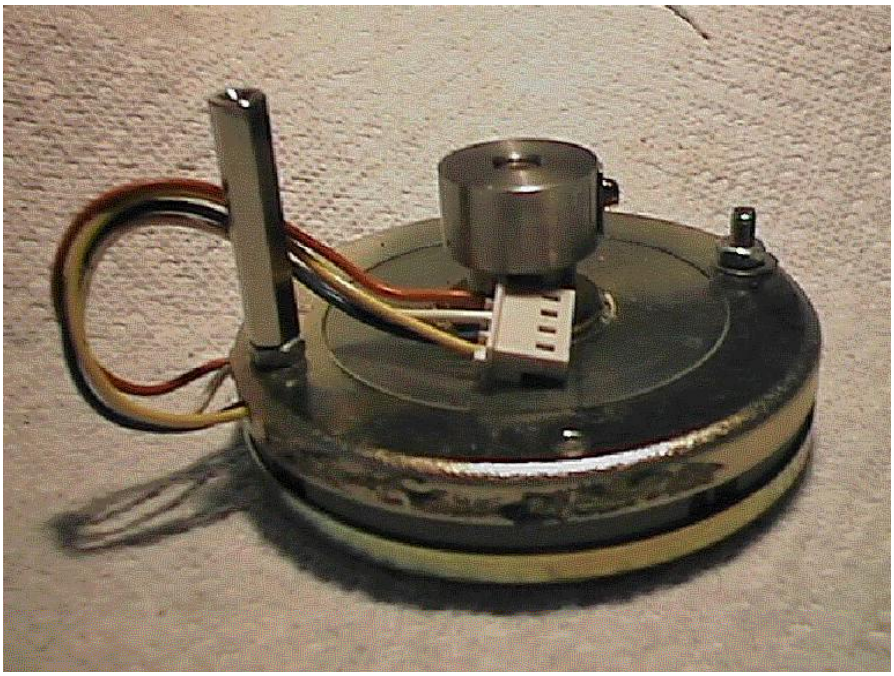
Der Schrittmotor

Von Wolfgang Back



Eine Bauform eines Schrittmotors.

Zu den interessantesten Kapiteln in der MSR – Technik gehört sicherlich die Beschäftigung mit Antriebsmotoren. Vor allem Schrittmotore sind hier beliebt, da man mit ihnen genaue Positionen anfahren kann, ohne eine aufwendige Wegstreckenerkennung zu implementieren. Jeder Schritt eines Schrittmotors entspricht einem exakten vorgegebenen Winkel. Die Elektronik zählt alle Schritte im Links- und Rechtslauf und kennt daher immer die exakte Position der Achse.

Die Schrittmotoren werden als hochpräzise Stellelemente in der Elektrotechnik und der Feinmechanik benötigt. Man findet sie z.B. in Diskettenlaufwerken, Scannern, Plottern, CD-Player, alten Festplatten usw. Überall dort, wo man ganz exakte Positionierungen ausführen muss, ist der Schrittmotor der richtige Antrieb.

Man unterscheidet grundsätzlich zwei verschiedene Schrittmotormodelle. Der unipolare Schrittmotor mit zwei Spulen (+ Mittelabgriff), der mit einer Spannung auskommt und der bipolare Schrittmotor mit vier Spulen, der die Spannungen ständig umpolt. Der Einfachheit halber wollen wir hier mit der Erklärung des unipolaren Motors beginnen.

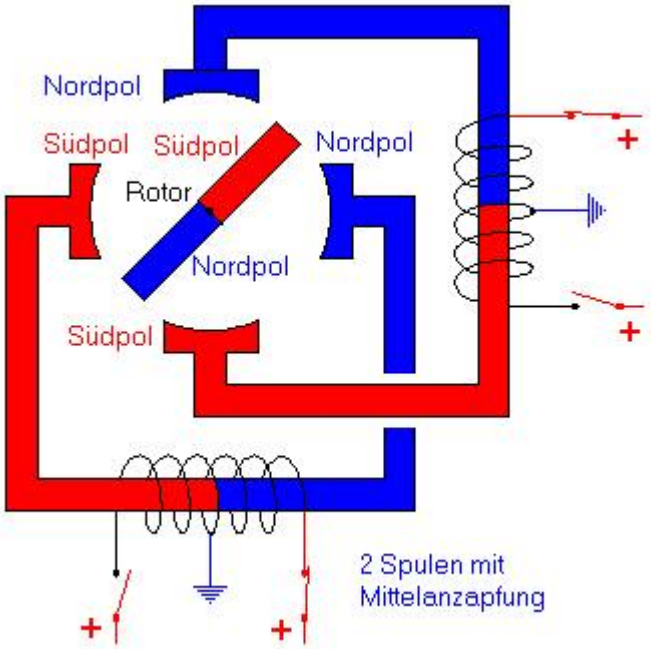


Bild xx : Schematischer Aufbau eines Schrittmotors (unipolar).

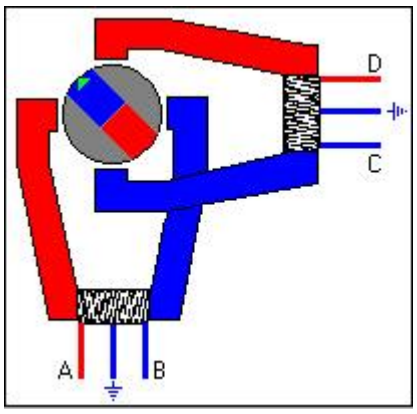
Der Rotor im Innern ist meist kugelgelagert und besitzt eine ganze Reihe permanentmagnetischer Pole (Dauermagnete). Im obigen Bild wurde zur Verdeutlichung nur ein Polpaar dargestellt.

Die beiden - sich kreuzenden – Statorzangen sind als Elektromagnete ausgeführt. Die vier Statorpole werden jeweils von zwei Spulen mit Mittelanzapfung ummagnetisiert, so dass die Statorpole abwechselnd Südpol oder Nordpol sein können. Die Bewegung entsteht durch das Ummagnetisieren der Statorzangen in einem vorgegebenen Rhythmus. Man könnte dies dadurch erreichen, indem man jeweils zwei Spulen mit umgekehrter Wickelrichtung aufbringen würde, die dann nacheinander eingeschaltet werden .

Der Einfachheit halber bedient man sich aber eines Tricks: Die Spulen sind in der gleichen Wickelrichtung aufgebracht: Sie besitzen jedoch eine Mittelanzapfung. Die Elektronik schaltet nun die eine Hälfte der Spule (die zweite Hälfte bleibt jeweils stromlos), dann die zweite Hälfte. Hierdurch erreicht man die gewollte Ummagnetisierung in der Statorzange. Die Drehung der Rotorwelle geschieht also schrittweise durch ein wanderndes Magnetfeld, das in den Statorwicklungen erzeugt wird.

Die Zeichnungen erläutern das Prinzip.

Gleichzeitg wird eine Tabelle angezeigt, die die spätere elektrische Beschaltung erläutert. Wenn die CControl den Schrittmotor steuern soll, so können wir gleichzeitg ablesen, wie die Digitalports zu anzusteuern sind. Für das Verständnis der Vorgänge wird vorausgesetzt, dass Digitalport 1 mit A (Wertigkeit 1), Port 2 mit B (Wertigkeit 2), Port 3 mit C (Wertigkeit 4) und Port 4 mit D (Wertigkeit 8) verbunden ist.

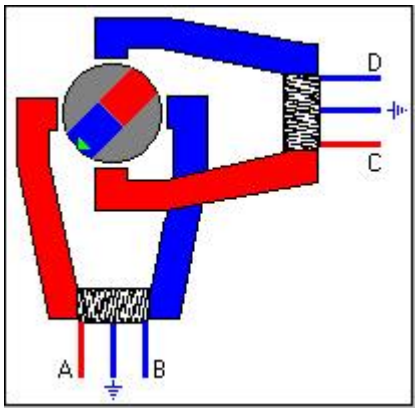


Vollschritt

	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	+	0
3	0	+	+	0
4	0	+	0	+
1	+	0	0	+

Schritt 1: Die beiden Südpole, bzw. Nordpole der Statorpole ziehen den Dauermagneten des Rotors in die gezeigte Position. Nur zwei Spulenhälften führen Strom.
Für die Elektronik bedeutet dies: A und D sind aktiv: Byteportwert 9.

