Kapazitätsmessung im Rampenverfahren

- Messung der Wasserstoff-Konzentration
- Bei Überschreiten einer kritischen Konzentration wird Alarm ausgelöst
- Eine blaue LED geht an, siehe Bild
- Der Alarmwert wird mit einem Mikrotaster gesetzt und im EEPROM des µC gespeichert

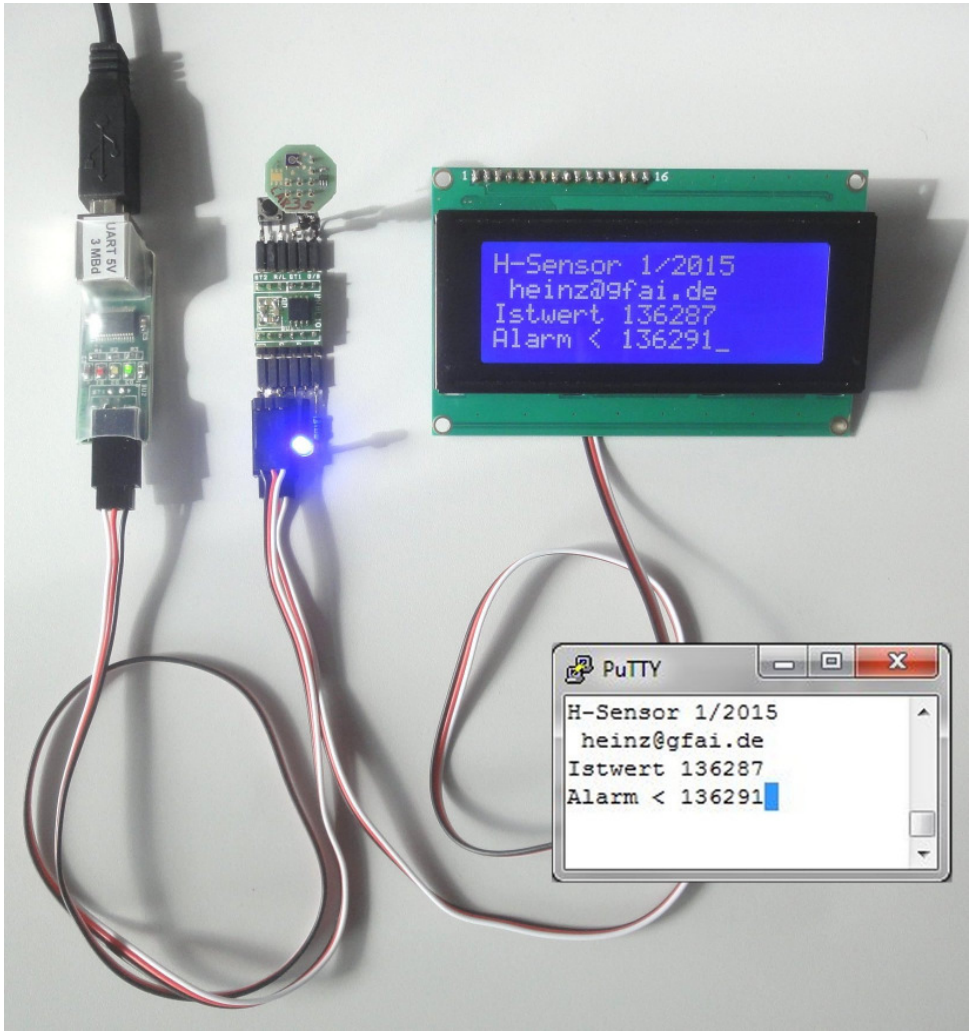
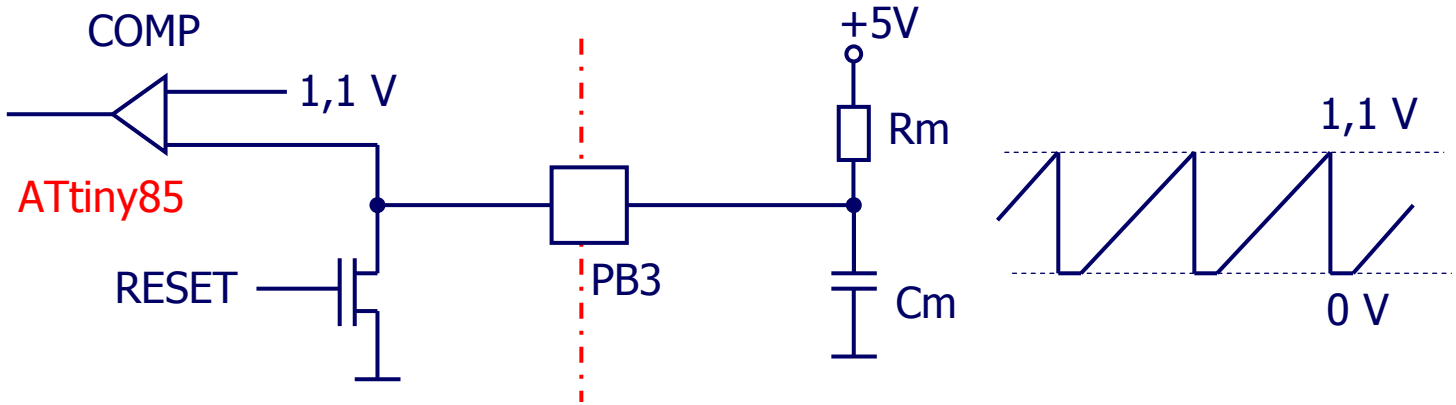


Bild: Testaufbau v39, vlnr: USB-SWART Schnittstelle, Testbord BRBL10, serielles Display SERLCDv1

