Die Prozedur Spülen

Sie enthält folgende Elemente:

- Ventil auf
- Schaukeln 3 mal
- Ventil zu
- Schaukeln 6 mal
- Pumpe ein
- Schaukeln 3 mal
- Pumpe aus

Aufgabe 1: Schreibe die Prozedur spülen mit den obigen Elementen.
Aufgabe 2: Füge folgende Texte für eine Bildschirmausgabe ein:
Ventil öffnen
Ventil schließen
Pumpe einschalten
Aufgabe 3: Teste die Prozedur aus, indem du die davor liegenden Prozeduraufrufe auskommentierst.

6.4.6 Die Prozedur Schleudern

Dieser Vorgang hat im Prinzip folgende Struktur:

- Pumpe ein
- Motor langsam
- Motor schnell
- Motor langsam
- Pumpe aus

Aufgabe: Schreibe die Prozedur Schleudern. Füge sinnvolle Bildschirmausgaben ein. Teste sie aus, indem die davor liegenden Prozeduren auskommentierst.

6.4.7 Das Hauptprogramm

Das Hauptprogramm kann z.B. folgende Anweisungen besitzen:

- vorwaschen
- waschen
- spülen
- schleudern

Aufgabe: Führe einen kompletten Waschvorgang durch.

7 Manipulator

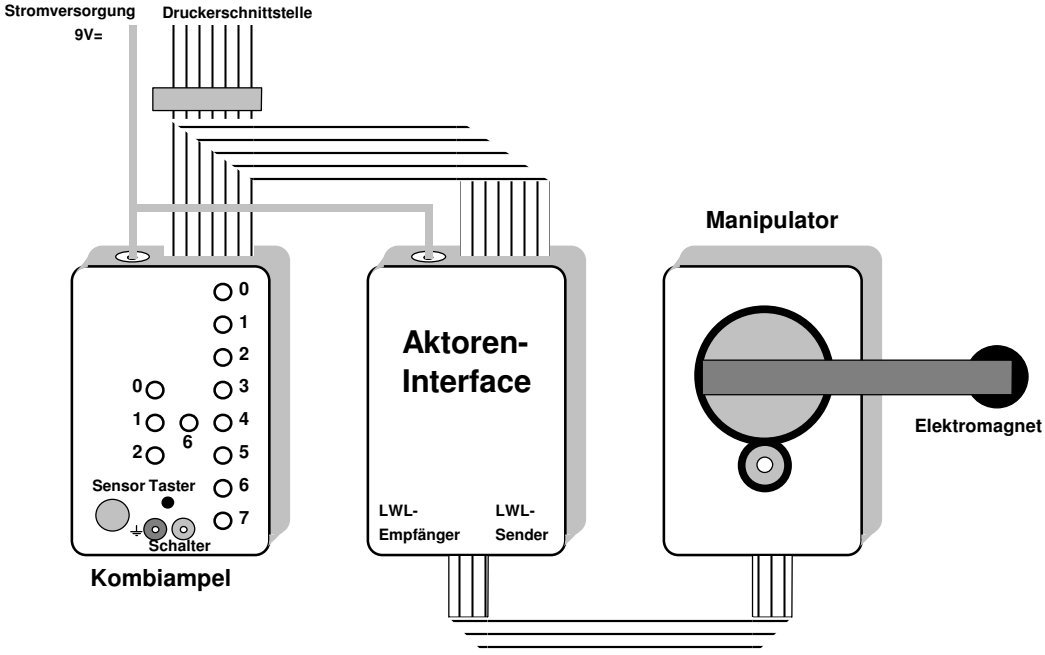


Abbildung 7-1: Anordnung des Manipulators

7.1 Aufbau des Manipulators

Das Manipulator-/Waschmaschinenmodell arbeitet als Manipulator, wenn der Manipulatorarm und das kleine Zahnrad montiert werden. Der Schrittmotor wird über die Ausgänge A4...A7 gesteuert.

7.2 Die Drehbewegungen des Manipulators

Damit der Manipulator eine Position anfahren kann, muß der Dreharm bewegt werden. Damit ein Groschen aufgenommen werden kann, muß zusätzlich zur Drehbewegung der Elektromagnet angesprochen werden.
Das Hinzuschalten der Kombiampel hilft, die gestellten Aufgaben besser zu bewältigen.

Aufgabe 1: Schreibe die Prozedur drehen_rechts und teste sie im Programm Manipulator. Speichere das Programm unter MANIPUL.PAS ab.
Aufgabe 2: Schreibe die Prozedur drehen_links.
Aufgabe 3: Schreibe die Prozedur drehen_links_magnet_ein.
Aufgabe 4: Schreibe die Prozedur drehen_rechts_magnet_ein.
Aufgabe 5: Schreibe die Prozedur Magnet_aus.

7.3 Positionssteuerung

7.3.1 Eine einfache Positionssteuerung durchführen

Sind einmal die Prozeduren zur Bewegung des Manipulatorarms und zur Aktivierung seines Elektromagneten vorhanden, ist die Positionierung des Elektromagneten in einem Hauptprogramm schnell vollzogen. Der Manipulator ist so konstruiert, daß der Groschen, der transportiert werden soll, auf einen freien Platz der Kombiampel oder des Gehäuses des Aktoreninterfaces liegen kann. Dazu muß die Kombiampel lediglich parallel links oder rechts neben den Manipulator gelegt werden. Der Elektromagnet kann dann einen Groschen ablegen bzw. aufnehmen. Der Manipulator soll im einfachsten Fall folgende Aufgabe bewältigen, wenn er von einer bestimmten Position aus gestartet wird:
(Zuvor mit der Hand den Manipulatorarm in Ausgangsposition über das kleine Zahnrad drehen)

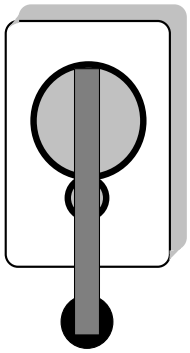


Abbildung 7-2: Manipulator in Ausgangsposition

- eine 1. Position anfahren
- einen Groschen aufnehmen
- eine 2. Position mit dem Groschen anfahren
- den Groschen ablegen
- ohne Groschen die 1. Position anfahren
- ohne Groschen wieder die 2. Position anfahren
- den Groschen aufnehmen
- usw.
- nach Beendigung der Aufgabe alles abschalten.

Aufgabe 1: Ermittle die Anzahl der Wiederholungen für den Aufruf der Prozedur drehen links, die nötig sind, um von der Ausgangsposition einen bestimmten Punkt anzufahren.

Aufgabe 2: Ergänze das Programm so, daß der Manipulator wieder an die Ausgangsposition zurückfährt.

Aufgabe 3: Überprüfe die Mechanik des Manipulators, ob die beiden Positionen auch über einen längeren Zeitraum sicher angefahren werden.

Aufgabe 4: Ergänze das Hauptprogramm so, daß ein Groschen transportiert wird. Nach Beendigung des Programms sollen alle Ausgänge abgeschaltet werden.

7.3.2 Die Startposition suchen

Der Manipulator ist so konstruiert, daß sein Elektromagnet sowohl einen Groschen aufnehmen bzw. ablegen als auch den Sensor verdunkeln kann, sofern die Kombiampel z.B. parallel links oder rechts neben den Manipulator gelegt wird.

Für die Sensorabfrage durch Verdunklung mit Hilfe des Elektromagneten gibt es drei Möglichkeiten von der Startposition aus (Der Manipulatorarm steht über dem kleinen Zahnrad): der Elektromagnet steht vor, auf oder hinter dem Sensor.
Somit gilt es zunächst einmal abzufragen, ob der Sensor durch den Elektromagneten verdunkelt ist. Ist dies nicht der Fall, kann dem Rechner mit Hilfe des Tasters mitgeteilt werden, daß der Elektromagnet vor dem Sensor steht. Andernfalls teilt die Betätigung des Schalters dem Rechner mit, daß der Elektromagnet hinter dem Sensor steht, der Schrittmotor den Manipulatorarm also in die andere Richtung drehen muß.

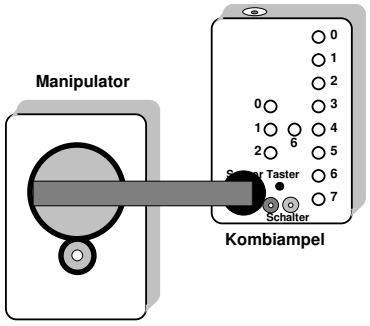


Abbildung 7-3: Anordnung zur Erkennung der Startposition

Aufgabe 1: Schreibe die Prozedur finde_startposition.
Aufgabe 2: Ergänze das Programm MANIPUL.PAS um die Suche der Startposition.

7.4 Den Manipulator über die Tastatur steuern

Der Programmkopf für das Programm MANIHAND.PAS berücksichtigt zwei Variablen.

```
program manipulator_handsteuerung;
uses komampel, crt;
var zeichen : char;
    zaehler : integer;
```

Das Hauptprogramm gibt zunächst die Bezeichnung und die Aufgabe der in Frage kommenden Tasten des Ziffernblocks auf dem Bildschirm aus. (Das Tastenfeld muß auf Ziffernbetrieb eingestellt sein, deshalb also die Num-Taste drücken)

```
begin
  lauflicht_ein;
  writeln ('Taste 8 : Magnet ein');
  writeln ('Taste 2 : Magnet aus');
  writeln ('Taste 6 : rechts drehen, Magnet aus');
  writeln ('Taste 4 : links drehen, Magnet aus');
  writeln ('Taste 3 : rechts drehen, Magnet ein');
  writeln ('Taste 1 : links drehen, Magnet ein');
  writeln ('Taste 0 : quit');
```

Danach folgen die Tastaturabfragen und die zugeordneten Anweisungen:

```
 repeat
  if keypressed then
(* Abfrage, ob eine Taste gedrückt wird *)
    begin
      zeichen := readkey;
(* Gedrückte Taste der Variablen zeichen zuordnen *)
      if zeichen = '4' then
        begin
          drehen_links_magnet_aus;
```

```
          ausgeben (0);
        end;
            .
            .
            .
    end;
 until zeichen = '0';
(* Taste 0 beendet die repeat-until-Schleife *)
end.
```

Aufgabe 1: Schreibe das Programm MANIHAND.PAS für die Handsteuerung. Ergänze dazu die fehlenden Befehle. Du kannst die Prozeduren zur Bewegung des Manipulators aus dem Programm MANIPUL.PAS übernehmen.

7.5 Teach-in mit einem Rechner

7.5.1 Lernen von Anweisungen

Ergänzt man das Programm MANIHAND.PAS um einige Anweisungen, so können die Anweisungen bei dem Betrieb des Manipulators mit Hilfe der Tasten gelernt und später automatisch ausgeführt werden. Ein solcher Vorgang heißt in der Robotertechnik Teach-In
Das Programm MANIHAND.PAS wird zunächst unter dem Namen MANTEACH.PAS gespeichert.

