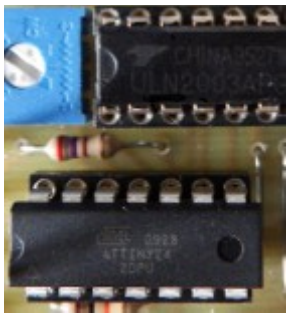
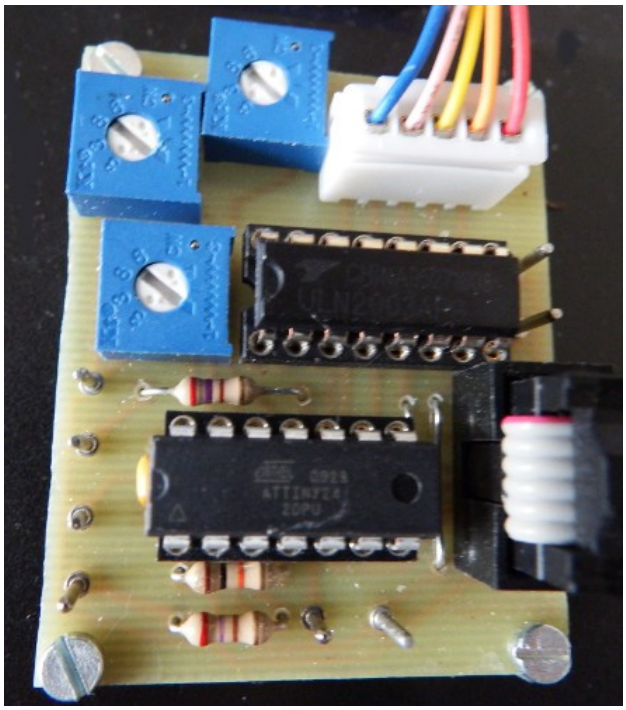


Applications of
AVR Single chip controllers AT90S,
ATtiny, ATmega and ATxmega
Stepper motor controller with an ATtiny24

Stepper motor 28BYJ-48 controller with an ATtiny24

Zipped version of these pages (291 kB)



Those who need a level-crossing or a gate closure for model-building are best suited with this stepper motor 48BYJ-24 and the ATtiny24 controller described here. It has the following properties:

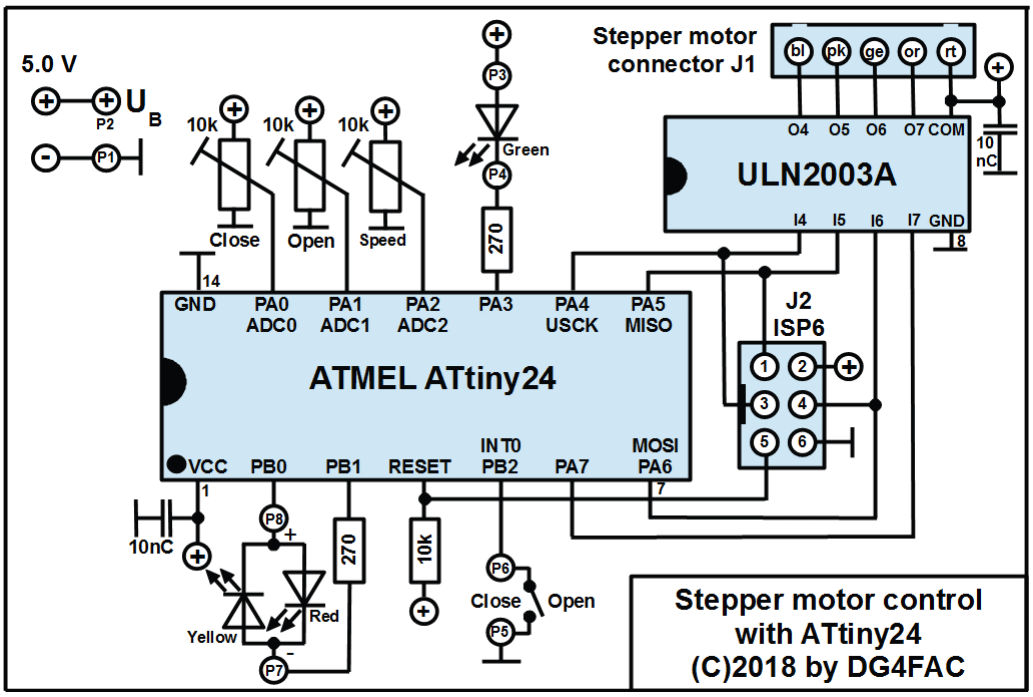
- low cost: in any case less than 10€ per piece,
- single operating voltage: motor and controller work with 5 V,
- very precise: rotates in 1,024 single steps over an angle of 180°,
- select-able speed: between two and 17 seconds per 180° rotation selectable with trim potentiometer,
- adjustable angles: maximum deflection in upper and lower position adjusted with trim potentiometers for a quick and convenient setup, no position sensors necessary,
- position storage: the last position is stored in EEPROM and is restored during start-up, possible correction of the middle position possible,
- reliable: runs like a Swiss clock, no sensible positioning equipment necessary,
- open source code: the assembler source is available here for free download and can be modified to fit to own needs, assembler source configurable for full- and half-step mode with only one small change.

For a voltage-controlled positioning system for a stepper motor with 12 V operating voltage that uses an ATtiny13 see this page.

Page top	1 Hardware	2 Mounting	3 Operation	4 Software	5 Trouble shooting

1 Hardware

1.1 Schematic



The stepper motor controller works with an ATtiny24. The motor control signals are outputted on the port bits PA4 to PA7 and drive the input pins of a darlington driver ULN2003A, which has protection diodes on its outputs for magnet applications. The output pins of the ULN2003A drive the magnets of the stepper motor 28BYJ-48 (see the schematic of the motor below). Software controls if only one single magnet is driven at a time (full step mode) or if one and two magnets are driven alternately (half step mode).

The AD converter input channels ADC0 and ADC1 are connected to trim potentiometers which determine the deflection in lower (closing) and upper (opening) direction.

AD converter input ADC2 is connected with a trim potentiometer that controls the motor speed.

