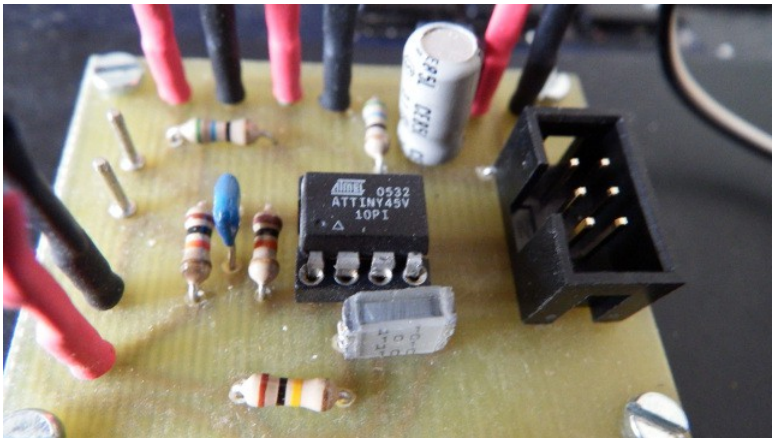
Anwendungen von
AVR-Single-Chip-Controller
AT90S, ATtiny, ATmega and
Atxmega

Audiosensor mit einem ATtiny45

Audiosensor mit einem ATtiny45

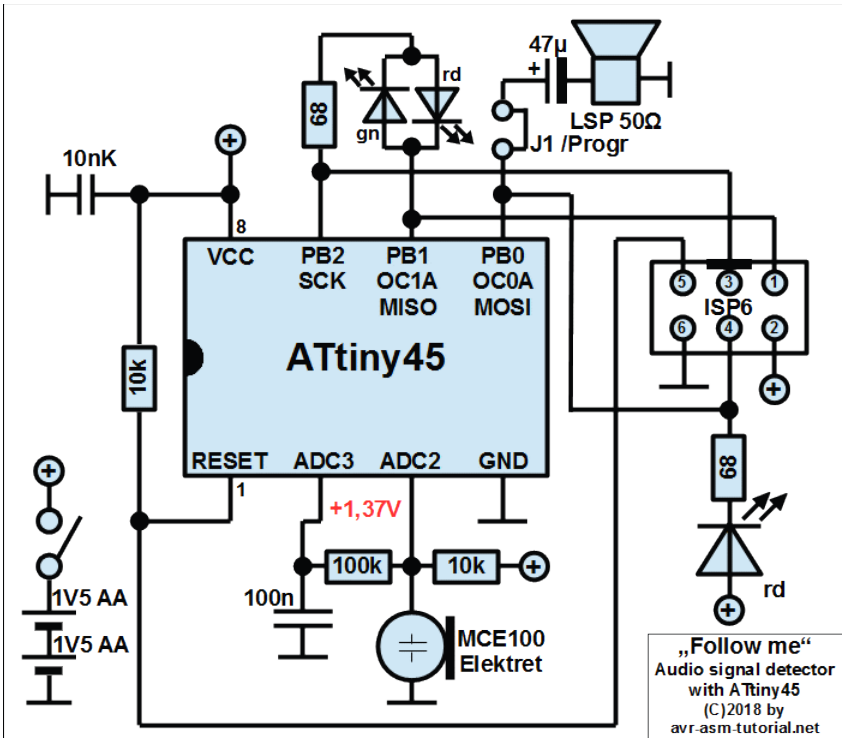


Dieser einfache Sensor reagiert auf steile Audiosignale und antwortet mit einem Signalton. Die Stärke des Audiosignals wird kontinuierlich auf einer rot-grünen Duo-Led dargestellt. Wenn Du schon einen Klatschschalter besitzt wirst Du erstaunt sein, mit wie wie wenig Elektronikkomponenten die Schaltung auskommt.

0 Überblick

1 Hardware

1.1 Schaltbild



Der Controller ATtiny45 erledigt die gesamte Arbeit und kontrolliert die wenigen externen Bauteile.

Die Stromversorgung erfolgt mit zwei AA- oder AAA-Batterien in Serie geschaltet. Der Strombedarf liegt normalerweise bei 1,5 mA, beim Alarm bis zu 15 mA.

Das Audiosignal wird mit einem Elektretmikrofon aufgenommen. Seine Betriebsspannung erhält das Mikrofon über den 10k-Widerstand, die resultierende

Spannung liegt bei 1,37 V und treibt den ADC2-Eingang des ATtiny45. Die Mikrofonspannung wird durch ein 100k/100nF-RC-Netzwerk gefiltert, das den Gleichspannungsanteil mittelt und als Vergleichsspannung am ADC3-Eingang anliegt. Die Differenz ADC2-ADC3 wird im Differenzverstärker des ATtiny45 um den Faktor 20 verstärkt und liefert ausreichend Pegel für die Messung am Signaleingang des AD-Wandlers.

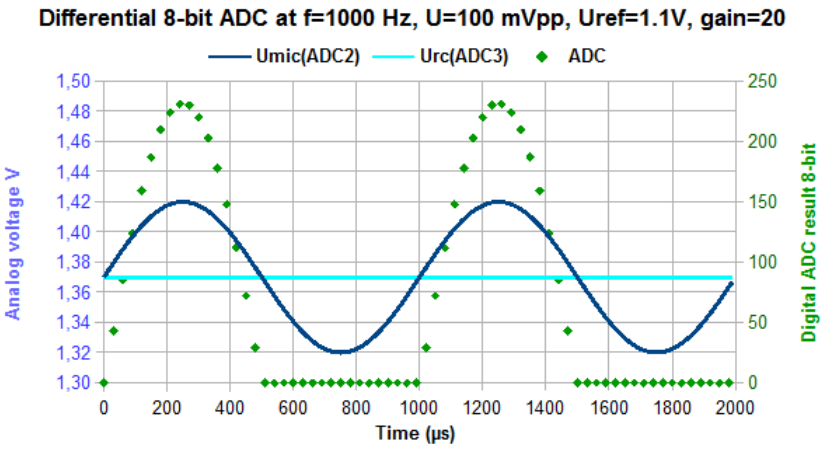
Die Duo-Led zeigt die Spitzenspannungen, gemittelt über 64 Einzelmessungen, in grün an. Je lauter das Signal wird, desto heller leuchtet die LED.