Zuerst wird der Programmkopf um die Variable datei : text; ergänzt:

```
program manipulator_teach_in_schreiben;
uses komampel, crt;
var zeichen : char;
    zaehler : integer;
    datei   : text;
```

Die Prozeduren bleiben vollständig erhalten. Im Hauptprogramm werden vor der repeat ... until-Schleife folgende Ergänzungen durchgeführt, sie sind jeweils kommentiert:

```
assign (datei, 'Liste.txt');
  (* Der Dateivariablen datei wird der Dateiname
  'Liste.txt'zugewiesen, unter dem  sie auf der Festplatte
   gespeichert wird *)
rewrite (datei);
  (* Eine Datei zum Beschreiben wird geöffnet; eine bereits
   bestehende Datei auf der Festplatte wirdüberschrieben. *)

begin
 lauflicht_ein;
      writeln ('Taste 8 : Magnet ein');
      writeln ('Taste 2 : Magnet aus');
      writeln ('Taste 6 : rechts drehen, Magnet aus');
      writeln ('Taste 4 : links drehen, Magnet aus');
      writeln ('Taste 3 : rechts drehen, Magnet ein');
      writeln ('Taste 1 : links drehen, Magnet ein');
      writeln ('Taste q : quit');
      assign (datei, 'Liste.txt');
      rewrite (datei);
(* Die Datei 'Liste.txt' wird neu angelegt *)
 repeat
  if keypressed then
    begin
      zeichen := readkey;
      write  (datei,zeichen);
(* Jede gelesene Taste wird Zeichen für Zeichen in die
   Datei 'Liste.txt' abgespeichert. *)
```

```
        write (zeichen);
(* gibt Zeichen auf dem Bildschirm aus*)
        if zeichen = '4' then
         begin
           drehen_links_magnet_aus;
           ausgeben (0);
         end;
            .
            .
            .
    end;
 until zeichen = 'q';
 close (datei);
(* Nach Abschluß der Tastaturabfrage wird die Datei 'Liste.txt'
geschlossen. *)
end.
```

7.5.2 Ausführen gelernter Anweisungen

Das Programm MANILISE.PAS, das die aufgezeichneten Befehle für den Manipulator automatisch ausführt, unterscheidet sich nur wenig vom Lernprogramm MANTEACH.PAS.

```
program manipulator_teach_in_lesen;
uses komampel, crt;
var zeichen : char;
    zaehler : integer;
    datei   : text;
```

Auch hier bleiben die Prozeduren vollständig erhalten. Das Hauptprogramm, die Änderungen sind kommentiert, wird wie folgt geändert:

```
begin
  lauflicht_ein;
  assign (datei, 'Liste.txt');
  reset (datei);
(*  Die Datei 'Liste.txt' wird zum Lesen geöffnet. *)
  repeat
    begin
      read (datei,zeichen);
(* Zeichen der Datei Liste.txt' lesen *)
      write (zeichen);
(* Gibt Zeichen auf dem Bildschirm aus. *)
       if zeichen = '4' then
         begin
           drehen_links_magnet_aus;
           ausgeben (0);
         end;
            .
            .
            .
    end;
 until zeichen = 'q';
 close (datei);
(* Auch nach dem Lesen schließt man die Datei. *)
end.
```

Aufgabe: Vervollständige dieses Programm und führe eine Transportaufgabe mehrfach durch.

7.6 Transport mit mehreren Manipulatoren

Links oder rechts parallel zum Manipulator wird eine Kombiampel gelegt. Legt man parallel zu dieser Kombiampel einen weiteren Manipulator, der von einem zweiten Rechner gesteuert wird, so läßt sich ein Groschen, der von einem Rechner mit Manipulator bewegt wurde, von einem zweiten Manipulator weiter transportieren.

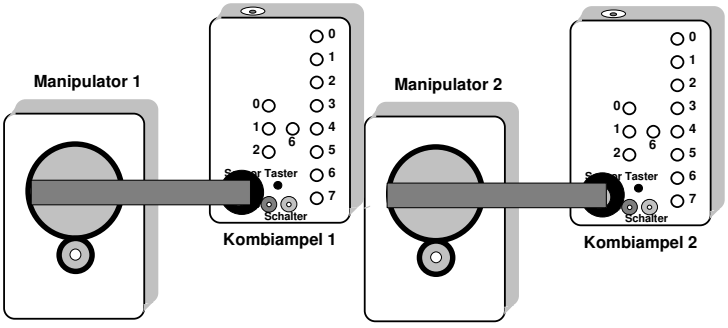


Abbildung 7-4: Transport mit zwei Manipulatoren

Zusätzlich müssen, damit ein Datenaustausch zwischen beiden Rechnern ermöglicht wird, beide Lichtwellenleiter der beiden Aktoren für die Kombiampel sende- und empfangsmäßig miteinander verbunden werden.
Wenn der erste Manipulator den Groschen abgelegt und den Arm wieder auf die Startposition zurückgereht hat, muss er dem zweiten Rechner über den Lichtwellenleiter ein Signalgeben, dass dieser jetzt den Groschen weitertransportieren kann.

Aufgabe 1: Schreibe ein Programm TRANSZWE.PAS, das einen Groschen von der Ablagefläche der Kombiampel des einen Rechners zur Ablagefläche der Kombiampel des anderen Rechners transportiert.

Aufgabe 2: Erweitere das Programm so, daß der Groschen vom zweiten Rechner weiter transportiert wird.

8 Codescheibenleser

8.1 Prinzipieller Aufbau

Das Modell eines Codescheibenlesers soll die prinzipielle Arbeitsweise einer Floppy-Disk, Hard-Disk, eines CD-Players, eines CD-ROM-Laufwerkes oder eines optischen Speicherssystems deutlich zu machen. Diese Geräte bestehen aus einem Motor für die Drehbewegung des Datenträgers und einem Antrieb für die Führung des Schreib- bzw. Lesekopfes. Ein Indexsystem übernimmt die Abstimmung der Drehbewegung mit der Leseeinrichtung.
Das Lesen der Informationen auf der Codescheibe und des Indexes, der an der Unterseite auf dem Codescheibenträger aufgebracht ist, übernehmen zwei Reflexlichtschranken. Bei dem Codescheibenleser erzeugt ein Schrittmotor die Rotationsbewegung der Codescheibe.

8.2 Die Codescheibe

Der Codescheibenleser liest Informationen, die auf einer Codescheibe in binärer Form aufgetragen wurden. Die Informationen sind auf vier Spuren verteilt, jede Spur ist in 32 Felder unterteilt, die jeweils ein Bit enthalten.

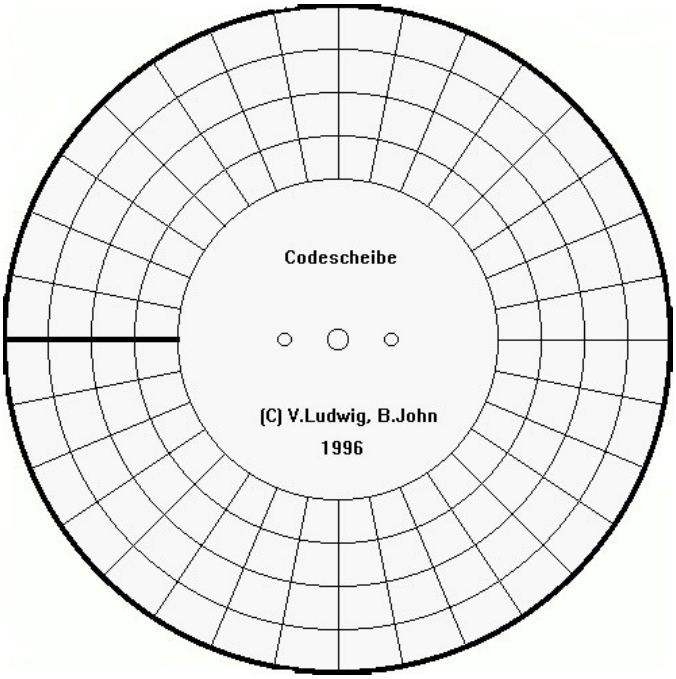


Abbildung 8-1: leere Codescheibe

Die Abbildung 8-1 zeigt eine unbeschriebene Codescheibe. Zu erkennen sind 4 Spuren mit 32 Feldern.

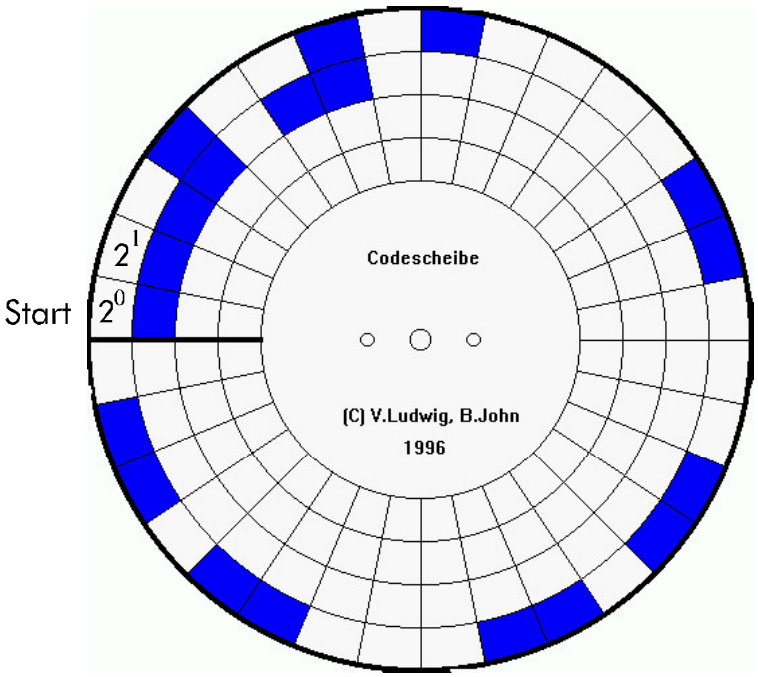


Abbildung 8-2: Codescheibe mit dem Wort "hallo"

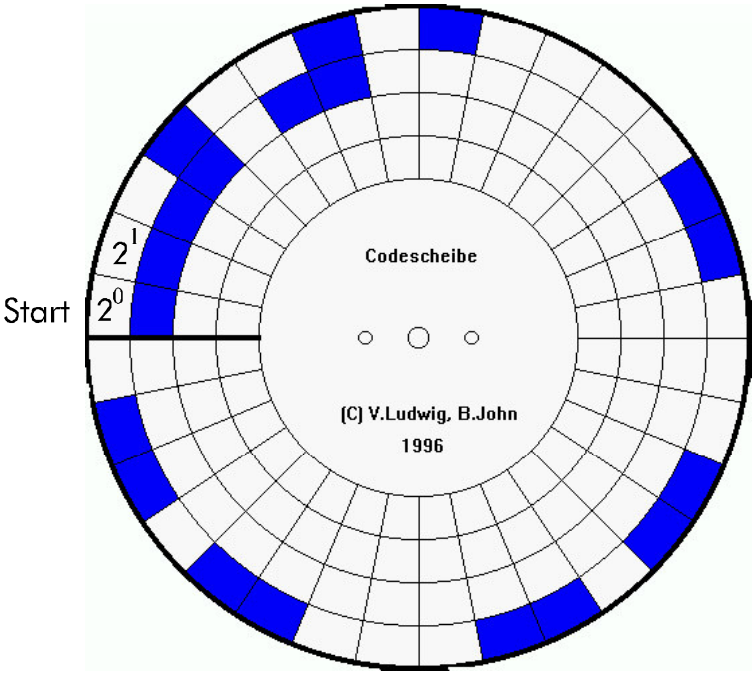


Abbildung 8-2 zeigt eine beschriebene Codescheibe. Die Informationen beginnen an der Startlinie auf der äußeren Spur. Die Informationen sind im Uhrzeigersinn geschrieben. Der Buchstabe "h" von "hallo" hat das ASCII-Bitmuster "01001000". Null-Bits sind weiß, Eins-Bits sind schwarz, das Bit 2^0 ist zuerst geschrieben. Die Codescheiben können mit dem Windows-Programm CDROMP.EXE von der Diskette hergestellt werden. Es sind auch andere Spur bzw. Feldzahlen möglich.