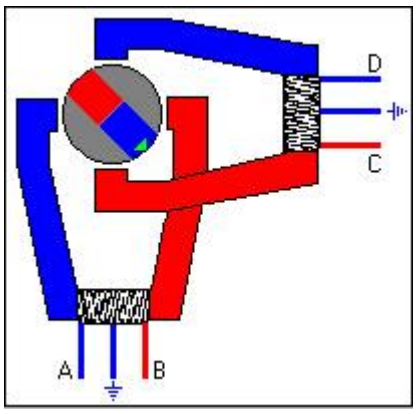
Danach wird von der Elektronik umgeschaltet und ein Elektromagnet wird in seiner Polarität umgeschaltet.



Vollschritt

	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	+	0
3	0	+	+	0
4	0	+	0	+
1	+	0	0	+

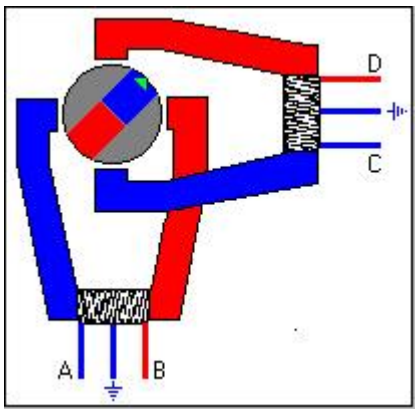
Schritt 2 : Nur eine Statorzange ändert ihre magnetische Polarität und bewegt den Rotordauermagneten exakt um einen Schritt. A und C sind jetzt aktiv. Byteportwert 1 + 4 = 5.



Vollschritt

	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	+	0
3	0	+	+	0
4	0	+	0	+
1	+	0	0	+

Schritt 3: Die andere Statorzange wurde umgepolt und sie zieht den Rotor um einen Schritt nach. Wert für den Byteport: 2 + 4 = 6.

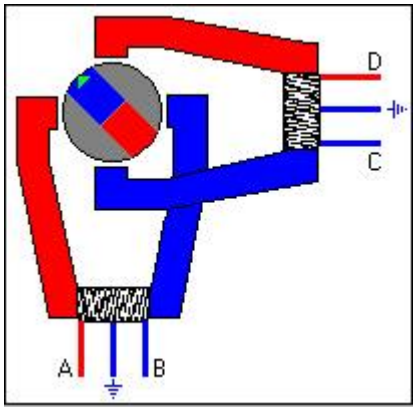


Vollschritt

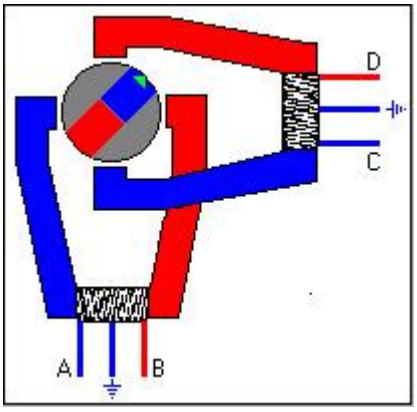
	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	+	0
3	0	+	+	0
4	0	+	0	+
1	+	0	0	+

Schritt 4: Die Statorwicklungen B und D sind jetzt aktiv. Der Wert des Byteports ist entsprechend mit 10 einzustellen. Mit vier Schritten wurde hier eine volle Umdrehung gegen die Uhrzeigerrichtung erreicht.

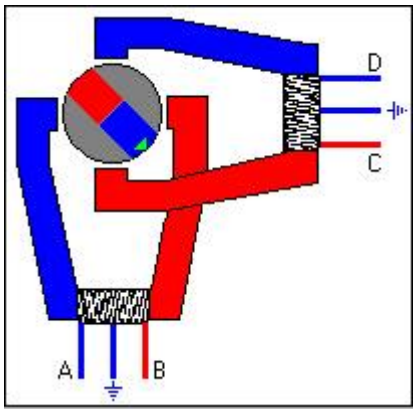
Nun hat ein Schrittmotor natürlich zwei Drehrichtungen. Die Drehung in Uhrzeigerrichtung (Rechtslauf) ist einfach zu erreichen. Die Byteportpotenziale sind in umgekehrter Reihenfolge einzuschalten. Auch dieses kann man sich an den Bildern verdeutlichen.

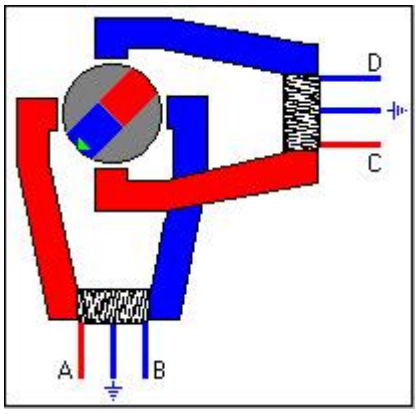


Schritt 1: Byteportwert 9.



Schritt 2: Byteportwert 10





Schritt 3: Byteportwert 6

Schritt 4: Byteportwert 5

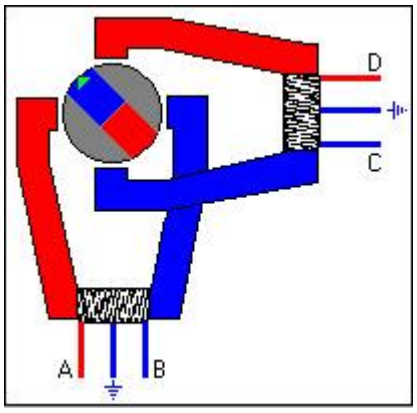
Damit ist die Bewegung eines Schrittmotors festgelegt. Bei dieser Methode spricht man von der Vollschrittmethode.

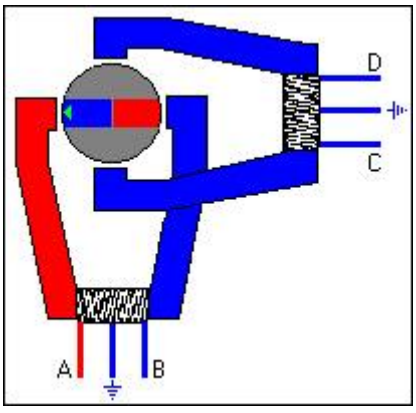
Man kann den Motor auch so steuern, dass er die doppelte Schrittzahl ausführt. Man spricht dann von einer Halbschrittmethode. Die Tabelle verdeutlicht dies.

Halbschritt

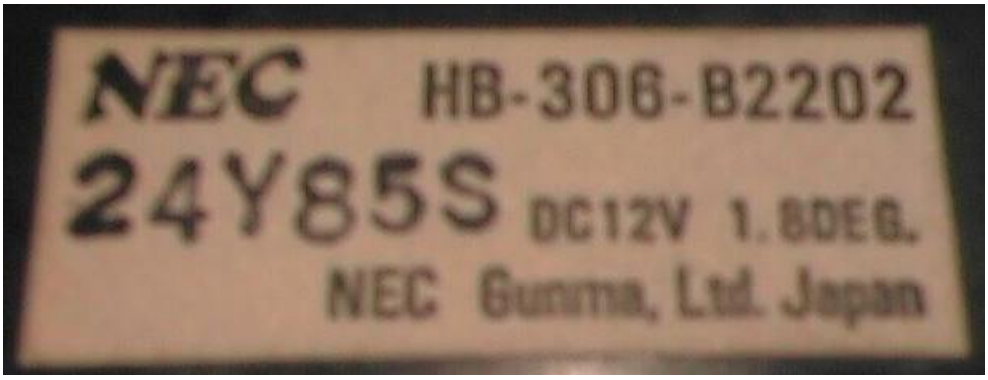
	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	0	0
3	+	0	+	0
4	0	0	+	0
5	0	+	+	0
6	0	+	0	0
7	0	+	0	+
8	0	0	0	+
1	+	0	0	+

Schritt 1 ist wie gehabt. Beide Statorzangen sind aktiv. Beim 2. Schritt wird eine Zange ganz ausgeschaltet und der Rotor bewegt sich direkt auf die aktiven Statorpole zu. Beim dritten Schritt sind wieder beide Statorzangen aktiv, danach wieder nur eine usw.





Der jeweilige Drehwinkel wird durch die Zahl der Magnetpole des Rotors festgelegt. Es gibt ganz verschiedene Motoren mit unterschiedlichstem Verhalten. Die genauesten Positionierungen kann man natürlich mit einem Motor erreichen, der viele Magnetpole aufweist. Ein Motor mit 200 Polen führt dann zu einem Schrittwinkel von 1.8 Grad im Vollschrittmodus (200*1.8=360) und entsprechend 0.9 Grad im Halbschrittmodus (400*.9=360).



Wenn man solche Typenschilder auf dem Motor vorfindet, hat man gewonnen. Hier ist es klar: Betriebsspannung: 12 Volt, 1.8 Grad Schrittweite.