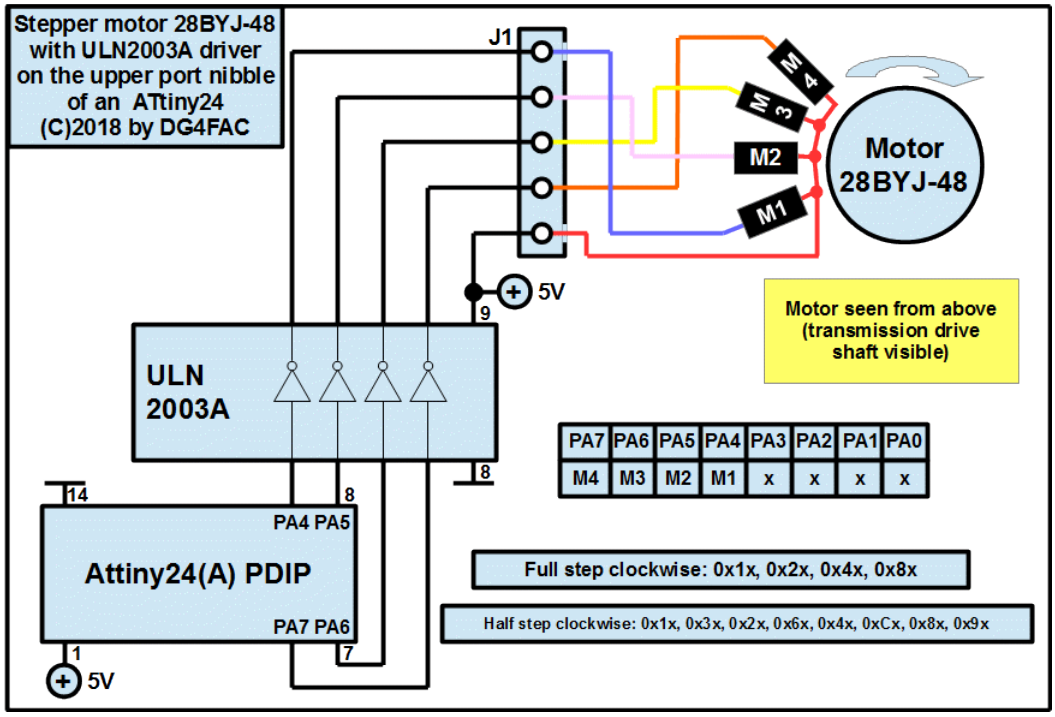
On the port outputs PB0 and PB1 a duo LED is driven. This LED blinks in red (closing) or yellow (opening) as long as the motor moves. If the motor reaches its end position the LED is permanently on and is terminated when an additional delay time has elapsed and the motor magnets are switched off (currently one second, can be up to 6.7 seconds).

On port pin PA3 a green LED is attached that is illuminated when the motor is on its end position and the delay time is elapsed. This also signals that the current position is stored in the EEPROM.

The Open/Close switch on PB2 (INT0) opens or closes the level-crossing or gate.

The controller can be programmed in the final application by In-System-Programming via the standard six-pin plug.

1.2 Schematic of the stepper motor



This is the schematic of the motor. Only four of the 16 magnets are displayed. Shown is the motor from the upper side where the axle is seen, magnet numbers are for movement in the right direction of the axle.

Together with the ULN2003A driver connections and the connections with the port pins of the ATtiny24 the magnet control for movement in right direction is shown in hexadecimal format, with which the upper port nibble has to be controlled.

The motor needs 64 single steps per round and has a built-in gear 1:64, so per round of the gear 64 * 64 = 2,048 single steps are required in full step mode. In half step mode 4,096 steps are required.

The motor can be driven in full step mode with a maximum speed of 4.2&s per gear round (488 Hz) without any load. At frequencies above the motor or at higher load it is not rotating smoothly any more or is even not moving at all. In half step mode the frequency can be slightly higher than doubled because of the 50% higher power. But slightly above 1,000 Hz it also reaches the end of correct rotation.

1.3 Component list

The necessary electronic parts are listed here. Prices are, of course, subject to changes.

The stepper motor 28BYJ-48 is not included in the list. This part can be ordered from traders on the internet quite cheaply for less than 5€ per piece. The driver board that is delivered with that has a plugged ULN2003A on board that can be re-

Stepper ATtiny24 components

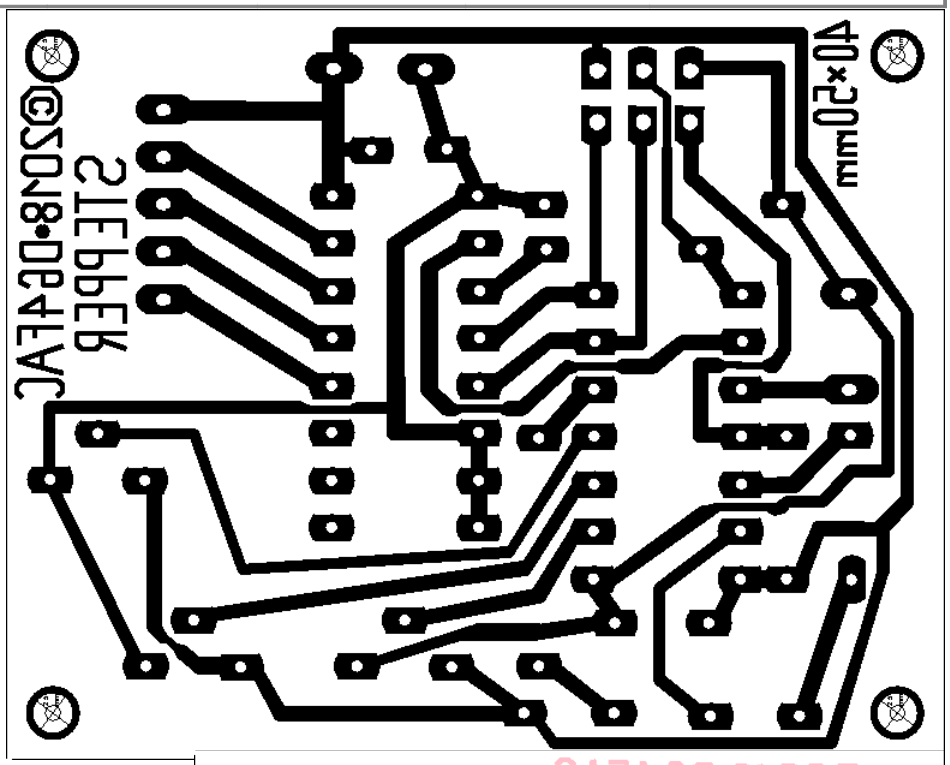
			Prices €	
Nmbr	Component	Reichelt	Single	Total
1	ATtiny24	ATTINY 24-20 PU	1,99	1,99
1	ULN2003A	ULN 2003A	0,32	0,32
1	LED 3mm green	LED 3MM GN	0,06	0,06
1	Duo LED 3 mm rot/yellow	LED 3 RY	0,18	0,18
3	Trim potentiometer 10k	76-10 10K	0,31	0,93
2	Resistor 270 Ohm	1/4W 270	0,103	0,21
1	Resistor 10 k	1/4W 10K	0,103	0,10
2	Ceramic capacitors 10n	Z5U-5 10N	0,03	0,06
1	Box connector 6-pin	WSL 6G	0,16	0,16
1	JST pin header 1x5 pin	JST XH5P ST	0,22	0,22
1	Switch	MS 244	1,7	1,70
8	Soldering tags 1mm	RF 1-100	0,0337	0,27
8	Soldering pins 1mm	RTM 1-100	0,018	0,14
0,08	Silver wire 0.6mm	SILBER 0,6MM	0,26	0,02
Total				**6,36**

moved and used here.