8.3 Die Programmierung des Codescheibenlesers

Je nach Leistungsfähigkeit der Lerngruppe lassen sich drei Wege einer Partner- bzw. Gruppenarbeit mit einem Codescheibenleser einschlagen:

- Eine sehr leistungsfähige Lerngruppe nimmt das im Anhang wiedergegebene professionelle Programm als Vorbild. Hierfür wird für eine Gruppe ein Aktoreninterface mit einem Codescheibenleser benötigt.
- Eine weniger leistungsstarke Lerngruppe hat die Möglichkeit, das Programm schrittweise aufzubauen. Bei dieser Gruppe lassen sich wiederum zwei Schwierigkeitsgrade einstellen:
- In Partnerarbeit werden die Bewegungen der Schrittmotoren und das Lesen und Auswerten der Informationen nacheinander programmiert. Auch hier werden wieder ein Aktoreninterface und ein Codierscheibenleser benötigt.
- In Gruppen- bzw. Partnerarbeit werden die Aufgaben für die Bewegungen der beiden Schrittmotoren aufgeteilt. Eine Gruppe programmiert auf ihrem Rechner die Drehbewegung des Codescheibenträgers, eine zweite Gruppe übernimmt auf ihrem Rechner die Programmierung des Lesearms. Für diese Form der Arbeitsteilung von zwei Gruppen werden zwei Aktoreninterfaces und nur ein Codescheibenleser benötigt. Hier müssen die im Anhang beschriebenen vier Anschlüsse des Anschlußbausteins geändert werden.

8.4 Programmierprojekt Codescheibenleser im Team

8.4.1 Gruppe 1: Die Drehbewegung der Codescheibe

Der Schrittmotor für die Drehbewegung des Tellers wird durch die Ausgänge A0 - A3 angesteuert. Damit das Aktoreninterface eine Spur der Codescheibe lesen kann, muß die Codescheibe auf dem Teller des Codescheibenlesers um 360 Grad gedreht werden. Dies wird in der Prozedur drehen durch einen Zaehler verwirklicht.
Die Geschwindigkeit der Drehbewegung soll, um die Lesegeschwindigkeit zu vergrößern, auf Maximum eingestellt werden.

****

8.4.2 Gruppe 2: Die Drehbewegung des Lesearms

Das Programm zum Lesen von Codescheiben für die Gruppe 2 baut ebenfalls auf dem Programm SCHRFUNC.PAS auf (vgl. Steuerung von zwei Schrittmotoren S. 123). Es erhält den Namen Codescheibenleser_2 und wird unter dem Namen CODESCH2.PAS abgespeichert.
Da dem Codescheibenleser zwei Schrittmotoren zugrunde liegen, erhält die Function schrittvor den Namen armschrittvor und die Anweisung if phase= then schrittvor:= die Bezeichnung if armphase= then armschrittvor.
Der Schrittmotor für die Drehbewegung des Lesearms wird durch diie Ausgänge A0 - A3 angesteuert.
Die Geschwindigkeit der Drehbewegung des Lesearms sollte überschaubar bleiben.

```
program Codescheibenleser_2
uses kombiampel;

var
   armphase, spur : integer;

(*Taster      : Index *)
(* Sensor    : Lesekopf *)
(* Schalter :  Endanschlag *)

function armschrittvor : integer;
  begin
    if armphase = 1 then armschrittvor := 16;
    if armphase = 2 then armschrittvor := 48;
    if armphase = 3 then armschrittvor := 32;
    if armphase = 4 then armschrittvor := 96;
    if armphase = 5 then armschrittvor := 64;
    if armphase = 6 then armschrittvor := 192;
    if armphase = 7 then armschrittvor := 128;
    if armphase = 8 then armschrittvor := 144;
    armphase := armphase + 1;
    if armphase > 8 then armphase := 1;
  end;

function armschrittrueck : integer;
  begin
    if armphase = 1 then armschrittrueck := 16;
    if armphase = 2 then armschrittrueck := 48;
    if armphase = 3 then armschrittrueck := 32;
    if armphase = 4 then armschrittrueck := 96;
    if armphase = 5 then armschrittrueck := 64;
    if armphase = 6 then armschrittrueck := 192;
    if armphase = 7 then armschrittrueck := 128;
    if armphase = 8 then armschrittrueck := 144;
    armphase := armphase - 1;
    if armphase < 1 then armphase := 8;
  end;
function armhalt : integer;
begin
```

```
    if armhalt = 1 then armhalt := 16;
    if armhalt = 2 then armhalt := 48;
    if armhalt = 3 then armhalt := 32;
    if armhalt = 4 then armhalt := 96;
    if armhalt = 5 then armhalt := 64;
    if armhalt = 6 then armhalt := 192;
    if armhalt = 7 then armhalt := 128;
    if armhalt = 8 then armhalt := 144;
end;

procedure schritt(wert : integer ; pause : real;
begin
   ausgeben(wert);
   warte(pause);
end;
```

8.4.3 Den Anschlag anfahren

Der Lesekopf des Codescheibenlesers wird durch den zweiten Schrittmotor über die Ausgänge A0 - A3 bewegt. Damit der Lesearm bei jedem Lesevorgang aus einer undefinierten in eine definierte Position gebracht wird, muß er zunächst bis an den Anschlag zurückgedreht werden. Das Zurückfahren des Lesearms geschieht mit Hilfe der Funktion armschrittrueck.

Aufgabe: Füge die Funktion armschrittrueck in das Programm Codescheiben_lesen ein. Benutze die Begriffe armphase bzw. armschrittrueck. Deklariere im Programmkopf die Variable armphase als integer.

Mit Hilfe der Prozedur procedure_an_den_Anschlag , die im Hauptprogramm aufgerufen werden muß, wird der Lesearm so lange zurückgefahren, bis der Kontakt zwischen dem Winkelstück unter dem blauen Zahnrad des Lesearms und dem Bolzen auf der Oberseite des Schrittmotors für den Lesearm geschlossen ist. Dieser Kontakt wird durch den Schaltereingang gelesen.

```
procedure an_den_Anschlag;
  begin
    repeat
      schritt(armschrittvor,0.05);
    until Schalter or Tastatur;
 end;
```

Das Hauptprogramm stellt armphase jeweils auf die erste Phase ein:

```
begin
  lauflicht_ein;
  armphase := 1;
  an_den_Anschlag;
  ausgeben(0);
end.
```

8.4.4 Spur 1 anfahren

Der Lesekopf soll vom Anschlag beginnend bis zur ersten Spur im Halbschrittverfahren mit geringem Tempo gefahren werden.

Aufgabe 1: Füge die Funktion armschrittvor in das Programm Codescheiben_leser2 ein. Benutze die Begriffe armphase bzw. armschrittvor.
Aufgabe 2: Ermittle die Anzahl der Schritte bis zur Spur 1.

Die Prozedur procedure Spur1_anfahren bringt den Lesekopf über die erste Spur.

```
procedure Spur1_anfahren;
var z : integer;
begin
  for zaehler := 1 to 42 do
    begin
      schritt(armschrittrueck,0.05);
    end;
end;
```

Aufgabe 3 : Überprüfe, ob der Lesekopf über die erste Spur geführt wird

Das Hauptprogramm wird um den Aufruf Spur1_anfahren ergänzt.

```
begin
  lauflicht_ein;
  tellerphase := 1;
  armphase := 1;
     an_den_Anschlag;
     Spur1_anfahren;
     ausgeben(0);
  end.
```

8.5 Index suchen

Bei der Inbetriebnahme des Codescheibenlesers befindet sich der Drehteller für die Codescheibe in einer undefinierten Lage. Wenn der Lesearm Spur 1 angefahren hat, befinden sich somit auch die Datenfelder auf der Spur 1 für den Lesekopf noch nicht in einer eindeutigen Position des Lesebeginns. Um diese Position festzulegen, muß der Index an der Unterseite des Drehtellers gelesen werden. Dies geschieht mit Hilfe des Tastereingangs.

Aufgabe 1: Überprüfe das einwandfreie Funktionieren des Tastereingangs.
Aufgabe 2: Füge die Prozedur index_suchen in das Programm ein und überprüfe, ob der Index gelesen wird.

```
procedure index_suchen;
  begin
      repeat
         if not Taster then
               write ('1');
            else write ('.');
     until not Taster or Tastatur;
  end;
```

Dem Hauptprogramm wird der Prozeduraufruf index_suchen zugefügt:

```
procedure index_suchen;
begin
  lauflicht_ein;
   tellerphase := 1;
   armphase := 1;
  an_den_Anschlag;
   Spur1_anfahren;
   index_suchen;
 end.
```

Wenn der Lesearm Spur 1 angefahren hat, muß zunächst der Teller von Hand um 360° gedreht werden. Wird der Index gelesen, so fährt der Lesearm wieder zurück.
Als Ergebnis des Lesevorgangs zeigt sich auf dem Bildschirm eine lange Folge von Punkten für den hellen Teil des Tellers und einer kurzen Sequenz von Einsen für den dunklen Index. Die Anzahl der Einsen hängt von der Geschwindigkeit der Drehbewegung des Tellers für die Codier-

scheibe und der Breite des Indexes ab. Auf dem Bildschirm ergibt sich, je nach Indexlage folgendes Bild:

... .111..... ...

8.6 Teamarbeit

Bis hierhin können beide Gruppen unabhängig voneinander ihre Aufgaben bewältigen. Um den auf dem Teller angebrachten Index jedoch während der rechnergesteuerten Drehbewegung des Tellers lesen und um den codierten Text lesen zu können, muß der Rechner mit dem Programm CODESCH1.PAS gestartet werden.

Aufgabe: Starte die Programme CODESCH1.PAS und CODESCH2.PAS und deute die Bildschirmausgabe.

Ist die Geschwindigkeit durch das Programm einmal festgelegt, so sollte die Anzahl der Einsen nach jeder Umdrehung gleich sein. Bei geringer Eindeutigkeit kann man durch Einstellen des Reglers für die Reflexlichtschranke des Index diesen Vorgang kalibrieren. (Vgl. Anhang).