Auch die elektrischen Kenndaten sind sehr unterschiedlich. Nennspannungen von 1.5 bis 15 und mehr Volt und Phasenströme von 0.1 bis mehrere Ampere sind verfügbar. Die Schrittwinkel variieren von 1.8 Grad bis 18 Grad (bei Minimotoren) und z.T. höher.

Unser Beispielmotor oben hat vier Schritte für eine Umdrehung und hätte daher einen Schrittwinkel von 90 Grad. Im Halbschrittmodus würde sich dies auf 45 Grad verringern. Man kann sich vorstellen, dass man damit nicht besonders genau steuern kann.

Schrittmotore sind keine Weltmeister in puncto Geschwindigkeit. Ein Maß für die Anzahl Umdrehungen pro Sekunde oder Minute ist die maximale Betriebsfrequenz in Hertz. Mein Experimentiermotor wird mit 1800 Hz angegeben. Damit käme man dann maximal auf 1800 / 200 = 9 Umdrehungen pro Sekunde im Vollschrittmodus.

Es wird problematisch, wenn man den Schrittmotor zu schnell bewegen will. Bei hoher Drehzahl kann es passieren, dass er Positionen überspringt, wenn er mechanische Lasten bewegen muss.

Der elektrische Anschluss

Es gibt spezielle Ansteuerungen für die Schrittmotoren. Diese sollen hier nicht betrachtet werden, da die CControl zum Einsatz kommen soll. Gehen wir einmal davon aus, dass Sie sich für die ersten Experimente einen Schrittmotor aus der Restpostenkiste besorgt haben.

Vielleicht schlachten Sie auch ein defektes Faxgerät, einen Scanner oder ein Diskettenlaufwerk aus. Nicht immer wird der Motor ausführlich beschriftet sein. Dann muss ein Messgerät die notwendigen Auskünfte liefern.

Wenn fünf oder sechs Kabelanschlüsse vorhanden sind, so handelt es sich um einen unipolaren Motor, der im folgenden beschrieben werden soll. Ist der Motor nur mit vier Leitungen bestückt, so ist es ein bipolarer Motor, der etwas aufwendiger beschaltet werden muss, da man die Spannungen umpolen muss (auch er wird beschrieben).

Ich konnte bisher noch nicht ermitteln, ob die Farben der Anschlüsse genormt sind. Bei meinen ca. 10 unipolaren Modellen ist es allerdings überall gleich. Rot ist immer der Mittelanschluss einer Spule. Schwarz und braun sind die zugehörigen Spulenenden. Bei der zweiten Spule sind es die Farben gelb und orange.

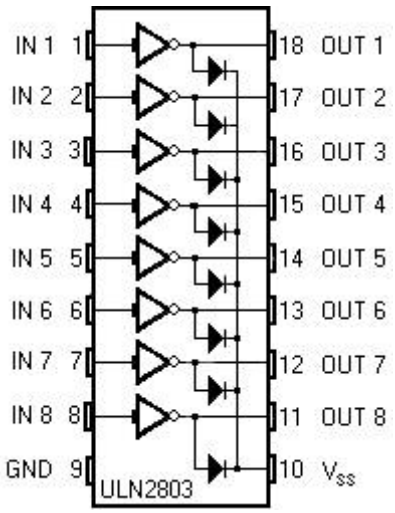
Ein Maß für die elektrische Beschaffenheit ist der Spulenwiderstand, den man jeweils zwischen rot und den beiden anderen Farben misst. Ein hoher Widerstand (> 30 Ohm) deutet auf eine höhere Betriebsspannung hin, die man problemlos aus dem 12 Volt – Netzgerät entnehmen kann. Man merkt es später im Betrieb an der sich einstellenden Temperatur des Motors. Wird er zu heiß, so ist die Spannung zu hoch gewählt. Ein paar Refernzwerte aus meinem Sortiment.

Motor 1: Nennspannung 11.5 Volt, Spulenwiderstand 75 Ohm, Schrittwinkel 1,8 Grad. Hier ist mit einem Spulenstrom von 160 mA zu rechnen (I = U / R).

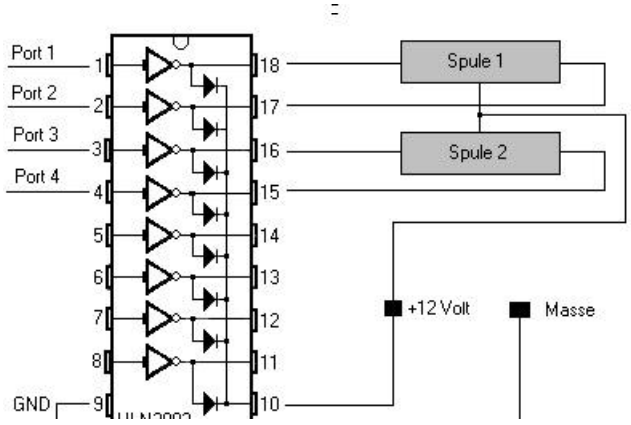
Motor 2: Nennspannung 17 Volt, Spulenwiderstand 150 Ohm, Schrittwinkel 7,5 Grad. Hier ist mit einem Spulenstrom von 120 mA zu rechnen.

Diesen Strom kann die CControl mit ihren Digitalports natürlich nicht direkt liefern. Ein entsprechender Leistungstreiber muß hier bereitgestellt werden. Man kann natürlich im einfachsten Falle 4 Transistoren für je ein Spulenende einsetzen. Einfacher und eleganter ist die Wahl eines Treiberchips. Der ULN2803 ist hierfür wie geschaffen. Jeder Ausgang (8) kann maximal 500 mA kontinuierlich liefern, die angelegte Spannung kann 50 Volt betragen. Der grosse Vorteil dieser Darlingtonausgänge ist die bereits integrierte Schutzdiode, die dafür sorgt, dass die bei induktiven Lasten auftretenden Spannungsspitzen die CControlports nicht zerstören.

Das Schaltbild zeigt den einfachen Aufbau. Alle Ausgänge liegen den Eingängen gegenüber. Lediglich Masse und +12 Volt (oder eine andere Spannung) werden angelegt. Wird dann der Eingang mit 5 Volt beschaltet, so zieht der Chip die Last gegen Masse (negative Logik).



ULN 2803 A – Treiberbaustein



den die beiden Mittelanzapfungen (rot) mit
ıg (hier 12 Volt) verbunden. Die beiden
en Ausgängen verbunden.

Jetzt muss noch ein Programm die entsprechenden Ausgänge im richtigen Schritttakt ein- und ausschalten. Es ist nicht sehr kompliziert, wenn man die oben beschriebenen Schritte nachvollzieht. Beginnen wir mit der Festlegung der Ports, die den Motor treiben sollen. Der Einfachheit halber definieren wir die Digiports 1 bis 8 als byteport. Port 1 und Port 2, Port 3 und Port 4 sind an die beiden Spulen angeschlossen. Zunächst sollen Vollschritte erzeugt werden.

```
define motor byteport[1]

#main
motor = 9
'pause 1
motor = 5
'pause 1
motor = 6
'pause 1
motor = 10
'pause 1
goto main
```

Vollschritt

	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	+	0
3	0	+	+	0
4	0	+	0	+
1	+	0	0	+

Nacheinander nehmen die Ausgänge `motor` die Werte 9 5 6 10 und danach wieder 9 5 6 10 usw. ein. An der Wahrheitstabelle kann man dies nachverfolgen. Je nach Anschluß der beiden Spulen wird sich der Motor rechts oder links drehen. Die Pause muss evtl. bei manchen Motoren mit grösseren Schrittwinkeln eingefügt werden. Ausprobieren!

Den Lauf in die entgegengesetzte Richtung erreichen wir durch eine einfache Umstellung der Schrittfolge: 9 10 6 5.

```
define motor byteport[1]

#main
motor = 9
'pause 1
motor = 10
'pause 1
motor = 6
'pause 1
motor = 5
'pause 1
goto main
```

Dieses funktioniert zwar – es ist jedoch nicht besonders gut programmiert. Wenn wir zum Halbschritt kommen, so sind 8 Schritte nötig; das Programm würde sich zu sehr aufblähen.