2 Mounting

2.1 PCB

The PCB that takes all the components is seen here. It is of a size of 50-by-40 mm and single sided and includes four mounting holes with a diameter of 2.5 mm.

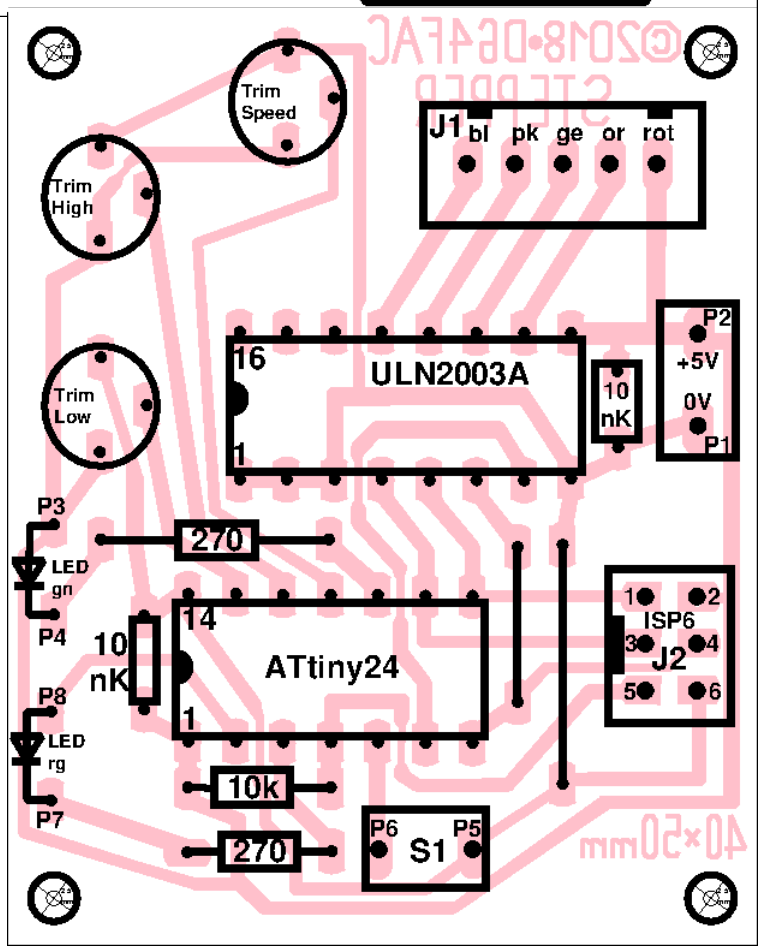


2.2 Component placement

Two wired bridges have to be soldered first. As trim potentiometer round as well as squared types can be used.

When adding the two plugs J1 and J2 please be aware of the spares, when mounting the IC sockets for the PDIP packages the mark for pin 1 is relevant.

The 6 pin box connector J2 is only to be mounted if you want to program the ATtiny24 in ISP mode.



3 Operation

3.1 Start-up

The stepper motor should be mounted so that he is roughly in the middle position (half open). Slight deviations from the middle position are tolerable. If that is impossible, consult chapter 3.3 on how to adjust the motor's middle position.

With the switch open the upper point is adjusted with the upper position trim potentiometer. After closing the switch adjust the lower position. If the motor keeps moving one step up and one step down again, choose a slightly different position to ensure that the position is stored in the EEPROM and that the motor consumes not more power than necessary.

The rotating speed can be adjusted as desired. At higher loads, select a lower speed that ensures that the motor is running smoothly.

3.2 Normal operation

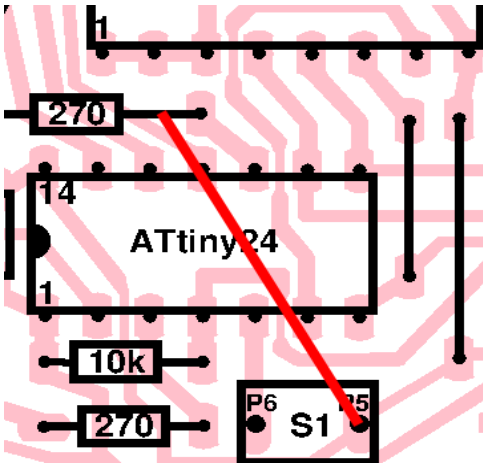
Under normal operation the switch opens and closes the level-crossing or the gate. If closing shall be initiated by an external signal, tie pin P5 to ground. Ensure that the signal duration is long enough to reach the green LED phase to ensure position storage.

Switch the operating voltage off, when the green LED is on, otherwise correct positioning does not work correct and re-adjustment might become necessary. This might become necessary if the upper or lower position cannot be adjusted any more. Consult chapter 3.3 in that case.

3.3 Re-adjusting middle position

The middle position is selected if both positioning trim potentiometers are in their left position (and switching does not result in motor movement any more). If, during mounting, the middle position is unreachable or if during operation this has moved too far up or down, re-adjust the middle position by following this procedure:

1. Bring both trim potentiometers to their leftmost position.
2. With the switch open (middle position to low) or closed (middle position to high) adjust the motor to the middle position with the respective trim potentiometer.
3. Switch the supply voltage off and either remove the supply or wait until it has fallen to below 1.7 V.
4. Connect the 270Ω resistor's lead that drives the green LED and is connected with pin 10 of the ATtiny24 with ground on pin P5.
5. Switch the operating voltage on and then remove the connection between pin 10 of the ATtiny24 and ground. The motor should then move to a different position.
6. Wait until the motor has reached its end position and until the green LED is on.



If the middle position could still not be reached with one re-adjustment, you can repeat this procedure.