Wenn das prinzipielle Funktionieren des Indexlesens gewährleistet war, muß die Prozedur procedure index_suchen für das Team umgeschrieben werden: Ist der Index gefunden, muß der Lesevorgang eingeleitet werden.

```
procedure index_suchen;
  begin
      repeat
         schritt(armhalt,0.01);
until
not Taster or Tastatur;
  end;
```

Um den Lesevorgang von Zeichen in Gang zu setzen, muß zunächst eine Codescheibe angefertigt werden. Da auch hier evtl. Kalibrierungsarbeiten vorgenommen werden müssen, ist es sinnvoll, eine Folge von 8 gleich großen hellen und dunklen Segmenten abzutasten.

Aufgabe 1: Berechne den Wert des Zeichens 10101010 und schlage in einer ASCII-Tabelle nach, welches Zeichen dahintersteckt.

Aufgabe 2: Berechne den Wert des Zeichens 01010101 und schlage in einer ASCII-Tabelle nach, welches Zeichen dahintersteckt.

Aufgabe 3: Stelle jeweils eine Codescheibe her.

8.7 Herstellen einer Codescheibe

Durch Aufruf des Programms CDROMP.EXE unter Windows gelangt der Benutzer in das Menü zur Herstellung von Codescheiben. Es soll eine Codescheibe mit 4 Spuren und 32 Sektoren gezeichnet werden. Auf dem Bildschirm erscheint eine unbeschriebene Codescheibe mit vier Spuren und 32 Sektoren. Durch Anklicken eines Feldes mit der Maus wird das jeweilige Feld gefärbt. Ein Laserdrucker ergibt ungewellte Codescheiben auch bei der Papierstärke 140g/m^2. Blätter mit Codescheiben, die mit einem Nadeldrucker oder Tintenstrahldrucker hergestellt worden sind, sollten auf möglichst starkem weißen Papier fotokopiert werden. Nach dem Ausdrucken der codierten Codescheibe wird die Codescheibe ausgeschnitten. Die beiden Löcher für die Aufnahme in die Justierschrauben des Codescheibentellers werden mit einem geeigneten Werkzeug ausgestanzt. Die aufgelegte Codescheibe muß mit Hilfe der beiden Muttern fixiert werden, da sonst der Durchmesser der beiden Löcher durch die Kräfte bei der Drehbewegung des Tellers vergrößert werden.

8.8 Lesen einer Spur

Wenn im Programm Codescheibenleser_1 Spur 1 angefahren worden ist, der Index erkannt worden ist und die Spur 1 gelesen werden soll, können in der Prozedur *procedure drehen;* die Pausen dazu genutzt werden, den Eingang Sensor erkennen zu lassen, ob das Feld schwarz bzw. weiß gefärbt ist.

Aufgabe 1: Überprüfe das einwandfreie Funktionieren des Sensoreingangs durch Aufruf der Sensorfunktion.

Aufgabe 2: Füge die Prozedur byte_lesen in das Programm ein. Berücksichtige eine Umdrehung der Codescheibe. Überprüfe, ob die markierten Felder der Codescheibe gelesen werden.

```
procedure lesen;
var z : integer;

begin
     for z:=1 to 200  do
       begin
          if not sensor then
          write ('1')
         else write ('.');
      end;
end;
```

Die Anzahl der gelesenen Zeichen hängt von der Geschwindigkeit der Drehbewegung und der Lesegeschwindigkeit ab.

8.9 Synchronisation des Lesevorgangs

8.10 Synchronisation über die Drehgeschwindigkeit der Codescheibe

Vgl. „Synchronisation über die Lesegeschwindigkeit“.
Damit die markierten Felder gelesen werden können, muß sich der Teller während des Lesevorgangs drehen. Dies geschieht mit Hilfe des Rechners der beteiligten Nachbargruppe. Bei dem bisherigen Stand des Programms wird bisher nur Spur 1 angefahren. Auf dem Bildschirm ergibt sich, je nach Geschwindigkeit der Drehbewegung, eine Folge von Punkten und Einsen:

... .1111....1111....1111....1111....1111 ...

Die Breite der Punkte bzw. Einsen ist auch abhängig von der Entfernung der Reflexlichtschranke vom Papier.

Aufgabe 3: Die Punkte und Einsen auf dem Bildschirm werden nach der ersten Umdrehung nahtlos dargestellt. Füge in das Programm an geeigneter Stelle einen Zeilensprung für den Bildschirm ein.

Aufgabe 4: Zähle die Anzahl der hellen und dunklen Felder auf dem Bildschirm. Stimmen sie mit der Anzahl auf der Codescheibe überein?

Aufgabe 5: Verändere die Geschwindigkeit der Drehbewegung des Codescheibentellers so, daß die Anzahl der hellen und dunklen Felder korrekt ist.

8.11 Synchronisation über die Lesegeschwindigkeit

Vgl. „Synchronisation über die Drehgeschwindigkeit der Codescheibe“.
Damit die markierten Felder gelesen werden können, muß sich der Teller während des Lesevorgangs drehen. Dies geschieht mit Hilfe des Rechners der beteiligten Nachbargruppe. Bei dem bisherigen Stand des Programms wird bisher nur Spur 1 angefahren. Auf dem Bildschirm ergibt sich, je nach Geschwindigkeit der Drehbewegung, eine Folge von Punkten und Einsen:

... .1111....1111....1111....1111....1111 ...

Die Breite der Punkte bzw. Einsen ist auch abhängig von der Entfernung der Reflexlichtschranke vom Papier.
Ist die Geschwindigkeit einmal festgelegt, so sollte die Anzahl der Einsen und Punkte je nach Sektor konstant sein. Ist dies nicht der Fall, hilft auch das Programm OSZI.PAS durch Darstellung der gelesenen Flanken auf dem Bildschirm festzustellen, ob die Reflexlichtschranke sauber arbeitet. Gegebenenfalls muß der Abstand des Lesekopfes zur Codescheibe durch Verändern des mit zwei Federn am blauen Zahnrad gehaltenen Lesearms variiert werden. Bei größeren Abweichungen kann man durch Einstellen des Reglers für die Reflexlichtschranke des Lesearms die Lesegenauigkeit optimieren. Nur bei einer waagrecht vollkommen glatt laufenden Codescheibe ergeben sich klare symmetrische Leseergebnisse. Geringe Abweichungen können jedoch in Kauf genommen werden, da sie durch das eigentliche Zeichenerkennungsprogramm ausgeglichen werden können.
Aufgabe 3: Die Punkte und Einsen auf dem Bildschirm werden nach der ersten Umdrehung nahtlos dargestellt. Füge in das Programm an geeigneter Stelle einen Zeilensprung für den Bildschirm ein.

Aufgabe 4: Zähle die Anzahl der hellen und dunklen Felder auf dem Bildschirm. Stimmen sie mit der Anzahl auf der Codescheibe überein?

Aufgabe 5: Verändere die Geschwindigkeit durch Einfügen einer geeigneten Warteschleife in der Prozedur PROCEDURE LESEN so, daß die Anzahl der hellen und dunklen Felder korrekt ist.

8.12 Wechseln auf die nächste Spur

Wenn Spur 1 gelesen worden ist und danach Spur 2 gelesen werden soll, muß Spur 2 angefahren werden. Dies geschieht, indem die Prozedur Armschrittvor vier mal aufgerufen wird.

```
procedure naechste_Spur_anfahren;
  var zaehler : integer;
  begin
    for zaehler := 1 to 4 do
      begin
        armschrittvor,0.01);
      end;
  end;
```

Sollen insgesamt 4 Spuren angefahren werden, wird das Hauptprogramm um eine repeat ... until-Schleife ergänzt.

```
begin
  lauflicht_ein;
  tellerphase := 1;
  armphase := 1;
  an_den_Anschlag;
  Spur1_anfahren;
  index_suchen;
  spur := 1;
   repeat
     naechste_Spur_anfahren;
     lesen;
     spur := spur + 1;
```

```
   until (spur > 4) or Tastatur;
  ausgeben(0);
end.
```

Wenn die markierten Felder sicher erkannt werden, wird nur ein Lesevorgang benötigt. Danach wird der Lesearm aus dem Tellerbereich gefahren. Somit lautet das endgültige Hauptprogramm:

```
begin
  lauflicht_ein;
  tellerphase := 1;
  armphase := 1;
        an_den_Anschlag;
        Spur1_anfahren;
        index_suchen;
        spur := 1;
        repeat
          lesen;
          naechste_Spur_anfahren;
          spur := spur +1;
        until (spur > 3) or tastatur;
      repeat lesen; until tastatur;
    ausgeben (0);
end.
```

8.13 Das Zeichenerkennungs- und Ausgabeprogramm

Werden Einsen und Punkte vom Programm sauber gelesen, kann das eigentliche Leseprogramm eingefügt werden. Es ist wie die vorangegangenen Testprogramme so aufgebaut, daß nach der Indexerkennung der erste Zeichen auf der ersten Spur gelesen wird und sein ASCII-Wert auf dem Bildschirm ausgegeben wird. Somit muß nur die Prozedur LESEN in die Prozedur ZEICHEN_LESEN geändert werden. Grundlage dieser Prozedur ist das Programm BYTEEMPFAENGER.PAS **(Fußnote).** Dies wurde um das Lesen des Startbits gekürzt, da das Erkennen des Zeichenanfangs durch das Lesen des Indexes gewährleistet ist.

```
procedure zeichen_lesen
var empfangsbyte, bitzaehler : integer;

begin
  empfangsbyte := 0;
  bitzaehler := 8;
  repeat
     empfangsbyte := empfangsbyte div 2;  (* Empfangene Bits um eine
Stelle*)
                                                                    (*
nach oben schieben                              *)

     if not sensor then
       begin
          empfangsbyte := empfangsbyte + 128;
       end;
     warte (0.195);
     bitzaehler := bitzaehler - 1;
 until bitzaehler = 0;
 writeln (empfangsbyte);
end.
```

Aufgabe 1: Ändere im Hauptprogramm den entsprechenden Prozeduraufruf und überprüfe das Programm.

Aufgabe 2: Stelle eine neue Codescheibe mit einem Text von maximal 16 Zeichen her und überprüfe, ob die Codescheibe dekodiert wird.

Aufgabe 3: Übertrage mit Hilfe eines Lichtwellenleiters den dekodierten Text auf den Rechner mit dem Steuerungsprogramm für den Codescheibenteller.