Wir erinnern uns an die AND – Funktion. `schritt AND 3` liefert uns genau das, was wir benötigen, um einen Tabellenaufruf vorzunehmen. Zum besseren Verständnis ist hier das

Ergebnis einer AND-Verknüpfung bis zum Wert 12 dargestellt.

```
0 AND 3 = 0
1 AND 3 = 1
2 AND 3 = 2
3 AND 3 = 3
4 AND 3 = 0
5 AND 3 = 1
6 AND 3 = 2
7 AND 3 = 3
8 AND 3 = 0
9 AND 3 = 1
10 AND 3 = 2
11 AND 3 = 3
12 AND 3 = 0
```

Damit ändert sich unser Programm folgendermassen:

```
define motor byteport[1]
define schritt word

#links
schritt = schritt - 1
looktab vollschritt , schritt and 3,motor
'pause 1 'evtl. Pause einfügen
goto links

#rechts
schritt = schritt + 1
looktab vollschritt, schritt and 3,motor
'pause 1 'evtl. Pause einfügen
goto rechts

table vollschritt
9 5 6 10
tabend
```

Die Variable `schritt` wird entweder hoch- oder runtergezählt. Es genügt hier die Definition einer Bytevariablen, da sie beim ‚Überlauf > 255' wieder bei 0 beginnt. Mit looktab wird der entsprechende Tabellenwert von 0 bis 3 gelesen.

Dieses kleine Programm kann jetzt einfach zum Halbschrittprogramm ergänzt werden, indem lediglich die Tabelleneinträge ausgetauscht werden. Dazu noch einmal die Wahrheitstabelle für Halbschritte:

Halbschritt

	phasen			
schritt	A	B	C	D
1	+	0	0	+
2	+	0	0	0
3	+	0	+	0
4	0	0	+	0
5	0	+	+	0
6	0	+	0	0
7	0	+	0	+
8	0	0	0	+
1	+	0	0	+

Wir erkennen die Wertigkeiten: 9, 1, 5, 4, 6, 2,10, 8. Wenn wir die Tabelle entsprechend verändern, so erhalten wir die doppelte Anzahl Schritte für eine Umdrehung. Natürlich wird der Motor dadurch entsprechend langsamer. Da jetzt 8 Schritte nacheinander ausgeführt werden, ist darauf zu achten, das man das Programm jetzt auf `schritt and 7` ändern muss.

```
define motor byteport[1]
define schritt word

#links
schritt = schritt - 1
looktab halbschritt , schritt and 7, motor
'pause 1 'evtl. Pause einfügen
goto links

#rechts
schritt = schritt + 1
looktab halbschritt , schritt and 7, motor
'pause 1 'evtl. Pause einfügen
goto rechts

table halbschritt
9 1 5 4 6 2 10 8
tabend

table vollschritt
9 5 6 10
tabend
```

Jetzt sollte alles funktionieren und man kann mit anderen Spielereien anfangen. Testen Sie einmal den Schrittwinkel mit einer Schleife, z.B.

```
for i=1 to 200:looktab vollschritt, i and 3,motor:next i
for i=1 to 400:looktab halbschritt, i and 7,motor:next i
```

oder

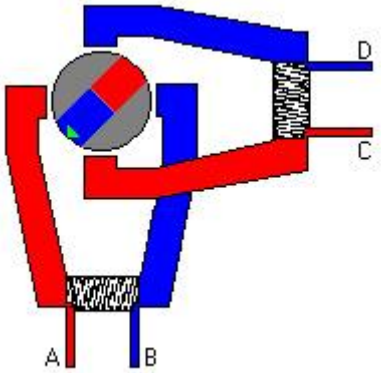
aus. Macht der Motor genau eine Umdrehung von 360°, so handelt es sich um einen Schrittmotor mit einem Schrittwinkel von 1.8° bzw. 0.9° im Halbschritt.

Der unipolare Motor wäre damit abgehandelt.

Der bipolare Schrittmotor

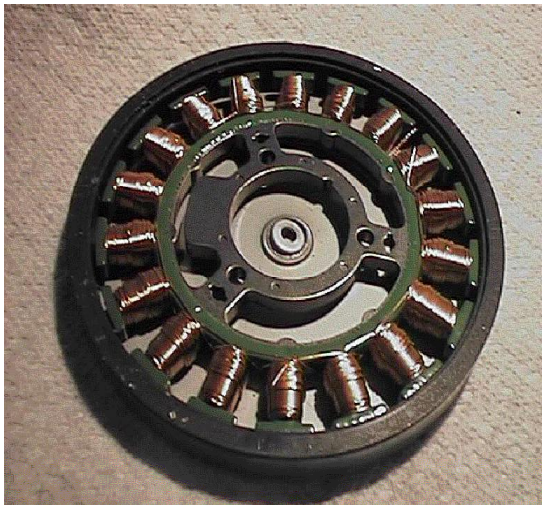
(erkenntlich an nur 4 herausgeführten Kabeln).

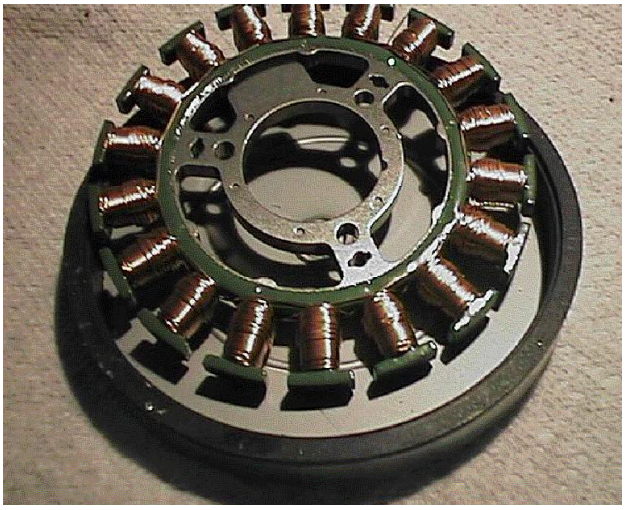
Hier müssen wir einen Mechanismus finden, um die Potenziale an den Spulenenden umzutauschen. Damit ändert sich die magnetische Polarisierung von Süd nach Nord und umgekehrt. Mit dem ULN2803 ist dies nicht möglich, da er nur nach Masse Leistung ziehen kann.



Der bipolare Schrittmotor; es fehlt die Mittelanzapfung.

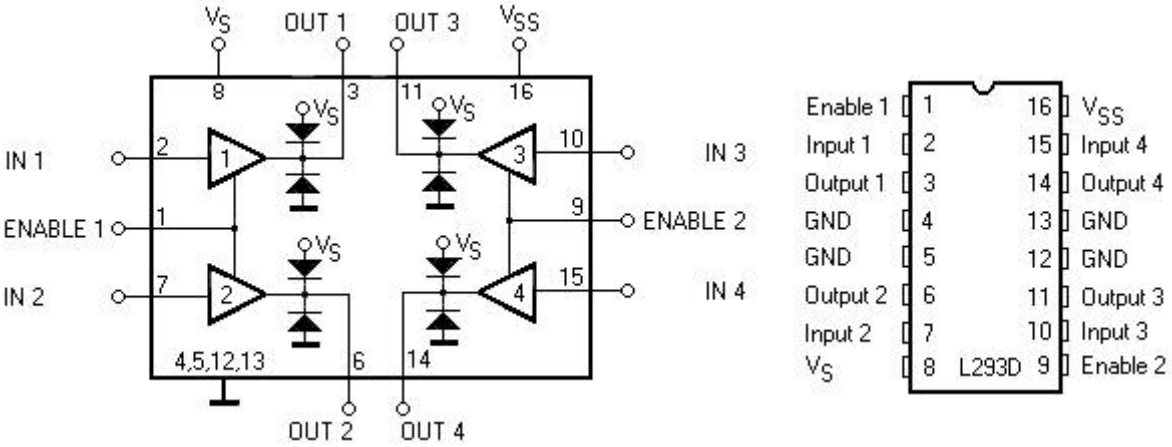
Dass Schrittmotore immer so aufgebaut sein müssen, dass sich im Innern der Rotor und außen der Statur befindet, widerlegt das nächste Bild. Der Rotor mit seinen Permamentmagneten liegt um den Statur herum. Es bewegt sich dann die große Scheibe. Zum Beispiel findet man solche Motore bei alten Diskettenlaufwerken. Die elektrische Ansteuerung ändert sich dadurch jedoch nicht. Insgesamt 18 Magnetpole kann man hier zählen. Das rechte Bild zeigt den herausgenommenen Stator. Im Hintergrund ist die Drehscheibe mit den Permanentmotoren im Außenring.



