



Anwendungen von
AVR-Einchip-Prozessoren AT90S, ATtiny,
ATmega und ATxmega
**Schrittmotorsteuerung mit
einem Attiny24**

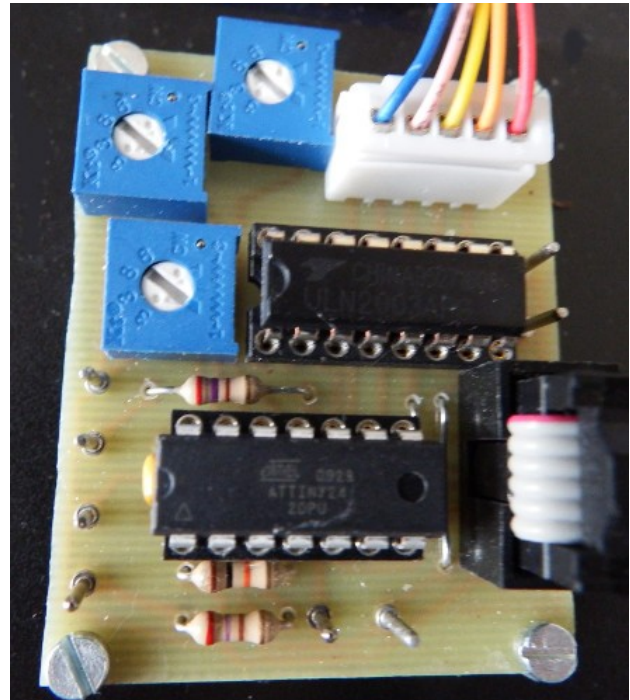


Schrittmotorsteuerung 28BYJ-48 mit einem ATtiny24

[Gezippte Version der Webseite \(504 kB\)](#)

Wer für seine Modelleisenbahn eine Schrankensteuerung oder eine Torsteuerung braucht, die mit einem Schrittmotor vom Typ 28BYJ-48 läuft, ist mit dieser Schaltung gut bedient. Sie hat folgende Eigenschaften:

- kostet alles in allem weniger als 10 € das Stück,
- lässt sich mit 5 V Gleichspannung betreiben,
- dreht maximal 180° in 1.024 Einzelschritten,
- die Drehgeschwindigkeit lässt sich zwischen zwei und 17 Sekunden mit einem Trimpotentiometer einstellen,
- die Auslenkung von Mittenstellung nach oben und nach unten lässt sich separat mit zwei Trimpotentiometern verstellen,
- die letzte Position wird jeweils im internen EEPROM gespeichert und beim Einschalten wieder hergestellt, Positionskorrektur der Mittenstellung möglich,
- läuft sehr zuverlässig wie ein Uhrwerk, keine kniffligen Stellungsmelder erforderlich,
- den Assembler-Quellcode gibt es hier für umme (für Nicht-Südhessen: kostenlos) zum Download, lässt sich bei Bedarf vom Vollschritt- auf das Halbschritt-Verfahren umstellen, wenn mehr Drehmoment nötig sein sollte.



[Seitenanfang](#)

[1 Hardware](#)

[2 Aufbau](#)

[3 Der Betrieb](#)

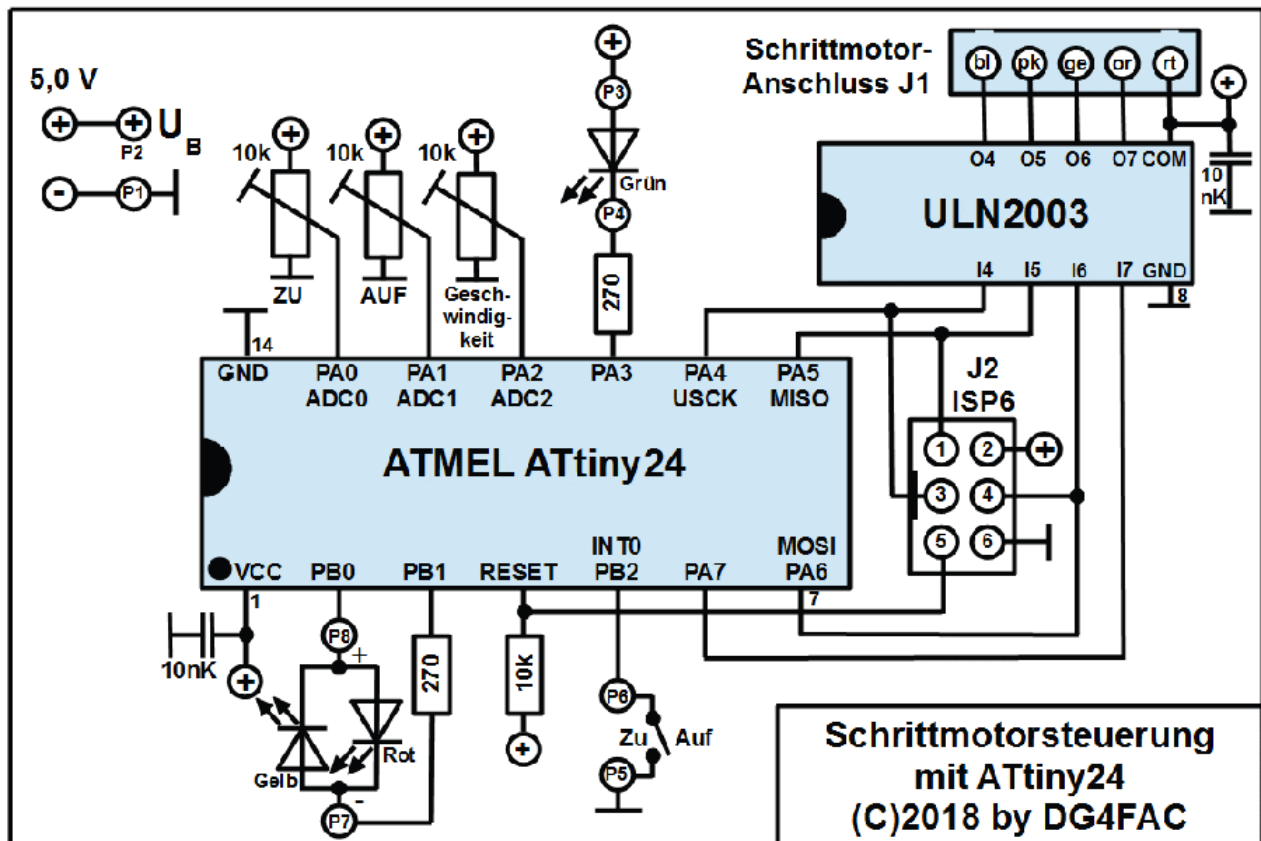
[4 Die Software](#)

[5 Trouble shooting](#)

1 Hardware

1.1 Schaltbild

Die Schrittmotorsteuerung arbeitet mit einem ATtiny24. Die Steuersignale für den Schrittmotor werden an den Portbits PA4 bis PA7 ausgegeben und treiben die Eingänge der Darlingtontreiber-Schaltung ULN2003A. Seine Ausgänge treiben die vier Magnete des Schrittmotors 28BYJ-24 (Schaltbild des Motors siehe unten) im Voll- (jeweils nur ein Magnet angesteuert) oder im Halbschrittverfahren (wechselweise ein bzw. zwei Magnete angesteuert).



An den AD-Wandler-Eingängen ADC0 und ADC1 sind die Positionstrimmer angeschlossen. Der Schrittmotor folgt deren Einstellung beim Schliessen (ADC0) bzw. beim Öffnen (ADC1).

Am AD-Wandler-Eingang ADC2 ist der Trimmer für die Geschwindigkeit angeschlossen.

An den Port-Ausgängen PB0 und PB1 ist eine Duo-LED angeschlossen. Sie blinkt rot (rechts herum) bzw. gelb (links herum) solange sich der Motor noch bewegt. Sie geht dauerhaft an, wenn der Motor seine Endposition erreicht hat. Sie erlischt, wenn die Magnete des Schrittmotors nach einer zusätzlichen Verzögerungszeit von einer Sekunde abgeschaltet sind.

Am Portausgang PA3 ist eine grüne LED angeschlossen. Sie leuchtet, wenn der Motor seine Endposition erreicht hat und die Verzögerungszeit abgelaufen ist. Sie zeigt ferner an, dass die erreichte Position im EEPROM gespeichert wird.