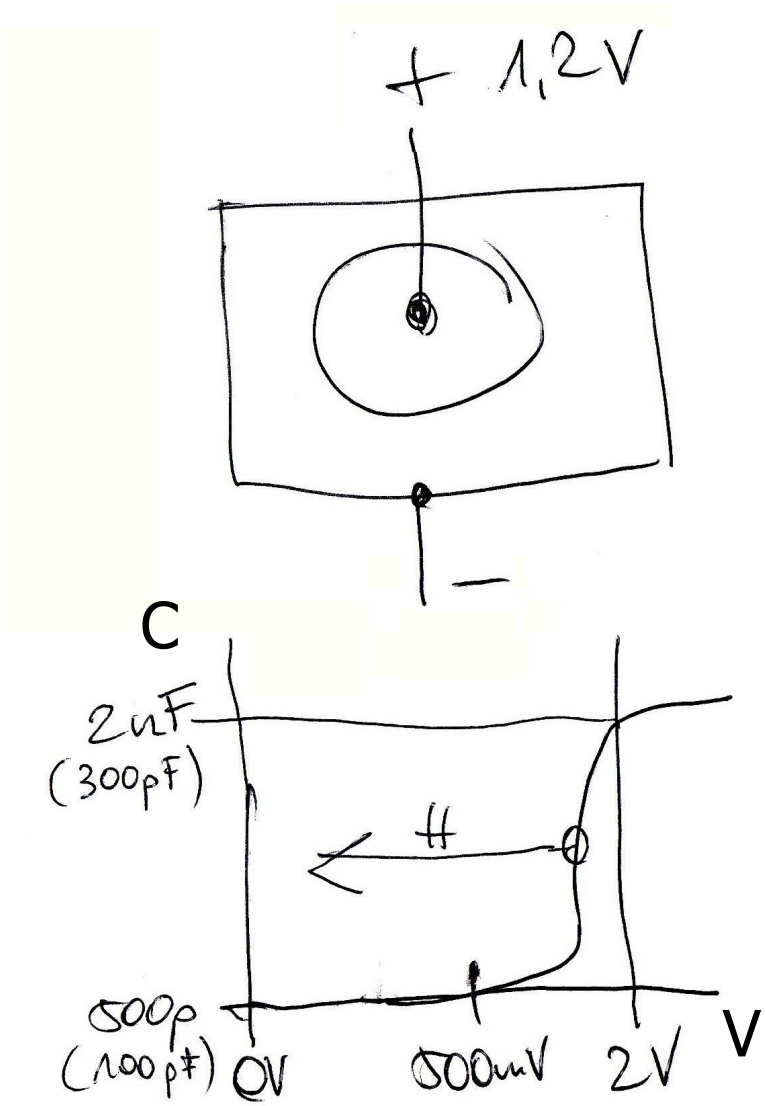
Meßprinzip

- Zur Messung der Kapazität Cm wurde ein spezielles Rampenverfahren entwickelt
- Ein Kondensator wird mittels Widerstand Rm = 220 kOhm permanent langsam hochgeladen
- Übersteigt die zu messende Spannung die Komparatorschwelle von 1,1 Volt (interner Komparator des ATtiny85), wird PB3 auf Ausgabe umgeschaltet
- PB3 zieht dann auf Null Volt (Reset) und verweilt kurz
- PB3 wird zurück auf Eingabe geschaltet
- Übersteigt die zu messende Spannung die Komparatorschwelle – von vorn
- In einem festen Zeitintervall werden die Iterationen gezählt, diese sind invers proportional zur zu messenden Kapazität

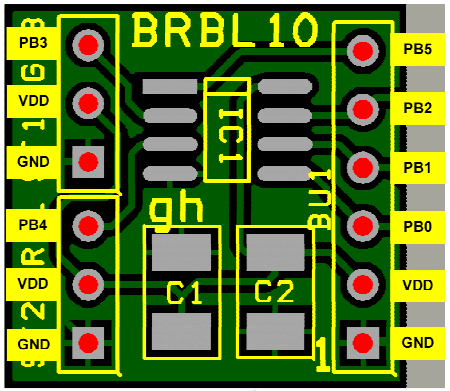


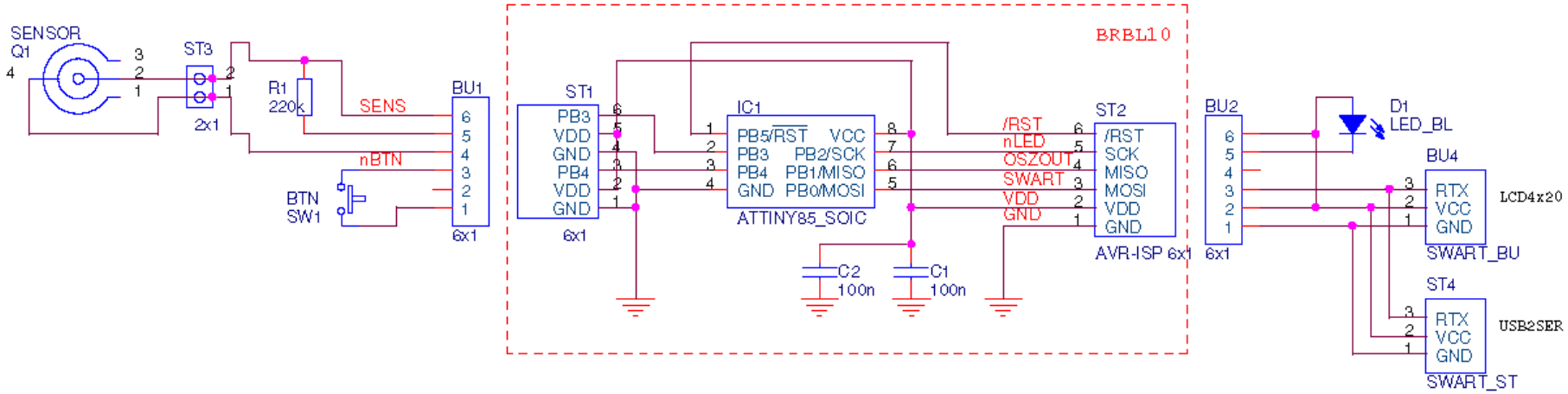
Sensor

- Der abzufragende Wasserstoff-Sensor zeigt eine Besonderheit: Die CV-Kurve verschiebt sich entlang der Volt-Achse
- Je höher die Wasserstoff-Konzentration, desto mehr nach links verschiebt sich die Kurve, siehe Bild
- Die Extrema liegen zwischen beim großen Sensor zwischen 500pF und 3nF, beim kleinen Sensor (hier verwendet) zwischen 100 und 300 pF
- Der optimale Arbeitspunkt liegt bei 1,2 Volt



Schaltung





H-Sensor: Elektroden Pins 2-4
Heizung 20 Ohm, Pins 1-3

Betrieb: 3-poliger Anschluß an CN1 (GND, VCC, SWART)
UART-simplex (Rx und Tx auf einem Pin mit pullup)
ATtiny85: 8kB + 512B
nBTN Automatische Kalibrierung (set to GND)
Programmierung: AVR-ISP an CN1 6-polig
PB1: digitalisiertes Signal von PB3

Title	Wasserstoffsensor Tiny auf Testboard BRBL10 - Rampe 0V bis 1,1V	
Size A4	Document Number heinz@gfai.de	Rev 39
Date: Tuesday, April 07, 2015	Sheet 1 of 7	

Meßergebnisse R = 220 kOhm (aktuell)