8.14 Dekodieren der Codescheibe mit nur einem Rechner

Will man das Dekodieren der Codescheibe einem Rechner übertragen, so müssen die Drehbewegung des Codescheibentellers und das Lesen der Zeichen gleichzeitig stattfinden.
Die Prozeduren für die Drehbewegung des Tellers für die Codescheibe und des Lesearms unterscheiden sich nicht von der Aufgabenstellung mit zwei Rechnern.:

```
program Codescheibenleser;
uses komampel;

var
  tellerphase, armphase : integer;
  spur : integer;

(* Taster   : Index *)
(* Sensor   : Lesekopf *)
(* Schalter : Endanschlag *)

function tellerschritt : integer;
  begin
    if tellerphase = 1 then tellerschritt := 1;
    if tellerphase = 2 then tellerschritt := 3;
    if tellerphase = 3 then tellerschritt := 2;
    if tellerphase = 4 then tellerschritt := 6;
    if tellerphase = 5 then tellerschritt := 4;
    if tellerphase = 6 then tellerschritt := 12;
    if tellerphase = 7 then tellerschritt := 8;
    if tellerphase = 8 then tellerschritt := 9;
    tellerphase := tellerphase - 1;
    if tellerphase < 1 then tellerphase := 8;
  end;

function armschrittvor : integer;
  begin
    if armphase = 1 then armschrittvor := 16;
    if armphase = 2 then armschrittvor := 48;
    if armphase = 3 then armschrittvor := 32;
    if armphase = 4 then armschrittvor := 96;
    if armphase = 5 then armschrittvor := 64;
    if armphase = 6 then armschrittvor := 192;
    if armphase = 7 then armschrittvor := 128;
    if armphase = 8 then armschrittvor := 144;
    armphase := armphase + 1;
    if armphase > 8 then armphase := 1;
  end;

function armschrittrueck : integer;
  begin
    if armphase = 1 then armschrittrueck := 16;
    if armphase = 2 then armschrittrueck := 48;
    if armphase = 3 then armschrittrueck := 32;
    if armphase = 4 then armschrittrueck := 96;
    if armphase = 5 then armschrittrueck := 64;
    if armphase = 6 then armschrittrueck := 192;
    if armphase = 7 then armschrittrueck := 128;
    if armphase = 8 then armschrittrueck := 144;
    armphase := armphase - 1;
    if armphase < 1 then armphase := 8;
  end;
```

```
procedure schritt(wert : integer;pause : real);
  begin
    ausgeben(wert);
    warte(pause);
  end;
```

Ohne gleichzeitige Drehbewegung des Tellers für die Codescheibe kommen auch die folgenden Prozeduren aus:

```
procedure an_den_Anschlag;
  begin
    repeat
      schritt(armschrittvor,0.05);
    until schalter or Tastatur;
 end;

procedure Spur1_anfahren;
var z : integer;
begin
  for z := 1 to 43 do
    begin
      schritt(armschrittrueck,0.01);
    end;
end;

procedure index_suchen;
  begin
    repeat
    zeichen_lesen;
      schritt(tellerschritt,0.01);
    until not Taster or Tastatur;
  end;
```

Jedoch muß die Prozedur LESEN in die Prozedur DREHEN_LESEN umgeschrieben werden.

```
procedure drehen_lesen;
var z : integer;
  begin
    for z:=1 to 200  do
      begin
        zeichen_lesen;
        schritt(tellerschritt,0.01);
      end;
  end;
```

Die Prozedur ZEICHEN_LESEN sorgt dafür, daß der Lesekopf die Zeichen liest und diese ausgibt.

```
procedure zeichen_lesen;
var empfangsbyte, bitzaehler : integer;

begin
  empfangsbyte := 0;
  bitzaehler := 8;
  repeat
    empfangsbyte := empfangsbyte div 2; (* Empfangene Bits um eine Stelle
*)
                                        (* nach oben schieben *)
    if not sensor then                  (* Datenbit einlesen *)
      begin
        empfangsbyte := empfangsbyte + 128;
      end;
      warte (0.01);
    bitzaehler := bitzaehler - 1;
  until bitzaehler = 0;
 writeln (chr(empfangsbyte));
```

```
end;
```

Startet man das Programm, so wird, bedingt durch die Laufzeit der Prozedur ZEICHEN_LESEN.PAS, die Drehbewegung für die Codescheibe rapide verlangsamt.
Dies liegt schon an der Ausgabe der gelesenen Werte. Die Wandlung der Werte zu Zeichen bedingt, daß eine zusätzliche Warteschleife eingebaut werden muß.
Somit ist es aus Zeitgründen günstiger, den Lese/Ausgabevorgang zunächst auf den Lesevorgang zu beschränken und die Ausgabe der Zeichen an das Programmende zu legen.
Dazu wird zunächst eine Variable als string am Programmanfang deklariert:

zeichenkette : string;

In der Prozedur ZEICHEN_LESEN.PAS wird die Anweisung zum Wandeln und Ausgeben des gelesenen Zeichens writeln (chr(empfangsbyte));
herausgenommen. An ihre Stelle tritt die Anweisung:

```
      zeichenkette := zeichenkette + chr(empfangsbyte);
```

Im Hauptprogramm werden die gelesenen Zeichen ausgegeben:

```
      writeln (zeichenkette);
```

Aufgabe: Starte das Programm und ermittle den richtigen Zeitablauf durch verändern des Wertes in der Anweisung warten ( ); in der Prozedur ZEICHEN_LESEN.PAS;

Zuletzt muß in der Prozedur NAECHSTE_SPUR_ANFAHREN berücksichtigt werden, daß sich auch hier der Teller für die Codescheibe weiter dreht.

```
procedure naechste_Spur_anfahren;
  var z : integer;
  begin
    for z := 1 to 4 do
      begin
        schritt(tellerschritt+armschrittrueck,0.01);
      end;
  end;
```

Das Hauptprogramm sieht somit wie folgt aus:

```
begin
  lauflicht_ein;
  tellerphase := 1;
  armphase := 1;
        an_den_Anschlag;
        Spur1_anfahren;
        index_suchen;
        spur := 1;
        repeat
          drehen_lesen;
          naechste_Spur_anfahren;
          spur := spur +1;
        until (spur > 4) or tastatur;
        an_den_Anschlag;
    repeat;until tastatur;
        ausgeben (0);
end.
```

9 Anzeigeelemente für alphanumerische Zeichen

Die 7-Segment-/Punktmatrixanzeige wird parralel zur Kombiampel an das Flachbandkabel[11] angeschlossen und mit der Stromversorgung verbunden. Durch einfaches Wechseln der 7-Segmentanzeige bzw. der Punktmatrix lassen sich zwei Anzeigesysteme herstellen.
Hierbei stellt die 7-Segmentanzeige das didaktisch einfache System mit einer einfachen Ansteuerung und dem Nachteil einer vollwertigen Anwendung nur auf den numerischen Bereich dar.
Dagegen ist die Punktmatrix in der Lage, alle gängigen alphanumerischen Zeichen anzuzeigen. Dies erfordert jedoch eine aufwendiger Programmierung.

9.1 Die 7-Segment-Anzeige

Die 7-Segment-Anzeige wird in die entsprechende Fassung gesteckt, die Punktmatrix gegebenenfalls entfernen.
Die einzelnen Segmente sind mit den Buchstaben a,b,c,d,e,f,g bezeichnet. Mit Hilfe der vorhanden 7 Segmente können alle Ziffern des Zehnersystems und mit Einschränkungen, die Buchstaben des Alphabets dargestellt werden.

9.1.1 Zifferndarstellung mit der 7-Segment-Anzeige

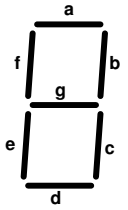


Abbildung 9-1: Sieben-Segment-Anzeige

Aufgabe 1: Übertrage die Anordnung der 7 Segmente in das Informatikheft.
Aufgabe 2: Ordne dem Segment a den Ausgabewert zu, der das Segment aktiviert. Notiere den Ausgabewert für das Segment a in deinem Informatikheft. Speichere das Programm unter SEGMEN_A.PAS ab.
Aufgabe 3: Verfahre so mit den restlichen Segmenten. Speichere die Programme unter SEGMEN_B.PAS bis SEGMEN_G.PAS ab.
Aufgabe 4: Setze auf dem Papier aus den vorhandenen Segmenten die Ziffer 1 zusammen. Dein Taschenrechner hilft dir bei der Darstellung.
Welche Ausgänge müssen gesetzt werden, damit die Ziffer 1 darstellt wird?
Schreibe ein Programm, das die Ziffer 1 darstellt. Schreibe den Ausgabewert als Summe. Speichere das Programm unter 7ZIFFER1.PAS ab.
Aufgabe 5: Stelle die restlichen Ziffern dar, indem der Wert des Ausgabebefehls als Summe geschrieben wird. Speichere die Programme unter 7ZIFFER2.PAS bis 7ZIFFER0.PAS ab.
Aufgabe 6: Führe einen Countdown durch und speichere das Programm unter 7COUNTDO.PAS ab.
Zusatz: Beende den Countdown mit einem Sound-Aufruf.
Aufgabe 7: Führe einen Countup durch und speichere das Programm unter 7COUNTUP.PAS
Aufgabe 8: Gestalte einen Würfel.

[11] Die Anzeigeschaltung kann auch direkt an der Druckerschnittstelle arbeiten, allerdings kann dann nicht auf die Anschauung der Ausgabebits zurückgegriffen werden.

9.1.2 Buchstabendarstellung mit der 7-Segment-Anzeige

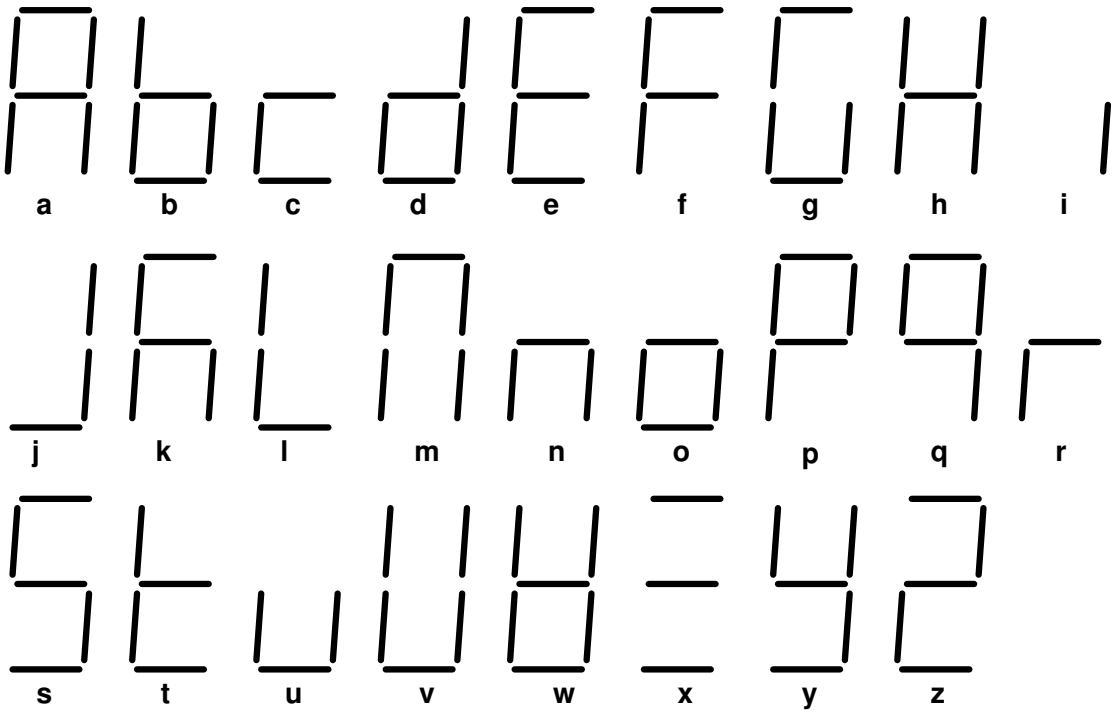


Abbildung 9-2: Buchstabenvorschlag für die 7-Segment-Anzeige

In der Abbildung ist ein Vorschlag zur Gestaltung des Alphabets mit Hilfe einer 7-Segmentanzeige dargestellt.