Sie brauchen zwei rot/gelbe oder grüne LEDs für beide Seiten der Schranke? Kein Problem: einfach Hintereinanderschalten und den Widerstand von 270Ω

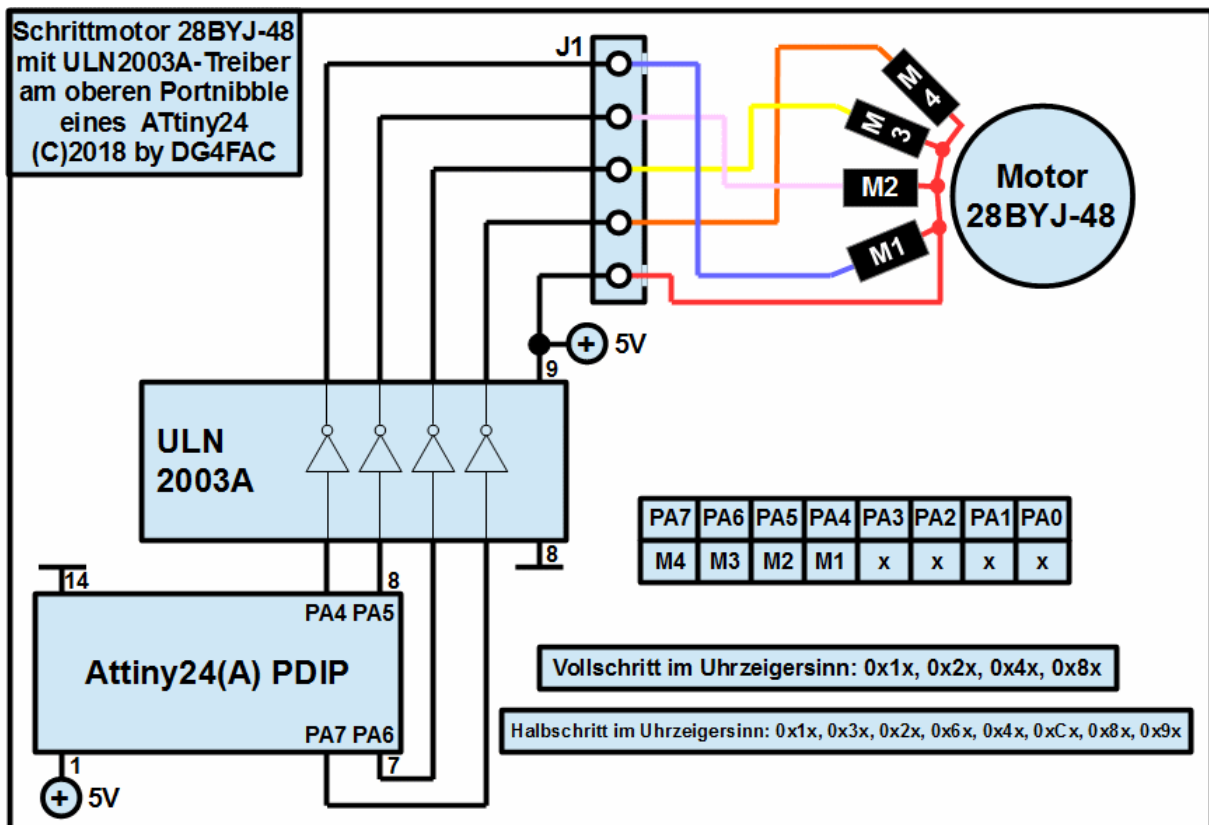
auf 100Ω verkleinern.

Mit dem Auf/Zu-Schalter an PB2 (INT0) wird die Schranke oder das Tor auf und zu gefahren.

Der Prozessor kann in der fertigen Schaltung mit der dafür vorgesehenen ISP6-Schnittstelle programmiert werden, die die Standardbelegung aufweist.

1.2 Schaltbild des Schrittmotors

Dies hier ist das Schaltbild des Motors. Eingezeichnet sind nur vier der insgesamt 16 Magnete. Gezeichnet ist der Motor in der Draufsicht, von der Getriebeachse her betrachtet.



Aus dem Treiber- und dem Anschlussschema des ATtiny24 ergeben sich für die Bewegung des Motors im Uhrzeigersinn die dargestellten Hexadezimalwerte, mit denen der obere Port des ATtiny24 angesteuert werden muss.

Der Motor macht 64 Einzelschritte pro Umdrehung und hat ein eingebautes Getriebe 1:64, so dass pro Umdrehung der Getriebewelle $64 * 64 = 2.048$ Einzelschritte im Vollschrittverfahren zu absolvieren sind (im Halbschrittverfahren 4.096).

Maximal kann der Motor im Vollschrittverfahren ohne Last mit ca. 4,2 Sekunden pro Getriebeumdrehung angesteuert werden (488 Hz), bei Frequenzen darüber dreht der Motor nicht mehr sauber oder gar nicht mehr. Im Halbschrittverfahren kann die Wechselfrequenz noch etwas höher als das Doppelte sein, weil 50% mehr Leistung aufgewendet werden. Bei etwas über 1.000 Hz ist aber auch hier Schluss.

1.3 Stückliste

Die benötigten elektronischen Teile sind in der Stückliste zusammengestellt. Natürlich unterliegen die angegebenen Preise Schwankungen.

Der Schrittmotor 28BYJ-48 ist in der Stückliste nicht enthalten. Er kann im Internet-Verbandhandel für deutlich weniger als 5 € pro Stück geordert werden.

Der auf dem mitgelieferten Board oft gesteckte ULN2003A kann ausgebaut und für die Bestückung verwendet werden.

Stepper ATtiny24 Stückliste

Anzahl	Bauteil	Reichelt	Preise €	
			Einzel	Gesamt
1	ATtiny24	ATTINY 24-20 PU	1,99	1,99
1	ULN2003A	ULN 2003A	0,32	0,32
1	LED 3mm grün	LED 3MM GN	0,06	0,06
1	Duo LED 3 mm rot/gelb	LED 3 RY	0,18	0,18
3	Trimmerpotentiometer 10k	76-10 10K	0,31	0,93
2	Widerstand 270 Ohm	1/4W 270	0,103	0,21
1	Widerstand 10 k	1/4W 10K	0,103	0,10
2	Keramikkondensator 10n	Z5U-5 10N	0,03	0,06
1	Wannstecker 6-polig	WSL 6G	0,16	0,16
1	JST Stecker 1x5 pin	JST XH5P ST	0,22	0,22
1	Schalter	MS 244	1,70	1,70
8	Lötöse 1mm	RF 1-100	0,0337	0,27
8	Lötstift 1mm	RTM 1-100	0,018	0,14
0,08	Silberdraht 0,6mm	SILBER 0,6MM	0,26	0,02
Gesamt				6,36

[Seitenanfang](#)

[1 Hardware](#)

[2 Aufbau](#)

[3 Der Betrieb](#)

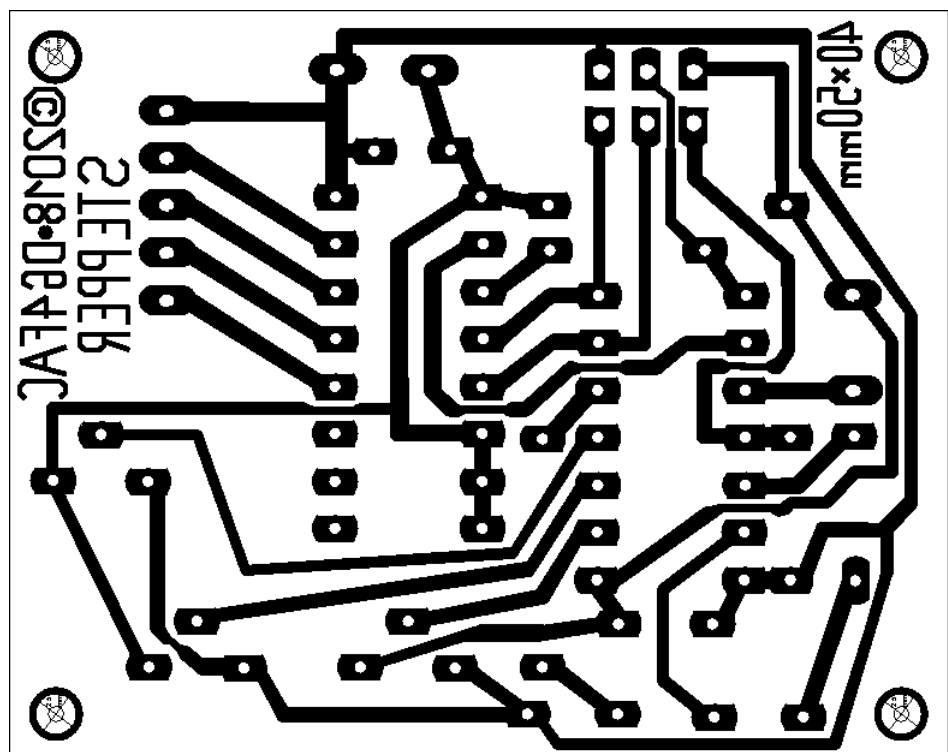
[4 Die Software](#)

[5 Trouble shooting](#)

2 Aufbau

2.1 Gedruckte Platine

Das Platinenlayout für die Schaltung ist hier zu sehen, es hat die Maße 50x40 mm (1/8 Euroformat) auf einseitig kaschiertem Fotoplatinmaterial. Es sind Befestigungslöcher mit 2,5 mm Durchmesser vorgesehen.