4 The software

The software is written in AVR assembler and is extensively commented. The source code can be downloaded here or viewed in the browser here. The code uses .if directives and can be assembled either with the gavrasm assembler or with ATMEL's assembler 2 (a how-to-page for assembling with gavrasm is available for Linux here and for Windows here).

Functioning of the software is elaborated in detail in the following chapters.

4.1 Positioning of the stepper motor

The current position of the stepper motor is held in the 16 bit register pair rCurrValH:rCurrValL. This value should be between 32,768 +/- 511 (full step mode) resp. 32,768 +/- 1,024 (half step mode). Change the mode in the head of the source code before assembling.

The desired target position of the step motor is in the register pair rSetValH:rSetValL. If CurrVal and SetVal differ, the motor moves in direction to SetVal (see the description in chapter 4.2 below).

The position in the open state is (32,768 - N) in register pair rOpenH:rOpenL, in closed position (32,768 + N) in rCloseH:rCloseL. Both positions are calculated from the respective trim potentiometer positions.

If the switch changes its state an INT0 interrupt is triggered and the T flag in the status register is set. Depending from the switch's position either the rOpen or the rClose value is written to rSetVal. The T flag is also set whenever new position values come in.

4.2 Stepper motor movement

Positioning of the stepper motor is performed by the 8 bit timer TC0. This timer runs with a prescaler of 64 (1 MHz / 64 = 15,625 Hz), each timer tick is 64 µ long. The timer runs in CTC mode (Clear Timer on Compare) and clears the counter after reaching and exceeding the Compare A value. Depending from the trimmed speed, this happens either after reaching 15 (half step mode, fastest speed) in 15*64 = 960 µs resp. 30 (full step mode, fastest speed) after 30*64 = 1,920 µs or after reaching 256 (0) (full step mode, lowest speed) after 256*64 = 16,384 µs.

Triggered by the TC0COMPA timer interrupt the interrupt service routine first checks if rCurrVal and rSetVal are equal. If that is the case, nothing else happens. If rCurrVal is smaller than rSetVal, rCurrVal is increased, otherwise it is decreased. In both cases the lowest two bits (full step mode) resp. three bits (half step mode) are used to read the magnets to be switched on from the respective table and to write those to the upper port nibble of port A of the ATtiny24. This also switches the green LED off (on pin PA3).

4.3 Switching the magnets off

The timing of the period after which the magnets of the motor are turned off following the last movement is performed by 16 bit timer TC1. This timer is clocked with a prescaler of 1,024.

Each TC0 timer interrupt, by which the position of the motor is changed, clears the timer TC1. Following the last movement TC1 runs and after the pre-selected time reaches its Compare A value. One second is selected by default, can be up to 6.7 seconds. The triggered TC1COMPA interrupt

- if the green LED was not lit, sets the flag bWrite to trigger the subsequent writing of the positions to the EEPROM,
- switches the motor magnets off (0000 to PA4 to PA7),
- switches the green LED on (PA3 to 0), and
- switches the red/yellow duo LED off (PB0 and PB1 to 0).

4.4 Reading positions from the EEPROM

At start-up of the controller the following bytes from address 0x0010 of the EEPROM are read and written to the respective registers:

1. a first mark Mark1 (should be 0xAA),
2. the currently desired close value (rCloseL, rCloseH),
3. the currently desired open value (rOpenL, rOpenH),
4. the current motor position (rCurrValL, rCurrValH),
5. a second mark Mark2 (should be 0x55).

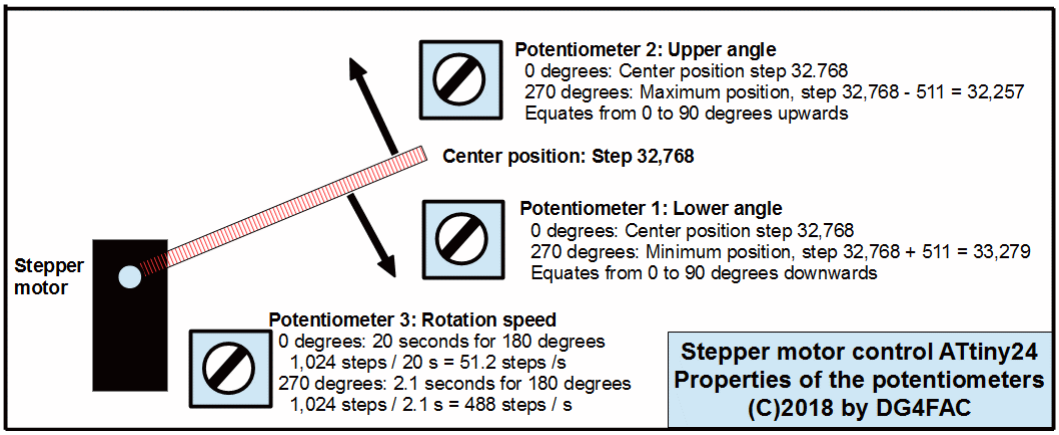
If an exclusive or of Mark1 and Mark2 yields 0xFF the EEPROM data read is correct and initiation continues. If not motor position and open and close value are set to 32,768 (0x8000).

4.5 Writing positions to the EEPROM

If the stepper motor reaches its end position and the additional delay time has elapsed, the same bytes are copied from the registers to the EEPROM. To do this, the first data byte, Mark1, is written to the EEPROM data port and the address port is set to 0x0010. The first EEPROM write is the started and the EERDY interrupt enable bit is set. Further EEPROM writes are performed in the interrupt service routine until all bytes are written.

4.6 Measuring trim potentiometer values and calculating positions/speed

This demonstrates the functions of the three potentiometers. Potentiometer 1 and 2 control the excursion from the middle position in upper and lower direction. Potentiometer 3 controls the speed of the stepper movement.



All three potentiometer positions are determined in cycles in which 64 single measurements are performed and summed up (10 bit ADC result, summed up to 16 bit sum values).

4.6.1 Calculating position values

In full step mode, the inverted MSB of the summed up result is multiplied by two, in half step mode by four. This value is subtracted from 32,768 to yield the rOpen value resp. added to 32,768 to yield the rClose value. The result is copied to the respective registers and the T flag is set, triggering update of the rSetVal.

4.6.2 Calculating speed and CTC values

The MSB of the summed up measurements is multiplied by (255-30)=225 (full step mode) resp. (255-14)=241 (half step mode) and 30 (full step mode) resp. 14 (half step mode) is added. The resulting value is written to the compare port OC0A of the 8 bit timer TC0 and controls the speed of motor movements.