Überschreitet eine voreinstellbare Anzahl an Messungen die Triggergrenze (1,5 mal der Durchschnitt), wird der Alarm aktiviert. Die Duo-Led wird rot und die rote Einzel-Led wird aktiviert. Im Lautsprecher ertönt ein voreinstellbarer 880Hz-Ton. Der Alarm erlischt nach zwei Sekunden.

Die ISP6-Schnittstelle ermöglicht die Programmierung des ATiny45 in der fertigen Schaltung. Zum Programmieren wird der Jumper J1 ausgeschaltet, damit der MOSI-Pin ungestört arbeiten kann.

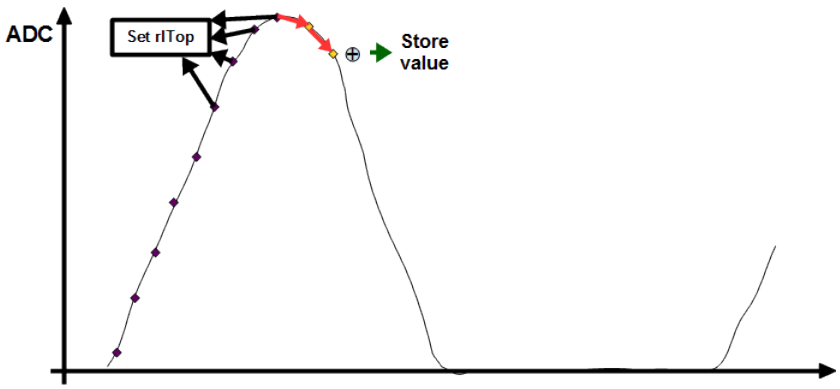
1.2 Wie es funktioniert

Der Controller misst kontinuierlich die Differenzspannung an den Eingangspins ADC2-ADC3, verstärkt diese um das 20-fache und vergleicht diese mit der internen Spannungsreferenz von 1,1 V. Das 8-Bit-Wandler-Ergebnis zwischen 0 und 255 deckt den Eingangspannungsbereich von 1,1 / 20 = 55 mV respektive 39 mV_{eff} ab. Nur die positiven Halbwellen werden gemessen, negative ergeben Null.



Der AD-Wandler arbeitet mit dem Controllertakt von 2 Mhz geteilt durch 2. Er benötigt 13 AD-Wandler-Takte. Die Interrupt-Service-Routine, die bei abgeschlossener Wandlung aufgerufen wird, benötigt zwischen 21 bis 38 Controllertakte bis der AD-Wandler neu gestartet wird. Die Wandlerfrequenz liegt folglich zwischen 26 und 43 kHz. Ein Ton von 1.000 Hz wird daher mindestens 25 Mal gemessen, wovon 12 bis 13 Messungen Null ergeben, der Rest die positive Halbwelle abbildet.

Um die Maximalamplitude zu erkennen wird das AD-Wandler-Resultat mit dem vorherigen verglichen. Steigt die Amplitude an, wird rTop auf das neue Ergebnis gesetzt. Sinkt das Wandler-Ergebnis und wird kleiner als rTop und auch kleiner als bei der letzten Messung wird der Spitzenwert im SRAM gespeichert, zusammen mit einem 16-Bit-Zeitstempel (jeweils 12 µs). Der Zeitstempel kann für die Beschränkung



auf bestimmte Frequenzen benutzt werden, wird aber in dieser Version nicht ausgewertet.

Sind 64 Spitzenwerte erkannt, werden diese gemittelt und auf der Duo-Led angezeigt (PWM-Zyklus des Timers TC1). Der Mittelwert wird um das 1,5-fache erhöht und als neuer Triggerwert gespeichert. Überschreiten eine voreingestellte Anzahl an Werten den Triggerwert wird die Alarmphase gestartet. Die LEDs werden umgeschaltet und der Timer TC0 erzeugt einen Ton von 880 Hz an seinem Ausgangspin OC1A.

1.3 Andere Einstellungen

Die Hard- und Software arbeitet auch mit einem ATtiny25 oder einem ATtiny85. Die Auswahl erfolgt zu Beginn des Quellcodes durch ändern von Konstanten. Wenn ein ATtiny25 gewählt wird, wird die Ausgabe der Zeitstempel abgeschaltet damit die 64 Messwerte in den verfügbaren SRAM-Speicher passt. Das Abschalten ändert nichts an der Funktionsweise.

Die Alarmschwelle, die Alarmdauer und der Alarmton können durch Ändern von Konstanten im Kopf der Software eingestellt werden.

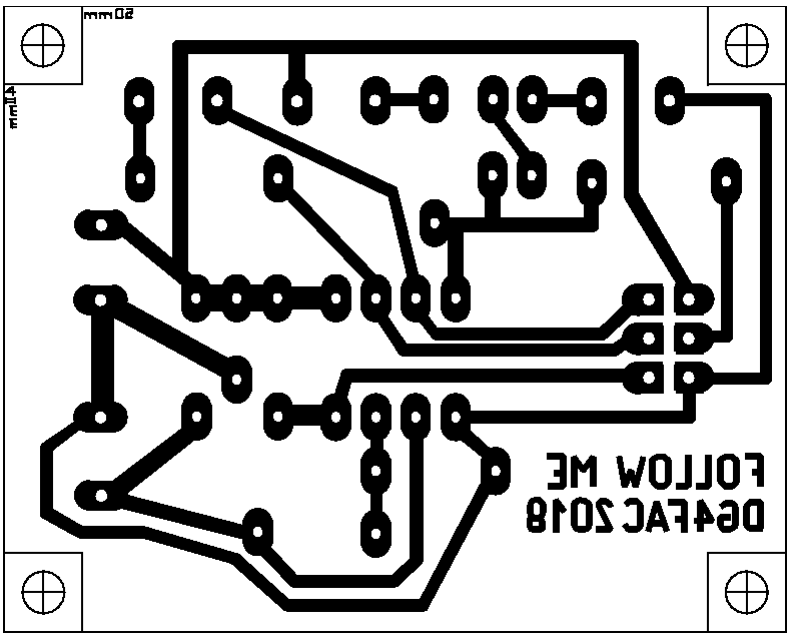
Alle angeschlossene Hardware kann separat getestet werden indem die entsprechenden "Debug switches" aktiviert werden. Die zwei LEDs, der Lautsprecher und das Mikrofon können getestet werden. In zwei weiteren Modi können AD-Wandler-Rohdaten oder erkannte Spitzenwerte in das EEPROM gespeichert werden. Von dort können sie über die ISP6-Schnittstelle ausgelesen werden.