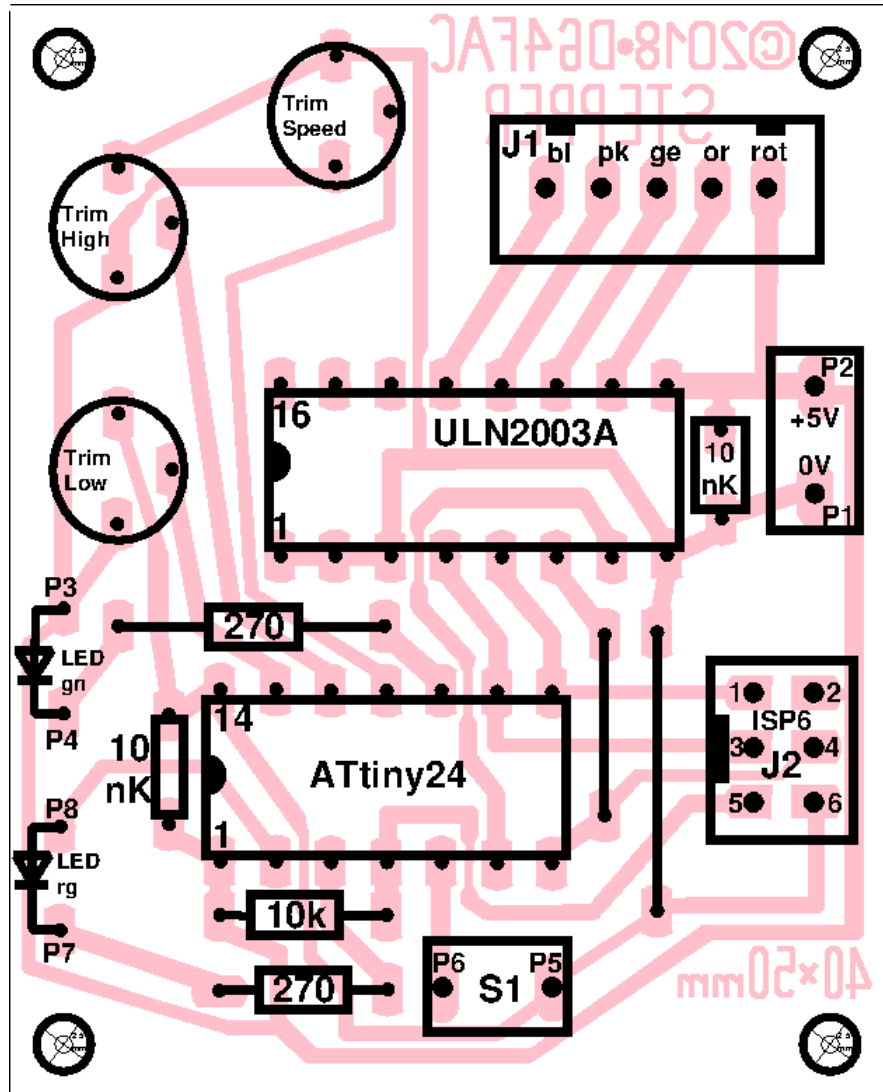


2.2 Bestückung

Es sind zwei Brücken zu bestücken. Als Trimmer können sowohl runde als auch quadratische Typen bestückt werden.

Bei den beiden Steckverbindern J1 und J2 ist auf die eingezeichneten Aussparungen zu achten, bei den IC-Sockeln auf die 1-Kennzeichnung.

Der ISP6-Wannenstecker J2 muss nur bestückt werden, wenn der ATtiny24 in der Schaltung programmiert werden soll.

[Seitenanfang](#)[1 Hardware](#)[2 Aufbau](#)[3 Der Betrieb](#)[4 Die Software](#)[5 Trouble shooting](#)

3 Der Betrieb

3.1 Inbetriebnahme

Vor dem ersten Einschalten ist der Schrittmotor etwa mittig einzustellen (bei Schranke oder Tor: halb geöffnet). Geringfügige Abweichungen von der Mitte um bis zu 90° sind unkritisch. Geht das nicht, ist ein- oder mehrmalig die Prozedur nach [3.3](#) auszuführen.

Bei offenem Schalter ist der obere Anschlagpunkt einzustellen, bei geschlossenem Schalter der untere Anschlagpunkt. Läuft der Motor dauerhaft um jeweils einen Schritt vorwärts und rückwärts, sollte die Einstellposition geringfügig geändert werden, um den Strombedarf und den Motorverschleiß zu mindern und um sicherzustellen, dass die Position auch korrekt gespeichert wird.

Die Geschwindigkeit kann je nach Geschmack eingestellt werden. Bei erhöhter Last die Geschwindigkeit etwas herunterfahren, bis der Motor sauber läuft.

3.2 Normalbetrieb

Im normalen Betrieb wird die Schranke/das Tor mit dem Schalter geschlossen bzw. geöffnet. Soll das Schliessen über ein externes Signal erfolgen, muss der Pin P6 auf Masse (Pin P5) gezogen werden. Dabei ist darauf zu achten, dass der Pin bis mindestens zum Erleuchten der grünen LED aktiviert bleibt.

Nur wenn die grüne LED erleuchtet ist, sollte der Strom abgeschaltet werden.

Lassen sich die obere oder untere Anschlagposition nicht mehr erreichen, ist ein- oder mehrmalig die Prozedur nach [3.3](#) auszuführen.

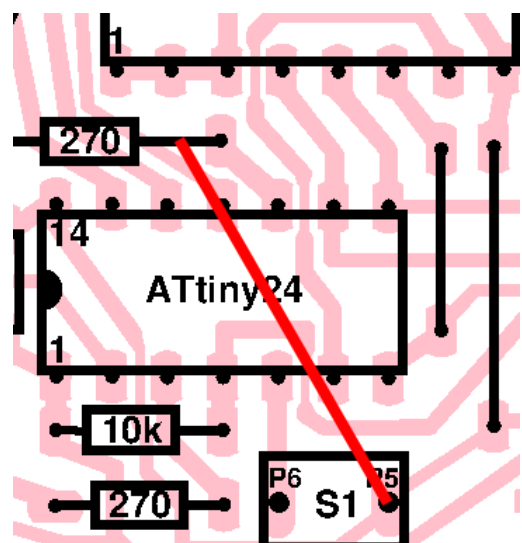
3.3 Mittenjustierung

Ist die werksseitige Stellung des Schrittmotors nach dem Einbau nicht mittig oder ist die Stellung des Schrittmotors z. B. durch kurzzeitige unkontrollierte Bewegungen beim Ausschalten der Schaltung oder durch Ausschalten der Versorgung im vollen Lauf verstellt, kann dies durch folgende Prozedur korrigiert werden.

Die Mittenverstellung stellt man fest, indem beide Positionstrimmer auf linken Anschlag gestellt werden (Öffnen und Schließen des Schalters führen in diesem Fall nicht zu Motorbewegungen). Falls die Schranke bzw. das Tor nicht ungefähr auf halber Höhe steht, kann sie mit der folgenden Methode wieder neu eingestellt werden.