Aufgabe 1: Stelle mit Hilfe von Prozeduren die Buchstaben des Alphabets dar und speichere das Programm unter dem Namen 7BUCHSTA.PAS ab.
Aufgabe 2: Laß die 7-Segmentanzeige buchstabieren.
Aufgabe 3: Laß Deinen Namen ausgeben.

9.2 Die Punktmatrixanzeige

Die Punktmatrixanzeige wird in die entsprechende Fassung gesteckt, die 7-Segmentanzeige wird gegebenenfalls entfernt.
Das Gerät ist so konzipiert, daß es nach Anschluß an den Druckerausgang des Rechners und Anlegen der Betriebsspannung sofort einsetzbar ist. Wegen der komlexeren Programmierung empfiehlt sich in der Anfangsphase das Hinzuschalten der Kombiampel.

9.2.1 Die Ansteuerung der Punktmatrix entdecken

Der Anzeigebaustein besteht aus 7 Reihen und 5 Spalten von Leuchtdioden.

```
LED-Matrix
                                        LEDs der Kombiampel
Zeile           OOOOO  S                O O O O O O O O
                OOOOO  p                O O O O O O O O
                OOOOO  a                O O O O O O O O
                OOOOO  l                O O O O O O O O
                OOOOO  t                O O O O O O O O
                OOOOO  e                O O O O O O O O
                OOOOO                   O O O O O O O O
```

Im Anhang findet sich eine Kopiervorlage für einArbeitsblatt mit Matritzen für die folgenden Aufgaben.

Aufgabe 1: Die Punktmatrix besteht aus 7 Zeilen und 5 Spalten. Numeriere die Zeilen, beginnend oben mit Z_0 und ebenso die Spalten, beginnend rechts mit S_0.
Aufgabe 2: Schreibe das Programm LEDMATRI.PAS. Schalte die Kombiampel auf Lauflicht. Schalte zunächst alle Ausgänge ein. Verwende zum Ausgeben den Befehl `binaus ('11111111');` *Markiere im Arbeitsblatt die aktivierten LEDs.*
Aufgabe 3: Verändere den Wert von `binaus (`11111111´);` *Finde heraus, welche Ausgänge die Auswahl der Zeilen und welche Ausgänge die Auswahl der Spalten ermöglichen.*
Aufgabe 4: Welcher binaus-Befehl bringt die LED im Zentrum der Punktmatrix zum Leuchten?
Aufgabe 6: Formuliere das Ergebnis: Die Ausgänge A..... - A..... steuern die Zeilen. Die 5 Spalten werden von den Ausgängen A..... - A..... aktiviert.
*Aufgabe 7: Ersetze den Befehl binaus (‘01101111`); durch dern Befehl ausgeben (Deziamalwert); Vergleiche den Dezimalwert mit dem Wert von ausgeben (32*Spaltenanzahl+Zeilenanzahl).*
Aufgabe 8: Schreibe für die Punktmatrix ein Lauflichtprogramm. Speichere es unter PUNKTMAT.PAS ab.

9.2.2 Prinzip der Zeichendarstellung

Das Bitmuster auf A5 ...A7 für das Ansprechen der Zeilen sieht wie folgt aus:

A7	A6	A5
0	0	**1**
0	**1**	0
0	**1**	**1**
1	0	0
1	0	**1**
1	**1**	0
1	**1**	**1**

Um ein X darzustellen, müssen bestimmte LEDs der Punktmatrixanzeige leuchten.
Das Bitmuster A0...A4 der Zeilen für ein X wird z.B.so festgelegt:

A4	A3	A2	A1	A0
1	0	0	0	**1**
1	0	0	0	**1**
0	**1**	0	**1**	0
0	0	**1**	0	0
0	**1**	0	**1**	0
1	0	0	0	**1**
1	0	0	0	**1**

Fügt man beide Bitmuster für A0 bis A7 zusammen, ergibt sich folgendes Bild:

A7	A6	A5	A4	A3	A2	A1	A0
0	0	**1**	**1**	0	0	0	**1**
0	**1**	0	**1**	0	0	0	**1**
0	**1**	**1**	0	**1**	0	1	0
1	0	0	0	0	**1**	0	0
1	0	**1**	0	**1**	0	**1**	0
1	**1**	0	**1**	0	0	0	**1**
1	**1**	**1**	**1**	0	0	0	**1**

Die Experimente aus dem letzten Abschnitt zeigen, dass immer nur *eine* Zeile von LEDs leuchtet. Um mit Hilfe der Punktmatrix ein ganzes Zeichen auszugeben, muß also dafür gesorgt werden, daß während der Darstellung die Zeilen nacheinander immer wieder schnell genug durchlaufen werden. Das geschieht durch die Ausgänge A5-A7. Bei jedem Zeilendurchlauf müssen die jeweils beteiligten LEDs angesprochen werden. Dafür sorgen die Ausgänge A0 - A4.

```
program Ledmatrix_zeichen_x;
uses komampel;

begin
lauflicht_ein;
while not tastatur do
    begin
      binaus('00110001');
      binaus('01010001');
      binaus('01101010');
      binaus('10000100');
      binaus('10101010');
      binaus('11010001');
      binaus('11110001');
    end;
  end.
```

Aufgabe 1: Schreibe das Programm LED-Matrix_zeichen_x ab und speichere es unter dem Namen LEDMAT_X.PAS.

Aufgabe 2: Füge hinter jede binaus-Anweisung eine Warteschleife ein, damit das Multiplex-Ansteuerverfahren der LED-Matrix erkennbar wird. Ermittle die maximale Wartezeit, mit der man die Darstellung noch flackerfrei sehen kann.

Aufgabe 3: Schreibe ein Programm, das ein I ausgeben soll. Speichere es unter LEDMAT_I.PAS ab.

Aufgabe 4: Schreibe für die Buchstaben deines Namens Prozeduren und laß deinen Namen ausgeben. Speichere das Programm unter LEDMAT_N.PAS ab.

9.2.3 Eine andere Art der Zeichenprogrammierung

Um ein Zeichen auf die Punktmatrix ausgeben zu können, müssen die 7 Bitmuster für dieses Zeichen vorbereitet sein. Für die Vorbereitung eines Zeichens sind in Pascal einige Vorarbeiten notwendig. Zunächst wird ein allgemeiner Datenyp definiert, der aus einem Feld (array) mit 7 Einträgen besteht. Jeder Eintrag ist ein Bitmuster mit 8 Stellen für eine Zeile.

```
type
  zeichenmatrixtyp = array[1..7] of string[8];
```

Speziell für das Zeichen "X" wird nun eine Konstante vom Zeichenmatrixtyp mit dem Namen zeichen_x festgelegt, die die notwendigen Bitmuster enthält.

```
const
  zeichen_X : zeichenmatrixtyp = ('00110001',
                                  '01010001',
                                  '01101010',
                                  '10000100',
                                  '10101010',
                                  '11010001',
                                  '11110001');
```

Nun kann eine Zählschleife im Hauptprogramm den Inhalt der einzelnen Bitmuster des Zeichens auf die Punktmatrix ausgeben helfen:

```
var
  zaehler : integer;

begin
  zaehler := 1;
  while not tastatur do
    begin
      binaus(zeichen[zaehler]);(* zeichen[zaehler] holt aus dem Feld  *)
                               (* mit den Bitmustern die Zeile mit der*)
                               (* Nummer heraus, die in der Variablen  *)
                               (* zaehler steht. *)
      zaehler := zaehler + 1;
      if zaehler > 7 then zaehler := 0;
      (*warte(0.001);*)
    end;
end.
```

Aufgabe: Schreibe das Programm Zeichenausgabe mit dem Namen ZEICAUSG.PAS.

9.2.4 Einen Text mit der Tastatur ausgeben

Druckersteuerzeichen, die Zeichen aller Tasten einer Tastatur sowie Graphikzeichen sind in jeder ASCII-Zeichen-Tabelle aufgeführt. Jedem Zeichen ist ein dezimaler Wert zwischen 0 und 255 zugeordnet. Die Tastaturbelegung innerhalb der ASCII-Zeichen-Tabelle beginnt mit der Leertaste und endet mit der Entfernungstaste für ein Zeichen.

Aufgabe 1: Stelle eine ASCII-Zeichen-Tabelle für die Zeichen zwischen 32 und 127 auf. Wenn du die "Alt"-Taste drückst und gleichzeitig eine Zahl auf dem numerischen Zahlenblock eingibst, erschent auf dem Schirm das dazu gehörende Zeichen, sobald du die "Alt"-Taste losläßt.

Nun kann diese Tabelle in einen Datentyp "asciityp" übersetzt werden. Dazu verwendet man ein Feld mit Elementen zwischen 32 und 127 (array[32..127]). Jedes dieser Feldelemente enthält die Bitmuster zum jeweiligen Zeichen.

```
program textausgabe;

uses crt, komampel;

type
  zeichenmatrixtyp = array[0..6] of string[8];
  asciityp = array[32..33] of zeichenmatrixtyp;

const
  pause = 0.001;
  tastaturzeichen : alphabettyp =  (('00100000'  (* 32 : Leerstelle *)
                                    ,'01000000'
                                    ,'01100000'
                                    ,'10000000'
                                    ,'10100000'
                                    ,'11000000'
                                    ,'11100000')
                                                 (* 33 : ! *)
                                    ,('00100110'
                                    ,'01000110'
                                    ,'01100110'
                                    ,'10000110'
                                    ,'10100000'
                                    ,'11000110'
                                    ,'11100110'));



var
  zeichen : char;

procedure gib_zeichen_aus(zeichen : char);
  var
    z : integer;
  begin
    if zeichen in [#32..#127] then
      begin
        for z := 0 to 6 do
          begin
            binaus(tastaturzeichen[ord(zeichen)][z]);
            warte(pause);
          end;
        binaus('00000000');
      end;
  end;
```

```
begin
  zeichen := ' ';
  while zeichen >= ' ' do
    begin
      if tastatur then zeichen := readkey;
      gib_zeichen_aus(zeichen);
    end;
end.
```

Aufgabe 2: Überprüfe mit dem obigen vereinfachten Programm, ob ein "!" und ein Leerzeichen ausgegeben werden, wenn du sie auf der Tastatur eingibst.