4.7 Re-adjust

On each start-up of the controller the pull-up resistor on PA3 (cathode green LED, 270Ω resistor) is switched on and it is checked if the PA3 input is clear. If that is the case, the current motor position rCurrVal, the target position rSetVal as well as the rClose and rOpen registers are all set to 32,768 and reading of the EEPROM data is skipped.

If the PA3 input pin reaches high, a period of 100 ms is started over which any low on this pin restarts this period (toggle protection). Only after this period normal processing continues.

Page top	1 Hardware	2 Mounting	3 Operation	4 Software	5 Trouble shooting

5 Trouble-Shooting

5.1 Motor runs over and over

If the motor runs permanently in the same direction and does not stop, this can be caused by spikes on the operating voltage. Those spikes restart the ATtiny24 and will occur if you did not mount the 10nF capacitor near the ULN2003A. Make sure that this capacitor is attached and is functioning.

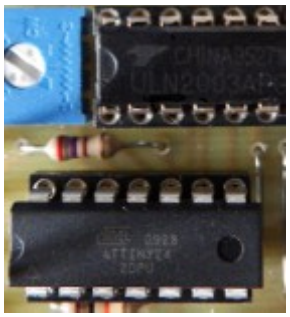
This can also occur if your supply is overloaded by the magnet current, which is slightly above 100 mA in full step mode or 200 mA in half step mode. In that case use a higher rated supply.

5.2 ISP programming of the ATtiny24 fails

If you program the ATtiny24 in the system with the ISP6 plug and if the motor is attached, the programming pulses cause a rapid switching of up to three magnets. If your power supply is only rated for 300 mA or less (e.g. in case of a supply over an USB interface), this can cause power spikes and programming fails. Use a higher rated supply for programming in that case.

Rapid switching of up to three magnets can also result in high back-currents, which can cause large spikes when short-circuited by the built-in diodes in the ULN2003A. Programming failure occurs also if the 10nF capacitor near the ULN2003A is not in place or not working correct. Increase its capacity to 100nF or tempoarily attach an electrolytical capacitor of 10µF to the power supply pins to cope with that.

Praise, error reports, scolding and spam please via the comment page to me. Spammers: please note that behind this is not a dump standard script but I personnally select the entries to be published. So, even if you try it five times (some spammers indeed do that) it will fail and ads for pharmacy will not occur here.



Applications of
AVR Single chip controllers AT90S,
ATtiny, ATmega and ATxmega
Stepper motor controller with an ATtiny24