Seitenanfang	1 Hardware	2 Aufbau	3 Betrieb	4 Software

2 Aufbau

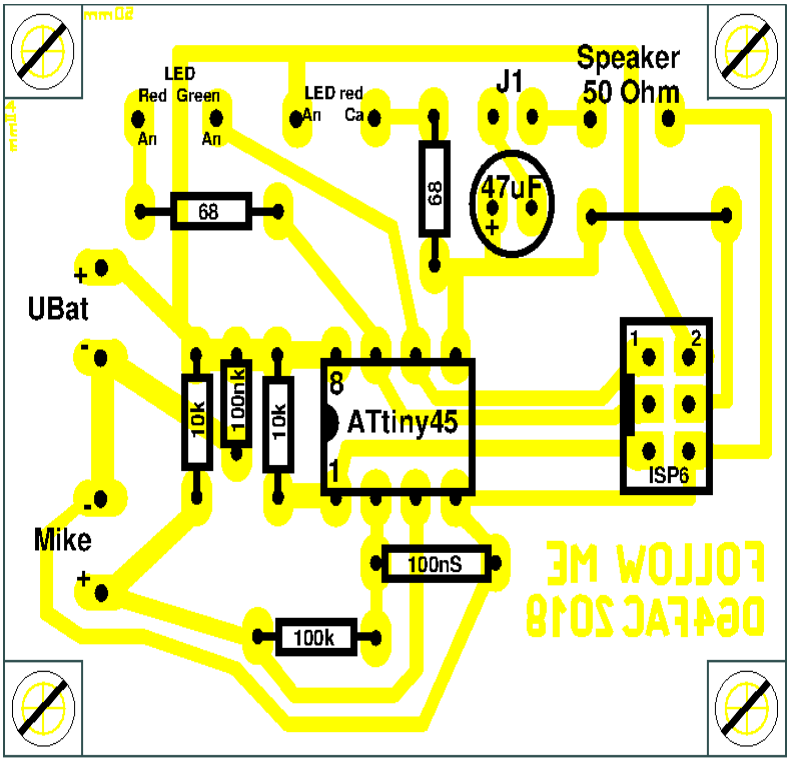
2.1 Gedruckte Schaltung

Die Platine mit 50-mal-40 mm nimmt alle Komponenten auf und besitzt vier Befestigungslöcher von 2,5 mm Durchmesser.



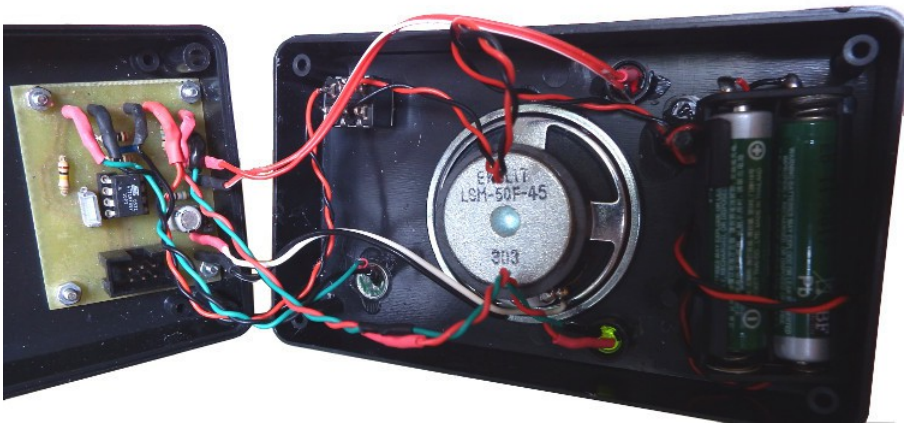
2.2 Komponentenmontage

Eine Brücke ist zu bestücken. Alle externen Komponenten sind mit 1 mm-Pins steckbar.



2.3 Einbau in eine Schachtel

Zusammen mit den beiden Batterien finden alle Komponenten in einer 120*70*40 mm Plastikschachtel Platz.



Seitenanfang	1 Hardware	2 Aufbau	3 Betrieb	4 Software

3 Betrieb

Der Betrieb ist einfach, weil der Hintergrund kontinuierlich gemessen und ausgewertet wird. Nur Signale, die den Hintergrund um das 1,5-fache übersteigen führen zum Alarm.

Da die Helligkeit der Duo-Led den Durchschnittswert fortlaufend anzeigt, kann durch langsam steigende Lautstärke über längeren Zeitraum die Empfindlichkeit trainiert werden. Das funktioniert aber nur bis das 1,5-fache nicht 255 erreicht.

Seitenanfang	1 Hardware	2 Aufbau	3 Betrieb	4 Software

4 Die Software

Die Software ist in AVR Assembler geschrieben und ausführlich kommentiert. Der Quellcode kann hier heruntergeladen oder hier im Browser gelesen werden. Der Quellcode verwendet .if-Direktiven und kann entweder mit dem gavrasm-Assembler oder mit ATMEL's Assembler 2 assembliert werden. Eine how-to-Seite für das Assemblieren mit gavrasm ist verfügbar für Linux hier und für Windows hier zu finden.

Änderungen von Konstanten können im Kopf der Software erfolgen wenn andere Eigenschaften gewünscht sind oder wenn die Software für Tests verwendet werden soll.

Ein besonderer Trick wurde im Quellcode verwendet. Da die AD-Wandler-Routine das Audiosignal in schneller Folge ausmessen muss, wird die Taktfrequenz des Controllers vom Default von 1 MHz auf 2 MHz umgestellt. Das erfolgt nicht durch Fuse-Änderung sondern durch Schreiben in das CLKPR-Portregister.

Die Quellcode-Datei enthält viele weitere Hinweise zur Funktionsweise.