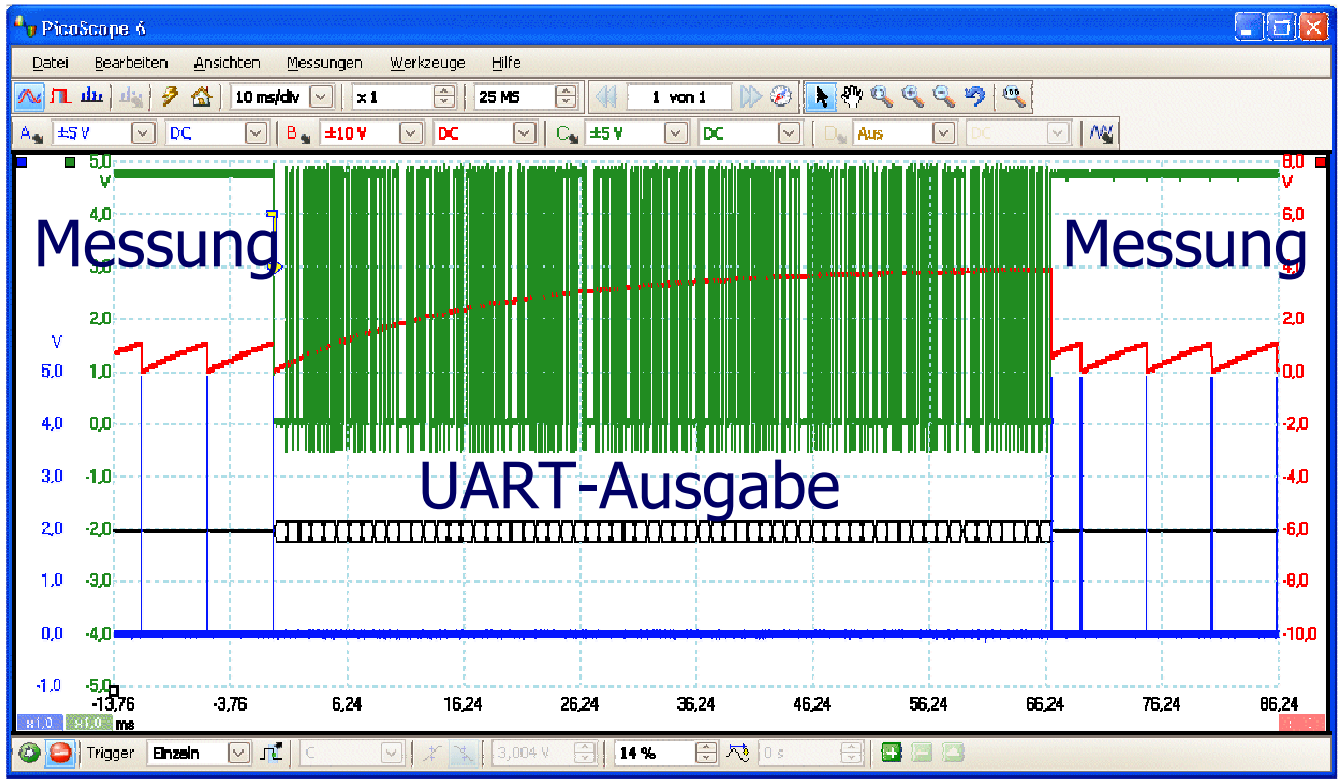
Meßergebnisse (Pins: Sensor Pd, GND RS)

C	R	"ist"
0 pF	220kOhm	136290
130 pF	220kOhm	112500
1 nF	220kOhm	28510
3,3 nF	220kOhm	9330
10 nF	220kOhm	2630
100 nF	220kOhm	697

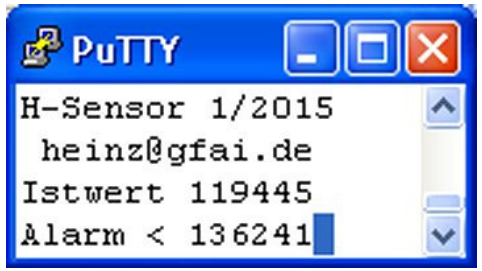
- Meßfluktuation etwa ±1% bei Meßzeit 800ms (Entladezeit 5µs)
- Meßunsicherheit ist gering temperaturabhängig (interner RC-Oszillator)
- Reproduzierbarkeit (Zimmertemperatur) nach 6 monatiger Laufzeit 100% (keine Veränderung des zuletzt eingestellten Schwellwertes)

Oszillogramme

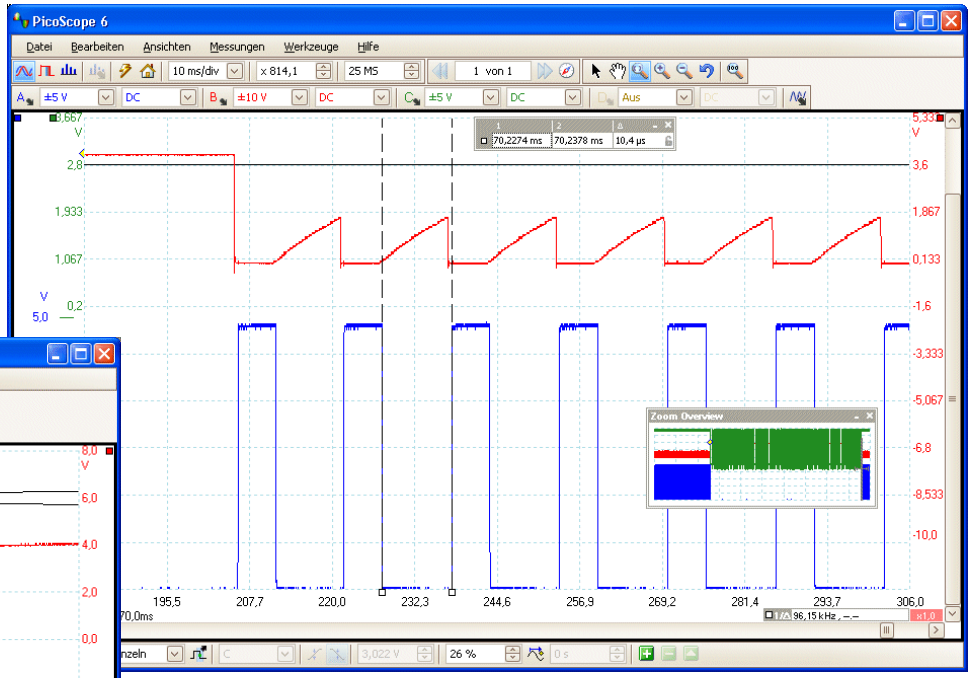
- ab v35: Rampe mit Iteration von 0V bis 1,1V
- Testboard BRBL10
- Bauelemente: ATtiny85 (SOIC8), Widerstand 220 kOhm, Taster