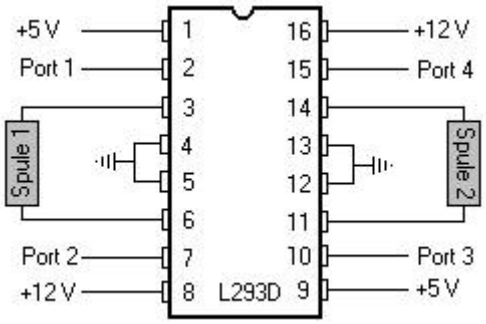


Der geeignete Leistungstreiber hierfür ist der L293D. Es ist ein Superchip, der für viele Anwendungen eingesetzt werden kann. Seine Ausgänge folgen dem angelegten Eingangspotenzial. Liegt hier ein Lowsignal, so ist der Ausgang als Masse anzusehen, liegt ein Highsignal an, so wird die angelegte Spannung Vs durchgeschaltet. Die Versorgungsspannung Vss darf maximal 36 Volt betragen. Legt man an Vs eine Spannung zwischen mindestens 4,5 Volt und maximal 36 Volt, so folgen die Ausgänge OUT1 bis OUT4 dieser Steuerspannung – ideal für Experimente mit Schrittmotoren verschiedener Nennspannungen.

Um an den L293D heranzukommen, erlebte ich eine kleine Odyssee. Conrad hat den Chip nicht und ich besorgte das erste Exemplar bei einem Münchenaufenthalt in der Schillerstrasse, ein Mekka für Elektronikfreunde. Beim ersten Händler: „führen wir nicht". Beim zweiten: „kostet 6.75, haben wir aber nicht da". Beim dritten Händler: „9.50 DM, einen habe ich noch". Hier entdeckte ich die Macht der Marktwirtschaft. Besser einen teuren in der Hand als leer ausgehen. In Köln dann durfte ich meinen zweiten Chip mit 14.50 DM bezahlen. Dabei war es nur ein B-Typ. Diesen Typ sollte man beim Experimentieren mit induktiven Lasten vermeiden, da er im Gegensatz zum L293D keine eingebauten Schutzdioden besitzt. Es passierte dann auch, was passieren muß: der vorgeschaltete 7400 gab bald seinen Geist auf. Jedenfalls wurde wenigstens kein Port der CControl zerschossen. Letztendlich wurde ich dann bei Reichelt fündig – 3.85 für den D –Typ; mittags Fax, am nächsten Morgen waren sie da.



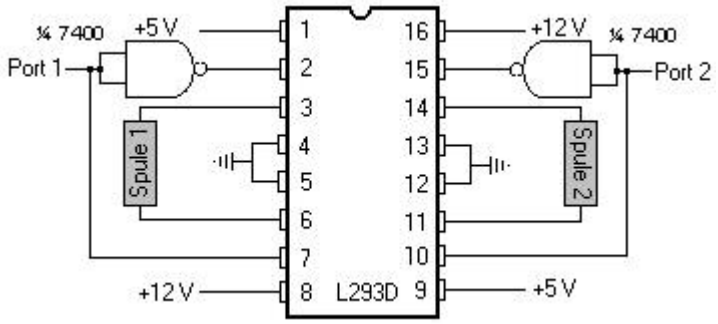
An diesem Blockschaltbild kann man die internen Dioden erkennen. Zu erklären sind noch die Eingänge ENABLE 1 und ENABLE 2. Liegt hier logisch 0 an, so sind die Ausgänge gesperrt, eine Highsignal (+5V) öffnet jeweils zwei Leistungstreiber.



Das Schaltbild zeigt den Anschluss eines bipolaren Schrittmotors mit einer Nennspannung von 12 Volt. Das unipolare Basicprogramm kann weiter benutzt werden.

Wer Digitalports sparen möchte und den Motor nur im Vollschritt betreiben möchte, der kann die unten aufgeführte Schaltung einsetzen. Mit einem NAND – Gatter (7400, 4011, 74132) lässt sich die Spannung an den Eingängen umkehren. Es werden nur noch 2 Ports benötigt.

Der Nachteil dabei ist: das Programm hat keine Möglichkeit mehr, den Motor ganz abzuschalten, da immer Spulenstrom fließt. Diese Methode sollte man also nur anwenden, wenn die Applikation ständig den Motor benutzt. Mit einem dritten Digitalport könnte man die ENABLE – Eingänge auf low setzen, um den Motor vom Netz zu nehmen.



Die Tabelle im Basicprogramm muss jetzt geändert werden, da nur noch zwei Ports im Spiel sind. Mit Hilfe der Zeichnungen von vorne kann man sich klarmachen, wie die Schritte aussehen müssen. Sind die Spulen so angeschlossen, dass eine 0 auf dem Byteport den Stator so magnetisieren wie das erste Bild des unipolaren Schrittmotors, so bezeichnen wir dies mit SüdSüd. Mit einer 1 auf dem Byteport erzeugen wir NordSüd. Danach muss NordNord eingestellt werden – Byteport = 3 und schliesslich wird mit SüdNord = 2 der nächste Schritt ausgeführt.

```
Table vollschritt
0 1 3 2
tabend
```

Assembler und Schrittmotor

Wie immer in der MSR – Technik ist es eleganter, wenn man die Aufgaben mit Assembler löst. Basic ist nun mal kein Weltmeister in puncto Geschwindigkeit. Der Leser, der noch nicht mit der Assemblersprache gearbeitet hat, der sollte sich zuerst dieses Kapitel im Buch ansehen.

Es soll ein universelles Steuerprogramm für Schrittmotore im Halbschrittmodus entstehen. Das Basicprogramm dazu ist winzig klein.

```
#main
sys &H0101 200, 0,1, 8,100
goto main
syscode "schrittm.s19“
```

Was bedeuten nun die Parameter hinter dem Sys &H0101 – Aufruf?

200 – das ist die Anzahl der gewünschten Schritte. Hier können Werte bis 65535 eingegeben werden, da mit einem Highbyte und einem Lowbyte gearbeitet wird.

0 – definiert die Richtung, z.B. rechts. Eine 1 würde dann einen Linkslauf erzeugen.

1 – liest die Halbschritttabelle, 0 entsprechend die Vollschritttabelle.

8 und 100 bestimmen die Schleifenwerte für eine Pause. Sie werden das später im Programm genauer sehen.

Das Assemblerprogramm:

```
**************************************
* Schrittmotorsteuerung
* Programmname: schrittm.asm
* Compilat:     schrittm.s19
* Basic:        schrittm.bas
*               #main
*               sys &H0101 200,0,1,8,100
*               goto main
*               syscode "schrittm.s19"
*
* (c) Wolfgang Back, 1999
**************************************
bport     equ $01 ; Digitalport 1 - 8
bpdir     equ $05 ; Richtung
schrittlo equ $9A ; Anzahl Schritte low
schritthi equ $99 ; Anzahl Schritte high
direct    equ $98 ; links oder rechts?
halbvoll  equ $96 : Halb- oder Vollschritt?
pau1      equ $94 ; Pausenwert 1
pau2      equ $92 ; Pausenwert 2
**************************************
         org $101; Programmstart

         lda #255        ; als Ausgang
         sta bpdir       ; einstellen

richtung lda direct      ; 0 oder 1 ?
         bne links       ; bei 1 verzweige nach links

rechts   ldx bport       ; lade den Portwert nach x
         lda halbvoll    ; lade Halbschritt oder Vollschritt
         bne halb        ; wenn nicht 0, dann Vollschritt
         lda vollright,x ; lese den indizierten Tabellenwert
         bra ausgang     ; Verzweige nach ausgang
halb     lda halbright,x
ausgang  sta bport       ; schalte den Port auf den Wert
         jsr pause       ; warten
         lda schrittlo   ; wieviele Schritte im Lowregister
         deca            ; verringere den Accu um 1
         sta schrittlo   ; speichere neuen Wert
         bne rechts      ; wenn nicht 0, verzweige nach rechts

schritte lda schritthi   ; lese das Highregister fuer Schritte
         cmp #0          ; ist es Null?
         beq ende        ; wenn ja, dann gehe nach ende
         deca            ; verringere den Accu um 1
         sta schritthi   ; speichere den neuen Wert
         lda #255        ; lade in den Accu 255
         sta schrittlo   ; speichere den Wert nach Lowregister
```

```
          lda direct      ; lade die richtung
          bne links       ; wenn nicht 0, dann links
          bra rechts      ; verzweige immer nach rechts

ende      rts             ; zurueck nach Basic

links     ldx bport       ; lade Portwert
          lda halbvoll    ; lade Halbschritt oder Vollschritt
          bne halb1       ; wenn nicht 0, dann Vollschritt
          lda vollleft,x  ; lese den indizierten Tabellenwert
          bra ausgang1    ; verzweige nach ausgang1
halb1     lda halbleft,x  ; Lade die Vollschritttabelle
ausgang1  sta bport       ; schalte den Port auf den Wert
          jsr pause       ; warten
          lda schrittlo   ; lade Lowregister
          deca            ; 1 abziehen
          sta schrittlo   ; neuen Wert speichern
          bne links       ; wenn nicht 0, gehe nach links
          jsr schritte    ; springe nach schritte

pause     lda pau1        ; lade Pausenwert aus Basic
loop1     ldx pau2        ; zweiter  Pausenwert
loop2     decx            ; x Register -1
          bne loop2       ; wenn nicht 0 dann loop2
          deca            ; accu -1
          bne loop1       ; wenn nicht 0 dann loop1
          rts

vollright
          fcb 5,0,0,0,0,9,5,0,0,10,6
halbright
          fcb 8,5,10,0,6,4,2,0,9,1,8
vollleft
          fcb 10,0,0,0,0,6,10,0,0,5,9
halbleft
          fcb 1,9,6,0,5,1,4,0,10,8,2
```

Bei Kenntnis der Assemblersprache ist das Programm leicht zu verstehen. Für den Anfänger soll erklärt werden, was sich im Unterprogramm `schritte` abspielt. Beim Basicaufruf können Werte bis 65535 als Schrittanzahl mitgegeben werden. Diese können bekanntlich nicht in einem 8 Bit Byte gespeichert werden. Das Basicprogramm macht hier automatisch eine Zerlegung. In $97 (Highregister) liegt jetzt die Anzahl der Schritte / 256. In $98 (Lowregister) wird der Rest der Division gespeichert.

