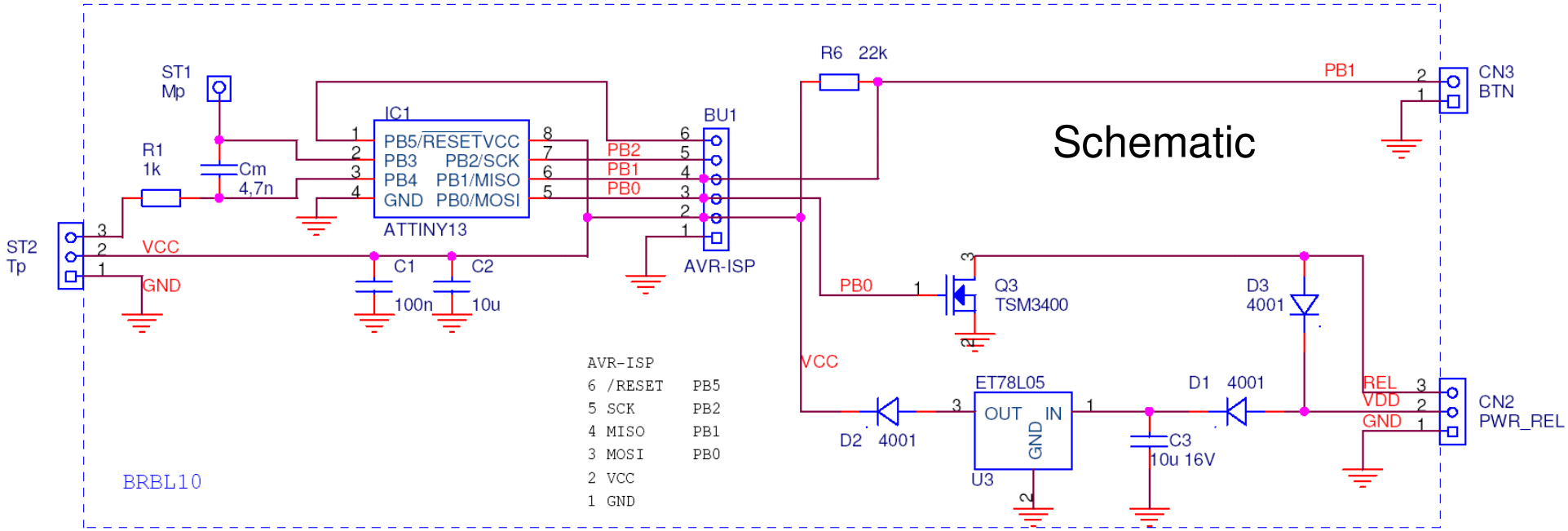
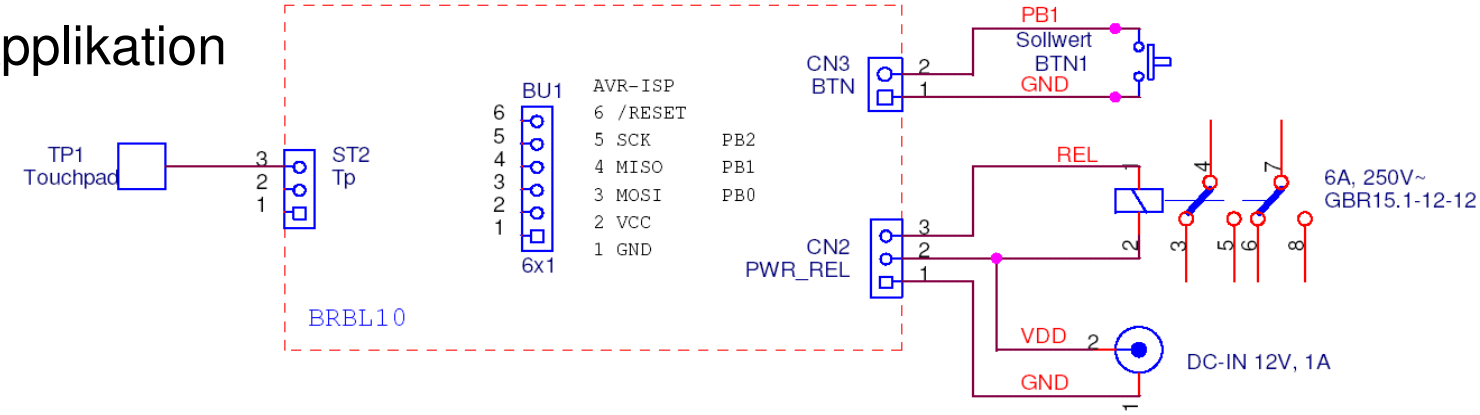
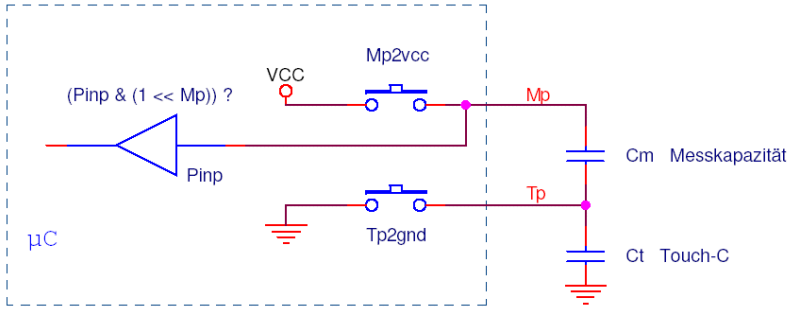
Versuchsschaltung auf Testboard BRBL10



Applikation



Sensor-Prinzip



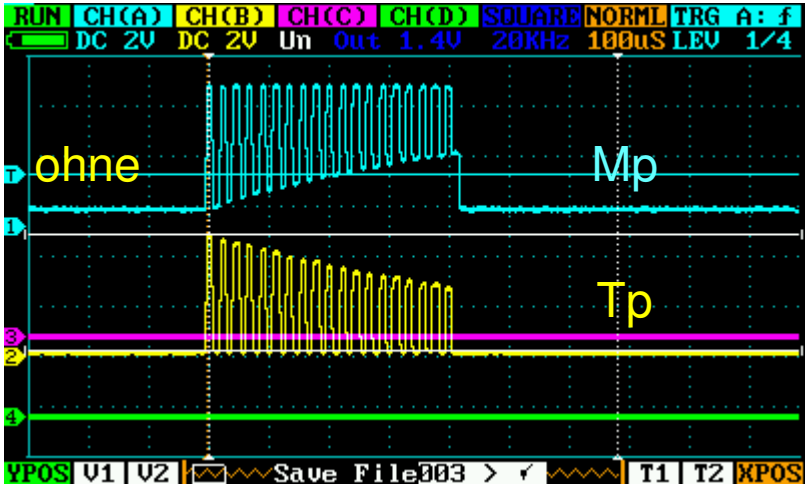
Phase	1	2	3	4
	Open	Tp2gnd	Open	Mp2vcc
Mp	-	-	-	VCC
Tp	-	GND	-	-