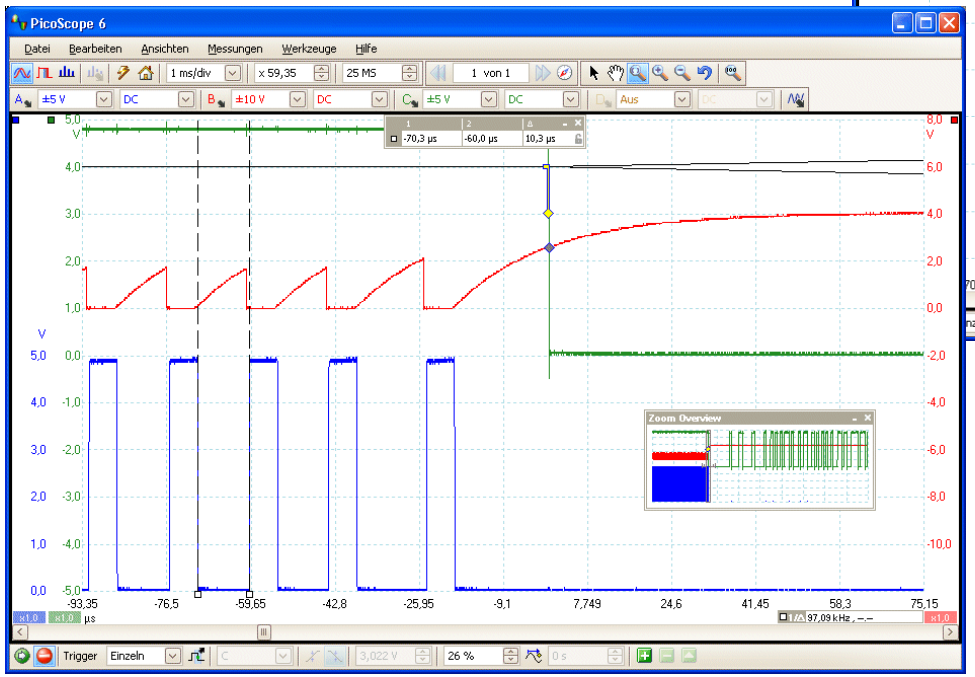


Testkapazität 20pF (+ Oszi-Pin)



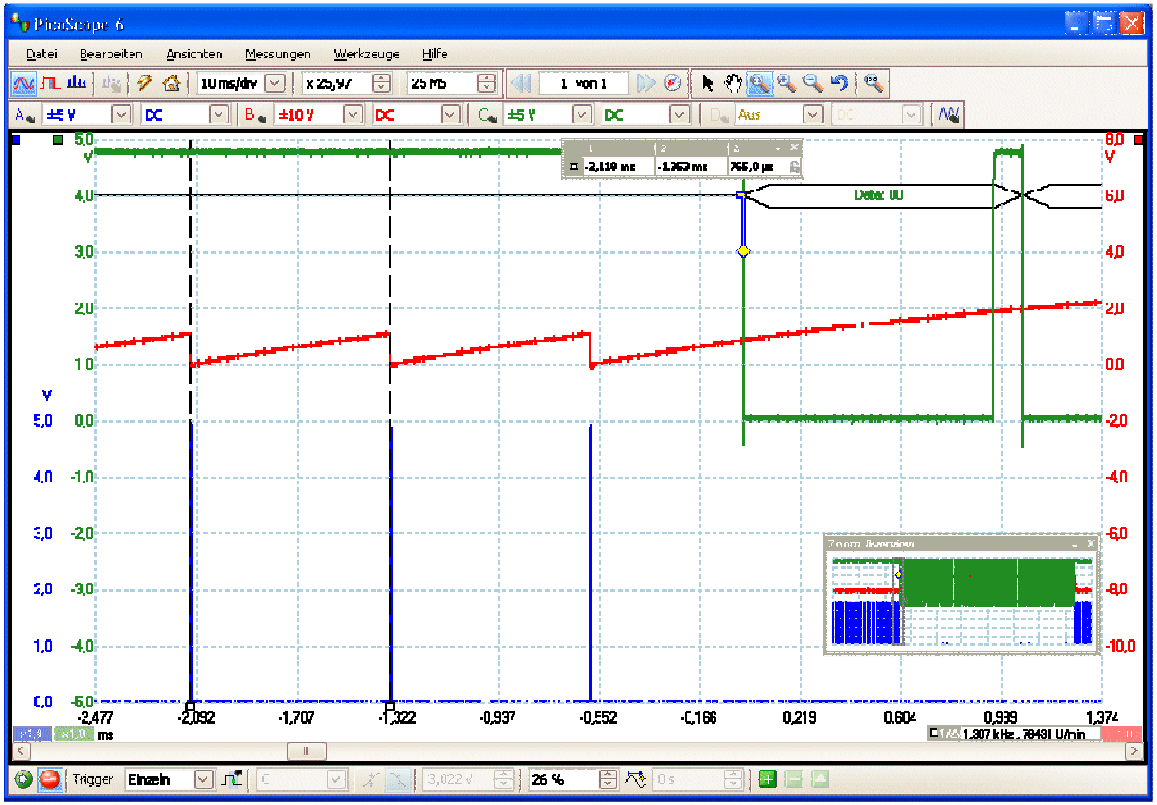
- Sensorpin PB3 (rot)
- Testausgang PB4 (blau)
- UART PB0 (grün)
- R=220 kOhm -> 10,35 µs





Testkapazität 10nF - Rampe

- Sensorpin PB3 (rot)
- Testausgang PB4 (blau)
- UART PB0 (grün)
- R=220 kOhm -> 765 µs



```
/*******************************************************************************


    Wasserstoff-Sensor mit ATtiny85 auf BRBL10
    Gibt Dezimalwert auf Soft-UART aus
    Übernimmt neuen Alarmwert aus dem Istwert auf Tastendruck
    Speichert Alarmwert im EEPROM

    AVR-Studio4 GUI Version 4,19,0,730
    Versuchsaufbau auf Board BRBL10
    Hotline     heinz@gfai.de
    www.gfai.de/~heinz/techdocs

    Emulation eines VT100 Terminals (ANSI)
    mit 9600 Baud, 8n1n

Prinzip:
    Kleinsignal-Kapazitätsmessung mit Sägezahn
    Nur ein Pin (SENS) wird gebraucht.
    Spannung iteriert von 0 bis 1,1Volt.
    R sollte 220 kOhm nicht übersteigen, Reststromgefahr.
    R sollte nicht kleiner als 220 kOhm sein, sonst wird Peak zu hoch,
    2V ohne C

    Status: Bug: uint16_t ist;      // ist int16_t
    Eignung zur Kap-Messung: hervorragend, auch für Wasserstoffsensor geeignet
    Erster erfolgreicher Test am 29.1.2015

Pins:   (Achtung ab v38 neues Pinout!)

    Port  in/out    Funktion
    ____________________________________________

    PB5     -       /RST fix auf VDD
    PB4     I       nBTN Auto-Kalibrierung low-aktiv
    PB3     I/O     Sensorpin
    PB2     O       nLED Schaltausgang low-aktiv
    PB1     O       Oszillation-Testpin
    PB0     I/O     Single Wire UART RTX


Pinout ATtiny85  (PDIP_8 oder SOIC_20SU)
                 --------
    /RESET  PB5 |1  \/  8| VDD  +5V
     Sensor PB3 |2      7| PB2  SCK   nLED
     nBTN   PB4 |3      6| PB1  MISO  Testpin
            GND |4      5| PB0  MOSI  RTX_UART
                 --------
```