Zunächst wird im Programm das Lowregister mit dem Rest geleert. Dann wird das Highregister auf Null geprüft. Ist es nicht Null, dann wird das Lowregister wieder mit 255 gefüllt, der Wert im Highregister um 1 erniedrigt. Ist das Lowregister wieder auf Null runtergezählt, so beginnt die Prüfung aufs Neue. Erst wenn beide Register leer sind, wird nach Basic zurückgesprungen.

Vielleicht noch ein paar Worte zu der Arbeit mit der Tabelle. Man lädt in das X-Register den Wert, den man aus der Tabelle mit dem Namen z.B. `halbleft` lesen möchte; beginnend mit 0 als erstem Wert.

```
Ldx  0
```

```
Lda  halbleft,x
```

Es wird eine 1 gelesen. Im Programm wird jeweils der Wert des Ports bestimmt und mit dem indizierten Wert der nächste Wert für den nächsten Schritt gelesen. Ist der Port am Anfang 0 (später kommt die 0 nicht mehr vor), wird der Port auf 1 gestellt.

Der nächste Aufruf: Index 1, Tabellenwert 9. Index 9 – Tabellenwert 8, Index 8 – Tabellenwert 10 usw.

Damit man den verwendeten Motor auf seine Fähigkeiten prüfen kann, werden die beiden Schleifenzähler für die Pause als Parameter mit dem Basicprogramm mitgeliefert. Sind die beiden Pausenparameter zu klein gewählt, wird der Motor irgenwann nicht mehr anlaufen. Probieren Sie es einfach aus.

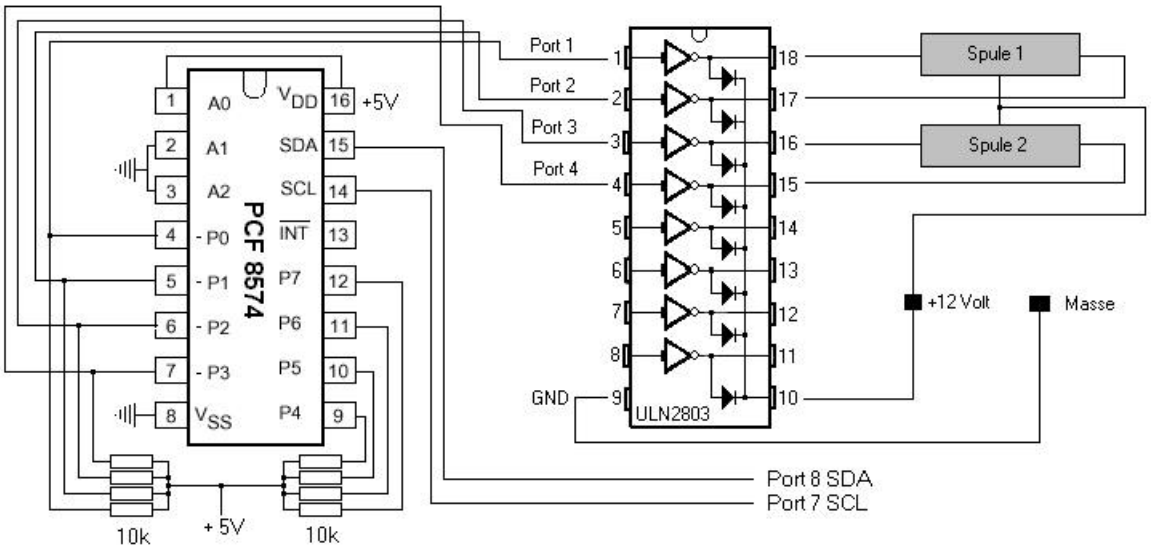
Ein wichtiger Tipp: Wenn das Assemblerprogramm einmal geladen ist, so braucht man es beim nächsten geänderten Basicaufruf nicht mehr neu zu übertragen. Schreiben Sie vor die Zeile `' syscode "schrittm.s19"` ein Hochkomma (REM). Dieses spart viel Zeit.

Schrittmotor im Lallussystem über I2C – Bus

Bisher wurden die Versuche mit Schrittmotoren mit einem Microcontroller unternommen, der 4 Digitalports zur Ansteuerung der Spulen frei hat. Im komplett aufgebauten Lallussystem sind die Digitalports weitgehend vergeben. Doch deshalb braucht man auf den Einsatz eines Motors nicht zu verzichten. Der I2C – Bus erfüllt hier die Anforderungen an die Steuerung.

Als Hardware benötigt man einen PCF8574 (I/O – Expander) und einen Motortreiber, z.B. den ULN2803A, der oben bereits beschrieben ist. Da die I2C – Routinen bereits im Lallusprogramm implementiert sind, bläht sich der Programmcode auch nicht besonders auf. Jedoch – dies sei vorab gesagt – es sollte das I2C – Assemblerprogramm geladen sein, wenn man auf einen einigermaßen schnellen Motor Wert legt. Der I2C – Bus in Basic ist eben kein Meister des Sprints.

Das nächste Schaltbild zeigt die Hardwarevoraussetzungen. Die Eingangsadresse ist hier auf 2 gelegt, das Programm bedient sich dieser Adresse. (A0 = +5V = 2, A1 und A2 auf GND). Damit erhöht sich die Grundsystemadresse 64 für den PCF8574 auf 64 + 2 = 66, oder &H42. Dieses ist bei eigenen Experimenten zu beachten.



Das Schaltbild für die Ansteuerung eines Schrittmotors über den I2C-Bus

Das Basicprogramm zur Ansteuerung soll hier für Experimentierzwecke aufgeführt werden. Der Motor dreht sich langsam – je nach Belegung der Spulen – entweder nach rechts oder links im Vollschritt.

```
define clock     port [7] ' clockt die I2C - Leitung (SCL)
define data      port [8] ' setzt die Dataleitung (SDA) für I2C

define i          byte    ' Variable für for ... next usw.
define schreibe   byte    ' Variable für I2C-Bus
define i2cein     byte    ' liefert die Schreibvariable für I2C-Bus
define adresse    &H42

#main
i2cein=9
gosub schreiben
i2cein=10
gosub schreiben
i2cein=6
gosub schreiben
i2cein=5
gosub schreiben
goto main

' ********Anfang I2C - Bus - Routinen************************
#SCHREIBEN
gosub start  : schreibe = adresse : gosub putbyte : gosub getack
schreibe = I2Cein  : gosub putbyte : gosub getack : gosub stop
return
#START
clock = 1 : data  = 1 : data  = 0 : clock = 0
return
#STOP
data  = 0 : clock = 1 : data  = 1
return
#PUTBYTE
for i = 7 to 0 step -1
```

```
if (schreibe and 1 shl i) = 0 then data = 0 else data = 1
pulse clock
next
return
#GETACK
data  = 1 : deact data : wait not data : pulse clock
return
```

Nutzt man die Routine I2Casm.s19 zur Ansteuerung über den I2C – Bus, so ergibt sich sofort ein besseres Ergebnis in puncto Geschwindigkeit. Für manchen Schrittmotor mag die Assembleransteurung sogar zu schnell sein, so dass man Pausen einfügen muß.