1. Beide Positionstrimmer auf linken Anschlag.
2. Bei offenem (Mitte zu niedrig) bzw. geschlossenem Schalter (Mitte zu hoch) mit dem jeweiligen Poti etwa die Mitte einstellen.
3. Die Stromversorgung abschalten und warten, bis die Betriebsspannung unter 1,7 V Restspannung entladen ist oder die Betriebsspannung von der Schaltung von der Platine abklemmen.
4. Den IC-seitigen Anschluss des Widerstands der grünen LED, der zu Pin 10 des ATtiny24 führt, mit GND verbinden (z. B. mit dem Anschlusspin P5).
5. Die Betriebsspannung einschalten und die Verbindung bei eingeschalteter Betriebsspannung wieder entfernen (der Motor sollte nun auf eine andere Position laufen).
6. Warten bis die grüne LED leuchtet und die Position gespeichert wird.

Ist die Mittenstellung um mehr als 90 Grad verschoben, kann die Prozedur zwei- oder mehrfach wiederholt werden.



4 Die Software

Die Software ist in Assembler geschrieben und ausführlich kommentiert. Sie kann [hier als Quellcode](#) heruntergeladen oder im Anhang: Quellcode in Assembler angesehen werden. Der Quellcode verwendet .if-Direktiven und kann mit [gavrasm](#) oder mit dem ATMEL-Assembler 2 assembliert werden (eine Anleitung zum Assemblieren mit gavrasm gibt es für [Linux hier](#) und für [Windows hier](#)).

Die Abläufe und Funktionsweise der Software sind im Folgenden detailliert erläutert.

4.1 Die Positionen des Schrittmotors

Die aktuelle Position des Schrittmotors wird im Registerpaar rCurrValH:rCurrValL als 16-Bit-Zahl verwaltet. Sie sollte im Bereich 32.768 +/- 511 (Vollschritt-Version, Default) bzw. +/- 1.023 (Halbschritt-Version, Änderung im Kopf der Software) liegen.

Die Soll-Position des Schrittmotors steht im Registerpaar rSetValH:rSetValL. Stimmt die Ist-Position mit der Sollposition nicht überein, dann erfolgen Links- oder Rechtsschritte (siehe [4.2](#)).

Die Position im offenen Zustand ($32.768 - N$) ist in rOpenH:rOpenL, im geschlossenen Zustand ($32.768 + N$) als CloseH:rCloseL verwaltet. Beide Positionen werden aus der Stellung des jeweiligen Trimmers berechnet.

Wird der Schalter betätigt, dann wird über den davon ausgelösten INT0-Interrupt die T-Flagge im Statusregister gesetzt. Je nach dem Zustand des Schalters wird entweder der Wert rOpen oder der Wert rClose in die Sollwert-Register rSetVal kopiert.

4.2 Positionsverstellung des Schrittmotors

Die Positionsverstellung des Schrittmotors arbeitet mit dem 8-Bit-Timer TC0. Er wird mit einem Vorteiler von 64 ($1 \text{ MHz} / 64 = 15.625$, Timer-Tick alle $64 \mu\text{s}$) im CTC-Modus betrieben (Rücksetzen des Timers bei Erreichen des Compare-Werts A). Abhängig vom eingestellten Geschwindigkeitspotiwert erfolgt dies nach Erreichen von 15 (Halbschrittmodus, höchste Geschwindigkeit) nach $15 \cdot 64 = 960 \mu\text{s}$ bzw. 30 (Vollschrittmodus, höchste Geschwindigkeit) nach $30 \cdot 64 = 1.920 \mu\text{s}$ oder beim Erreichen von 255 (niedrigste Geschwindigkeit) nach $256 \cdot 64 = 16.384 \mu\text{s}$.

Ausgelöst durch den Timer-Interrupt TC0COMP A wird in der Interrupt-Service-Routine überprüft, ob rCurrVal und rSetVal gleich sind. Ist das der Fall, steht der Stepper an der gewünschten Soll-Position und es passiert nichts weiter.

Ist rCurrVal kleiner als rSetVal, wird rCurrVal um Eins erhöht. Ist es größer, wird rCurrVal um Eins nach unten verstellt. In beiden Fällen werden die untersten zwei (Vollschrittverfahren) bzw. drei Bit (Halbschrittverfahren) dazu benutzt, um aus einer Tabelle die nächstfolgende Magnetkombination auszulesen und an den Ausgangsport PA4 bis PA7 zu schreiben. Gleichzeitig wird an Portpin PA3 die grüne LED abgeschaltet.

4.3 Die Magnetabschaltung

Das Timing zur Magnetabschaltung arbeitet mit dem 16-Bit-Timer TC1. Er wird mit einem Vorteiler von 1.024 getaktet.

Bei jedem TC0-Timer-Interrupt, bei dem die Motorposition geändert wird, wird der Timer TC1 auf Null gesetzt. Ab der letzten Motorverstellung läuft der Timer TC1 durch und erreicht den voreingestellten Endwert (eine Sekunde ist voreingestellt). Der ausgelöste TC1COMPA-Interrupt

- schaltet den Motor ab (0000 auf PA4 bis PA7),
- schaltet die grüne LED an (PA3 auf 0),
- setzt die Flagge bWrite, damit die Position im EEPROM abgelegt wird.

Nachfolgende Interrupts lösen nur dann das Setzen der Flagge bWrite aus, wenn die grüne LED noch nicht an ist.

4.4 Position aus EEPROM lesen

Zu Beginn, nach dem RESET des Prozessors, werden ab Adresse 0x0010 des EEPROMs folgende Bytes gelesen:

1. eine erste Kennung Mark1 (sollte 0xAA sein),
2. der aktuell eingestellte Schließen-Wert (rCloseL, rCloseH),
3. der aktuell eingestellte Öffnen-Wert (rOpenL, rOpenH),
4. die aktuelle Motorposition (rCurrValL, rCurrValH),
5. die zweite Kennung Mark2 (sollte 0x55 sein).

Die gelesenen Bytes werden in den entsprechenden Registern abgelegt.

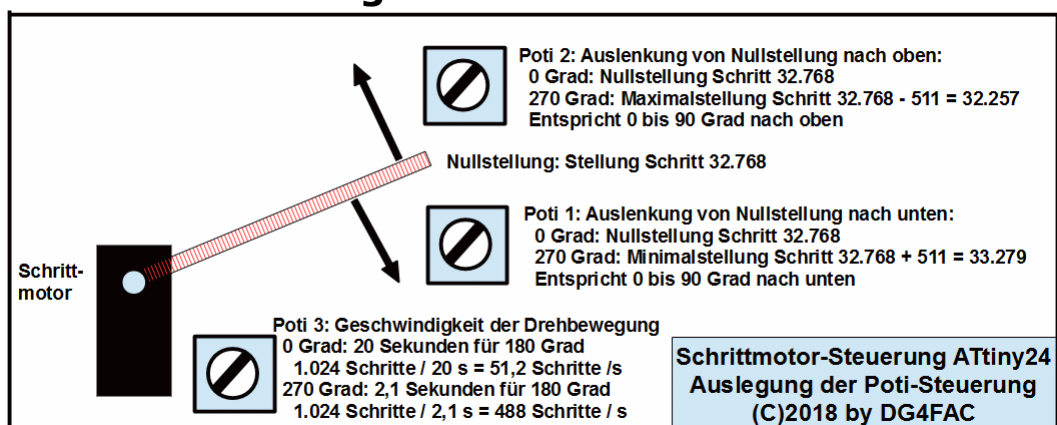
Ergibt das Exklusiv-Oder von Mark1 und Mark2 0xFF, dann sind die EEPROM-Daten korrekt. Wenn nicht, wird die Motorstellung und werden die Open- und Close-Werte auf 32.768 (0x8000) gestellt.

4.5 Position ins EEPROM schreiben

Erreicht der Motor seinen Endstand und läuft die Verzögerungszeit ab, dann werden diese Bytes an die gleiche Position im EEPROM geschrieben. Dazu wird der Schreibzyklus mit Mark1 begonnen und nach jedem abgeschlossenen Schreiben der EEPRDY-Interrupt ausgelöst. Sind alle Bytes geschrieben, wird der EEPRDY-Interrupt ausgeschaltet.

4.6 Messen und Umrechnung der Potentiometerwerte

Dies hier zeigt die Funktion der drei Trimmer-Potentiometer. Poti 1 und 2 bestimmen die Auslenkung von der Mitte



nach unten bzw. oben. Potentiometer 3 bestimmt, mit welcher Geschwindigkeit diese Bewegung des Schrittmotors erfolgt.