Seitenanfang	1 Hardware	2 Aufbau	3 Betrieb	4 Software

Lob, Fehlerberichte, Nörgeln und Spam bitte in die Kommentarseite an mich. Spammer: bitte beachten, dass hinter diesem Link kein Standardscript steckt und ich persönlich die Einträge auswähle. Daher sind auch wiederholte Versuche, Einträge zu platzieren zum Scheitern verurteilt und Werbung für Online-Apotheken wird hier nicht erscheinen.

Anwendungen der
AVR Single chip controller
AT90S, ATtiny, ATmega and
Atxmega

Audiosensor mit einem ATtiny45 Assembler-Quellcode

Assembler-Quellcode für einen Audiosensor mit dem ATtiny45

Der Quellcode ist hier im asm-Format.

```
;
; ************************************
; * Folge mir Audio Sensor mit       *
; * ATtiny25/45/85, Version 1 April 2018 *
; * (C)2018 by avr-asm-tutorial.net  *
; ************************************
;
; Arbeitet wahlweise mit einem ATtiny45 oder
;   einem ATtiny85 ohne Aenderungen,
;   kann an einen ATtiny25 angepasst werden
;   durch Setzen des entsprechenden Schal-
;   ters indem das Speichern der 16-Bit-Zeit-
;   stempel unterbunden wird.
;
; ********************************
;     T Y P E N A U S W A H L
; ********************************
;
; Waehlt den Zieltyp aus, entweder den
;   ATtiny 45 oder den ATtiny85, bei Auswahl
;   des ATtiny25 werden Zeitstempel abge-
;   schaltet
.equ dev_sel = 45 ; kann 25, 45 oder 85 sein
;
;
.nolist
.if dev_sel == 25
  .include "tn25def.inc"
  .set tstmp = 0
  .endif
.if dev_sel == 45
  .include "tn45def.inc"
  .set tstmp = 1
  .endif
.if dev_sel == 85
  .include "tn85def.inc"
  .set tstmp = 1
  .endif
.list
;
; Erzwingen der Zeitstempel-Einstellung
;   ueberschreibt die Einstellung bei der
```

```
;   Typauswahl
.set tstamp = 1 ; 0 = keine Zeitstempel
;
.if (tstamp == 1)&&(dev_sel == 25)
  .error "Nicht genug Platz im EEPROM!"
  .endif
;
; *********************************
;    D E B U G - S C H A L T E R
; *********************************
; Alle Debug-Schalter = 0 in der Endversion
;
; Testen der angeschlossenen Hardware
;
; Schalte beide roten LEDs ein und
;   unendliche Schleife
.equ debug_ledsRed = 0 ; 0 oder 1
;
; Schalte gruene LED an und unendliche
;   Schleife
.equ debug_ledGreen = 0 ; 0 oder 1
;
; Schalte den Lautsprecher an und
;   unendliche Schleife
.equ debug_speaker = 0 ; 0 oder 1
;
; Teste das Mikrofon und die ADC-Funktion
;
; Sammle AD-Wandler-Datensets, mittele und
;   verwende den Durchschnitt als Pulsweiten-
;   modulation fuer beide LEDs
.equ debug_mikeadc = 0
;
; Sammle die Maxima bei 64 AD-Wandler-Daten-
;   sets, schalte die Interrupts aus, kopie-
;   re die Daten in das EEPROM und unendliche
;   Schleife
.equ debug_eep = 0
;
; Sammle 64 AD-Wandler-Datensets-Rohdaten,
;   schalte die Interrupts aus, kopiere die Daten
;   ins EEPROM, schalte die rote LED an und
;   unendliche Schleife
.equ debug_eep_raw = 0
;
; *********************************
;        H A R D W A R E
; *********************************
;
; AVR-Typen: ATtiny25/45/85
;   Packung: 8-pin-PDIP_SOIC
;
;                 ________
;           1 /         |8
;   RESET o--|RES   VCC|--o VCC
;           2|         |7
; Udiff - o--|ADC3  PB2|--o R/G-Led rote anode
;           3|         |6
; UMike + o--|ADC2  PB1|--o R/G-Led gruene Anode
;           4|         |5
;     GND o--|GND  OC0A|--o Lautsprecher, R-Led Kathode
;            |_________|
;
;
```

```
; *********************************
;  P O R T S   U N D   P I N S
; *********************************
;
; LED-Ports und -Pins
.equ pLedO = PORTB ; Led-Port Output
.equ bLedRCO = PORTB0 ; Rote Led Kathodenpin
.equ bLedRGRAO = PORTB2 ; Rot/gruen Led rote Anode Pin
.equ bLedRGRCO = PORTB1 ; Rot/gruen Led rote Kathode Pin
.equ bLedRGGAO = PORTB1 ; Rot/gruen LED gruene Anode Pin
.equ bLedRGGCO = PORTB2 ; Rot/gruen LED gruene Kathode Pin
.equ pLedD = DDRB ; Led-Port Richtung
.equ bLedRCD = DDB0 ; Rote Led Kathode Richtung Pin
.equ bLedRGRAD = DDB2 ; Rot/gruene Led rote Anode Richtung Pin
.equ bLedRGGAD = DDB1 ; Rot/gruene Led gruene Anode Richtung Pin
; Lautsprecher-Port und -Pins
.equ pSpkO = PORTB ; Lautsprecher Ausgangsport
.equ bSpkO = PORTB0 ; Lautsprecher Ausgangspin
.equ pSpkD = DDRB ; Lautsprecher Richtungsport
.equ bSpkD = DDB0 ; Lautsprecher Richtungspin
;
; *********************************
; E I N S T E L L K O N S T A N T E N
; *********************************