Anschlüsse PCB BRBL10

```
 SENS   PB3    -------------    VDD     LED_A
|       VDD   |             |   PB2     LED_K
 SGND   GND   |             |   PB1     Test-Ausgang oder Heizung
 BTN    PB4   |             |   PB0     RTX |
|       VDD   |             |   VDD     +5V | UART
 BTN    GND    -------------    GND     GND |
```

Prozessor: ATMEL ATtiny85
AVR Studio 4.19: Set controller in
1) "Project -> Configuration options"
2) "Debug -> Select Platform and Device"
Der Prozessor läuft mit 8 MHz (interner Oszillator) auf 2,7 bis 5 Volt

Funktion

```
                              Bandgap   |\   Komparator im ATtiny85
                 ___            1,1V  --|+ \____
        +5V o--|___|----o-------o----|- /    ACO
                 R      |  SENS      |/
                     C ===
                        |
                       --- GND
```

Zwischen +5V und Sensorpin S befindet sich ein Ladewiderstand R
und die zu messende Kapazität C. Am Sensorpin ist der interne Komparator
angeschlossen. Schaltet der Ausgang ACO des Komparators auf low, wird
das Sensorpin auf low geschaltet und entlädt kurz den Knoten.
Anschließend bestimmt der Widerstand R die steigende Flanke.
Es entsteht eine Dreiecksoszillation im Bereich von 0 bis 1,1 Volt,
deren Frequenz proportional zur Kapazität C ist.

Aktuelle Dimensionierung des Sensor Pullup Widerstands R
R = 220kOhm führt ohne Last auf maximalen Hub von 0...1,8 Volt
Ziel: Abschaltung bei 1,1V.
R besser größer, aber R = 1 MOhm ist bereits zu feuchtempfindlich
#define messinterval   // Meßintervall mal 100ms siehe ups.h
#define discharge   // Entladezeit in µsec siehe ups.h
bei Einstellung 5µs werden gemessen 5,7µs