Die drei Potentiometerstellungen werden in drei Messzyklen zu je 64 Einzelmessungen per ADC-Interrupt durchgeführt. Die Ergebnisse der je 64 Einzelmessungen werden zu einem 16-Bit-Wert aufsummiert.

4.6.1 Umrechnen der beiden Positionswerte

Beim Vollschrittverfahren wird das MSB der Messwertsumme mit zwei multipliziert, beim Halbschrittverfahren mit vier. Zu diesem Wert wird 32.768 addiert (Offen-Position) bzw. er wird von 32.768 subtrahiert (Geschlossen-Position) und in den entsprechenden Registern abgelegt.

4.6.2 Umrechnen des Geschwindigkeitswerts

Vom MSB der Messwertsumme wird das Zweierkomplement gebildet und mit $(255-30)=225$ beim Vollschrittverfahren bzw. $(255-14)=241$ beim Halbschrittverfahren multipliziert und das resultierende MSB der Multiplikation mit 30 bzw. 14 addiert. Dieser Wert wird in das Vergleichsregister OC0A des 8-Bit-Timers geschrieben und steuert die Geschwindigkeit der Schrittmotorverstellung.

4.7 Neujustierung

Bei jedem Start wird der Pull-Up-Widerstand am Portpin PA3 eingeschaltet und festgestellt, ob der Eingang Low ist. Ist das der Fall, werden die aktuelle Motorposition, der Sollwert und die Offen- und Geschlossen-Stellungen auf 32.768 gestellt und das Lesen der EEPROM-Daten unterbunden.

Wird der Eingangspin wieder freigegeben, wird für 100 ms lang festgestellt, ob der Pin High bleibt. Ist das der Fall, wird das Programm normal fortgesetzt, ansonsten beginnt die Periode von vorne (Vermeidung von Schalterprellen).

[Seitenanfang](#)[1 Hardware](#)[2 Aufbau](#)[3 Der Betrieb](#)[4 Die Software](#)[5 Trouble shooting](#)

5 Trouble-Shooting

5.1 Motor läuft dauerhaft/endet nicht

Falls der Motor dauerhaft läuft und seinen Lauf nicht beendet, kann das daran liegen, dass die Magnete des Motors beim Aus- und Umschalten eine Spikespannung erzeugen, die den Prozessor rückt. Das passiert gerne im Programmiermodus, wenn die Schaltung aus der USB-Stromversorgung des Programmiergeräts gespeist wird. Falls dies passiert, kann der Keramikkondensator am ULN2003A auf 100 nK vergrößert werden. Das sollte die Spikes beseitigen. Wenn nicht, kann ein weniger sensibles Netzteil abhelfen.

5.2 ATtiny24 lässt sich nicht programmieren

Erfolgt die Programmierung des ATtiny24 in der Schaltung über die ISP6-Schnittstelle und ist der Motor während der Programmierung angeschlossen,

verursachen die Programmierimpulse ein schnelles An- und Ausschalten von bis zu drei Magneten des Motors gleichzeitig. Das kann bei einer schwachbrüstigen Stromversorgung, z. B. aus der USB-Schnittstelle, kurzzeitig mit bis zu 300 mA Strom Spikes auf der Versorgungsleitung verursachen. Ausserdem produzieren die Magnete beim Abschalten ebenfalls Spannungsspitzen, die zwar von den im ULN2003A verbauten Dioden kurzgeschlossen werden, aber dennoch schnelle Spikes in der Versorgung verursachen. Ohne den eingezeichneten Keramikkondensator von 10 nF neben dem ULN2003A ist es erfahrungsgemäß daher nicht möglich, den ATtiny24 über die ISP6-Schnittstelle zu programmieren.

Gegen Schwachbrüstigkeit der Stromversorgung hilft auch ein kleiner Elko von z. B. 10 μ F, die kurzen Lastspikes auf der Versorgungsspannung wegzupuffern. Nach der Programmierung kann der Elko wieder entfernt werden.

Lob, Tadel, Fehlermeldungen, Genöle und Geschimpfe oder Spam bitte über das [Kommentarformular](#) an mich.

©2018 by <http://www.avr-asm-tutorial.net>

Anhang: Quellcode in Assembler

```
;
; *****
; * Schrittmotorsteuerung ATtiny24 *
; * 16-Bit-Position, Voll/Halbschritt *
; * Version 1, Maerz 2018, V1 *
; * (C)2018 by avr-asm-tutorial.net *
; *****
;
.nolist
.include "tn24adef.inc" ; Define device ATtiny24A
.list
;
; *****
;   M O D U S - S C H A L T E R
; *****
;
.equ Halfstep = 0 ; 1=Halbschritt, 0=Vollschritt
;
; *****
;   D E B U G - S C H A L T E R
; *****
;
; Debug ausschalten: alle Schalter auf 0
;
; Teste ADC-Wert-Umrechnung
.equ TestAdc = 0 ; Teste Berechnung Kanal 2
.equ TestChannel=2 ; Channel zu Simulieren
.equ TestValue = 255*256
;
; Teste Ausgabe Motor
.equ TestMotor = 0 ; Teste Motorstellung
.equ TestCurr = 32768 ; aktuelle Stellung
.equ TestNext = 32767 ; naechste Stellung
;
;   H A R D W A R E
; *****
;
; Device: ATtiny24, Package: 14-pin-PDIP_SOIC
;
;
;   1 /-----|14
;   +Ub o--|VCC   GND|--o -Ub
;   2|      |13
; Duo AnRt o--|PB0  ADC0|--o Poti ZU
;   3|      |12
; LED AnGe o--|PB1  ADC1|--o Poti AUF
;   4|      |11
;   RESET o--|RES   ADC2|--o Poti Speed
;   5|      |10
; Schalter o--|PB2   PA3|--o LED Ka Gn
;   6|      |9
;   ULN IN7 o--|PA7  PA4|--o ULN IN4
;   7|      |8
;   ULN IN6 o--|PA6  PA5|--o ULN IN5
;   |-----|
;
; *****
;   P O R T S   A N D   P I N S
; *****
; Duo-LED-Ansteuerung
.equ pDuoO = PORTB ; Duo-LED-Ausgabeport
.equ pDuoD = DDRB ; Duo-LED-Richtungsport
.equ bDuoAR = PORTB0 ; Rote Anode Portbit
.equ bDuoAY = PORTB1 ; Gelbe Anode Portbit
; Schaltersteuerung
.equ pSwO = PORTB ; Schalter-Ausgabeport
.equ pSwD = DDRB ; Schalter-Richtungsport
.equ pSwI = PINB ; Schalter-Eingabeport
.equ bSwO = PORTB2 ; Schalter-Ausgabepin
.equ bSwD = DDB2 ; Schalter-Richtungspin
.equ bSwI = PINB2 ; Schalter-Eingabepin
; Bereit-LED
.equ pOkO = PORTA ; Bereit-LED-Ausgabeport
.equ pOkD = DDRA ; Bereit-LED-Richtungsport
.equ pOkI = PINA ; Eingabeport
.equ bOkO = PORTA3 ; Bereit-LED-Ausgabepin
```

```

.equ bOkD = DDA3 ; Bereit-LED-Richtungspin
.equ bOkI = PINA3 ; Lese-Pin
; Schrittmotor
.equ pStpO = PORTA ; Schrittmotor-Ausgabeport
.equ pStpD = DDRA ; Schrittmotor-Richtungsport
.equ mStpD = 0xF0 ; Maske Schrittmotor-Richtung
;
; *****
;   A D J U S T A B L E   C O N S T
; *****
;
; Taktgenerator Default (CLKDIV8 gesetzt)
.equ clock=1000000 ; Define clock frequency
;
; Grad maximaler Drehwinkel =
; maximale Getriebewinkel-Aussteuerung):
.equ cAngleMax = 180 ; Grad maximaler Drehwinkel
;
; Startgeschwindigkeit, Default
; bis zur ersten Potentiometermessung
.equ cSecOpenClose = 20 ; 20 Sekunden
;
; Zeit nach letzter Bewegung bis Motor stromlos
; Minimum: 1 (1 ms), Maximum: 6710 (6,71 s)
.equ cMotOffms = 1000 ; Millisekunden bis Motor aus
;
; *****
;   F I X   &   D E R I V.   C O N S T
; *****
;
; Motor mit Getriebe 1:64
; Schritte pro Umdrehung Motor 64
; Schritte pro Getriebeumdrehung:
.equ cStepsRound = 64 * 64
; Mittenstellung:
.equ cMiddle = 32768
; Schritte pro Maximalwinkeldrehung:
.equ cStepsAngle = (cStepsRound*cAngleMax+180)/ 360
; Schritte pro halber Maximalwinkeldrehung
; (von Mitte aus nach oben bzw. unten):
.equ cStepsUpDown = (cStepsAngle+1) / 2
;
; Berechnung der Startgeschwindigkeit
.equ cTCOPresc = 64 ; Vorteiler TC0
; TCOTicks in Mikrosekunden:
.equ cTCOTick = (1000000*cTCOPresc)/clock
; Dauer in Mikrosekunden fuer volles Oeffnen/
; Schliessen ueber den Maximalwinkel
.equ cUsSwing = (1000000*cSecOpenClose)/cStepsAngle
; CTC-Wert fuer 8-Bit-TC0
.equ cCTCDef = (cUsSwing+cTCOTick/2)/cTCOTick
;
; CTC-Wert fuer 16-Bit-TC1 Motor aus
.equ cMotOff = (1000*cMotOffms+512)/1024-1
;
; *****
;   T I M I N G
; *****
;
; TC0 ist Schrittmotor-Ansteuerung
; laeuft im CTC-Modus mit Compare A
; clock = 1.000.000 Hz
; Vorteiler = 64
; Timer-Tick = 64 us
; Vollschrirtverfahren:
; Schritte 180 Grad = 1.024
; Langsame Bewegung = 16,78 s
; Zeit pro Schritt = 16.384 us
; CTC-Ticks = 256
; Compare-A-Wert = 255
; Schnelle Bewegung = 2,03 s
; Zeit pro Schritt = 1.984 us
; CTC-Ticks = 31
; Compare-A-Wert = 30
; Berechnung aus MSB ADC-Summe ADC2:
; Compare A = (ADC-MSB * 225)/256+30
; Halbschrirtverfahren:
; Schritte 180 Grad = 2.048
; Langsame Bewegung = 17,7 s
; Zeit pro Schritt = 16.384 us