; Taktfrequenz Prozessor, wird zu Beginn durch
;   Schreiben in den CLKPR-Port eingestellt
.equ clock=2000000 ; Taktfrequenz
;
; Anzahl der Triggerwertueberschreitungen die zu
;   Alarm fuehren, zwischen 1 und 100
.equ cTriggr = 50 ; Triggerpegelueberschreitungen in %
;
; Alarmdauer in Zehntel-Sekunden
;   Kann zwischen 1 und 41 eingestellt werden
.equ cAlrm = 10 ; Vielfache von 0,1 Sekunden
;
; Alarmtonhoehe, 62 Hz .. 15 kHz
.equ cTone = 880 ; Ton in Hz
;
; *********************************
;    F E S T K O N S T A N T E N
; *********************************
;
; Alarmzaehler fuer TC1
.equ cAlrmC = clock / 128 /256 * cAlrm / 10
;
; Alarmtonumrechnung
.equ cToneC = clock / 64 / cTone ; CTC-A TC0
;
; Triggerpegelschwelle
.equ cTrigger = (64*cTriggr+50)/100 ; Triggerschwelle
;
;
; Pruefe Einstellungen
.if cAlrmC > 255
  .error "Alarmdauer-Konstante zu gross!"
  .endif
.if (cToneC<1)||(cToneC>255)
  .error "Alarmfrequenz ausserhalb zulaessiger Bereich!"
  .endif
.if (cTrigger<1) || (cTrigger>64)
  .error "Triggerschwelle ausserhalb zulaessiger Bereich!"
  .endif
;
```

```
; *********************************
;         T I M I N G
; *********************************
;
; Timing des AD-Wandlers:
;   Der AD-Wandler wird mit einem Vorteiler
;   von 2 getaktet. Er wandelt das 20-fach
;   verstaerkte Differenzsignal an den Ein-
;   gaengen (ADC2 - ADC3) aus.
;   Takt Controller              = 2 MHz
;   Vorteiler ADC                = 2
;   Wandlungstakt                = 1 MHz
;   Wandlungsschritte            = 13
;   Wandlungszeit                = 13 us
;   Zeit von Wandlungsstart bis
;     ADC-Interrupt              = 13 us
;   Controllertakte hierfuer     = 26
;   Takte bis zum Neustart: 21/23/25/26/35/38
;     Minimum bis zum Neustart   = 10,5 us
;     Maximum bis zum Neustart   = 19 us
;   Wandlungszeit Minimum        = 23,5 us
;   Wandlungszeit Maximum        = 32 us
;   Wandlungszeit mit TC0-Int    = 39 us
;   Minimum Wandlungsfrequenz    = 25,64 kHz
;   Maximum Wandlungsfrequenz    = 42,55 kHz
;
; 8-Bit Timer TC0 als Zeitbasis und als Tongenerator
;   Als Zeitbasis waehrend Messphase:
;     Modus: 8-Bit Normalbetrieb
;     Kontrollertakt             = 2 MHz
;     Prescaler                  = 256
;     TC0-Tick                   = 128 us
;     Ueberlaufzeit              = 32,77 ms
;     Ueberlauffrequenz          = 30,5 Hz
;     Ueberlauf MSB in TC0H      = 8,38 s
;
;   Als Tongenerator in der Alarmphase:
;     Modus: 8-Bit CTC mit OCR0A und OC0A-Togglen
;     Controller Takt            = 2 MHz
;     Precaler                   = 64
;     Tonfrequenz, OCR0A=0x01    = 15,6 kHz
;     Tonfrequenz, OCR0A=0xFF    = 61 Hz
;
; 8-Bit Timer TC1 als PWM fuer Pegelanzeige
;   zeigt auf der rot/gruenen LED den Pegel an
;   und stellt Dauer in der Alarmphase fest
;   Als LED-PWM (Normalbetrieb):
;   Controllertakt               =   2 MHz
;   Prescaler                    = 128
;   Zeit pro Timertick           =  64 us
;   Zeit fuer 256 Timerticks     =  16.384 ms
;   Zeit bis Ueberlauf           =  16,4 ms
;   PWM-Frequenz rot/gruene LED  =  61 Hz
;
;   Als Alarmzeit Timer (bei Alarm):
;   Controllertakt               =   2 MHz
;   Prescaler                    = 128
;   Zeit pro Timertick           =  64 us
;   Zeit fuer 256 Timerticks     =  16,384 ms
;   Zaehlwert fuer 1 Sekunde     =  61
;   Zaehlwert fuer 4 Sekunden    = 244
;
; *********************************
;       R E G I S T E R
```

```
; *********************************
;
; Verwendet: R1:R0 fuer Addierer Pegelwerte
; Verwendet: R2 fuer Maximalwert
; frei: R3 bis R8
.def rMaxA = R9 ; Maximum Analogwert
.def rLast = R10 ; Letzter gemessener Wert
.def rTrigger = R11 ; Triggerwert
.def rTc0L = R12 ; TC0-Zeitstempel, LSB
.def rTc0H = R13 ; TC0-Zeitstempel, MSB
.def rTop = R14 ; Spitzenwert innerhalb von Ints
.def rSreg = R15 ; Sichern/Wiederherstellen Statusregister
.def rmp = R16 ; Vielzweckregister
.def rimp = R17 ; Vielzweckregister innerhalb Ints
.def rFlag = R18 ; Flaggenregister
  .equ bOvf = 0 ; Ueberlauf ADC-Wertepuffer, nicht verwendet
  .equ bDwn = 1 ; Signalwert am ADC abwaerts
  .equ bDsc = 2 ; Pufferende, werte Datensaetze aus
  .equ bAlrm = 3 ; Alarmphase aktiv
.def rOvfC = R19 ; Ueberlaufzaehler, nicht verwendet
.def rDscC = R20 ; Datensatzzaehler
.def rAlrmC = R21 ; Alarmzaehler