UART-Ausgabe für LCD 4x20: 9600 Baud, 8n1n

```
0123456789abcdef0123
====================
H-Sensor 1/2015
 heinz@gfai.de
```

```
    Istwert 136437
    Alarm < 119241
    ====================

    Istwert:    "ist"   Momentanwert
    Alarmwert:  "alrm"  Wenn Istwert kleiner, wird LED eingeschaltet
    Werte werden ASCII dezimal ausgegeben als Byte-Array erg[0]...[n]

Single-Wire UART Prinzip: (RTX wird mit Pullup 4,7 kOhm hochgezogen)

        RX <--o----...------<> RTX
              |
    USB       V A              Sensor
          D1  - K
              |
        TX >--


UART Protokoll (Rx und Tx)

                 S   0   1   2   3   4   5   6   7   P   P
    H   ------     --- --- --- --- --- --- --- --- --- --- .......
    L         `--- --- --- --- --- --- --- --- ---´

     S: Startbit (low), P: Stopbit (high, Tx: 2x), 0...7 Databits, LSB first
     Wir senden mit 9600 Baud d.h. 104 µs pro Bit und mit zwei Pausen P

Taster zur Kalibrierung nBTN
    Der Alarmwert kann eingestellt werden, indem der Sensor der Alarmbedingung
    (H-Konzentration) ausgesetzt wird, und nBTN gedrückt wird (nach GND).

Alarmwertspeicher im EEPROM
    Es wird ein uint16_t als Alarmwert gespeichert, und
    bei Startup automatisch geladen.

Fuses bitte einstellen:
    BODLEVEL = 1,8V
    CKDIV8 rücksetzen (default ist 1 MHz) --> auf 8 MHz
    SUT_CKSEL = int.RC Osc. 8 MHz
    EXT     0xFF
    HIGH    0xDE
    LOW     0xE2

AVR Memory Usage
    Device: attiny85
    Program:    2620 bytes (32.0% Full)
    (.text + .data + .bootloader)
    Data:         31 bytes (6.1% Full)
```

```
    (.data + .bss + .noinit)
    Build succeeded with 0 Warnings...

    Mit ### markierte Werte sind an PCB und Prozessor anzupassen.

*******************************************************************************
*******************************************************************************/

// Zubehör
#include <stdint.h>         // uint8_t
#include <stdlib.h>         // itoa
#include <stdio.h>          // sprintf
#include <util/delay.h>     // ms, µs, braucht F_CPU
#include <avr/io.h>         // ports and pins
#include <avr/interrupt.h>  // interrupts
#include <avr/pgmspace.h>   // PROGMEM prog_char pgm_read_byte für EEPROM

#include "ups.h"            // alle Unterprogramme des Projekts


/*******************************************************************************/
/*******************************************************************************/


int main(void){

    cli();                  // Interrupts sperren
    ButtonInit;             // BTN mit pullup
    LEDinit;                // LED ohne pullup
    LEDein;                 // output low
    UARTinit;               // UART receiving

    // Oszillator
    Sinit;                  // Sensor-Pin
    // Ainit;               // Aktor-Pin Apin als Testsignal-Ausgabe
    compInit();             // Komparator

    wait_ms(1000);          // auf Terminal-Init warten
    LEDaus;
    initBittaktTimer();     // für UART
    sei();                  // für UART
    showscrn();             // Splashscreen zeigen
    wait_ms(1000);          // eine Weile stehenlassen

    EEPROM_read(EEadr, &alrm);  // Alarmwert aus EEPROM holen
    messTimInit();          // Meß-Timer, Zeitintervall für Zählung
```

```
 while(1) {
    ist  =  0;              // Zahl der Umladungen
    sei();                  // Interrupts freigeben

    // neue Messung
    messung(messinterval);  // ist-Wert im Meßinterval hochzählen

    // LED-Ausgabe
    if (ist > alrm)     {LEDaus;}   // LED-Ausgang als Schalter
    else                {LEDein;}   // Alarm --> nLED = low

    // UART-Ausgabe
    showscrn();             // Splashscreen zeigen
    showresult();           // Messwerte an UART ausgeben

    // Button abfragen
    isbuttonpressed();      // soll ein neuer Alarmwert eingestellt werden?

    // alles von vorn, neuer Zyklus
 }  // while(1)

}   // main(void)
```

```
// Zubehör laden
#include <stdint.h>         // uint8_t
#include <stdlib.h>         // itoa, utoa
#include <stdio.h>          // sprintf
#include <avr/io.h>         // ports and pins
#include <avr/interrupt.h>  // interrupts
#include <avr/eeprom.h>     // eeprom_read_byte
#include <util/delay.h>     // ms, µs, braucht F_CPU
#include <avr/pgmspace.h>   // PROGMEM prog_char pgm_read_byte für EEPROM
#include "ups.h"

// String-Konstanten            "0123456789abcdef0123"  20-Zeichen LCD, vierzeilig
const char scrn1[]  PROGMEM =   "H-Sensor 1/2015\r\n";  // erste  Zeile
const char scrn2[]  PROGMEM =   " heinz@gfai.de";       // zweite Zeile
const char ist_s[]  PROGMEM =   "Istwert";      // aktueller Meßwert "ist"
const char thr_s[]  PROGMEM =   "Alarm <";      // Auslöseschwelle  "alrm"

/*******************************************************************************/
/***** UPs *********************************************************************/
/*******************************************************************************/

/*** Komparator ******************************************************************/
// Der Comparator invertiert zwischen Sensor Spin und Comp-Out Compin

ISR(ANA_COMP_vect){         // comparator interrupt, falling edge detected, SENS > 1,1V
        setLow;             // discharge node to 0 Volt
        // setAhigh;        // Testsignal
        wait_us(discharge); // Entladezeit in µsec
        setInp;             // charge node slowly with R
        // setAlow;         // Testsignal
        ist++;              // Meßwert erhöhen bei jeder Umschaltung
}


void compInit(void) {       // Comparator Init, neginput = SENS, posinput = bandgap 1,1V ###
    bclr(ADCSRA,ADEN);      // Disable ADC p.119 tab. 16-1
    bset(ADCSRB,ACME);      // enable ADC-mux for AC
    bset(ADMUX,MUX1); bset(ADMUX,MUX0); // ADC-Input ADC3 -> SENS, MUX[3:0]=0011, p.135 tab17-4
    bset(ADMUX,REFS1);      // Bandgap 1,1V ohne external capacitor
    bset(ACSR,ACBG);        // Bandgap einschalten
    bset(ACSR,ACIS1); bclr(ACSR,ACIS0); // interrupt on falling output edge p121, tab16-2
    bclr(ACSR,ACIE);        // interrupt disable
}


void compStart (void) {
    bset(ACSR,ACIE);        // comparator interrupt enable ###
}
```

```
void compStop (void) {
    bclr(ACSR,ACIE);        // comparator interrupt disable ###
}

/*** Timer1 für Meßinterval ***************************************************/
// Interrupt occured:       // Meßzeit-Timer

ISR(TIM1_COMPA_vect)  {     // timer1 compare match A, klingelt alle 100ms
// ISR(TIM1_OVF_vect) {     // interrupt if timer overflow, 255ms,  p.44
    mtime++;                // Zahl durchlaufener Meßintervalle a 100ms
}


void messTimInit(void) {    // Meßzeit-Intervall Timer timer1
    // timer1 prescal: 8 MHz/8192 ~ 1,024 ms -> timer1 clock, p.83

    TCCR1 |= (1<<CS13)|(1<<CS12)|(1<<CS11) ; // CK/8192 prescaler division, p.90
    OCR1A  = 97;            // bset output compare match register 100ms, p.89
    bset(TCCR1,CTC1);       // bset CTC-mode für output compare match, p.89
    TCNT1  = 0;             // reset timer value -> macht CTC
    bclr(TIMSK,OCIE1A);     // disable output compare A interrupt, p.87
    // bclr(TIMSK,TOIE1);   // disable ovf interrupt, p.87
}


void messTimStart(void) {   // vorgegebene messintanzahl mal 100ms
    TCNT1 = 0;              // reset timer value, see p.81 ff
    mtime = 0;
    //bset(TIMSK,TOIE1);    // disable ovf interrupt, p.87