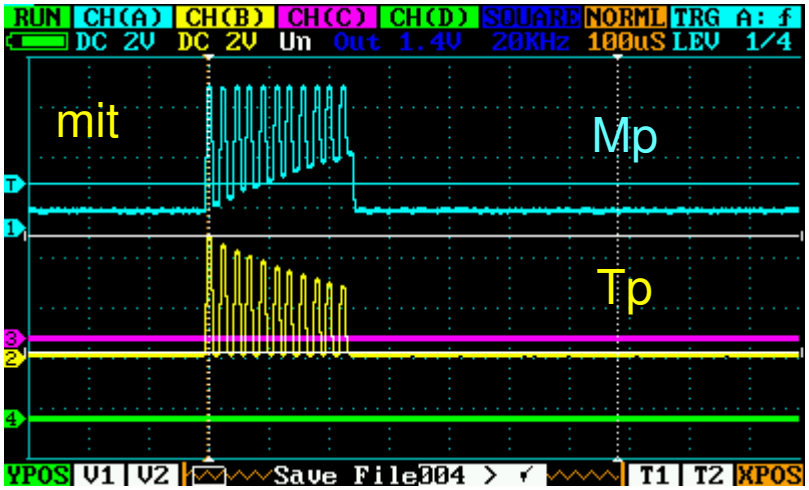
- Messpin Mp, Touchpin Tp
- Cm fix, Ct variierend
- Spannung über Cm: Um = Ump – Utp
- Ist Ct vorhanden (Touch) und Cm leer, wird Cm jeweils in Phase 4 um eine winzige Ladungsmenge
  ΔQm = Qt ~ Ct (Vcc-Um) aufgeladen
- Start: Um = 0V (Cm ist entladen)
- Ende: Um ~ Vcc / 2 („0“→“1“)
- Initial: Mp und Tp werden auf GND gezogen → Entladen von Cm vor Beginn eines Meßzyklus
  Mp = Tp = GND; Qm = 0

- Zyklisch folgen die Phasen 1…4, bis Cm aufgeladen ist
- Mp und Tp werden abwechselnd auf VCC bzw. GND gelegt, aber nie werden beide Schalter gleichzeitig geschlossen
- In Phase 2 wird Ct entladen. Hier kann jeweils an Mp die Ladespannung Um des Messkondensators Cm gemessen werden, da dabei Tp auf GND liegt. Ist Um hoch genug, Abbruch und Auswertung der Zyklenzahl
- In Phase 4 wird Mp hochgezogen → kapazitive Spannungsteilung zwischen Cm und Cp, Ladungstransfer

Oszillogramme

- Versuchsweise wurden die Portpins Mp und Tp mit dem Oszilloskop beobachtet (PB3 und PB4)
- Sie werden dabei zusätzlich durch den Tastkopf des Oszilloskops belastet
- Entsprechend weniger Zyklen werden im Leerlauf durchlaufen
- Die Bilder zeigen Oszillogramme mit und ohne Touch
- Zu Beginn eines Meßzyklus ist Cm entladen
- Der Meßzyklus ist beendet, wenn an Messpin Mp high-Pegel gemessen wird
- Man erkennt sehr schön die Aufladung des Messkondensators Cm
- Die Anzahl durchlaufener Zyklen ist proportional zur Kapazität Cp am Touch- Eingang Tp





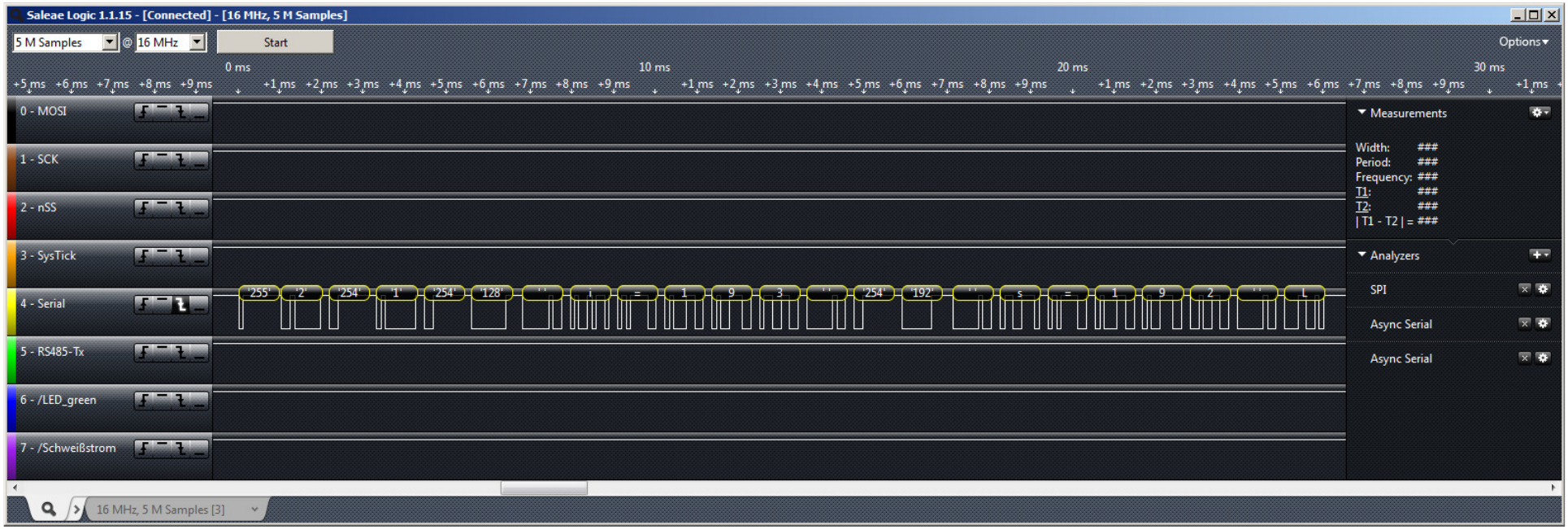
UART-Ausgabe

Soft-UART gibt an serielles LCD aus:

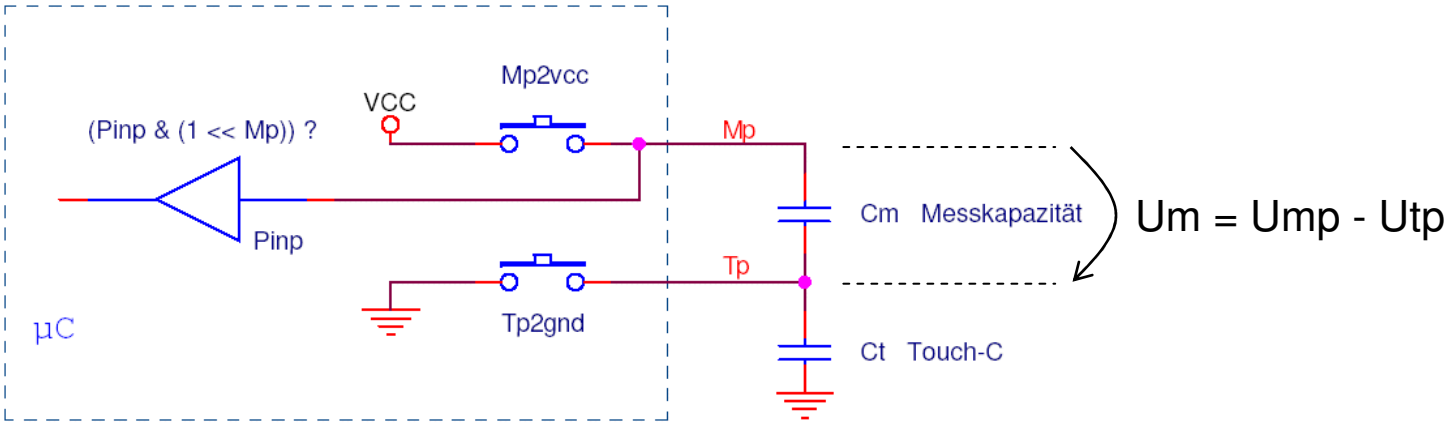
Zahl der Iterationen **i**,
eingestellter Sollwert **s**,
Füllstand **L**ow oder **H**igh