; frei: R22 bis R27
; Verwendet: R27:R26 = X fuer EEPROM-Schreiben
; Verwendet: R29:R28 = Y als Pufferzeiger
; Verwendet: R31:R30 = Z fuer diverse Zwecke
;
; *********************************
;           S R A M
; *********************************
;
.dseg
.org SRAM_START
.equ nDataSets = 64 ; Datenset=zwei Byte Zeit, ein Byte AD-Wert
sBuffer:
.if tstmp == 1
  .byte 3*nDataSets ; Reserviere 192 Bytes fuer Datensaetze
  .else
  .byte nDataSets ; Reserviere 64 Bytes fuer Daten
  .endif
sBufferDataEnd:
.byte 1 ; Durschschnitt
.byte 1 ; Maximalwert
.byte 1 ; Triggerwert
sBufferEnd:
;
; *********************************
;            C O D E
; *********************************
;
.cseg
.org 000000
;
; ************************************
; R E S E T  &  I N T - V E K T O R E N
; ************************************
  rjmp Main ; ResetveKtor
  reti ; INT0
  reti ; PCI0
  reti ; OC1A
  rjmp Ovf1Int ; OVF1
  rjmp Ovf0Int ; OVF0
  reti ; ERDY
  reti ; ACI
```

```
  rjmp AdcRdy ; ADCC
  reti ; OC1B
  reti ; OC0A
  reti ; OC0B
  reti ; WDT
  reti ; USI_START
  reti ; USI_OVF
;
; *********************************
;  I N T - S E R V I C E   R O U T .
; *********************************
;
; Ueberlauf-Interrupt TC0: 8-Bit-Zaehler
;   Messphase: Normaler Zaehler, liefert Zeitstempel MSB
;   Alarmphase: Interrupt aus, CTC fuer Lautsprecherausgang
Ovf0Int: ; 7 Takte fuer Interrupt&Vektorsprung
  in rSreg,SREG ; Sichere SREG, +1=8
  inc rTc0H ; Erhoehe MSB, #1=9
  out SREG,rSreg ; Stelle SREG wieder her, +1=10
  reti ; +4=14
;
; Ueberlauf-Interrupt TC1
;   Messphase: 8-Bit PWM, Interrupt aus
;   Alarmphase: Normalbetrieb, Alarmdauer zaehlen
Ovf1Int:
  in rSreg,SREG ; Sichern SREG
  tst rAlrmC ; Alarmzeit aktiv?
  breq Ovf1Int1 ; Alarmzeit aus
  dec rAlrmC ; Abwaerts zaehlen
Ovf1Int1:
  out SREG,rSreg ; Stelle SREG wieder her
  reti
;
; ADC-Wandlung komplett Interrupt
;   Taktzyklen: +n=nc: ISR-Ausfuehrungstakte
;   #n: Taktzyklen vom letzten ADC-Start bis Neustart
AdcRdy: ; 7 Takte fuer Interrupt, #12
  in rTc0L,TCNT0 ; Lese LSB Zeit, +1=8, #13
  in rSreg,SREG ; Sichern SREG, +1=9, #14
  in rimp,ADCH ; Lese MSB ADC, +1=10, #15
.if debug_eep_raw == 0
  tst rimp ; Null? +1=11, #16
  breq AdcZero ; Ja, loesche Abwaertsflagge +1/2=12/13, #17/18
  cp rimp,rLast ; Vergleiche mit letztem Wert, +1=13, #18
  mov rLast,rimp ; Letzter=Aktueller Wert, +1=14, #19
  brcc AdcGreater ; Wert steigt, +1/2=15/16, #20/21
  cp rimp,rTop ; Kleiner als ITop?, +1=16, #21
  brcc AdcGreater ; Steigt wieder, +1/2=17/18, #22/23
  sbrc rFlag,bDwn ; Ueberspringe wenn bDwn-Flagge Null, +1/2=18/19, #23/24
  rjmp AdcRet ; +2=20, #25
  .endif
.if tstmp == 1 ; Speichere Zeitstempel in Puffer
  st Y+,rTc0L ; LSB Zeit in Puffer, +2=21, #26
  st Y+,rTc0H ; MSB Zeit in Puffer, +2=23, #28
  .endif
.if debug_eep_raw == 1
  st Y+,rimp ; Speichere Messwert in Puffer, +2=25, #30
  .else
  st Y+,rTop ; Speichere Maximalwert in Puffer, +2=25, #30
  .endif
  sbr rFlag,1<<bDwn ; Setze Datenflagge, +1=26, #31
  dec rDscC ; Vermindere Datensatzzaehler, +1=27, #32
  brne AdcRet ; Nicht am Pufferende, +1/2=28/29, #33/34
  ldi YH,High(sBuffer) ; Y-Zeiger neu beginnen, +1=29, #34
```

```
  ldi YL,Low(sBuffer) ; +1=30, #35
  ldi rDscC,nDataSets ; Datensatzzaehler neu beginnen, +1=31, #36
  sbr rFlag,1<<bDsc ; Datenpuffer voll Flagge, +1=32, #37
AdcRet: ; 19/20/21/30/32 Takte, #23/24/25/34/37
  ; Neustart ADC
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS0); +1=20/21/22/31/33,
#25/26/27/35/38
  out ADCSRA,rimp ; Start naechste Wandlung, +1=22/32/34, #0
  out SREG,rSreg ; Stelle SREG wieder her, +1=23/33/35, #1
  reti ; +4=27/37/39, #5
AdcGreater: ; 16/18 Takte, #21/23
  cpi rimp,0x10 ; Ignore Kleinstwerte, +1=17/19, #22/24
  brcs AdcClrTop ; Loesche rTop, +1/2=18/19/20/21, #23/24/25/26
  mov rTop,rimp ; Setze neuen Top-Wert, 1=19/21, #24/26
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS0); +1=20/22, #25/27
  out ADCSRA,rimp ; Starte naechste Wandlung, +1=19/21, #0
  out SREG,rSreg ; Stelle SREG wieder her, +1=20/22, #1
  reti ; +4=24/26, #5
AdcZero: ; ADC ist Null, 13 cycles, #18
  cbr rFlag,1<<bDwn ; Loesche bDwn-Flagge, +1=14, #19
AdcClrTop:
  clr rTop ; Loesche rTop-Wert, +1=15, #20
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS0); +1=16, #21
  out ADCSRA,rimp ; Starte naechste Wandlung, +1=17, #0