    bset(TIMSK,OCIE1A);     // enable output compare A interrupt, p.87
}


void messTimStop(void) {    // Meßzeit-Intervall Ende
    //bclr(TIMSK,TOIE1);    // disable ovf interrupt, p.87
    bclr(TIMSK,OCIE1A);     // disable output compare A interrupt, p.87
    mtime = 0;              // bereit für neue Messung
}

/*** EEPROM *******************************************************************/
uint8_t EEarr[eesize] EEMEM;


void EEPROM_write(uint8_t addr, int32_t value) {        // addr (0...63-eesize)
    eeprom_write_block (&value, EEarr, sizeof(value));  // (RAM-addr, EEPROM-addr, size_t)
}
```

```
void EEPROM_read(uint8_t addr, int32_t *value) {            // addr (0...63-eesize)
    eeprom_read_block (value, EEarr, sizeof(*value));   // (RAM-addr, EEPROM-addr, size_t)
}

/*** Taste ****************************************************************/
void isbuttonpressed(void) {
    if (!(BtnInport & (1 << BtnPin))){  // wenn Taste gedrückt ist (auf GND)
        alrm = ist - 1;     // neuer Alarmwert ist um Eins kleiner als Istwert
        // ins EEPROM schreiben
        LEDein;                 // test
        EEPROM_write(EEadr, alrm);  // "alrm"-Wert speichern
        ucmd(NUL,FF);           // Cursor auf Zeile 1, Pos.0
        uputsf(scrn1);          // Splashscreen zeigen
        uputsf(scrn2);          // Splashscreen zeigen
        wait_ms(2000);          // sichtbar lange Pause nach dem Brennen
        LEDaus;                 // test
        wait_ms(800);           // zeigen
    }
}

/*** Soft-UART mit Bittakt-Timer0 *************************************************/
    // Wir senden mit 9600 Baud d.h. 104 µs pro Bit
    // Der Prozessor ATtiny85 läuft mit 8 MHz (interner RC-Oszillator)
    // Der Timer hat insgesamt einen Teiler von
    // 8.000.000 Hz / 9600 Baud = 833 zu realisieren.
    // Schalten wir einen Vorteiler von 8 ein, folgt ein einzustellender Timerwert
    // von 833 / 8 ~ 104 für den Bittakt, Fehler 0,16%.
    // pro Halbbit sind 52, pro 1,5 Bit sind 156 einzustellen.
#define HalfBitDelay        52      // timer 0,5 bittime for UART
#define OneBitDelay         104     // timer 1,0 bittime for UART
#define ThreeHalfBitDelay   156     // timer 1,5 bittime for UART

volatile uint8_t nextbitsig = 0;    // global timer signal

// Bittakt erzeugen:mit Timer0
ISR(TIM0_COMPA_vect){           // interrupt if timer0 compare match A, p.44
    nextbitsig = 1;             // set nextbitsig-signal
    TCNT0    = 0;               // reset timer value
}


void initBittaktTimer(void){    // UART bit clock, timer0, prescal 8, see p.65 ff
    TCCR0B |= (1<<CS01);        // prescaler division by 8, p.72
    TCCR0A |= (1<<WGM01);       // set CTC-mode for output compare match A, p.79
    TIMSK  &= ~(1<<OCIE0A);     // disable output compare interrupt, p.74
    nextbitsig = 0;             // reset nextbitsig
```

```
    TCNT0   = 0;                // reset timer value
}


void starteBittakt(uint8_t time2wait){  // UART bit clock, timer0, stop at time2wait
    TIMSK  &= ~(1<<OCIE0A);     // disable output compare interrupt, p.74
    nextbitsig = 0;             // reset nextbitsig
    TCNT0    = 0;               // reset timer value
    OCR0A    = time2wait;       // set output compare match register, p.74
    TIFR  |= (1<<OCF0A);        // falls Interrupt bereits gesetzt war, löschen (!)
    TIMSK   |= (1<<OCIE0A);     // enable output compare A interrupt, p.74
}


void beendeBittakt(){           // UART bit clock ATtiny13, see p.64 ff
    TIMSK &= ~(1<<OCIE0A);      // disable output compare interrupt, p.74
    nextbitsig = 0;             // reset nextbitsig
    TCNT0    = 0;               // reset timer value
}


void waitabit(){                // wait for timer signal "nextbitsig"
    while (!nextbitsig)         // polling timer
    {;}                         // wait until nextbitsig = 1
    nextbitsig = 0;             // reset nextbitsig
}


// UART ////////////////////////////////////////////////////////////

// Soft_UART_Tx
void uputc(uint8_t ch) {        // ein UART Zeichen senden
    // non-inverting-UART       // soft-send one character bit by bit
    uint8_t i=0;
    // send startbit low:
    starteBittakt(OneBitDelay); // start timer with bit clock
    sendL;                      // send startbit = low
    waitabit();                 // wait for next clock
    // send 8 bits:
    for (i=0; i<8; i++)         // send all bits of "ch"
    {                           // sende jeweils das LSB (rechtes)
        if ((ch & 0x01))        // if right most bit is one
            { sendH; }          // send one
        else                    // if right most bit is zero
            { sendL; }          // send zero
        ch = ch >> 1;           // bit shift right (send LSB)
        waitabit();             // wait for next clock
    }
```

```
    // send 2 stop bits (lieber bissel zuviel, erhöht Fehlertoleranz):
    sendH;                      // send stopbit = high
    waitabit();                 // erstes Stopbit
    // waitabit();              // zweites Stopbit kann man senden
    beendeBittakt();
    // receive;                 // gehe auf Empfang mit pullup == sendH
}


// UART String aus RAM senden
void uputs(char *strg){ // ab Zeiger *strg senden
    while (*strg) {     // solange *strg != '\0' (ungleich String-Ende)
        uputc(*strg++);
    }
}


// UART String aus dem Flash senden
void uputsf(const char *strg){      // ab Zeiger *strg senden
    char ch = pgm_read_byte(strg);  // erstes char aus Flash holen,
    strg++;                         // ptr increment
    while (ch) {        // solange ch != '\0' (ungleich String-Ende)
        uputc(ch);
        ch = pgm_read_byte(strg++); // String abarbeiten
    }
}


void ucmd(uint8_t char1, uint8_t char2){    // send a two-char-command to LCD
    uputc(char1);
    uputc(char2);
}


// volatile char outstrg[16] = {};      // bei utoa/itoa ist Ergebnis null-terminiert '\0'

/*  // Integer to ASCII Ausgabe mit itoa, utoa: sparsame Variante
void uout(const char *string, uint16_t messwert){   // Meßwert-Ausgabe
    uputsf(string);                 // Ausgabe flash string
    utoa((uint16_t) messwert, outstrg, 10); // int to ascii: itoa(int,chararray,radix)
    uputs(outstrg);
}*/

//  uint to ascii with sprintf - kostet 1,4kB flash mehr
void uout(const char *strg, int32_t messwert){      // Meßwert-Ausgabe
    uputsf(strg);                       // Ausgabe string
    sprintf(outstrg, " %ld", messwert); // d: int; u: unsigned integer; ld: long int
    uputs(outstrg);
```

```
}


/*** Messung ***********************************************************/


void messung(uint8_t messintanzahl){    // Anzahl der Meßintervalle a 100ms
    mtime = 0;      // Zeit auf Null
    ist = 0;        // Zähler zurücksetzen
    messTimStart(); // Zeitmessung starten (Timer1), zählt "ist" hoch
    compStart();    // Comparator starten, siehe ISR
    while ( (mtime < messintanzahl) ){
    }   // verweile, solange Meßzeit nicht abgelaufen
    compStop();     // Comparator-Interrupt stoppen
    messTimStop();  // Meßzeit stoppen (Timer1). Interrupt disable
}



/*** Screens ***********************************************************/

void showscrn(void){
    ucmd(NUL,FF);           // reset terminal
    uputsf(scrn1);          // Splashscreen zeigen
    uputsf(scrn2);          // Splashscreen zeigen
}