Unten: Ausgabe der Soft-UART an serLCD mit 9600 Baud

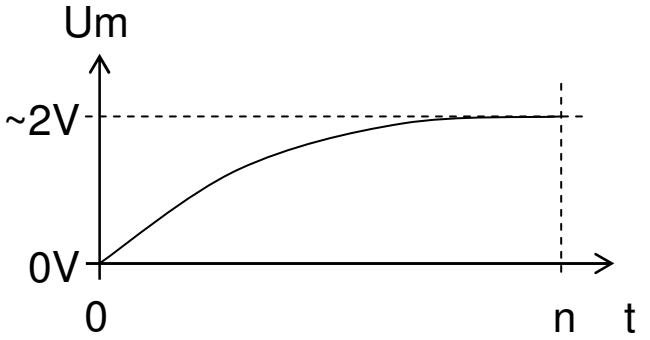
Ausgabe des seriellen LCD
Beispiel Füllstandsmessung



Ermittlung der Touch-Kapazität Ct aus der Zyklenzahl n



1) Qm = Cm Um
2) Qm = n Qt
3) Qt = Ct (Vcc – Um)



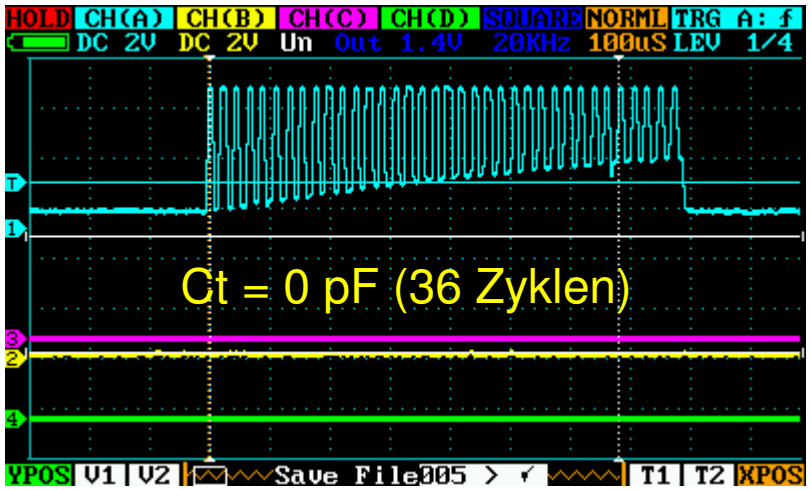
Aus 3) Ct = Qt / (Vcc - Um)
Aus 2) Ct = Qm / n (Vcc - Um)
Aus 1) Ct = Cm Um / (n (Vcc - Um))

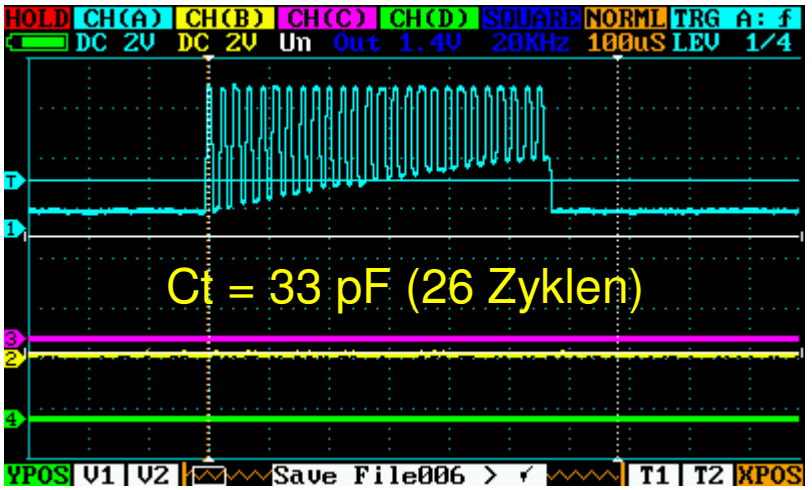
$$Ct = \frac{Cm}{n\left(\frac{Vcc}{Um} - 1\right)}$$

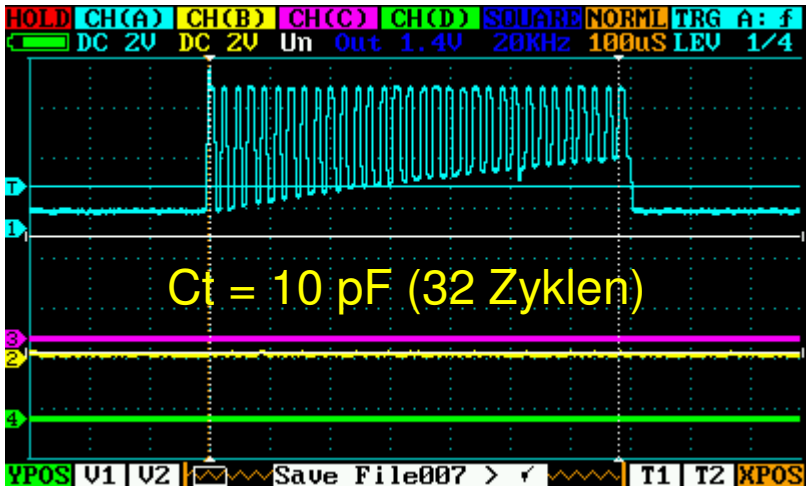
Mit der Formel kann Ct aus der beobachteten Zyklenzahl n bestimmt werden. Cm, Vcc und Um sind bekannt. Die Ladespannung Um ist eine Funktion der Zyklenzahl, siehe Bild.

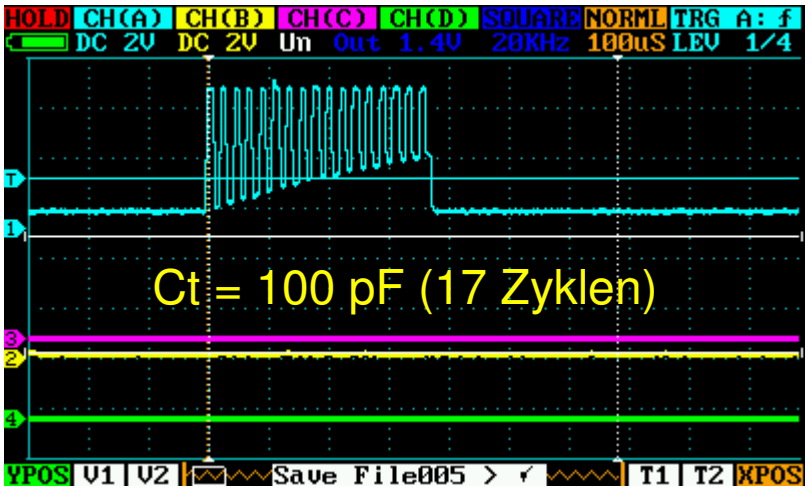
Variation von Ct (Ctouch)