Software for the stepper motor 28BYJ-48 control with an ATtiny24

Assembler source code (as asm source code here)

```
;
; ************************************
; * Stepper motor control with ATtiny24 *
; * 16 bit position, Full/Half step     *
; * Version 1, March 2018, V1           *
; * (C)2018 by avr-asm-tutorial.net     *
; ************************************
;
.nolist
.include "tn24adef.inc" ; Define device ATtiny24A
.list
;
; *********************************
;    M O D E - S E L E C T I O N
; *********************************
;
.equ Halfstep = 0 ; 1=Halb step, 0=Full step
;
; *********************************
;    D E B U G - S W I T C H E S
; *********************************
;
; Debug off: all switches to 0
;
; Test ADC value calculation
.equ TestAdc = 0 ; Test ADC channel number
  .equ TestChannel=2 ; Channel to simulate (0..2)
  .equ TestValue = 255*256
;
; Test motor driver output
.equ TestMotor = 0 ; Test motor driver
  .equ TestCurr = 32768 ; current position
  .equ TestNext = 32767 ; next position
;
;        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 AnRd o--|PB0   ADC0|--o Potentiometer CLOSE
;             3|          |12
; LED AnYe o--|PB1   ADC1|--o Potentiometer OPEN
;             4|          |11
;    RESET o--|RES   ADC2|--o Potentiometer Speed
;             5|          |10
;   Switch o--|PB2    PA3|--o LED Cathode 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 control
.equ pDuoO = PORTB ; Duo-LED output port
.equ pDuoD = DDRB ; Duo-LED direction port
.equ bDuoAR = PORTB0 ; Red anode portpin
.equ bDuoAY = PORTB1 ; Yellow anode portpin
; Switch control
.equ pSwO = PORTB ; Switch output port
.equ pSwD = DDRB ; Switch direction port
.equ pSwI = PINB ; Switch input port
.equ bSwO = PORTB2 ; Switch output portpin
.equ bSwD = DDB2 ; Switch direction portpin
.equ bSwI = PINB2 ; Switch input portpin
; OK-LED control
.equ pOkO = PORTA ; OK-LED output port
.equ pOkD = DDRA ; OK-LED direction port
.equ pOkI = PINA ; OK-LED input port
.equ bOkO = PORTA3 ; OK-LED output portpin
.equ bOkD = DDA3 ; OK-LED direction portpin
.equ bOkI = PINA3 ; OK-LED input portpin
; Stepper motor control
.equ pStpO = PORTA ; Stepper output port
.equ pStpD = DDRA ; Stepper direction port
.equ mStpD = 0xF0 ; Mask stepper direction
;
; *********************************
;   A D J U S T A B L E   C O N S T
; *********************************
;
; Clock frequency default (CLKDIV8 set)
.equ clock=1000000 ; Define clock frequency
;
; Degrees maximum angle (maximum gear):
.equ cAngleMax = 180 ; Degrees maximum angle
;
; Start speed, default until first measurement
;   is completed
.equ cSecOpenClose = 10 ; 10 seconds
;
; Time after last motor move until magnets off
;   Minimum: 1 (1 ms), Maximum: 6,710 (6,71 s)
.equ cMotOffms = 1000 ; Milli seconds until motor off
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
; *********************************
;
; Motor with gear 1:64
; Stepps per round:
.equ cStepsRound = 64 * 64
; Medium position:
.equ cMiddle = 32768
; Steps per maximum angle:
.equ cStepsAngle = (cStepsRound*cAngleMax+180)/ 360
; Stepps per half maximum angle
; (from middle to max. up or down):
.equ cStepsUpDown = (cStepsAngle+1) / 2
;
; Calculation of CTC value for start speed
.equ cTC0Presc = 64 ; Prescaler TC0
; TC0Ticks in micro seconds:
.equ cTC0Tick = (1000000*cTC0Presc)/clock
; Duration in micro seconds for full opening/closing
```

```
;   over maximum angle
.equ cUsSwing = (1000000*cSecOpenClose)/cStepsAngle
; CTC value for 8 bit TC0
.equ cCTCDef = (cUsSwing+cTC0Tick/2)/cTC0Tick
;
; CTC value for 16 bit TC1 motor off
.equ cMotOff = (1000*cMotOffms+512)/1024-1
;
; *********************************
;          T I M I N G
; *********************************
;
; TC0 is stepper motor position control
;   runs in CTC mode with Compare A
;   clock               = 1,000,000 Hz
;   Vorteiler           =        64
;   Timer tick          =        64 us
; Full step mode:
;   Steps for 180 deg   =   1,024
;   Slow motion         =        16.78 s
;     Time per step     =    16,384 us
;     CTC ticks         =       256
;     Compare A value   =       255
;   Rapid motion        =         2.03 s
;     Time per step     =     1,984 us
;     CTC ticks         =        31
;     Compare A value   =        30
;     (Remark: limited to fastest motor movement)
;   Calculation of CTC from MSB ADC sum on ADC2:
;     Compare A = (ADC-MSB * 225)/256+30
; Half step mode:
;   Steps 180 degrees   =     2,048
;   Slow motion         =        17.7 s
;     Time per step     =    16,384 us
;     CTC ticks         =       135
;     Compare A value   =       134
;   Rapid motion        =         1.97 s
;     Time per step     =       960 us
;     CTC ticks         =        15
;     Compare A value   =        14
;   Calculation of CTC from MSB ADC sum on ADC2:
;     Compare A = (ADC-MSB * 241)/256+14
;
; TC1 is timing the motor off (magnets off)
;   runs in normal mode with Compare A
;   Clock               = 1,000,000 Hz
;   Prescaler           =     1,024
;   Timer tick          =     1,024 us
;   Maximum until OVF   =    65,536
;   Respective time     =         6.7 s
;   Adjusted to         =         1 s
;
; ADC measures potentionmeter 1, 2 and 3
;   Clock               = 1,000,000 Hz
;   ADC prescaler       =       128
;   ADC tick            =       128 us
;   Conversion clocks   =        14
;   Time per meassurem. =     1,792 us
;   64 measurements     =   114,688 us
;   Three channels      =   344,064 us
;   Cycles per second   =         2.9
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; R0, R1, R2, used for multiplications etc.
; free: R4
; ADC value summing
.def rAdcL = R5 ; Sum ADC value, LSB
.def rAdcH = R6 ; dto., MSB
```

```
; Register values for stepper positioning
  .equ cStepSetS = 7 ; for Reading/Writing 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 ; End of the data set
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside interrupts
.def rFlag = R18 ; Flags
  .equ bWrite = 0 ; Write data set to EEPROM
  .equ bAdc = 1 ; 64 AD conversions completed
.def rAdc = R19 ; ADC counter
.def rSetValL = R20 ; Current position stepper, LSB
.def rSetValH = R21 ; dto., MSB