Das Programm mit der Assemblerroutine ist einfach, Die Adresse wird zuerst übergeben, dann der Wert des Ports, mit 0 wird ein Schreibvorgang eingeleitet. Jetzt dürfte der Schrittmotor schon ‚Beine' bekommen haben.

```
#main
sys &H0101 &H42,8+1,0
sys &H0101 &H42,8+2,0
sys &H0101 &H42,4+2,0
sys &H0101 &H42,4+1,0
goto main
syscode "i2casm.s19"
```

Da der Aufruf für manche Schrittmotoren schon zu schnell erfolgen wird (der Motor wackelt und brummt), muß eine Pause eingefügt werden, damit er sich kontinuierlich dreht. Die einfachste Methode wäre die Eingabe `pause 1, pause 2` o.ä.

Dadurch wird aber mindestens 20 msec gewartet und der Motor evtl. zu stark gebremst.

Ein Tipp: geben Sie einen Printbefehl ein, der etwa so aussehen könnte: `print "hallo"` oder `print "hallihallo"`. Dadurch entstehen ebenfalls Pausen für die Abarbeitung; doch sie sind kleiner als der Befehl `pause 1`. Je länger der zu „printende" Text ist, desto länger dauert die Pause.

Das Programm könnte also so aussehen:

```
define adr &H42
#main
sys &H0101 adr,255-9,0
print "hallo"
sys &H0101 adr,255-10,0
print "hallo"
sys &H0101 adr,255-6,0
print "hallo"
sys &H0101 adr,255-5,0
print "hallo"
goto main

syscode "i2casm.s19"
```

Hier noch einmal der Hinweis: Wenn Sie mit dem Programm experimentieren, so schalten Sie das **erneute** Laden des Assemblercodes aus: Dieses kostet viel Zeit. Ein Hochkomma am Anfang der Zeile erspart viel Ladezeit, also : `' syscode "i2casm.s19"`

Man kommt sicherlich auf die Idee, dass man mit dem PCF-Chip ja 8 Ausgänge hat und damit auch zwei Schrittmotoren ansteuern könnte. Natürlich ist dieses möglich. Das entsprechende Programm kann jetzt sicherlich selbst entwickelt werden. Hier wird eine andere Variante der Pausenerzeugung angewandt. Im Unterprogramm `#warten` kann jetzt zentral eine Pause eingegeben werden.

```
Define adr &H42
#main
sys &H0101 adr,255-(128+16),0
gosub warten
sys &H0101 adr,255-(128+32),0
gosub warten
sys &H0101 adr,255-(64+32),0
gosub warten
sys &H0101 adr,255-(64+16),0
gosub warten
goto main

#warten
print "ABC"
return

'syscode "i2casm.s19"
```

Natürlich kann auch hier das Auslesen über Tabellen erleichtert werden. Vorwärtslauf und Rückwärtslauf ist so leicht möglich. Das folgende Programm steuert zwei Schrittmotore gleichzeitig. Je nachdem, welcher Tabellenaufruf aktualisiert wird, werden entweder Voll- oder Halbschritte ausgeführt. Natürlich kann man auch so programmieren, dass beide Motore unterschiedlich laufen usw. usw. Nicht vergessen: das Terminalprogramm muß aufgerufen werden, um die Entscheidung 'links' oder 'rechts' treffen zu können.

```
define richtung word
define position byte
define zeichen  byte
define wert     byte

print "links  <1>"
print "rechts <2>"

#main
if rxd then get zeichen
if zeichen = 49 then richtung =  1
if zeichen = 50 then richtung = -1
position=position + richtung
looktab vollschritt,position and 3,wert
'looktab halbschritt,position and 7,wert
sys &H0101 &H042,wert,0
goto main

table vollschritt 153 170 102 85
tabend
table halbschritt 153 17 85 68 102 34 170 136
tabend

'syscode "i2casm.s19"
```

Einen Sekundenzeiger mit einem Schrittmotor realisieren.

Es macht sicherlich keinen großen Sinn, es ist jedoch eine Anwendung, die man direkt realisieren kann. Mit dem Schrittmotor soll ein Sekundenzeiger realisiert werden. Wenn man es weiterspinnt, so kann daraus sogar ein Dekorationselement werden. Drei Schrittmotore – mit jeweils einem Zeiger - bilden eine komplette Uhr.

Auf die Achse meines Motors klebte ich einen Strohhalm als Sekundenzeiger. Der Motor hatte einen Schrittwinkel von 1.8 Grad im Vollschrittmodus, das sind also 200 Schritte pro Umdrehung. Eine Minute hat bekanntlich 60 Sekunden. Damit paßt der Motor zunächst einmal nicht in das Raster. Es muß etwas getrickst werden.

Jede Sekunde besteht aus drei Schritten – macht 180 Schritte. Die verbleibenden 20 Schritte müssen wir daher aufteilen. Bei jeder dritten Sekunde wird ein Schritt zusätzlich gemacht.

Nicht vergessen: das Terminalprogramm muß zum Start aufgerufen werden.

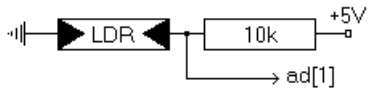
```
' Programmname: I2CUHR.BAS

define position byte
define altsec   byte
define wert     byte

sys &H0101 &H42,1+2+4+8,0 ' macht Motor stromlos

print "Start mit <Taste>"
#warten
if not rxd then goto warten ' wartet auf Tastendruck

altsec = second ' wartet eine Sekunde
#main
if altsec = second then goto main
gosub schritt        ' Einzelschritt
gosub schritt        ' Einzelschritt
gosub schritt        ' Einzelschritt
if second mod 3 = 0 then gosub schritt ' Zusatzschritt
altsec = second      ' Merker
goto main

#schritt
position=position - 1
looktab vollschritt,position and 3,wert
sys &H0101 &H042,wert,0
return

table vollschritt 9 10 6 5
tabend

'syscode "i2casm.s19"
```

Nullpunkteinstellung mit LDR

Oftmals benötigt man einen definierten Anfangspunkt für den Motor – auch wenn man immer die Schritte links und rechts per Programm mitgezählt hat. Wird der Motor z.B. ausgeschaltet, so bleibt er irgendwo stehen und die Synchronisation mit dem Programm ist

verloren. Oder bei starker Belastung kann evtl. ein Schritt verlorengehen und man muß den Motor wieder auf Null setzen. Zum einen kann man das mit einem mechanischen Anschlag realisieren, wenn der Motor nur Teile der 360 Grad abfahren kann. Eleganter ist die elektronische Lösung.

Wir benutzen dazu einen lichtabhängigen Widerstand (LDR), der mit dem Analogport ad[1] verbunden wird. Der LDR wird in einer kleinen lichtdichten Box eingebracht. Nur ein kleines Loch läßt etwas Licht auf den LDR fallen. Dieses Loch wird direkt unter dem Zeiger angebracht. Ist der Zeiger irgendwo, so fällt etwas Licht auf den LDR, fährt er über das Loch, so wird es auf jeden Fall dunkler.



Um mit allen Lichtverhältnissen (bis auf absolutes Dunkel) klarzukommen, wird zunächst einmal durch eine volle Umdrehung das Lichtminimum gesucht. Das Programm merkt sich den größten und den kleinsten Lichtwert und bildet einen Mittelwert daraus.

Bei der zweiten Umdrehung wird der Analogport ständig auf `'<= mini + mittel'` überprüft. Ist dieser Wert gefunden, ist der der Nullpunkt eingestellt.

```
' Programmname I2CNULLP.BAS

define licht    ad[1]
define position word
define wert     byte
define maxi     byte
define mini     byte
define mittel   byte

#nullpunkt_einstellen
maxi =0
mini =255
for position =0 to 199 ' eine komplette Umdrehung
looktab  vollschritt,position and 3,wert
sys &H0101 &H42,wert,0
if licht > maxi then maxi =licht ' Analogwerte merken
if licht < mini then mini = licht' Analogwerte merken
next position

mittel=(maxi-mini)/2
print mini,maxi,mittel
for position=0 to 199   ' zweiter Umlauf
looktab  vollschritt,position and 3,wert
sys &H0101 &H42,wert,0
if licht <= mini + mittel then goto nullpunkt
next position
goto nullpunkt_einstellen

#nullpunkt
print "Nullpunkt gefunden"
end

table vollschritt 9 10 6 5
```

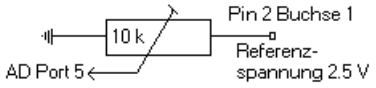
```
tabend
'syscode "I2casm.s19"
```

Der Schrittmotor als Messgerät

Die folgende Anwendung ist vielleicht etwas „durch die Nase gebohrt“; doch manchmal sind es gerade diese etwas verschrobenen Anwendungen, die letztendlich zum Einsatz kommen und für den einen oder anderen recht nützlich sein können. Es wurde bereits erwähnt, dass der Schrittmotor nur dann richtig arbeiten kann, wenn er keine großen Lasten zu bewegen hat, die schwerer sind als sein Durchzugsvermögen. Muß er zuviel Last transportieren, so kommt es leicht zum „Durchrutschen“ und die mitgezählten Schritte sind verloren, der Motor ist aus der Synchronität herausgefallen.