Anschaltung von Festkondensatoren an Ct, Keramik, +/- 5%; Cm = 4.7nF
Vorsicht: hier streift uns die Heisenbergsche Unschärferelation: Entweder wir beobachten mit dem Oszilloskop, oder wir messen genau. Ohne Oszilloskop beobachten wir bei Ct = 0 pF bis zu 194 Zyklen









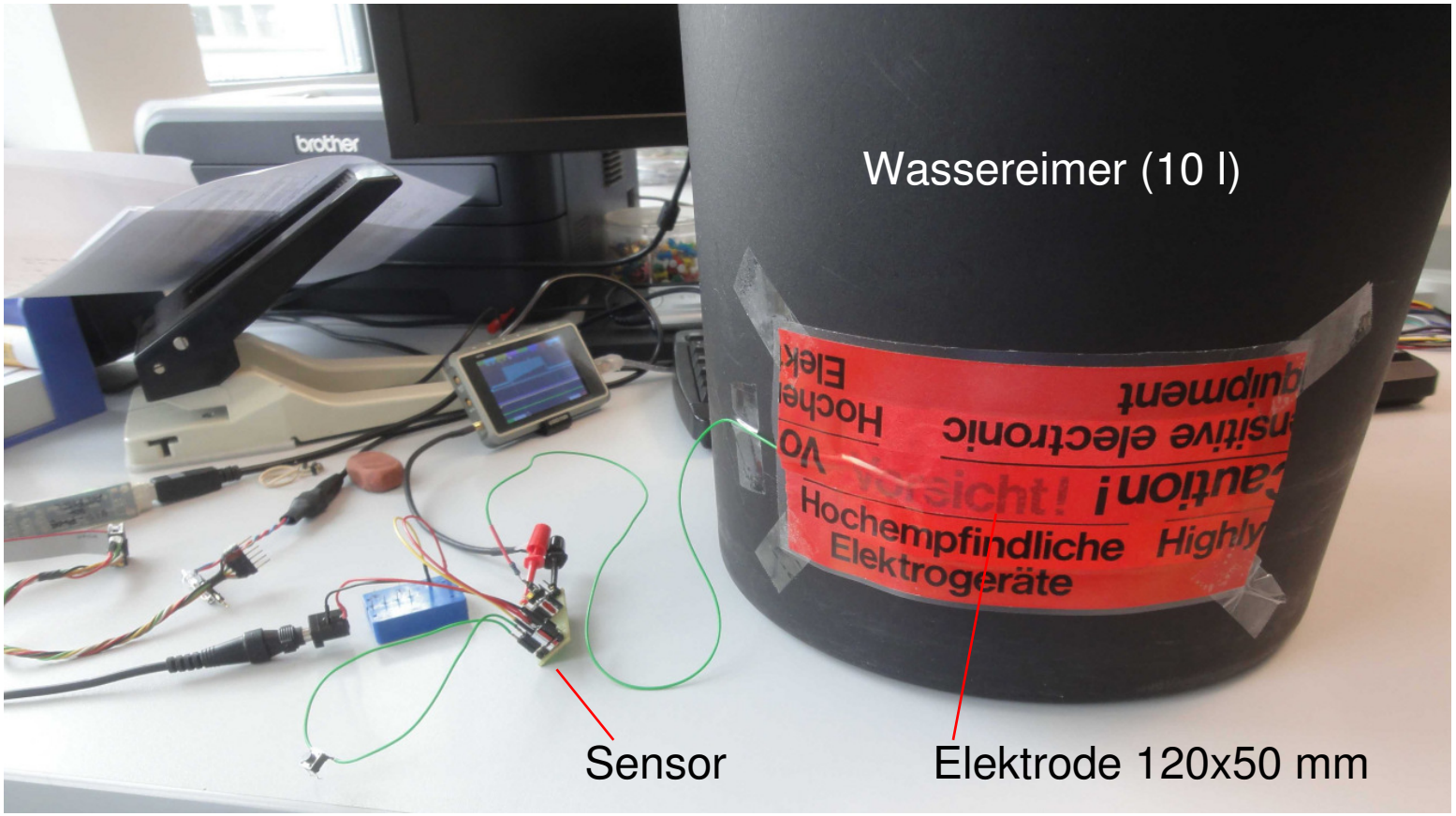
Sensorkalibrierung

- Messung mit Festkondensatoren an Ct, Keramik, +/- 5%
- Messkapazität Cm = 4.7nF, Zyklenzahl n
- Annahme1) Um ~ 2V: K = 1 / (Vcc/Um - 1) = 1 / (5V/2V - 1) → K = 2/3
- Ct = K Cm/n, Cm = 4,7 nF, siehe Tabellenspalte 3
- Fehler-Iteration ergibt K = 0,629, siehe Spalte 4
- damit ist die mittlere Abschaltspannung Um = Vcc/(1/K + 1) = 1,93 Volt
- Die parasitäre Kapazität Cto des Portpins ist hier 18,36 pF,
  ohne Adapterstecker messen wir maximal i = 194, dabei ist Cto = K Cm/n = 15,24 pF

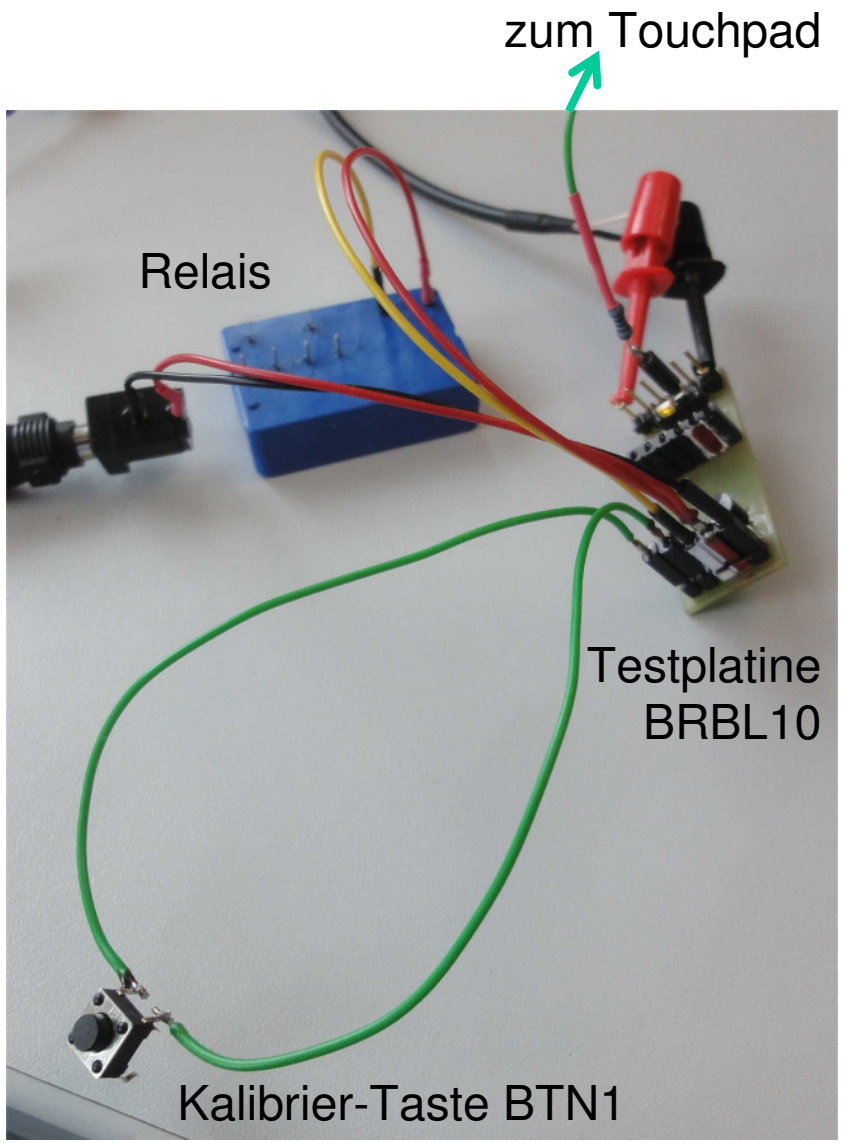
Cextern pF	Zyklen i	Ct = K Cm/n für K = 2/3	Ct = K Cm/n, K = 0,629
0	161	19,46 pF	18,36 pF
10	102	30,72 pF	28,98 pF
33	57	54,97 pF	51,86 pF
100	25	125,33 pF	118,25 pF