void showresult(void){  // Messwerte an UART ausgeben
    // erste Zeile:
    ucmd(CR,LF);            // Cursor auf Zeile 1, Pos.0
    uout(ist_s, ist);       // Ausgabe des Istwerts
    // zweite Zeile:
    ucmd(CR,LF);            // Cursor auf Zeile 2, Pos.0
    uout(thr_s, alrm);      // Ausgabe des Alarmwerts
}
```

```
#ifndef UPS_H ////////////////// header ////////////////////////////////
#define UPS_H

// Hardware-Konfiguration ////////////////////////////////////////////////

// Project-Options:
// #define F_CPU 8000000    // interner RC-Oszillator ATtiny85 für delay.h
// #define __AVR_ATtiny85__ // bei AVR-Studio-4 im Project-Menue einzustellen
// zusätzlich unter Debug/Select Platform and Device wählen: Simulator2 ATtiny85

// Hardwarespezifische Eigenheiten sind markiert mit ###



// Parameter ////////////////////////////////////////////////////////

// Integrationszeit für eine Messung in 100 Millisekunden
#define messinterval 8      // Meßintervall mal 100 ms

// Löschzeit der zu messenden Kapazität in µsec
#define discharge 5         // reicht bis 100nF

// Ausgabebuffer: int16 ~ 6-stellig, int32 ~ 11-stellig
char outstrg[12];           // for utoa/itoa/sprintf

// EEPROM-Speicherplatzpaar für den Alarmwert "alrm"
#define EEadr 12            // Adresse zwischen 0...(63-eesize)

// Format zur Speicherung alrm im EEPROM
#define eesize 4            // bytes stored



// Pins und Ports ///////////////////////////////////////////////////


// UART Pin und Port
#define Upin        PB0     // SWART-RTX-Pin
#define Udir        DDRB    // direction
#define Uoport      PORTB   // outport
#define Uiport      PINB    // inport

// LED (low aktiv)
#define LEDpin      PB2     // LED-Pin
#define LEDport     PORTB   // outport
#define LEDdir      DDRB    // direction

// Taste zur Festlegung der Schwelle
```

```
#define BtnPin      PB4     // button pin
#define BtnDir      DDRB    // direction
#define BtnOport    PORTB   // outport
#define BtnInport   PINB    // inport

// Sensorpin
#define Spin        PB3     // oscillator input pin, sensor S
#define Sdir        DDRB    // direction
#define Soport      PORTB   // outport
#define Siport      PINB    // inport

// Testpin
#define Apin        PB1     // oscillator output pin, actor A
#define Adir        DDRB    // direction
#define Aoport      PORTB   // outport
#define Aiport      PINB    // inport

// Comparator
#define Compin      ACO     // Ausgabewert im Comparator
#define Comreg      ACSR    // im Register


// serial UART für VT100/ANSI Terminals
// verwendete ASCII-Steuerzeichen
#define NUL 0x00    // null character \0
#define BEL 0x07    // bell
#define BS  0x08    // backspace
#define TAB 0x09    // horizontal tabulator \t
#define LF  0x0a    // line feed, new line \n
#define FF  0x0c    // formfeed, hier: clear screen
#define CR  0x0d    // carriage return, \r
#define ESC 0x1b    // escape
#define SPC 0x20    // space sign


// Macros ////////////////////////////////////////////////////////////

#define wait_ms(value) _delay_ms(value)
#define wait_us(value) _delay_us(value)


// port can be: Pdir, Pout, Pinp ...; bit is PB0, PB1 ...
#define bset(port,bit) ((port) |= (1 << (bit)))
#define bclr(port,bit) ((port) &= (unsigned) ~(1 << (bit)))
#define btgl(port,bit) ((port) ^= (1 << (bit)))
```

```
// Pin-Configuration ATtiny85 ////////////////////////////////////////

/* UART Tx/Rx Zustände für 1-wire UART (Rx und Tx auf einem Pin):
    Uout    Tx sendet low       outport low
    Uout    Tx sendet high      inport mit pullup
    Uin     receive             inport mit pullup
    --> nur zwei Zustände sind nötig: receive und sendL     */

#define receive { bclr(Udir, Upin); bset(Uoport,Upin); } // input with pullup
#define sendL   { bset(Udir, Upin); bclr(Uoport,Upin); } // output low
#define sendH    { receive; }                            // output with pullup
#define UARTinit { receive; }                            // Ruhezustand

// push-button BTN1
#define ButtonInit  {bclr(BtnDir, BtnPin); bset(BtnOport, BtnPin);} // input mit pull-up

// Sensor S
#define Sinit   { bclr(Sdir, Spin); bclr(Soport,Spin); } // inport ohne pullup
#define setLow  { bset(Sdir,Spin); bclr(Soport, Spin);} // Kap. entladen
#define setInp  { Sinit; }                               // inport ohne pullup

/* // Aktor A (for tests only)
#define Ainit    { bset(Adir, Apin);  } // as output
#define setAhigh { bset(Aoport,Apin); } // high
#define setAlow  { bclr(Aoport,Apin); } // low
*/
// LED-init
#define LEDinit { bset(LEDdir, LEDpin); bclr(LEDport,LEDpin); } // output low
#define LEDein  { bclr(LEDport,LEDpin); }
#define LEDaus  { bset(LEDport,LEDpin); }
#define LEDtgl  { btgl(LEDport,LEDpin); }


// globale Messzeit für Timer1-Interrupt
volatile uint16_t mtime;        // global time in 100 ms steps



/******************************************************************************/
/***** UPs ********************************************************************/
/******************************************************************************/


/*** Komparator ***************************************************/
volatile uint8_t compsig;       // global comparator signal for out-toggle
ISR(ANA_COMP_vect);             // comparator interrupt

void compInit(void);            // neginput = SENSpin, posinput = bandgap 1,1V
```

```
void compStart (void);          // interrupt enable
void compStop (void);           // interrupt disable



/*** Timer1 für Meßinterval ****************************************/
volatile uint8_t endsignal;     // global timer signal für Meßende
ISR(TIM1_COMPA_vect);           // interrupt if timer compare match A, p.44

void messTimInit(void);         // Meßzeit-Intervall Timer timer1, p.83
void messTimStart(void);        // Meßzeit Start
void messTimStop(void);         // UART bit clock ATtiny13, see p.64 ff



/*** Messung *******************************************************/
volatile int32_t ist;           // Meßergebnis Ist-Wert
volatile int32_t alrm;          // Alarmschwelle des Ist-Wertes

void messung(uint8_t messintanzahl);



/*** EEPROM ********************************************************/

void EEPROM_write(uint8_t addr, int32_t value);
void EEPROM_read(uint8_t addr, int32_t *value);


/*** Soft-UART mit Bittakt-Timer0 **********************************/

// Timer0 (Bittakt)
volatile uint8_t nextbitsig;    // signal for next bit
ISR(TIM0_COMPA_vect);           // interrupt if timer compare match A, p.44

void initBittaktTimer(void);    // UART bit clock ATtiny13, see p.64 ff
void starteBittakt(uint8_t time2wait);  // UART bit clock ATtiny13, see p.64 ff
void beendeBittakt();           // UART bit clock ATtiny13, see p.64 ff
void waitabit();                // wait for timer signal "nextbitsig"

// Soft-UART
void uputc(uint8_t ch);         // ein UART Zeichen senden
void uputs(char *strg);         // RAM
void uputsf(const char *strg);  // Flash
void ucmd(uint8_t char1, uint8_t char2); // send a two-char-command
void uout(const char *string, int32_t messwert);    // Meßwert-Ausgabe
```

```
/*** Button *******************************************************/
void isbuttonpressed(void);


/*** Screens ******************************************************/
void showscrn(void);            // flash-screen
void showresult(void);          // value- und alarm-Wert


#endif ///////////////// header ////////////////////////////////
```