```

```

;   CTC-Ticks           =       135
;   Compare-A-Wert     =       134
;   Schnelle Bewegung  =         1,97 s
;   Zeit pro Schritt   =         960 us
;   CTC-Ticks         =         15
;   Compare-A-Wert     =         14
;   Berechnung aus MSB ADC-Summe ADC2:
;   Compare A = (ADC-MSB * 241)/256+14
;
; TC1 ist Motor-Aus-Schalter
; laeuft im Normalmodus mit Compare A
; Clock                = 1.000.000 Hz
; Vorteiler           =         1.024
; Timer-Tick          =         1.024 us
; Maximum bis Ueberl. =        65.536
; Entspricht Zeit     =          6,7 s
; Eingestellt auf     =          1 s
;
; AD-Wandler misst Potistellungen 1, 2 und 3
; Clock                = 1.000.000 Hz
; Vorteiler           =          128
; ADC-Tick            =          128 us
; Wandlertakte/Messung=          14
; Zeit pro Messung    =         1.792 us
; 64 Messungen        =       114.688 us
; Drei Kanäle         =       344.064 us
; Messungen pro Sek. =          2,9
;
; *****
;           R E G I S T E R S
; *****
;
; R0, R1, R2, R3 benutzt fuer Multiplikationen etc.
; free: R4
; ADC-Werte-Summierung
.def rAdcL = R5 ; Summe ADC-Wert, LSB
.def rAdcH = R6 ; dto., MSB
; Registerwerte fuer Schrittmotorstellung
.equ cStepSetS = 7 ; fuer Lesen/Schreiben EEPROM
.def rMark1 = R7
.def rCloseL = R8
.def rCloseH = R9
.def rOpenL = R10
.def rOpenH = R11
.def rCurrValL = R12
.def rCurrValH = R13
.def rMark2 = R14
.equ cStepSetE = 15 ; Ende des Datensets
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose Interrupts
.def rFlag = R18 ; Flaggen
.equ bWrite = 0 ; Datenset in EEPROM schreiben
.equ bAdc = 1 ; 64 Messwerte sind aufsummiert
.def rAdc = R19 ; ADC-Summierungszaeher
.def rSetValL = R20 ; Sollwert, LSB
.def rSetValH = R21 ; Sollwert, MSB
; free: R22 to R29
; used: R27:R26 = X Zeiger ausserhalb Interrupts
; used: R29:R28 = Y Zeiger fuer EEPROM-Schreiben
; used: R31:R30 = Z Zeiger Vielzweck, fuer LPM
;
; *****
;           S R A M
; *****
;
.dseg
.org SRAM_START
; (Nicht benutzt, nur fuer den Stapel)
;
; *****
;           C O D E
; *****
;
.cseg
.org 000000
;
; *****
; R E S E T & I N T - V E C T O R S
; *****

```

```

    rjmp Main ; Reset vector
    rjmp Int0Isr ; EXT_INT0
    reti ; PCIO
    reti ; PCI1
    reti ; WATCHDOG
    reti ; ICPL
    rjmp OC1AIsr ; OC1A
    reti ; OC1B
    reti ; OVFl
    rjmp OC0AIsr ; OC0A
    reti ; OC0B
    reti ; OVFO
    reti ; ACI
    rjmp AdcIsr ; ADCC
    rjmp EepIsr ; ERDY
    reti ; USI_STR
    reti ; USI_OVF
;
; *****
; I N T - S E R V I C E   R O U T .
; *****
;
; INT0 Interrupt Service Routine
;   Schalter-Aktionen loesen INT0 aus
;   bei steigenden und fallenden Flanken
Int0Isr:
    set ; Update-Flagge fuer Sollwert
    reti
;
; OC0A Interrupt Service Routine
;   bedient Schrittmotor
OC0AIsr: in rSreg,SREG ; rette SREG
    cp rCurrValL,rSetValL ; LSB rIst = rSoll?
    brne OC0AIsr1 ; LSB ungleich
    cp rCurrValH,rSetValH
    breq OC0AIsr8 ; gleich, tue nichts
    cp rCurrValL,rSetValL ; LSB-Vergleichen
OC0AIsr1:
    ; rIst ungleich rSoll, Motorbewegung
    cpc rCurrValH,rSetValH ; MSB testen
    brcs OC0AIsr4 ; rIst kleiner rSoll
    tst rCurrValL ; Pruefe auf Null
    brne OC0AIsr2 ; LSB nicht Null
    dec rCurrValH ; MSB eins abwaerts
OC0AIsr2:
    dec rCurrValL ; Ein Schritt zurueck
    sbi pDuoO,bDuoAY ; gelbe LED an
    cp rCurrValL,rSetValL
    breq OC0AIsr3 ; nicht gelb blinken
    sbrc rCurrValL,3
    cbi pDuoO,bDuoAY ; gelbe LED blinkt
OC0AIsr3:
    cbi pDuoO,bDuoAR ; Kathode gelbe LED Low
    rjmp OC0AIsr7 ; Schrittmotor verstellen
    ; rIst kleiner rSoll
OC0AIsr4:
    inc rCurrValL ; Ein Schritt vorwaerts
    brne OC0AIsr5 ; LSB nicht Null
    inc rCurrValH ; MSB erhoehen
OC0AIsr5:
    sbi pDuoO,bDuoAR ; rote LED an
    cp rCurrValL,rSetValL
    breq OC0AIsr6 ; nicht rot blinken
    sbrc rCurrValL,3
    cbi pDuoO,bDuoAR ; rote LED blinken
OC0AIsr6:
    cbi pDuoO,bDuoAY ; Kathode rote LED Low
OC0AIsr7:
    push ZH ; rette Z
    push ZL
    mov rimp,rCurrValL ; Ist-Wert kopieren
.if Halfstep == 1
    ldi ZH,High(2*StepTab8) ; Z auf 8-Step-Tabelle
    ldi ZL,Low(2*StepTab8)
    andi rimp,0x07 ; die untersten drei Bit
.else
    ldi ZH,High(2*StepTab4) ; Z auf 4-Step-Tabelle
    ldi ZL,Low(2*StepTab4)
    andi rimp,0x03 ; die untersten zwei Bit
.endif