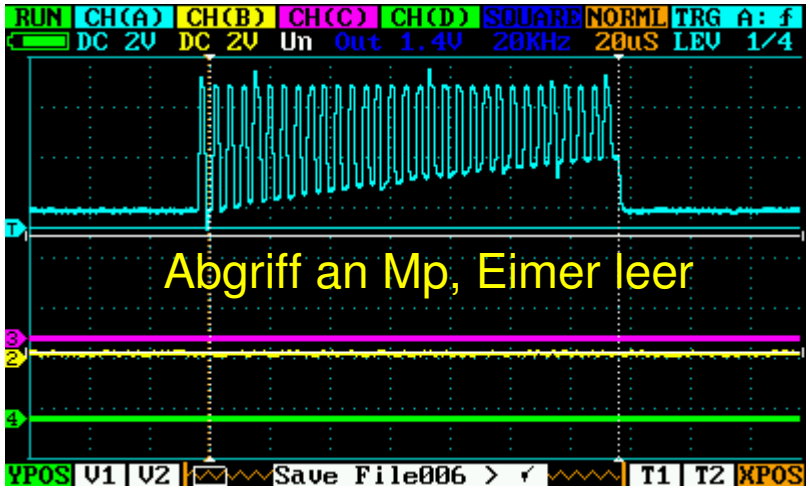
Füllstands-App

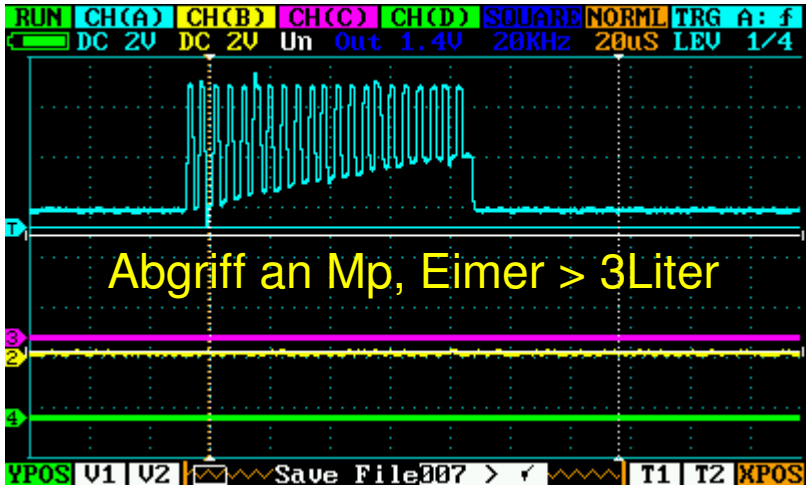
- Zur Überprüfung des Wasserstandes in einem Behälter aus Plast wurden Versuche gemacht
- Behältermaterial PP05 (Polypropylen), Wandstärke ca. 3 mm
- Es wurde eine 50 mm breite Alu-Folie als Elektrode verwendet.