  out SREG,rSreg ; Stelle SREG wieder her, +1=18, #1
  reti ; +4=22, #5
;
; ************************************
;  H A U P T P R O G R A M M,  I N I T
; ************************************
;
Main:
.if dev_sel != 25 ; Nicht bei ATtiny25
  ldi rmp,High(RAMEND) ; Init MSB Stapel
  out SPH,rmp ; Init MSB Stapelzeiger
  .endif
  ldi rmp,Low(RAMEND) ; Init LSB Stapel
  out SPL,rmp ; Init LSB Stapelzeiger
  ; Setze Controllertakt auf 2 MHz
  ldi rmp,1<<CLKPCE ; Taktvorteiler-Enable
  out CLKPR,rmp
  ldi rmp,1<<CLKPS1 ; Taktvorteiler auf 4
  out CLKPR,rmp
  ; Init rote LED Portpin
  sbi pLedD,bLedRCD ; Rote Led Kathode als Ausgang
  sbi pLedO,bLedRCO ; Rote Led Kathode high
  ; Init rot/gruene Duo-LED
  sbi pLedD,bLedRGRAD ; Rot/gruen rote Anode Ausgang
  sbi pLedD,bLedRGGAD ; Rot/gruen gruene Anode Ausgang
  cbi pLedO,bLedRGRAO ; Rot/gruen rote Anode low
  cbi pLedO,bLedRGGAO ; Rot/gruen gruene Anode low
.if debug_ledsRed == 1 ; Test der beiden roten LEDs
  cbi pLedO,bLedRCO ; Rote Led-Kathode an
  sbi pLedO,bLedRGRAO ; Rote Anode high
  cbi pLedO,bLedRGGAO ; Gruene Anode low
  test_leds_Red_Loop:
    rjmp test_leds_Red_Loop ; Unendliche Schleife
  .endif
.if debug_ledGreen == 1 ; Teste die gruene LED
  sbi pLedO,bLedRGGAO ; Gruene Anode high
  cbi pLedO,bLedRGRAO ; Rote Anode low
  test_led_green_loop:
    rjmp test_led_green_loop ; Unendliche Schleife
  .endif
```

```
  ; Init Lautsprecher Portpin
  sbi pSpkD,bSpkD ; Lautsprecher Portpin Ausgang
  sbi pSpkO,bSpkO ; Lautsprecher Portpin high (rote LED aus)
.if debug_speaker == 1 ; Lautsprecher testen
  ldi rmp,clock / 8 / 2000 ; Vergleicher A auf 1000 Hz
  out OCR0A,rmp
  ldi rmp,(1<<COM0A0)|(1<<WGM01) ; CTC bei OCR0A, Toggle OC0A
  out TCCR0A,rmp
  ldi rmp,(1<<CS01) ; Prescaler = 8
  out TCCR0B,rmp
  test_speaker_loop:
    rjmp test_speaker_loop ; Unendliche Schleife
  .endif
  ; Abschalten der Portpin-Treiber ADC2 und ADC3
  ldi rmp,(1<<ADC2D)|(1<<ADC3D)
  out DIDR0,rmp
  ; Init des ADC und des Timers zum Start im Messmodus
  rcall InitAdcTimer
  ; Interrupts und sleep mode idle
  ldi rmp,1<<SE ; Sleep mode idle
  out MCUCR,rmp
  sei ; Enable Interrupts
;
; *********************************
;   P R O G R A M M S C H L E I F E
; *********************************
;
Loop:
  sleep ; Schlafen legen
  nop ; Dummy fuer Wecken
  sbrc rFlag,bAlrm ; Alarm aktiv?
  rcall Alarm ; Ja, behandle Alarm
  sbrc rFlag,bDsc ; Datensaetze komplett Flagge?
  rcall Update ; Ja, behandle Update
  rjmp loop
;
; Aktiver Alarm
Alarm:
  tst rAlrmC ; Alarmzaehler auf Null?
  brne AlarmRet ; Nein, warte weiter
  ldi rmp,0 ; Alarm zu Ende, loesche Puffer
  ldi ZH,High(sBuffer) ; Z auf Puffer
  ldi ZL,Low(sBuffer)
Alarm1:
  st Z+,rmp ; Nullen schreiben
  cpi ZH,High(sBufferDataEnd) ; Ende Puffer MSB?
  brcs Alarm1 ; Nein, weiter
  cpi ZL,Low(sBufferDataEnd) ; Ende Puffer LSB?
  brcs Alarm1 ; Nein, weiter
  ldi rmp,0xFF ; Alarm naechste Runde aus
  mov rTrigger,rmp ; in Triggerwert
  cbr rFlag,1<<bAlrm ; Loesche Alarmflagge
  rcall InitAdcTimer ; Neustart Timer und ADC
AlarmRet:
  ret
;
; Behandle Datentabelle
Update:
  cbr rFlag,1<<bDsc ; Loesche Puffer-Komplett-Flagge
.if (debug_eep == 1)||(debug_eep_raw == 1) ; EEPROM-Schalter
  clr rmp ; ADC, Timer und Interrupts abschalten
  out ADCSRA,rmp
  out TCCR0A,rmp
  out TCCR0B,rmp
```

```
  out TCCR1,rmp
  out GTCCR,rmp
  out TIMSK,rmp ; Timer-Interrupts aus
  .endif
  ; Berechne Durchschnitt, Maximum und Triggerueber-
  ;   schreitungen im SRAM-Puffer
.if tstmp == 1 ; Mit Zeitstempeln
  ldi XH,High(sBuffer+2) ; Zeige auf ersten Wert mit Zeitstempel
  ldi XL,Low(sBuffer+2)
  .else
  ldi XH,High(sBuffer) ; Zeige auf ersten Wert ohne Zeitstempel
  ldi XL,Low(sBuffer)
  .endif
  clr ZH ; Loesche die Summe
  clr ZL
  clr rMaxA ; Loesche Maximalwert
  clr R1 ; Loesche MSB Addierer
  clr R2 ; Anzahl Ueberschreitungen
  ldi rmp,nDataSets ; Anzahl Datensaetze
UpDate1:
  ld R0,X+ ; Lese Datenbyte aus Puffer
  cp R0,rMaxA ; Vergleiche mit Maximum
  brcs UpDate2 ; Kleiner
  mov rMaxA,R0 ; Speichere neues Maximum
Update2:
  cp R0,rTrigger ; Vergleiche mit Triggerwert
  brcs UpDate3 ; Kleiner
  inc R2 ; Zaehle Ueberschreitung
UpDate3:
  add ZL,R0 ; Addiere LSB zu Summe
  adc ZH,R1 ; Addiere MSB wenn Carry
.if tstamp == 1
  adiw XL,2 ; Zeige auf naechstes Datenbyte