```

```

add ZL,rimp ; zur Tabellenadresse addieren
ldi rimp,0
adc ZH,rimp
lpm rimp,Z ; Tabellenbyte lesen
in ZL,pStpO ; Port lesen
andi ZL,0x0F ; unterste drei Bit erhalten
or ZL,rimp ; Motor- und Bereit-Led-Bits setzen
out pStpO,ZL ; an Stepmotorausgang, Bereit-LED aus
pop ZL ; stelle Z wieder her
pop ZH
clr rimp ; Motor-Aus-Zaehler neu starten
out TCNT1H,rimp
out TCNT1L,rimp
OC0AIsr8:
out SREG,rSreg ; stelle SREG wieder her
reti
;
.if Halfstep == 1
; Schrittfolge-Tabelle Halbschrittmodus
StepTab8:
.db 0x18,0x38,0x28,0x68,0x48,0xC8,0x88,0x98
.else
; Schrittfolge-Tabelle Vollschrittmodus
StepTab4:
.db 0x18,0x28,0x48,0x88
.endif
;
; OC1A-ISR schaltet Motormagnete aus
;
OC1AIsr:
in rSreg,SREG ; SREG sichern
sbic pOkO,bOkO ; gruene LED schon an? Wenn nicht...
sbr rFlag,1<<bWrite ; Schreibe Position ins EEPROM
in rimp,pStpO ; Lese Motor-Port Output
andi rimp,0x07 ; Untere drei Bits erhalten
out pStpO,rimp ; Motor aus, gruene LED an
cbi pDuoO,bDuoAR ; rote/gelbe LED aus
cbi pDuoO,bDuoAY
out SREG,rSreg ; SREG wieder herstellen
reti
;
; ADC complete Interrupt
AdcIsr:
in rSreg,SREG ; rette Status
in rimp,ADCL ; Lese LSB Ergebnis
add rAdcL,rimp ; addiere zur Summe
in rimp,ADCH ; lese MSB Ergebnis
adc rAdcH,rimp
dec rAdc ; Zaehler abwaerts
brne AdcIsr1 ; noch nicht 64 Werte
sbr rFlag,1<<bAdc ; setze Flagge
rjmp AdcIsr2 ; nicht neu starten
AdcIsr1:
ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
out ADCSRA,rimp
AdcIsr2:
out SREG,rSreg ; stelle Status wieder her
reti
;
; EEPROM Interrupt
EepIsr:
in rSreg,SREG ; SREG retten
in rimp,EEARL ; Lese LSB Adresse
inc rimp ; naechste Adresse
out EEARL,rimp
ld rimp,Y+ ; Lese naechstes Byte
out EEDR,rimp
sbi EECR, EEMPE ; EEPROM-Schreiben starten
sbi EECR, EEPE
cpi YL,cStepSetE ; Letztes Byte?
brcs EepIsr1
cbi EECR,EERIE ; Keine Interrupts mehr
cbr rFlag,1<<bWrite ; Schreibflagge loeschen
EepIsr1:
out SREG,rSreg ; SREG wieder herstellen
reti
;
; *****
; M A I N P R O G R A M I N I T
; *****

```

```

;
Main:
    ldi rmp,Low(RAMEND)
    out SPL,rmp ; Init LSB stack pointer
;
; Debugschalter auswerten
.if TestAdc == 1
; Teste Berechnung ADC
    ldi rmp,High(TestValue)
    mov rAdcH,rmp
    ldi rmp,Low(TestValue)
    mov rAdcL,rmp
    ldi rmp,TestChannel
    out ADMUX,rmp
    rcall AdcFlag
TestAdcLoop:
    rjmp TestAdcLoop
.endif
.if TestMotor == 1 ; Teste Motorstellung
    ldi rmp,High(TestCurr) ; aktuelle Stellung
    mov rCurrValH,rmp
    ldi rmp,Low(TestCurr)
    mov rCurrValL,rmp
    ldi rmp,High(TestNext)
    mov rSetValH,rmp
    ldi rmp,Low(TestNext)
    mov rSetValL,rmp
TestMotorLoop:
    rcall OC0AIsr
    rjmp TestMotorLoop
.endif
;
; Neujustierung?
    cbi pOkD,bOkD
    sbi pOkO,bOkO ; Pull-up einschalten
    sbic pOkI,bOkI ; Pin auf Null?
    rjmp LeseEeprom ; Nein
    ldi rmp,High(32768)
    mov rSetValH,rmp
    mov rCurrValH,rmp
    mov rOpenH,rmp
    mov rCloseH,rmp
    ldi rmp,Low(32768)
    mov rSetValL,rmp
    mov rCurrValL,rmp
    mov rOpenL,rmp
    mov rCloseL,rmp
    sbr rFlag,1<<bWrite
WaitRestart:
    ldi ZH,High(16667) ; 100 ms warten
    ldi ZL,Low(16667)
WaitInactive:
    sbis pOkI,bOkI ; Ueberspringe wenn Pin high
    rjmp WaitRestart ; Pin ist auf Null
    sbiw ZL,1 ; Abwaerts zaehlen
    brne WaitInactive ; Noch nicht 100 ms
    rjmp NoEepromRead ; Fertig gewartet
; Lese EEPROM-Inhalt
LeseEeprom:
    rcall ReadEep ; Lese den Datenset im EEPROM
NoEepromRead:
; Init Duo-LED
    sbi pDuoD,bDuoAR ; Richtung rote Anode Ausgang
    sbi pDuoD,bDuoAY ; Richtung gelbe Anode Ausgang
    cbi pDuoO,bDuoAR ; rote Anode auf Low
    cbi pDuoO,bDuoAY ; gelbe Anode auf Low
; Init Bereit-LED
    sbi pOkD,bOkD ; Bereit-LED Richtung Ausgang
    sbi pOkO,bOkO ; Bereit-LED aus
; Init Schalter
    cbi pSwD,bSwD ; Schalter Richtung auf Input
    sbi pSwO,bSwO ; Schalter Pull-Up An
; Init AD-Wandler
    clr rmp ; Mit Kanal 0 beginnen, URef=UB
    out ADMUX,rmp
    ldi rAdc,64 ; 64 Messungen
    ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
    out ADCSRA,rmp ; Starte erste Wandlung mit Interrupts
; Init Schrittmotor
; Schrittmotor

```



```

; .equ pStpO = PORTA ; Schrittmotor-Ausgabeport
; .equ pStpD = DDRA ; Schrittmotor-Richtungsport
; .equ mStpD = 0xF0 ; Maske Schrittmotor-Richtung
in rmp,pStpO ; Output-Port lesen
andi rmp,0x0F ; Unterste Bits erhalten, oberste = Null
out pStpO,rmp ; in Output-Port schreiben
in rmp,pStpD ; Richtungsregister lesen
ori rmp,mStpD ; mit Richtungsmaske odern
out pStpD,rmp ; Richtung Ausgang
; Init TC1 als Motormagnet-Abschaltung bei inaktiv
ldi rmp,High(cMotOff) ; Zeit bis Abschaltung
out OCR1AH,rmp
ldi rmp,Low(cMotOff)
out OCR1AL,rmp
clr rmp ; Normal Mode
out TCCR1A,rmp
ldi rmp,(1<<CS12)|(1<<CS10) ; Prescaler=1024
out TCCR1B,rmp
ldi rmp,1<<OCIE1A ; Compare Match A Interrupt
out TIMSK1,rmp
; Init TC0 als Schrittmotor-Steuerung
ldi rmp,cCtcDef ; niedrige Anfangsgeschwindigkeit
out OCR0A,rmp ; CTC-Wert setzen
ldi rmp,1<<WGM01 ; No OC output, WGM01 fuer CTC
out TCCR0A,rmp
ldi rmp,(1<<CS01)|(1<<CS00) ; CTC, Presc=64
out TCCR0B,rmp
ldi rmp,1<<OCIE0A ; Compare Match A Interrupt
out TIMSK0,rmp
; Sleep, externe ints und int enable
ldi rmp,(1<<ISC00)|(1<<SE) ; Externer INT0, Schlafmodus
out MCUCR,rmp
ldi rmp,1<<INT0 ; Ext INT0 enable
out GIMSK,rmp
sei ; Enable interrupts
;
; *****
;   P R O G R A M   L O O P
; *****
;
Loop:
    sleep
    nop
    sbrc rFlag,bAdc
    rcall AdcFlag
    sbrc rFlag,bWrite
    rcall WriteEep
    brtc Loop ; Kein Update Sollwert
    clt
    sbis pSwI,bSwI ; Ueberspringe bei Schalter offen
    rjmp SetOpen
    cli
    mov rSetValL,rCloseL ; Sollwert = ZU
    mov rSetValH,rCloseH
    sei
    rjmp Loop
SetOpen:
    cli
    mov rSetValL,rOpenL ; Sollwert Offen-Stellung
    mov rSetValH,rOpenH
    sei
    rjmp loop
;
; ADC-Flagge gesetzt, 64 Messungen fertig
AdcFlag:
    cbr rFlag,1<<bAdc ; Loesche Flagge
    in rmp,ADMUX ; Lese Wandlerkanal
    cpi rmp,1 ; Kanal ADC1?
    breq AdcFlag1
    brcs AdcFlag0
    rjmp AdcFlag2
AdcFlag0:
    ; Kanal 0 auswerten
    ; ADC-Ergebnissumme MSB invertieren und mit vier malnehmen
    ; ADCH(MSB) ADCL(LSB)
    ; xxxx.xxxx.xx(xx.xxxx)
    ; MSB nach R0, die beiden oberen
    ; Bits vom LSB von rechts
    ; einschieben
    ; R1      R0