Die Anwendung, die hier beschrieben wird, benötigt keine großen Kräfte. Lediglich ein Zeiger, der auf der Achse angebracht wird, muß bewegt werden. Dieser Zeiger soll das analoge Ergebnis einer Messung anzeigen. Also: zuerst analog messen, dann alles digitalisieren, dann wiederum analog anzeigen. Warum nicht?

In dem vorliegenden Beispiel war an dem Analogport 5 ein Potentiometer von 10k angeschlossen, das zum einen mit der Referenzspannung von 2.5 Volt und mit Masse verbunden wurde. Die Mittelanzapfung wurde mit dem Porteingang verbunden. Damit lassen sich dann Werte zwischen 0 und 255 erreichen. Dreht man am Potentiometer, so folgt der Zeiger dem neuen eingestellten Wert.




```
' Programmname anamess.bas

define poti ad[5]
define weg     word     ' Wieviele Schritte?
define altweg  word     ' speichert Wegvariable
define merker  word     ' Merkervariable
define schritt byte     ' Schritte fuer Stepmotor
define wert    byte     ' Variable
define i       byte     ' Zaehlervariable
define adr     &H40     ' Chipadresse

' ************************************************************

#main
merker = poti
print merker
if merker > altweg  then gosub rechts
if merker < altweg  then gosub links
sys &H0101 adr,255,0    ' Motor freischalten
altweg=merker
goto main

' ************************************************************
```

```
#links
weg = altweg - merker
for i = 1 to weg
schritt = schritt - 1
looktab halbschritt , schritt and 7, wert
sys &H0101 adr,255-wert,0
next i
return
' ************************************************************

#rechts
weg = merker - altweg
for i=1 to weg
schritt = schritt + 1
looktab halbschritt , schritt and 7, wert
sys &H0101 adr,255-wert,0
next i
return

table halbschritt 144 16 80 64 96 32 160 128 tabend

'syscode "i2casm.s19"
```

Zum guten Schluß dieses Kapitels soll noch ein Programm vorgestellt werden, das sicherlich nicht jedermann aufbauen wird. Die Schaltung beinhaltet einen I/O – Expander PCF8574, ein LC – Display mit 2*8 Zeichen, einen Treiber L293D zum Ansteuern eines Schrittmotors, ein Schieberegister 4094 zum Datentransport der Bits für das LC - Display und einen Temperatursensor DS1621, der über den I2C – Bus angesteuert wird.

1.) Die Temperatur wird gemessen. Zwei Bytes werden geliefert. Das erste Byte ist der Ganzzahlwert der Temperatur. Der zweite Wert ist entweder 0 oder 128. Ist der Wert 128, dann ist die Temperatur = Ganzzahlwert + .5, ansonsten ist die Temperatur = Ganzzahlwert.

2.) Die gemessene Temperatur wird über das LC – Display ausgegeben – und zwar z.B. als 20.5 mit anschließendem °C.

3.) Die Veränderung der Temperatur wird dem Schrittmotor mitgeteilt, der hier auch als analoges Messgerät dient.

4.) Da zwei Bytes nacheinander gelesen werden müssen, wird die Assemblerroutine i2c_sysd.s19 installiert. Hier wird der interne I2C – Bus genutzt. Dieses bedeutet: Die Informationen werden über Pin 11 (SDA) und Pin 12 (SCL) geliefert. Der interne I2C-Bus ist in dem Kapitel Assembler genauer beschrieben.

5.) Es kommen hier einige Spezialitäten zusammen, die in verschiedenen Kapiteln einzeln besprochen wurden. Die Messung der Temperatur, die Anzeige über ein LC _ Display, die Nutzung der internen I2C – Routinen, die Ansteuerung mehrere Komponenten über den Zweidrahtbus I2C.

```
' Programmname messgera.bas

define aus1    byte     ' Rückgabe aus Assembler
define aus2    byte     ' Rückgabe aus Assembler
```

```
define temp    word     ' speichert auch Minuswerte
define alttemp word     ' speichert Temperaturwert
define weg     word     ' Wieviele Schritte?
define altweg  word     ' speichert Wegvariable
define merker  word     ' Merkervariable
define schritt byte     '
define wert    byte     ' Variable
define i       byte     ' Zaehlervariable
define zeile   bit[192] ' Zeilenumschaltung

define led  8
define data 4
define cp   2
define rs   1

' ************************************************************
sys &H0101 &H40,255,0,0,0  ' Dummyaufruf für I2C- Bus (Start)
sys &H0101 &H40,255,0,0,0  ' Dummyaufruf für I2C- Bus (Start)

sys &H0101 &H9E,&HAC,0,0,2 ' kont. Wandlung, neg. Polarität
sys &H0101 &H9E,&HEE,0,0,0 ' Wandlung ein

' ************************************************************

sys &H0101 &H9E,&H17,0,0,6  ' gespeicherten Wert 1 lesen
altweg = aus1 * 256
sys &H0101 &H9E,&H17,1,0,6  ' gespeicherten Wert 2 lesen
altweg = altweg + aus1
print "alt";altweg

' ************************************************************
' Character 0 (Gradzeichen) erstellen

wert= 64 : gosub kommando
for aus1 = 0 to 7
looktab grad,aus1,wert
gosub schreiben
next

' ************************************************************

wert=&H0C : gosub kommando ' Display ein, ohne Cursor
wert=&H38 : gosub kommando ' zwei Zeilen
gosub cls

' ************************************************************

#main
sys &H0101 &H9E,&HAA,0,0,0 ' Temperaturlesen ein
sys &H0101 &H9E,0,0,0,3    ' zwei Bytes lesen
temp =aus1*10+(aus2=128)*-5
pause 20
print temp
if alttemp = temp then goto weiter
gosub cls
wert = (temp/100) + 48           : gosub schreiben
wert = ((temp mod 100)/10)  + 48 : gosub schreiben
```

```
wert=46                          : gosub schreiben
wert=(temp mod 10) + 48          : gosub schreiben
wert = 32                        : gosub schreiben
wert=0                           : gosub schreiben  ' Gradzeichen
wert=67                          : gosub schreiben  ' C schreiben
alttemp = temp

' ************************************************************

#weiter
if temp > altweg then gosub rechts
if temp < altweg then gosub links

altweg = temp

' ************************************************************

if altweg <> merker then gosub merken
merker =altweg
print merker
goto main

' ************************************************************

#merken
sys &H0101 &H9E,&H17,0,altweg / 256,5   ' Hibyte schreiben
sys &H0101 &H9E,&H17,1,altweg mod 256,5 ' Lobyte schreiben
return

' ************************************************************

#links
weg = altweg - temp
for i=1 to weg
schritt = schritt - 1
looktab halbschritt , schritt and 7, wert
sys &H0101 &H40,255-wert,0,0,0
next i
sys &H0101 &H40,255,0,0,0
return

' ************************************************************

#rechts
weg = temp-altweg
for i=1 to weg
schritt = schritt + 1
looktab halbschritt , schritt and 7, wert
sys &H0101 &H40,255-wert,0,0,0
next i
sys &H0101 &H40,255,0,0,0
return

' ************************************************************

#kommando
for i=7 to 0 step -1
```

```
if wert and 1 shl i then sys &H0101 &H40,rs + led,0,0,0
sys &H0101 &H40,cp + led,0,0,0 ' cp
sys &H0101 &H40,0 + led,0,0,0
next i
sys &H0101 &H40,data + led,0,0,0 ' RS = 0
sys &H0101 &H40,0 + led,0,0,0
return

' ************************************************************

#schreiben
for i=7 to 0 step -1
if wert and 1 shl i then sys &H0101 &H40,rs + led,0,0,0
sys &H0101 &H40,cp + led,0,0,0     ' cp
sys &H0101 &H40,0 + led,0,0,0
next i
sys &H0101 &H40,data + rs + led,0,0,0 ' RS = 1
sys &H0101 &H40,0 + led,0,0,0
return

' ************************************************************

#cls
wert=&H01 : gosub kommando
return

' ************************************************************

#zeile_schalten
zeile = not zeile
if zeile then wert = &H80 + &H40 else wert=&H80
gosub kommando
return

' ************************************************************

table vollschritt 144 80 96 160 tabend
table halbschritt 144 16 80 64 96 32 160 128 tabend
table  grad  &B00001100  &B00010010  &B00010010  &B00001100  0  0  0  0
tabend

'syscode "i2c_sysd.s19"
```