  .endif
  dec rmp ; Anzahl Datensaetze abwaerts
  brne Update1 ; Nicht fertig, weiter
  lsl ZL ; Multipliziere Summe mit zwei
  rol ZH
  lsl ZL ; Multipliziere Summe mit vier
  rol ZH
  mov rTrigger,ZH ; Kopiere das MSB
  out OCR1A,ZH ; Schreibe in TC1 Vergleichswert A
  mov ZL,ZH ; Kopiere MSB
  lsr ZL ; Dividiere durch zwei
  add rTrigger,ZL ; Addiere zum MSB
  brcc UpDate4 ; Kein Carry
  ldi ZL,0xFF ; Setze Triggerwert auf Maximum
  mov rTrigger,ZL ; Und in rTrigger
UpDate4:
.if (debug_eep==0)&&(debug_eep_raw==0) ; Debug-Schalter sind aus
  mov rmp,rTrigger ; Kopiere Triggerwert
  cpi rmp,0xFF ; Am Maximum?
  breq UpDate5 ; Alarm aus
  ldi rmp,cTrigger ; Anzahl Ueberschreitungen fuer Alarm
  cp R2,rmp ; Vergleiche Ueberschreitungen
  brcs UpDate5 ; Kleiner, kein Alarm
  ; Alarm-Bedingung erreicht, starte Alarmphase
  cbi ADCSRA,ADIE ; ADC-Interrupts aus
  ldi rmp,cToneC ; Alarmton-Frequenz
  out OCR0A,rmp ; Auf Vergleicher TC0
  ldi rmp,(1<<COM0A0)|(1<<WGM01) ; CTC, toggle OC0A
  out TCCR0A,rmp ; An TC0 Control A
  ldi rmp,(1<<CS01)|(1<<CS00) ; Prescaler =64
  out TCCR0B,rmp ; An TC0 Control B
```

```
  ldi rAlrmC,cAlrmC ; Setze Alarmdauer-Wert
  ldi rmp,1<<TOIE1 ; Nur TC1-Interrupts
  out TIMSK,rmp ; In Timer-Int-Maske
  sbi pLedO,bLedRGRAO ; Setze RG rote Anode high
  cbi pLedO,bLedRGGAO ; RG gruene Anode low
  sbr rFlag,1<<bAlrm ; Alarmflagge
Update5:
  ret ; Ende des Puffer-Updates
  .else
; Schreibe den SRAM-Inhalt ins EEPROM
Write2Eep:
  st X,rTrigger ; Haenge Triggerwert an
.if tstmp == 0
  adiw XL,2 ; Ohne Zeitstempel zwei Byte vorwaerts
  .endif
  st -X,rMaxA ; Speichere Maximum
  st -X,ZH ; Speichere Durchschnitt
  clr rmp ; Abschalten ADC und Interrupts
  out ADCSRA,rmp
  out TIMSK,rmp ; Abschalten Timer-Interrupts
  cli ; Keine Interrupts mehr
EepWait:
  sbic EECR,EEPE ; Warte bis EEPROM bereit
  rjmp EepWait
  ldi rmp,0 ; Setze EEPROM Atomic loeschen/schreiben
  out EECR,rmp
  ldi XH,0 ; Zeiger in EEPROM
  ldi XL,0
  ldi ZH,HIGH(sBuffer) ; Z auf Puffer
  ldi ZL,Low(sBuffer)
EepWr:
  sbic EECR,EEPE ; Warte bis EEPROM bereit
  rjmp EepWr
.if dev_sel != 25 ; Nur bei ATtiny45/85
  out EEARH,XH ; Setze EEPROM Adresse MSB
  .endif
  out EEARL,XL ; Setze LSB Adresse
  ld rmp,Z+ ; Lese Datenbyte aus SRAM
  out EEDR,rmp ; Setze EEPROM-Datenport
  sbi EECR,EEMPE ; Program Enable
  sbi EECR,EEPE ; Program EEPROM
  adiw XL,1 ; Naechste Adresse
  cpi ZH,High(sBufferEnd) ; MSB Pufferende?
  brne EepWr ; Nein, weiter
  cpi ZL,Low(sBufferEnd) ; LSB Pufferende?
  brne EepWr ; Nein, weiter
EepLedOn:
  cbi pLedO,bLedRCO ; Rote Led an
EepLoop:
  rjmp EepLoop ; Unendliche Schleife
  .endif
;
; Init des ADC und der Timer fuer Messbetrieb
;   beim Beginn und im Anschluss an Alarmphasen
InitAdcTimer:
  ; Init ADC
  ldi YH,High(sBuffer) ; Y-Zeiger auf SRAM-Puffer-Start
  ldi YL,Low(sBuffer)
  ldi rDscC,nDataSets ; Anzahl Daten setzen
  ; ADC mit interner Reference 1.1V, ADLAR, Differential ADC2-ADC3, gain=20
  ldi rmp,(1<<REFS1)|(1<<ADLAR)|(1<<MUX2)|(1<<MUX1)|(1<<MUX0)
  out ADMUX,rmp ; ADC-Multiplexer
  ; Enable, Start conversion, int enable, clock/2
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS0)
```

```
  out ADCSRA,rmp ; ADC starten
  ; Starte TC0 als 8-Bit-Zaehler als Zeitbasis mit Prescaler 256
  ldi rmp,(1<<COM0A1)|(1<<COM0A0); 8-Bit-Zaehler, Compare setzt OC0A, Mode
normal
  out TCCR0A,rmp
  ldi rmp,1<<CS02 ; Prescaler = 256
  out TCCR0B,rmp
  ; Starte TC1 als invertierte PWM mit prescaler 128
  ldi rmp,0x00 ; Compare-Match-Wert kurz
  out OCR1A,rmp
  ldi rmp,0xFF ; Zaehlen bis 256
  out OCR1C,rmp
  ldi rmp,(1<<COM1A1)|(1<<CS13)|(1<<PWM1A) ; PWM-Signal an OC1A, prescaler 128
  out TCCR1,rmp
  clr rmp ; Kein PWM-Signal auf OC1B
  out GTCCR,rmp
  ldi rmp,1<<TOIE0 ; Nur Ueberlauf-Interrupt TC0
  out TIMSK,rmp
  cbi pLedO,bLedRGRAO ; Rote Anode low
  sbi pLedO,bLedRGGAO ; Gruene Anode high
  ret
;
; Copyright
.db "(C)2018 by Gerhard Schmidt  "
.db "C(2)10 8ybG reahdrS hcimtd  "
;
; Ende Quellcode
;
```

Lob, Fehlerberichte, Nörgeln und Spam bitte in die Kommentarseite an mich. Spammer: bitte beachten, dass hinter diesem Link kein Standardscript steckt und ich persönlich die Einträge auswähle. Daher sind auch wiederholte Versuche, Einträge zu platzieren zum Scheitern verurteilt und Werbung für Online-Apotheken wird hier nicht erscheinen.