Will man alle Zeichen ausgeben, so muss das Feld "Tastaturzeichen" auf die Einträge 32...127 erweitert werden.

```
type
  zeichenmatrixtyp = array[0..6] of string[8];
  asciityp = array[32..127] of zeichenmatrixtyp;

const
  pause = 0.001;
  tastaturzeichen : alphabettyp =  (('00100000'  (* 32 : Leerstelle *)
                                    ,'01000000'
                                    ,'01100000'
                                    ,'10000000'
                                    ,'10100000'
                                    ,'11000000'
                                    ,'11100000')
                                                  (* 33 : ! *)
                                   ,('00100110'
                                    ,'01000110'
                                    ,'01100110'
                                    ,'10000110'
                                    ,'10100000'
                                    ,'11000110'
                                    ,'11100110')
                                                   (* 34 : " * )
                                   ,('xxxxxxxx'
                                    ,'xxxxxxxx'
                                         .
                                         .
                                         .
                                                     (* 127 :   )
                                         .
                                    ,'xxxxxxxx'));
```

Aufgabe 3: Wie werden im Feld Tastaturzeichen die einzelnen Zeichen getrennt?
Aufgabe 4: Ergänze die Konstante Tastaturzeichen um die fehlenden Zeichen.
Aufgabe 5: Gib einen Text aus.
Aufgabe 6: Zeichne Bilder auf der Matrixanzeige.

10 Empfang der Atomzeit des Zeitzeichensenders DCF 77

10.1 Der Zeitsignal- und Normalfrequenzsender DCF77[12]

Standort:
Sendefunkstelle Mainflingen (50 01' Nord 09 00' Ost), etwa 25 km südöstlich von Frankfurt/Main. Das Steuersignal wird am Sendeort von Atomuhren der PTB abgeleitet und von Braunschweig aus kontrolliert.

Trägerfrequenz:
Normalfrequenz 77,5 kHz
relative Abweichung der Trägerfrequenz vom Nennwert im Mittel über:

$$1\text{Tag} <= 1 * 10^{-12}$$
$$100\text{Tage} <= 2 * 10^{-13}$$

Die Phasenzeit von DCF77 wird so nachgeregelt, daß sie am Sendeort näherungsweise (±0,3µs) mit UTC (PTB) in Übereinstimmung bleibt. Am Empfangsort beobachtete größere Phasen- bzw. Frequenzschwankungen werden durch die Überlagerung von Raum- und Bodenwelle hervorgerufen.

Leistung:
Senderleistung 50 kW, geschätzte abgestrahlte Leistung etwa 30 kW
Reichweite etwa 2000 km

Antenne:
150 m (im Falle der Aussendung mit Reserveantenne 200 m) hohe vertikale Rundstrahlantenne mit Dachkapazität.

Sendezeit:
24-h-Dauerbetrieb. Kurze Unterbrechungen (einige Minuten) sind möglich, wenn bei Störungen oder Wartungsarbeiten auf einen Reservesender oder eine Reserveantenne umgeschaltet werden muß. Bei Gewitter können auch längere Abschaltungen vorkommen.

Zeitsignale:
Der Träger wird amplitudenmoduliert mit Sekundenmarken: Zu Beginn jeder Sekunde (mit Ausnahme der 59. Sekunde jeder Minute) wird die Trägeramplitude für die Dauer von 0,1 s oder 0,2 s auf etwa 25% abgesenkt. Der Beginn der Trägerabsenkung ist der genaue Sekundenbeginn. Das Fehlen der 59. Sekundenmarke kündigt die nächstfolgende Minutenmarke an. Die Sekundenmarken sind phasensynchron mit dem Träger. Bedingt durch die Übertragung ist die Unsicherheit, mit der die Zeitpunkte der Sekundenmarken empfangen werden können, größer als die der steuernden Atomuhren. Die Ursachen dafür sind die geringe Bandbreite der Sendeantenne, Raumwelleneinflüsse und mögliche Interferenzen. Trotzdem sind beim Empfang der Sekundenmarken in Entfernungen bis zu einigen hundert Kilometern vom Sendeort Unsicherheiten der aufgenommenen Zeitpunkte von weniger als 0,1 ms erreichbar.

[12] Auszug aus der Broschüre "Zur Zeit" PTB Braunschweig Deutschland (mit Ergänzungen durch DJ1XK bei Codierschema-Tabelle)

Zeitcode:
Während jeder Minute werden die Nummern von Minute, Stunde, Tag, Wochentag, Monat und Jahr BCD-codiert durch Impulsdauermodulation der Sekundenmarken Übertragen. Dieses "Telegramm" gilt jeweils für die folgende Minute. Dabei entsprechen Sekundenmarken mit einer Dauer von 0,1 s der binären Null und solche mit einer Dauer von 0,2 s der binären Eins. Die Zuordnung der einzelnen Sekundenmarken auf die Übertragene Zeitinformation zeigt das Codierschema. Die drei Prüfbits P1, P2 und P3 ergänzen jeweils die vorhergehenden Informationswörter (7 Bits für die Minute, 6 Bits für die Stunde und 22 Bits für das Datum, einschließlich Nummer des Wochentags) auf eine gerade Anzahl von Einsen.

Codierschema:

Sekunde

0	M	Minutenmarke (0,1 s)
1	bis 14	frei
15	R	Sekundenmarke Nr. 15 hat die Dauer von 0,2 s, wenn die Aussendung über die Ersatzantenne erfolgt
16	A1	Ankündigung eines bevorstehenden Wechsels von MEZ auf MESZ oder umgekehrt
17	Z1	Zonenbit
18	Z2	Zonenbit
19	A2	Ankündigung einer Schaltsekunde
20	S	Startbit der kodierten Zeitinformation (0,2 s)
21	M1	Minute Wertigkeit 1
22	M2	Minute Wertigkeit 2
23	M4	Minute Wertigkeit 4
24	M8	Minute Wertigkeit 8
25	M10	Minute Wertigkeit 10
26	M20	Minute Wertigkeit 20
27	M40	Minute Wertigkeit 40
28	P1	Prüfbit für Minuten
29	S1	Stunde Wertigkeit 1
30	S2	Stunde Wertigkeit 2
31	S4	Stunde Wertigkeit 4
32	S8	Stunde Wertigkeit 8
33	S10	Stunde Wertigkeit 10
34	S20	Stunde Wertigkeit 20
35	P2	Prüfbit für Stunden
36	T1	Tag Wertigkeit 1
37	T2	Tag Wertigkeit 2
38	T4	Tag Wertigkeit 4
39	T8	Tag Wertigkeit 8
40	T10	Tag Wertigkeit 10
41	T20	Tag Wertigkeit 20
42	W1	Wochentag Wertigkeit 1
43	W2	Wochentag Wertigkeit 2
44	W4	Wochentag Wertigkeit 4
45	MO1	Monat Wertigkeit 1
46	MO2	Monat Wertigkeit 2
47	MO4	Monat Wertigkeit 4
48	MO8	Monat Wertigkeit 8
49	MO10	Monat Wertigkeit 16
50	J1	Jahr Wertigkeit 1
51	J2	Jahr Wertigkeit 2
52	J4	Jahr Wertigkeit 4
53	J8	Jahr Wertigkeit 8
54	J10	Jahr Wertigkeit 10
55	J20	Jahr Wertigkeit 20
56	J40	Jahr Wertigkeit 40
57	J80	Jahr Wertigkeit 80
58	P3	Prüfbit für Datum/ Wochentag

Sekunde 59 fehlt

Die Sekundenmarken Nr. 17 und 18 zeigen an, auf welches Zeitsystem sich die ausgesandte Zeitinformation bezieht. Bei Aussendung der MEZ wird die Sekundenmarke Nr. 18 auf 0,2 s verlängert; die Sekundenmarke Nr. 17 hat die Dauer von 0,1 s. Bei Aussendung der MESZ ist es umgekehrt. Vor einem Übergang von MEZ nach MESZ oder zurück wird außerdem jeweils eine Stunde lang die Sekundenmarke Nr. 16 als verlängerte Marke (0,2 s) ausgesendet. Diese Verlängerung beginnt beim Übergang von MEZ auf MESZ (von MESZ nach MEZ) um 01.00.16 Uhr MEZ (02.00.16 Uhr MESZ) und endet um 01.59.16 Uhr MEZ (02.59.16 Uhr MESZ). Die Sekundenmarke Nr. 19 kündigt eine Schaltsekunde an. Sie wird dann ebenfalls eine Stunde lang vor Einfügung der Schaltsekunde als verlängerte Marke (0,2 s) ausgesendet. Schaltsekunden werden weltweit zum gleichen Zeitpunkt in die Koordinierte Weltzeitskala UTC Eingeführt, vorzugsweise am Ende der letzten Stunde des 31. Dezember oder des 30. Juni. Dies bedeutet, daß Schaltsekunden in der gesetzlichen Zeit der Bundesrepublik Deutschland eine Sekunde vor 1 Uhr MEZ am 1. Januar oder vor 2 Uhr MESZ am 1. Juli eingeschoben werden. Bei einer Schaltsekunde am 1. Januar (1. Juli) beginnt

daher die Verlängerung der Sekundenmarke Nr. 19 um 00.00.19 Uhr MEZ (01.00.19 Uhr MESZ) und endet um 00.59.19 Uhr MEZ (01.59.19 Uhr MESZ). Beim Einfügen einer Schaltsekunde hat die zugehörige Minute eine Dauer von 61 Sekunden, und die der Marke 01.00.00 Uhr MEZ bzw. 02.00.00 Uhr MESZ vorhergehende 59. Sekundenmarke wird mit einer Dauer von 0,1 s ausgesendet. Die zur eingefügten 60. Sekunde gehörige Marke wird weggelassen (keine Trägerabsenkung).

10.2 Eine DCF77-Empfangsanlage im Informatikraum installieren

Die DCF77-Empfängerplatine (Fußnote: Bezug; Betrieb mit einem NPN-Transistor, der vom invertierten Ausgang betrieben wird.) benötigt zum Betrieb eine Stromversorgung zwischen 1,2 V und 15 V. Die Stromaufnahme beträgt 3 mA.. In Verbindung mit einem Rechner muß die Betriebsspannung 5 V betragen, da die Spannung für einen Eingang der Druckerschnittstelle des Rechners max. 5 V beträgt. Beim Betrieb einer Emfängerplatine mit einem Rechner kann somit die Stromversorgung aus dem Rechner mit Hilfe eines Ausgangs der Druckerschnittstelle vorgenommen werden: Die Öffnungen der 4 Schraubklemmen der DCF77-Empfängerplatine schauen den Betrachter an: Die Masseleitung der Druckerschnittstelle wird an Klemme 1 gelegt, der Ausgang A0 der Druckerschnittstelle (der liefert, wenn er aktiviert ist, +5V) an Klemme 2. An den nicht invertierenden Ausgang (Klemme 3) wird ein 10 kOhm-Widerstand geschraubt. Das andere Ende wird mit Klemme 2 verbunden. Der Eingang Ex der Druckerschnittstelle wird mit Klemme 3 verbunden.