Versuchsaufbau Füllstandsmessung

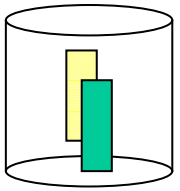


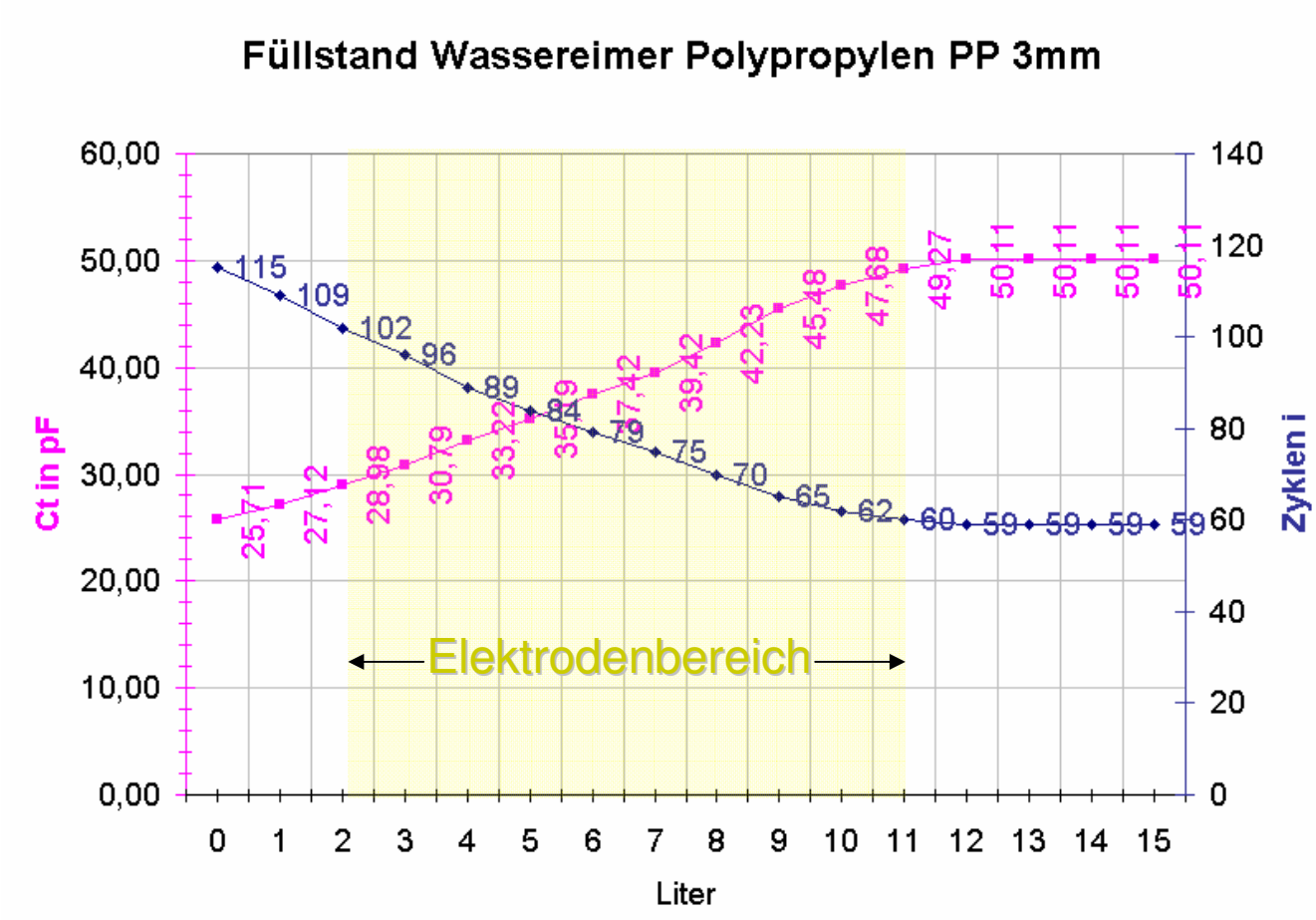




Kapazität als Funktion des Füllstands

- Elektrode und Gegenelektrode 120 x 55 mm sind vertikal gegenüber angebracht





Fazit der Füllstandsversuche

Nachrechnung der Serienkapazität Cser des Behälters:

Polypropylen $\varepsilon_r = 2.1$, Elektrodenfläche A = 120x50 mm² = 6000 mm²,

Wandstärke d = 3 mm → Cser = $\varepsilon_o \varepsilon_r A/d$ = 37 pF

Cser wirkt in Reihe zu Ct und sollte größer Cto gewählt werden

Fazit:

- Es ist prinzipiell möglich, kapazitiv Füllstände zu detektieren
- Die Serienkapazität Cser des Behälters liegt in Reihe zur Touchkapazität Ct und ist nicht vernachlässigbar
- Mit zu kleiner Cser vermindert sich die nutzbare Variation der Zyklenzahl, der Sensor reagiert empfindlicher auf parasitäre Kapazitäten
- So reicht die Näherung einer Hand auf 5 cm bereits aus, um auszulösen
- Es sind relativ großflächige Elektroden erforderlich, um die störende Serienkapazität der Behälterwand zu kompensieren
- In Anbetracht einer parasitären GND-Kapazität des Mikrocontroller-Pins von 15 pF ist die hier gezeigte Elektrode mit Cser = 37 pF noch akzeptabel
- Überraschede Entdeckung: Tp und Mp können vertauscht werden!

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


    Kapazitiver Touch-Sensor mit ATtiny13
    nach dem Prinzip einer kapazitiven Ladungspumpe
    Prüft ein Touchpad durch Kapazitätsmessung
    im Bereich von etwa 20 bis 100 pF
    Gibt Dezimalwert auf Soft-UART aus
    Übernimmt neuen Sollwert aus dem Istwert auf Tastendruck
    Speichert Sollwert im EEPROM
    Brennt Splash-Screen eines Sparkfun Serial-LCD,
    wenn Taste beim Power-Up gedrückt war

    Prinzip siehe Burkhard Kainka http://www.elektronik-labor.de/AVR/Touch.html

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

    - Open Source -


Funktion

    Am Touchpin Tp können unterschiedlich große kapazitive Berührungsflächen angeschlossen werden.
    Ist eine geeignete Anordnung gefunden, wird der Sensor durch einmaligem Druck auf die
    angeschlossene Taste BTN1 kalibriert. Der gemessene Ist-Wert wird als neuer Soll-Wert
    im EEPROM gespeichert und beim nächsten Hochfahren der Betriebsspannung automatisch geladen.

    Auf dem UpinW erfolgt UART-Ausgabe mit 9600 Baud z.B. für
    serielles LCD "Sparcfun ADM1602U-NSW-FBS"

    In einer Anwendung als Füllstandsensor wurden folgende Werte ermittelt:
    - Plast-Behälter Wandstärke 3 mm, Material PP05 (Polypropylen)
    - Elektrodenfläche 120 x 50 mm an Seite aufgeklebt, Kapazität gegen Wasser 37pF
    - Genauigkeit der Messung einer Füllstandsmarke etwa +/- 10 mm

    Zu beachten ist eine hohe Senstivität nach rückwärts. Das System reagiert
    auf eine Hand bereits im Abstand von etwa 3...5 cm. Auch wird der Sensor
    durch metallische Kabelkanäle/Netzkabel im Abstand bis etwa 30 cm gestört.
```

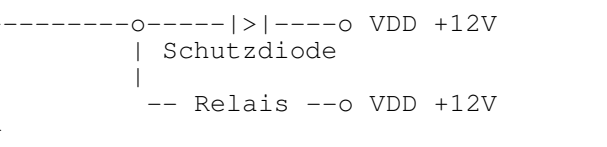
```
Pinout ATtiny13  (PDIP_8 und SOIC_20SU)
                 --------
    /RESET, PB5 |1  `´  8| VCC
            PB3 |2      7| PB2, SCK
            PB4 |3      6| PB1, MISO
            GND |4      5| PB0, MOSI
                 --------
    AVR Studio 4.19: Set controller in "Project -> Configuration options"


Prinzipbild

    µC: Messpin  Mp -----
                         |
                        ---
                        ---  Cm - Mess-Kondensator 4.7nF
                         |
    µC: Touchpin Tp -----o----- R ---- Touchpad
                               1kQ        |
                                         ---
                                         --- Ct - Touch-Kondensator (variiert)
                                          |
                                         --- GND
Funktionsprinzip
    Der Sensor arbeitet als Ladungspumpe mit kapazitivem Spannungsteiler. Dabei
    wird ein großer Kondensator Cm durch viele Ladungen eines kleinen Kondensators Ct
    geladen.
    Um die Funktion zu verstehen, setzen wir voraus, Cm sei initial entladen,
    Mp und Tp haben dann identisches Potential.
    1) Abwechselnd werden Messpin Mp bzw. Touchpin Tp an VCC bzw. GND geschaltet.
    Ist Tp an GND angeschlossen, ist Mp frei und umgekehrt. Somit fließt keinerlei Strom durch Cm.
    Wirkt nun zusätzlich eine Berührungskapazität Ct an Tp gegen GND, entsteht ein
    Spannungsteiler aus Cm und Ct im Verhältnis von ca. 4,7 nF / 25 pF = 188,
    wenn das Messpin hochgezogen wird.
    Schutzwiderstand R sei vernachlässigbar klein (1kQ). Er dient nur als Dämpfung gegen HF.
    Cm wird allmählich um die Ladung von Ct aufgeladen, Ct wird um dieselbe Ladung geladen,
    nur mit entgegengesetztem Vorzeichen.
    Im Folgezyklus wird Tp wieder an GND gelegt, um Ct zu entladen, dann geht es
    wieder zu Punkt 1)
    Nach Cm/Ct Zyklen sollte Cm nennenswert geladen sein.
    Ist Tp gerade nach GND gelegt, wird die Ladung von Cm an Mp gemessen. Wenn diese größer ist,
    als eine Schwelle, dann ist ein Berührung erkannt, die Schleife wird verlassen.
    Der Schleifenzähler (Variable "ist") ist Maß der Kapazität.
    Knackpunkt aber ist eine Kalibrierung, da das Tauchpad eine sehr variable, parasitäre
    Kapazität darstellt. Beim Drücken der angeschlossenen Taste ("Sollwert") wird der letzte,
    gemessene Istwert als neuer Sollwert gespeichert - natürlich im EEPROM, damit der µC
    damit wieder bootet.
```

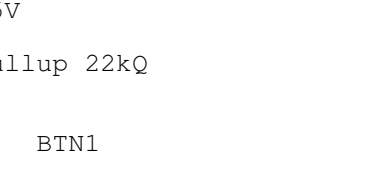
Prozessor: ATMEL ATtiny13 mit 5V und 9,6 MHz

Relaisausgang



LED über Q3 oder Vorwiderstand > 1.8 kQ anschliessen,
um MISO (AVR-ISP) nicht kurzzuschliessen

Taster zur Kalibrierung



Regler:
TS78L05 5V/100mA

Ports:

Port	Funktion
PB0	Relaisausgang: Gate des Transistors Q3 oder LED mit Vorwiderstand
PB1	Sollwert-Taster BTN1
PB2	UART-Tx
PB3	Messpin Mp ---\ Cm
PB4	Touchpin Tp ---/ Cm

Fuses bitte prüfen:
BODLEVEL = 4.3V
CKDIV8 nicht gesetzt --> für 9600 Baud wichtig (default ist gesetzt)
SUT_CKSEL = 9.6 MHz --> für 9600 Baud
HIGH 0xF9
LOW 0x7A

Die Register in den Funktionen EEPROM_write(), EEPROM_read(), StartTimer() und StopTimer()
sowie folgende Werte sind an Leiterkarte und Prozessor anzupassen: ****************/