; free: R22 to R25
; used: R27:R26 = X Pointer outside interrupts
; used: R29:R28 = Y Pointer for EEPROM write
; used: R31:R30 = Z Pointer multipurpose, for LPM
;
; *********************************
;          S R A M
; *********************************
;
.dseg
.org SRAM_START
; (Not used, only for stack)
;
; *********************************
;         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, switch
        reti ; PCI0
        reti ; PCI1
        reti ; WATCHDOG
        reti ; ICP1
        rjmp OC1AIsr ; OC1A, motor time out
        reti ; OC1B
        reti ; OVF1
        rjmp OC0AIsr ; OC0A, motor movement
        reti ; OC0B
        reti ; OVF0
        reti ; ACI
        rjmp AdcIsr ; ADCC, Potentiometers
        rjmp EepIsr ; ERDY, EEPROM write
        reti ; USI_STR
        reti ; USI_OVF
;
; *********************************
;  I N T - S E R V I C E   R O U T .
; *********************************
;
; INT0 Interrupt Service Routine
;   Switch actions trigger INT0
;   on rising and falling edges
Int0Isr:
  set ; Update set value for motor
  reti
```

```
;
; OC0A Interrupt Service Routine
;   moves stepper if necessary
OC0AIsr:  in rSreg,SREG ; Save SREG
  cp rCurrValL,rSetValL ; LSB rCurr = rSet?
  brne OC0AIsr1 ; LSB differs
  cp rCurrValH,rSetValH
  breq OC0AIsr8 ; MSB equals, too, do nothing
  cp rCurrValL,rSetValL ; LSB compare again
OC0AIsr1:
  ; rCurr and rSet differ, Move motor
  cpc rCurrValH,rSetValH ; Test MSB
  brcs OC0AIsr4 ; rCurr smaller than rSet
  ; rCurr larger than rSet, move backwards
  tst rCurrValL ; Check LSB=0
  brne OC0AIsr2 ; LSB not zero
  dec rCurrValH ; Decrease MSB
OC0AIsr2:
  dec rCurrValL ; One step back
  sbi pDuoO,bDuoAY ; Yellow LED on
  cp rCurrValL,rSetValL ; Last step
  breq OC0AIsr3 ; Yes, do not blink
  sbrc rCurrValL,3 ; Bit 3 for blink
  cbi pDuoO,bDuoAY ; Yellow LED off
OC0AIsr3:
  cbi pDuoO,bDuoAR ; Cathode yellow LED low
  rjmp OC0AIsr7 ; Adjust stepper position
  ; rCurr smaller than rSet
OC0AIsr4:
  inc rCurrValL ; One step forward
  brne OC0AIsr5 ; LSB not zero
  inc rCurrValH ; Increase MSB
OC0AIsr5:
  sbi pDuoO,bDuoAR ; Anode red LED on
  cp rCurrValL,rSetValL ; Last step?
  breq OC0AIsr6 ; Yes, do not blink
  sbrc rCurrValL,3 ; Bit 3 for blink
  cbi pDuoO,bDuoAR ; Red LED off
OC0AIsr6:
  cbi pDuoO,bDuoAY ; Cathode red LED low
OC0AIsr7:
  push ZH ; Save Z
  push ZL
  mov rimp,rCurrValL ; Copy current position
.if Halfstep == 1
  ldi ZH,High(2*StepTab8) ; Z to 8 step table
  ldi ZL,Low(2*StepTab8)
  andi rimp,0x07 ; Isolate the lowest three bit
  .else
  ldi ZH,High(2*StepTab4) ; Z to 4 step table
  ldi ZL,Low(2*StepTab4)
  andi rimp,0x03 ; Isolate the lowest two bit
  .endif
  add ZL,rimp ; Add to table address
  ldi rimp,0 ; Add carry
  adc ZH,rimp
  lpm rimp,Z ; Read table byte
  in ZL,pStpO ; Read current port outputs
  andi ZL,0x0F ; Save the lowest four bits
  or ZL,rimp ; Set motor and Ok output bits
  out pStpO,ZL ; To motor output, clear OK LED
  pop ZL ; Restore Z
  pop ZH
  clr rimp ; Restart motor off timer
  out TCNT1H,rimp
  out TCNT1L,rimp
OC0AIsr8:
  out SREG,rSreg ; Restore SREG
  reti
;
.if Halfstep == 1
```

```
  ; Step table half step mode
  StepTab8:
  .db 0x18,0x38,0x28,0x68,0x48,0xC8,0x88,0x98
  .else
  ; Step table full step mode
  StepTab4:
  .db 0x18,0x28,0x48,0x88
  .endif
;
; OC1A-ISR switches motor magnets off
;
OC1AIsr:
  in rSreg,SREG ; Save SREG
  sbic pOkO,bOkO ; Green LED already on? If not ...
  sbr rFlag,1<<bWrite ; Write data set to EEPROM
  in rimp,pStpO ; Read motor output port
  andi rimp,0x07 ; Save lower three bits
  out pStpO,rimp ; Motor off, green LED on
  cbi pDuoO,bDuoAR ; Red/Yellow LED off
  cbi pDuoO,bDuoAY
  out SREG,rSreg ; Restore SREG
  reti
;
; AD Conversion complete interrupt
AdcIsr:
  in rSreg,SREG ; SaVE status
  in rimp,ADCL ; Read LSB ADC result
  add rAdcL,rimp ; Add to sum
  in rimp,ADCH ; Read MSB ADC result
  adc rAdcH,rimp ; Add to sum with carry
  dec rAdc ; Count downwards
  brne AdcIsr1 ; Not yet 64 measurements
  sbr rFlag,1<<bAdc ; Set flag
  rjmp AdcIsr2 ; Do not start next
AdcIsr1:
  ; Start next measurement
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  out ADCSRA,rimp
AdcIsr2:
  out SREG,rSreg ; Restore status
  reti
;
; EEPROM Ready interrupt
EepIsr:
  in rSreg,SREG ; Save SREG
  in rimp,EEARL ; Read LSB address
  inc rimp ; Next address
  out EEARL,rimp ; To address port
  ld rimp,Y+ ; Read next byte from register
  out EEDR,rimp ; To data port
  sbi EECR, EEMPE ; Enable EEPROM write
  sbi EECR, EEPE ; Start EEPROM write
  cpi YL,cStepSetE ; Last byte?
  brcs EepIsr1 ; No, continue write
  cbi EECR,EERIE ; No more EEP-RDY interrupts
  cbr rFlag,1<<bWrite ; Clear write flag
EepIsr1:
  out SREG,rSreg ; Restore SREG
  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
;
; Check debug switches
.if TestAdc == 1
  ; Test calculation from ADC values
```

```
  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 ; Test motor positioning
  ldi rmp,High(TestCurr) ; Current position
  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
  ;
  ; Re-adjust?
  cbi pOkD,bOkD ; Direction of green LED input
  sbi pOkO,bOkO ; Switch on pull-up resistor
  sbic pOkI,bOkI ; Pin low?
  rjmp LeseEeprom ; No, no re-adjust
  ; Re-adjust to middle position
  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 ; Set write flag EEPROM
WaitRestart:
  ldi ZH,High(14287) ; Wait for 100 ms
  ldi ZL,Low(14287)
WaitInactive:
  sbis pOkI,bOkI ; Skip next if pin  is high
  rjmp WaitRestart ; Pin is low, restart loop
  sbiw ZL,1 ; Count down
  brne WaitInactive ; Not yet 100 ms delay
  rjmp NoEpromRead ; Ready, do not read EEPROM
  ; Read EEPROM data set
LeseEeprom:
  rcall ReadEep ; Read the data set from EEPROM
NoEpromRead:
  ; Init Duo-LED
  sbi pDuoD,bDuoAR ; Direction red anode output
  sbi pDuoD,bDuoAY ; Direction yellow anode output
  cbi pDuoO,bDuoAR ; Red anode low
  cbi pDuoO,bDuoAY ; Yellow anode low
  ; Init green Ok LED
  sbi pOkD,bOkD ; Ok LED direction output
  sbi pOkO,bOkO ; Ok LED off
  ; Init switch
  cbi pSwD,bSwD ; Direction of switch to input
  sbi pSwO,bSwO ; Switch input pull-Up resistor on
  ; Init AD converter
  clr rmp ; Start with channel 0 beginnen, URef=5V