Beim Betrieb einer Empfängerplatine mit mehreren Rechnern im Informatikraum sollte die Stromversorgung z.B. mit Hilfe einer 4,5V Flachbatterie vorgenommen werden. So kann ein vom Rechnerbetrieb unabhängiges Signal aus nur einem Empfänger allen Schülergruppen dauernd zur Verfügung stehen. In diesem Fall reicht der zur Verfügung stehende Ausgangsstrom von 1mA nicht aus. Somit muß ein Verstärker, bestehend aus zwei Widerständen und einem NPN-Transistor hinter den invertierenden Ausgang (Klemme 4) geschraubt/gelötet werden. Dazu wird ein 10kOhm Basiswiderstand an Klemme 4 geschraubt und an das mittlere Bein (Basis, es zeigt auf die Klemme 4) eines NPN-Transistors (z.B. BC 107) so gelötet, daß die flache Seite des Transistorkörpers (auf jeden Fall Transistor mit Plastikkörper verwenden)nach oben schaut. Nun wird das linke Bein (Emitter)des Transistors mit Klemme 1 mit Hilfe eines Drahtes und eines Lötkolbens verbunden. In Klemme 2 wird ein 1 KOhm-Widerstand geschraubt. Das andere Ende dieses Widerstandes wird an das rechte Bein des Transistors (Kollektor) gelötet. Hieran wird auch die Eingangsleitung für die Druckerschnittstelle gelötet.

Der DCF77-Empfänger reagiert zudem sehr empfindlich auf die Einstrahlung der Monitore im Informatikraum. Aus diesem Grunde ist es sinnvoll, die Empfängerplatine an eine störarme Stelle des Informatikraumes zu installieren (Wenn möglich, Südfenster, vom nächsten Monitor weit entfernt). Dabei wird die Ferritantenne so plaziert, daß sie mit ihrer Breitseite Richtung Frankfurt/Main zeigt. (Die Ferritantenne an die Schraubklemmen ankleben, dann bilden Ferritantenne und DCF77-Empfängerplatine eine Einheit). Mit Hilfe eines Stereo-Mikrofonkabels bei Stromversorgung aus dem Rechner oder eines 2-adrigen Kabels bei Betrieb mit einer Flachbatterie werden die einzelnen Rechnerplätze z.B. unter Verwendung von Lüsterklemmen mit dem DCF77-Signal erreicht.

10.3 Die Signale des DCF77-Senders auf der Kombiampel anzeigen

Der DCF77-Empfängerbaustein mit Ferritantenne hat eine Betriebsspannung von 1,2 - 15 Volt und eine Stromaufnahme von 3 mA. Somit kann die Stromversorgung direkt aus einer der 8 Ausgänge genommen werden.
Der DCF77-Empfänger ist so verdrahtet, daß Ausgang A0 die Stromversorgung übernimmt und der Schaltereingang den nicht invertierten Empfängerausgang liest.

Aufgabe 1: Schreibe ein Testprogramm, das den Ausgang A7 der Kombiampel ansteuert, wenn der DCF77-Empfänger ein empfangenes Signal liefert.

Wenn A7 dauernd leuchtet: Den DCF77-Empfänger möglichst weit vom Rechner entfernen, da dann Störsignale den Empfang beeinflussen.

Aufgabe 2: Beobachte die LED von Ausgang A7 über den Zeitraum von 60 Sekunden. Zähle die Anzahl der Impulse. Vergleiche die Länge der Impulse.

10.3.1 Den Startimpuls mit der Kombiampel finden

Wenn 58 statt 60 Impulse pro Minute gezählt werden, dann fehlen an einer Stelle
zwei Impulse. D.h. hier kann eine Pause von 2 Sekunden registriert werden. Das folgende Programm DCFSTART.PAS zeigt diese Zeitlücke innerhalb 1 Minute 1 mal an A7:

```
program DCF77start;
uses komampel;

begin
  lauflicht_ein;
  ausgeben(1);
  repeat
    if schalter then
      ausgeben (129)          (* DCF77 low auf A7 anzeigen *)
      else ausgeben (65);    (* DCF77 high auf A6 anzeigen *)
    until tastatur;
    ausgeben (0);
end.
```

10.3.2 Die Signale von DCF77 auf dem Bildschirm anzeigen

Da der Ausgang des DCF77-Empfängers mit dem Schaltereingang verbunden ist, verbindet man die beiden Buchsen des Schalters der Kombiampel, die die DCF77-Signale empfängt mit den entsprechenden Buchsen der Kombiampel für die Darstellung des Programms OSZI.EXE. Startet man beide Programme, so kann die unterschiedliche Abfolge der Signale beobachtet werden.

Aufgabe 1: Vergleiche die Signale und beschreibe sie.
Aufgabe 2: An welcher Stelle taucht das längste Signal auf?

10.3.3 Die Signale von DCF77 dekodieren

Beispiel:

Am 6.7.1996 um 4.22 h wurde folgende Sequenz von Sekundenmarken auf den Bildschirm geschrieben:

```
000000000000000001001001000010100010011000011111000110100 11
```

Von den 59 Sekundenmarken sollen hier zunächst nur die Angaben über Minuten, Stunden, Tag, Wochentag, Monat und Jahr ausgewertet werden. Liest man die Zeichenkette beginnend mit Null, sind dies die Sekundenmarken Nr. 21 bis Nr. 57:

```
00000000000000000001001***00100001010001001100001111100011010011***
```

Entsprechend dem obigen Codierschema wird wie folgt zugeordnet (PB = Prüfbit; E = Einer; Z = Zehner; m = Minute; S = Stunde; W = Wochentag; M = Monat; J = Jahr):

```
Prüfbit:              PB          PB
Code:       0010 000   1  0100 01   0  0110 00 011 1110   0  0110
1001
Bedeutung:    mE  mZ       SE  SZ       TE  TZ  W   ME   MZ  JE   JZ
Wert:          4   0       2   2        6   0   6   7    0    6   9
```

Beachte: Bei der Umwandlung Binär -> Dezimal wird entgegen der normalen Leserichtung von links nach rechts ausgewertet, d. h. die Wertigkeit für z.B. 0010 lautet:

$$\begin{array}{cccccc} 0 & 0 & 1 & 0 & & \\ 2^0 & 2^1 & 2^2 & 2^3 & \ldots & 2^n \\ 0 & + 0 & + 4 & + 0 & = 4 & \end{array}$$

11 Betrieb mit LOCAD

Die Eingänge der Aktoren für die Kombiampel sind wie bei der Kombiampel auf das LOCAD-System abgestimmt. Für LOCAD muß der entsprechende Jumpertausch bei der Kombiampel vorgenommen werden.

12 Aufbauanleitung

Das Zusatzgerät Aktoren für die Kombiampel kann direkt an den Druckerausgang des Rechners gesteckt werden. Es entfällt jedoch die Anzeigemöglichkeit als wesentliche Hilfe bei der Bewältigung gestellter Aufgaben.

Der 25-polige SUB-D-Stecker (male) der Aktoren für die Kombiampel benötigt ein mitgeliefertes 25-poliges Gegenstück (female), das auf das Flachbandkabel der Kombiampel in der Nähe der Kombiampel so gepreßt wird, daß die Anschlußseite nach oben schaut und die Reihe mit den **12** Polen.

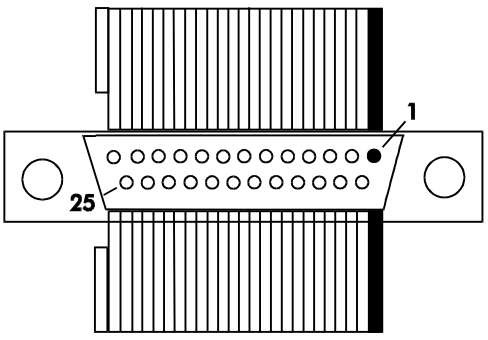
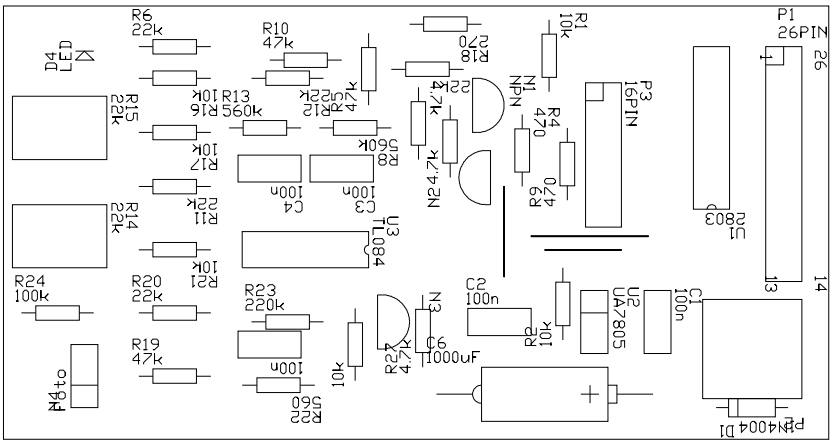
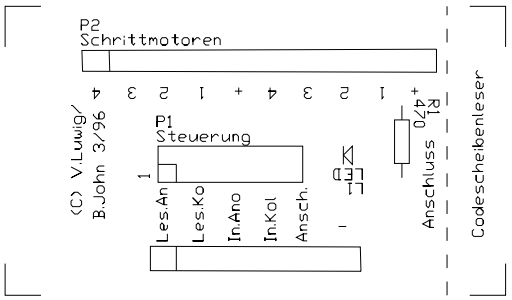


Abbildung 12-1: Hinzufügen des Anschlußsteckers

13 Bestückungspläne



CDROM1 Top Overlay



CDROMA Top Overlay

14 Anhang

14.1 Arbeitsblatt für die Aufgaben zur Punktmatrixanzeige

```
LED-Matrix
                                                   LEDs der Kombiampel
Zeile            OOOOO  S                            O O O O O O O O
                 OOOOO  p                            O O O O O O O O
                 OOOOO  a                            O O O O O O O O
                 OOOOO  l                            O O O O O O O O
                 OOOOO  t                            O O O O O O O O
                 OOOOO  e                            O O O O O O O O
                 OOOOO                               O O O O O O O O
LED-Matrix
                                                   LEDs der Kombiampel
Zeile            OOOOO  S                            O O O O O O O O
                 OOOOO  p                            O O O O O O O O
                 OOOOO  a                            O O O O O O O O
                 OOOOO  l                            O O O O O O O O
                 OOOOO  t                            O O O O O O O O
                 OOOOO  e                            O O O O O O O O
                 OOOOO                               O O O O O O O O
LED-Matrix
                                                   LEDs der Kombiampel
Zeile            OOOOO  S                            O O O O O O O O
                 OOOOO  p                            O O O O O O O O
                 OOOOO  a                            O O O O O O O O
                 OOOOO  l                            O O O O O O O O
                 OOOOO  t                            O O O O O O O O
                 OOOOO  e                            O O O O O O O O
                 OOOOO                               O O O O O O O O
LED-Matrix
                                                   LEDs der Kombiampel
Zeile            OOOOO  S                            O O O O O O O O
                 OOOOO  p                            O O O O O O O O
                 OOOOO  a                            O O O O O O O O
                 OOOOO  l                            O O O O O O O O
                 OOOOO  t                            O O O O O O O O
                 OOOOO  e                            O O O O O O O O
                 OOOOO                               O O O O O O O O
LED-Matrix
                                                   LEDs der Kombiampel
Zeile            OOOOO  S                            O O O O O O O O
                 OOOOO  p                            O O O O O O O O
                 OOOOO  a                            O O O O O O O O
                 OOOOO  l                            O O O O O O O O
                 OOOOO  t                            O O O O O O O O
                 OOOOO  e                            O O O O O O O O
                 OOOOO                               O O O O O O O O
```