```

```

; 0000.00xx.xxxx.xxxx
mov R0,rAdcH ; MSB ADC-Ergebnis
com R0 ; Potentiometer-Umkehr
clr R1 ; Loeschen obere Bits
rol rAdcL
rol R0 ; in LSB einschieben
rol R1 ; in MSB schieben
.if HalfStep == 1
lsl R0 ; in LSB rotieren
rol R1 ; in MSB rotieren
.endif
; Ergebnis zu 32.768 addieren
ldi ZH,High(32768)
ldi ZL,Low(32768)
sub ZL,R0
sbc ZH,R1
mov rOpenH,ZH ; Wert in rOpen kopieren
mov rOpenL,ZL
ldi rmp,1 ; auf Kanal 1 weitermessen
out ADMUX,rmp
set ; T-Flagge update Sollwert
rjmp AdcFlag9
AdcFlag1:
; Kanal 1 auswerten
; ADC-Ergebnis mit vier malnehmen
; ADCH(MSB) ADCL(LSB)
; xxxx.xxxx.xx(xx.xxxx)
; MSB nach R0, die beiden oberen
; Bits vom LSB von rechts
; einschieben
; R1 R0
; 0000.00xx.xxxx.xxxx
mov R0,rAdcH ; MSB ADC-Ergebnis in LSB
com R0 ; Invertieren
clr R1 ; Loeschen obere Bits
rol rAdcL
rol R0 ; in LSB einschieben
rol R1 ; in MSB schieben
.if Halfstep == 1
lsl R0 ; in LSB einschieben
rol R1 ; in MSB schieben
.endif
; Wert von 32.768 abziehen
ldi ZH,High(32768)
ldi ZL,Low(32768)
add ZL,R0
adc ZH,R1
mov rCloseH,ZH ; Und in rClose ablegen
mov rCloseL,ZL
ldi rmp,2 ; mit Kanal 2 weitermachen
out ADMUX,rmp
set ; T-Flagge fuer Sollwert setzen
rjmp AdcFlag9
AdcFlag2:
; Kanal 2 auswerten
.if Halfstep == 1
ldi rmp,120 ; Multiplikator halb
ldi ZH,14 ; Minimum CTC-Wert
.else
ldi rmp,225 ; Multiplikator ganz
ldi ZH,30 ; Minimum CTC-Wert
.endif
clr ZL
mov R0,rAdcH
clr R1
AdcFlag21:
lsr rmp ; Multiplikator rechts in Carry
brcc AdcFlag22 ; Null, nicht addieren
add ZL,R0 ; Eins, Multiplikator addieren
adc ZH,R1
AdcFlag22:
; Multiplikator mal zwei
lsl R0
rol R1
tst rmp ; Alle Einsen multipliziert?
brne AdcFlag21 ; nein, weiter
out OCR0A,ZH ; in Compare A schreiben
ldi rmp,0 ; auf Kanal 0 weitermessen
out ADMUX,rmp
AdcFlag9:

```

```

clr rAdcH ; Loeschen der Summen
clr rAdcL
ldi rAdc,64 ; 64 Messungen neu starten
ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
out ADCSRA,rmp ; Messung starten
ret

;
; *****
;   E E P R O M - S T E U E R U N G
; *****
;
; Schreibe aktuellen Datenset in das EEPROM
;
; Datenstruktur:
; .equ cStepSetS = 7 ; fuer Lesen/Schreiben EEPROM
; .def rMark1 = R7
; .def rCloseL = R8
; .def rCloseH = R9
; .def rOpenL = R10
; .def rOpenH = R11
; .def rCurrValL = R12
; .def rCurrValH = R13
; .def rMark2 = R14
; .equ cStepSetE = 15 ; Ende des Datensets
;
WriteEep:
sbic EECR, EEPE ; warte bis EEPROM bereit
rjmp WriteEep ; noch nicht bereit
cbr rFlag,1<<bWrite
ldi ZH,High(EepAdrs) ; Z auf EEPROM-Adresse
ldi ZL,Low(EepAdrs)
ldi YH,High(cStepSetS) ; Y auf Register
ldi YL,Low(cStepSetS)
out EEARH,ZH ; in EEPROM-Adressport
out EARL,ZL
ld rmp,Y+ ; Lese erstes Byte
out EEDR,rmp ; in EEPROM Datenport
cli ; Interrupts deaktivieren
sbi EECR, EEMPE ; EEPROM-Schreiben starten
sbi EECR, EEPE
sei ; Interrupts wieder aktivieren
sbi EECR,EERIE ; EEP-Interrupts einschalten
ret

;
; Lese EEPROM-Werte ein
; Datenstruktur:
; .equ cStepSetS = 7 ; fuer Lesen/Schreiben EEPROM
; .def rMark1 = R7
; .def rCloseL = R8
; .def rCloseH = R9
; .def rOpenL = R10
; .def rOpenH = R11
; .def rCurrValL = R12
; .def rCurrValH = R13
; .def rMark2 = R14
; .equ cStepSetE = 15 ; Ende des Datensets
ReadEep:
ldi ZH,High(EepAdrs) ; Z auf EEPROM-Werte
ldi ZL,Low(EepAdrs)
ldi XH,High(cStepSetS) ; X auf Step-Register
ldi XL,Low(cStepSetS)
ldi rmp,cStepSetE-cStepSetS ; Anzahl Bytes
mov R0,rmp
ReadEep1:
sbic EECR, EEPE ; warte bis EEPROM bereit
rjmp ReadEep1
ReadEep2:
out EEARH,ZH ; Setze Lese-Adresse EEPROM
out EARL,ZL
sbi EECR, EERE ; Lese-Befehl EEPROM
in rmp, EEDR ; Lese Byte
st X+,rmp ; speichere in Register
adiw ZL,1 ; naechste Adresse
dec R0 ; alle Bytes?
brne ReadEep2 ; nein, weiter
mov rmp,rMark1 ; Pruefe Werte
eor rmp,rMark2 ; Exklusiv-Oder
cpi rmp,0xFF ; Korrekt?
breq ReadEep3 ; Ja, korrekte Kennungen
; Falsche Kennungen, Ignoriere EEPROM-Daten

```

```

ldi rmp,High(32768) ; Setze Default
mov rOpenH,rmp ; auf Mitte
mov rCloseH,rmp
mov rCurrValH,rmp
clr rOpenL
clr rCloseL
clr rCurrValL
ReadEep3:
ret
;
; EEPROM-Inhalte Default
;
; Datenstruktur:
; .equ cStepSetS = 7 ; fuer Lesen/Schreiben EEPROM
; .def rMark1 = R7
; .def rCloseL = R8
; .def rCloseH = R9
; .def rOpenL = R10
; .def rOpenH = R11
; .def rCurrValL = R12
; .def rCurrValH = R13
; .def rMark2 = R14
; .equ cStepSetE = 15 ; Ende des Datensets
.eseg
.org 0x0010 ; EEPROM-Segment ab 0x10
EepAdrs:
; Defaultwerte: Kennung (0xAA), ZU-Wert, Auf-Wert, IST-Wert,
; SOLL-Wert, Kennung (0x55)
.db 0xAA ; Kennung 1
.dw 32768 ; Close-Wert
.dw 32768 ; Open-Wert
.dw 32768 ; Aktuelle Position Mitte
.db 0x55 ; Kennung 2
;
; Ende Quellcode
;

```