  out ADMUX,rmp
  ldi rAdc,64 ; 64 measurements
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  out ADCSRA,rmp ; Start first conversion and interrupts
```

```
  ; Init stepper motor
  in rmp,pStpO ; Read output on port
  andi rmp,0x0F ; Save lower four bits, upper nibble=0
  out pStpO,rmp ; To output port motor
  in rmp,pStpD ; Read direction port
  ori rmp,mStpD ; OR with motor direction mask
  out pStpD,rmp ; To direction port as output
  ; Init TC1 as motor magnet off timer
  ldi rmp,High(cMotOff) ; Time until motor off
  out OCR1AH,rmp ; to OCR1A
  ldi rmp,Low(cMotOff)
  out OCR1AL,rmp
  clr rmp ; TC1 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 as stepper motor control
  ldi rmp,cCtcDef ; Lowest start speed
  out OCR0A,rmp ; Set CTC value
  ldi rmp,1<<WGM01 ; No OC output, WGM01 for 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, external ints and int enable
  ldi rmp,(1<<ISC00)|(1<<SE) ; External INT0, sleep mode idle
  out MCUCR,rmp
  ldi rmp,1<<INT0 ; External 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 ; Convert ADC value
  sbrc rFlag,bWrite
  rcall WriteEep ; Start EEPROM write
  brtc Loop ; No update of set value
  clt ; Clear set value update flag
  sbis pSwI,bSwI ; Skip next if switch open
  rjmp SetOpen ; Switch is open
  cli ; Disable interrupts
  mov rSetValL,rCloseL ; Set value = rClose
  mov rSetValH,rCloseH
  sei ; Enable interrupts
  rjmp Loop
SetOpen:
  cli ; Disable interrupts
  mov rSetValL,rOpenL ; Set value = rOpen
  mov rSetValH,rOpenH
  sei ; Enable interrupt
  rjmp loop
;
; ADC flag set, 64 measurements complete
AdcFlag:
  cbr rFlag,1<<bAdc ; Clear flag
  in rmp,ADMUX ; Read current channel from MUX
  cpi rmp,1 ; Channel = ADC1?
  breq AdcFlag1 ; Yes
  brcs AdcFlag0 ; Channel = ADC0
  rjmp AdcFlag2 ; Channel = ADC2
AdcFlag0:
  ; ADC0 was measured
```

```
  ; Invert ADC value and multiply by 2
  mov R0,rAdcH ; Copy MSB ADC result to LSB
  com R0 ; Invert result
  clr R1 ; Clear MSB
  rol rAdcL ; Shift upper bit of LSB result to carry
  rol R0 ; Rotate into LSB
  rol R1 ; And into MSB
.if HalfStep == 1
  ; Half step mode, multiply by 2 again
  lsl R0 ; Rotate LSB left
  rol R1 ; Rotate into MSB
  .endif
  ; Subtract result from 32.768
  ldi ZH,High(32768)
  ldi ZL,Low(32768)
  sub ZL,R0
  sbc ZH,R1
  mov rOpenH,ZH ; Copy value to rOpen
  mov rOpenL,ZL
  ldi rmp,1 ; Continue measurements on channel 1
  out ADMUX,rmp
  set ; Set T flag for updating set value
  rjmp AdcFlag9
AdcFlag1:
  ; Channel 1 was measured
  mov R0,rAdcH ; MSB ADC result to LSB
  com R0 ; Invert result
  clr R1 ; Clear upper bits
  rol rAdcL ; Shift upper bit of LSB result to carry
  rol R0 ; Rotate into LSB
  rol R1 ; And into MSB
.if Halfstep == 1
  ; Half step mode, multiply by 2 again
  lsl R0 ; Left shift LSB
  rol R1 ; Rotate into MSB
  .endif
  ; Add value to 32.768
  ldi ZH,High(32768)
  ldi ZL,Low(32768)
  add ZL,R0
  adc ZH,R1
  mov rCloseH,ZH ; Copy result to rClose
  mov rCloseL,ZL
  ldi rmp,2 ; Continue with channel 2
  out ADMUX,rmp
  set ; Set T flag for updating set value
  rjmp AdcFlag9
AdcFlag2:
  ; Channel 2 was measured
.if Halfstep == 1
  ldi rmp,120 ; Multiplicator for half step
  ldi ZH,14 ; Minimum CTC value
  .else
  ldi rmp,225 ; Multiplicator for full step
  ldi ZH,30 ; Minimum CTC value
  .endif
  clr ZL ; Clear LSB result
  mov R0,rAdcH ; Copy MSB ADC result to LSB
  clr R1 ; Clear R1 as MSB
AdcFlag21:
  lsr rmp ; Multiplicator right shift into carry
  brcc AdcFlag22 ; Zero, do not add
  add ZL,R0 ; One, add multiplicator
  adc ZH,R1
AdcFlag22:
  ; Multiply multiplicator by two
  lsl R0
  rol R1
  tst rmp ; All ones multiplied?
  brne AdcFlag21 ; No, continue
  out OCR0A,ZH ; Write MSB result to Compare A
```

```
  ldi rmp,0 ; Continue with channel 0
  out ADMUX,rmp
AdcFlag9:
  clr rAdcH ; Clear sum
  clr rAdcL
  ldi rAdc,64 ; Restart 64 measurements
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  out ADCSRA,rmp ; Start first measurement
  ret
;
; ************************************
;    E E P R O M - C O N T R O L
; ************************************
;
; Write current data set in registers to EEPROM
;
WriteEep:
  sbic EECR, EEPE ; Wait until EEPROM is ready
  rjmp WriteEep ; Not yet ready
  cbr rFlag,1<<bWrite ; Clear write flag
  ldi ZH,High(EepAdrs) ; Z to EEPROM start address
  ldi ZL,Low(EepAdrs)
  ldi YH,High(cStepSetS) ; Point Y to first register
  ldi YL,Low(cStepSetS)
  out EEARH,ZH ; Z to EEPROM address port
  out EEARL,ZL
  ld rmp,Y+ ; Read first byte
  out EEDR,rmp ; Write to EEPROM data port
  cli ; Disable interrupts
  sbi EECR, EEMPE ; Enable EEPROM write
  sbi EECR, EEPE ; Start EEPROM write
  sei ; Re-enable interrupts
  sbi EECR,EERIE ; Enable EEP RDY interrupts
  ret
;
; Read EEPROM values to registers
ReadEep:
  ldi ZH,High(EepAdrs) ; Z to EEPROM address
  ldi ZL,Low(EepAdrs)
  ldi XH,High(cStepSetS) ; X to register address
  ldi XL,Low(cStepSetS)
  ldi rmp,cStepSetE-cStepSetS ; Number of bytes
  mov R0,rmp ; to R0
ReadEep1:
  sbic EECR, EEPE ; Wait until EEPROM ready
  rjmp ReadEep1 ; Not yet ready
ReadEep2:
  out EEARH,ZH ; Set Read address EEPROM
  out EEARL,ZL
  sbi EECR, EERE ; Read enable EEPROM
  in rmp, EEDR ; Read byte from EEPROM
  st X+,rmp ; And store in register
  adiw ZL,1 ; Next address
  dec R0 ; All bytes read?
  brne ReadEep2 ; No, continue
  mov rmp,rMark1 ; Check mark bytes
  eor rmp,rMark2 ; Exclusive-OR
  cpi rmp,0xFF ; Correct?
  breq ReadEep3 ; Yes, correct marks
  ; Incorrect marks, ignore EEPROM data
  ldi rmp,High(32768) ; Set default
  mov rOpenH,rmp ; To middle position
  mov rCloseH,rmp
  mov rCurrValH,rmp
  clr rOpenL
  clr rCloseL
  clr rCurrValL
ReadEep3:
  ret
;
; EEPROM content default
```

```
;
.eseg
.org 0x0010 ; EEPROM segment from 0x10
EepAdrs:
; Default values: Mark1 (0xAA), Close value, Open value, SET value,
;   Mark2 (0x55)
.db 0xAA ; Mark 1
.dw 32768 ; Close
.dw 32768 ; Open
.dw 32768 ; Set value
.db 0x55 ; Mark 2
;
; End of source code
;
```

Praise, error reports, scolding and spam please via the comment page to me. Spammers: please note that behind this is not a dump standard script but I personally select the entries to be published.