```
    // Controller
    #define F_CPU 9600000           // interner RC-Oszillator
    // #define __AVR_ATtiny13__     // entfällt bei AVR-Studio4

    // LED oder Relaisausgang (high = on, low = off)
    #define RelPin  PB0     // Relais-Pin
    #define RelPort PORTB   // Relais-Port
    #define RelDir  DDRB    // Relais-Pindirection Output

    // Taste BTN1 zur Schwellwerteinstellung
    #define BtnPin  PB1     // Pin
    #define BtnInport   PINB    // Taste zur Festlegung der Schwelle
    #define BtnDir  DDRB    // pull-up einschalten (unnütz, da Pups gesperrt werden)

    // UART write
    #define UpinW           PB2     // write pin
    #define ComPortOut      PORTB   // output port
    #define ComPortDir      DDRB    // port direction
    #define WaitOneBit  125     // timer 2/2 = one bittime for UART

    // Qtouch-Sensor
    #define Mp      PB3     // Messpin
    #define Tp      PB4     // Touchpin
    #define Pout    PORTB   // Port-Out für Mp, Tp
    #define Pdir    DDRB    // Port-Direction für Mp, Tp
    #define Pinp    PINB    // Port-Input für Mp, Tp

    // EEPROM-Speicherplatz für den Sollwert "soll"
    #define EEadr   11      // ein Byte zwischen 0...63

/*** Ende der Anpassung ************************************************************

    AVR Memory Usage
    ----------------
    Device: attiny13
    Optimization: -0s
    Program:     982 bytes (95.9% Full)
    (.text + .data + .bootloader)
    Data:         35 bytes (54.7% Full)
    (.data + .bss + .noinit)
    Build succeeded with 0 Warnings...
    */

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

```
// disable all pullups
#define DisablePullups  bitset(MCUCR,PUD)   // MCUCR |= (1 << PUD);

// relais or led
#define SetRelPin       bitset(RelDir,RelPin)   // Led oder Relais hier über Transistor anschliessen
#define Relein          bitset(RelPort,RelPin)  // high ist ein
#define Relaus          bitclr(RelPort,RelPin)  // low ist aus

// push-button BTN1
#define SetButton   {bitclr(BtnDir, BtnPin); bitset(BtnInport, BtnPin);}    // input mit pull-up

// charge transfer
#define Mp2vcc  {bitclr(Pdir,Tp); bitset(Pout,Mp); bitset(Pdir,Mp);}    // Mp nach VCC ziehen, Tp offen
#define Tp2gnd  {bitclr(Pdir,Mp); bitclr(Pout,Tp); bitset(Pdir,Tp);}    // Tp nach GND ziehen, Mp offen
#define Open    {bitclr(Pdir,Mp); bitclr(Pdir,Tp);}                     // Mp und Tp öffnen
#define DisCharge   {bitclr(Pout,Tp); bitset(Pdir,Tp);  bitclr(Pout,Mp); bitset(Pdir,Mp);}  // Cm entladen

// soft-uart
#define ComPortInit {bitset(ComPortDir,UpinW); bitset(ComPortOut,UpinW);} // UART-Pin: Out, High
#define SendL       {bitclr(ComPortOut,UpinW);}                         // send bit = low
#define SendH       {bitset(ComPortOut,UpinW);}                         // send bit = high

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

// folgendes Zubehör wird benötigt
#include <stdint.h>
#include <stdlib.h>         // itoa
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


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

void EEPROM_write(uint8_t addr, uint8_t data){  // write one byte
    // function-source: Atmel-manual ATtiny13 p.18
    // Wait for completion of previous write
    while(EECR & (1<<EEPE))     // warte
    ;   // das ist kein Schreibfehler
    // Set Programming mode
    EECR |= (0<<EEPM1)|(0>>EEPM0);
    // Set up address and data registers
    EEARL = addr;
    EEDR = data;
    // Write logical one to EEMPE
    EECR |= (1<<EEMPE);
```

```
    // Start eeprom write by setting EEPE
    EECR |= (1<<EEPE);
}

uint8_t EEPROM_read(uint8_t addr){              // read one byte
    // function-source: Atmel-manual ATtiny13 p.19
    // Wait for completion of previous write
    while(EECR & (1<<EEPE))     // warte
    ;   // das ist kein Schreibfehler
    // Set up address register
    EEARL = addr;
    // Start eeprom read by writing EERE
    EECR |= (1<<EERE);
    // Return data from data register
    return EEDR;
}


/*** soft UART ******************************************************************/
// wir erzeugen folgenden UART Code:

    // H ------- S  -0- -1- -2- -3- -4- -5- -6- -7- -E- -E- ---------
    // L        --- --- --- --- --- --- --- --- ---

    // S: Startbit (low), E: Stopbit (high, 2x), 0...7 Databits, LSB first

    // Wir senden mit 9600 Baud d.h. 104 µs pro Bit
    // Der Prozessor läuft mit 9,6 MHz (interner Oszillator)
    // Der Timer hat insgesamt einen Teiler von
    // 9.600.000 Hz / 9600 Baud = 1000 zu realisieren.
    // Schalten wire einen Vorteiler von 8 ein, folgt ein einzustellender Timerwert
    // von 1000 / 8 = 125 für den Bittakt.

volatile uint8_t nextbit = 0;   // timer signal to send next bit

// Das Timer-Interrupt gibt den Bittakt vor:
ISR(TIM0_COMPA_vect){       // interrupt if timer compare match A, p.44
    nextbit = 1;            // write/read next bit
}

void startTimer(uint8_t time2wait){ // UART bit clock ATtiny13, see p.64 ff
    // timer config (prescal 8; stop at 125 for 1 bitclock or 187 for 1.5 bitclocks
    TCCR0B |= (1<<CS01);    // prescaler division by 8, p.72
    TCCR0A  = (1<<WGM01);   // set CTC-mode für output compare match, p.69
    TCNT0 = 0;              // reset timer value
    OCR0A   = time2wait - 1;    // set output compare match register, p.74
    TIMSK0  = (1<<OCIE0A);  // enable output compare A interrupt, p.74
    sei();
```

```
}

void stopTimer(){           // UART bit clock ATtiny13, see p.64 ff
    cli();
    TIMSK0 = 0;
}

void wait4clk(){            // wait for timer signal nextbit = 1
    while (!nextbit)        // polling UART
    ; nextbit = 0;          // reset nextbit
}

void sendc(uint8_t value){  // send character to soft UART Tx
    // non-inverted - UART-style
    uint8_t i;

    // start timer with bit clock
    startTimer(WaitOneBit);

    // send startbit
    SendL;                      // send startbit = low

    wait4clk();                 // wait for next clock

    // send 8 bits of value
    for (i=0; i<8; i++)         // send all bits of value
    {
        if ((value & 0x01))             // if last bit is one
            { SendH; }  // send one
        else                            // last bit is zero
            { SendL; }  // send zero

        value = value >> 1;     // bit shift right (LSB was first)
        wait4clk();             // wait for next clock
    }

    // send 1 1/2 stopbits
    // startTimer(Wait3halfbit);    // 1.5 bit times

    // send 2 stop bits (lieber bissel mehr)
    SendH;  // send stopbit = high
    wait4clk();
    wait4clk();

    // finish
    stopTimer();
}
```

```
/*** serial LCD: Sparkfun serLCD v2.5 ****************************************/

// serLCD.c and serLCD.h see http://playground.arduino.cc/Code/SerLCD

// LCD commands, see http://www.sparkfun.com/datasheets/LCD/SerLCD_V2_5.PDF
#define clear           0x01    // Display löschen
#define home            0x02    // Cursor an den Anfang
#define selectLineOne   0x80    // 128, Zeile 1, Position 0
#define selectLineTwo   0xc0    // 192, Zeile 2, Position 0

void LCDcmd(uint8_t value){     // send any command to LCD
    sendc(0xFE);    // LCD_SENDCOMMAND
    sendc(value);   // command value
    _delay_us(10);
// Verwendung:
//  LCDcmd(home);
//  LCDcmd(clear);
//  LCDcmd(selectLineOne);
//  LCDcmd(selectLineTwo);
}

void LCDoutput(char name, uint8_t messwert){    // Ausgabe auf serial LCD
    // messwert "ist" soll dezimal ausgegeben werden als 3 Byte-Array erg[0]...[3]
    char erg[4];        // bei itoa ist Ergebnis null-terminiert, d.h. erg[4] = '\0'
    itoa(messwert, erg, 10);    //  int to ascii, itoa(int, chararray, radix)
    if ((erg[2] == '\0')) erg[2] = ' '; // itoa String-Terminator gegen Space ersetzen
    if ((erg[3] == '\0')) erg[3] = ' ';

    sendc(' ');
    sendc(name);    // ein Byte, i oder s
    sendc('=');
    sendc(erg[0]);
    sendc(erg[1]);
    sendc(erg[2]);
    sendc(' ');
}

void fill8spaces(){ // LCD Macke überschreiben
    uint8_t i = 0;
    for (i=0; i<8; i++){
        sendc(' ');     // 8 spaces
    }
}

void sendScreen(char scrn1[16], char scrn2[16]){    // to serialLCD
    uint8_t i;                  //  see Sparkfun serLCD manual
    LCDcmd(home);
```

```
    LCDcmd(selectLineOne);
    for (i=0; i<16; i++){
        if (scrn1[i] == '\0'){  // String Terminator gegen Space ersetzen
            scrn1[i]  = ' ';}
        sendc(scrn1[i]);
    }

    LCDcmd(selectLineTwo);
    for (i=0; i<16; i++){
        if (scrn2[i] == '\0'){
            scrn2[i]  = ' ';}
        sendc(scrn2[i]);
    }
}

void burnSplashScreen(){    // for serialLCD
// create your own splash screen, use this command to burn
    sendc(0x7c);    // 124  see Sparkfun serLCD v2.5 manual
    sendc('\n');    // CTRL-J = 0x0a = new line
    _delay_ms(500);
}



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


int main(void){

    // inits
    uint8_t ist, soll;  // reicht aus, nicht größer als 255

    SetButton;          // pullup muß extern sein, siehe folgende Zeile
    DisablePullups;     // disable all pullups, sonst geht Ladung verloren
    SetRelPin;          // Relais, LED
    Relaus;             // aus = 0
    ComPortInit;        // set UART pin: out, high
    LCDcmd(home);       // reset LCD
    LCDcmd(clear);      // reset LCD

    // Soll Sparcfun Splash Screen neu gebrannt werden?
    if (!(BtnInport & (1 << BtnPin))){  // wenn Taste gedrückt war (auf GND)
        _delay_ms(2000);                // auf LCD lange genug warten
        char *scrn1 = " Q-Sensor 9/2013";   // erste Zeile
        char *scrn2 = "  heinz@gfai.de ";   // zweite Zeile
        sendScreen(scrn1, scrn2);   // load screen
```

```
        burnSplashScreen();         // execute burning
    }

    soll = EEPROM_read(EEadr);  // Sollwert laden
    ist  =  0;          // Istzahl der Umladungen
    DisCharge;          // definierten Zustand schaffen, dazu Cm entladen
    _delay_ms(200);     // lange genug warten
    Tp2gnd;             // Tp laden auf GND -> zum Messen erforderlich
    // Mp ist jetzt low, da Cm entladen wurde und Tp auf GND liegt


  while(1){             // Schleifhaupte

    // Cm iterativ aufladen, dabei Variable "ist" hochzählen
    while(!(Pinp & (1 << Mp))){     // solange Mp low ist
        // toggeln, insgesamt Cm / Ct_min = 4,7nF/20pF < 235 steps
        // Cm mit Qt laden
        Open;           // Tp und Mp öffnen
        Mp2vcc;         // Mp laden auf VCC
        _delay_us(1);   // kurz warten

        // Ct entladen
        Open;           // Tp und Mp öffnen
        Tp2gnd;         // Tp laden auf GND, auch zum Messen erforderlich
        _delay_us(1);   // kurz warten
        ist++;          // hochzählen
    }
    // Mp wurde als high erkannt
    // Relais/LED schalten
    if (ist < soll)     {Relein;}
    else                {Relaus;}

    // Messwerte auf Serial-LCD ausgeben
    LCDcmd(home);           // Cursor auf Zeile 1, Pos.0

    LCDcmd(selectLineOne);
    LCDoutput('i', ist);    // Ausgabe des Istwerts
    if (ist < soll)     {sendc('H');}   // high capacity ~ low i
    else                {sendc(' ');}
    fill8spaces();      // remove a LCD-bug, send 8 spaces

    LCDcmd(selectLineTwo);
    LCDoutput('s', soll);   // Ausgabe des Sollwerts
    if (ist > soll)     {sendc('L');}   // low capacity ~ high i
    else                {sendc(' ');}
    fill8spaces();      // remove a LCD-bug, send 8 spaces
```

```
    _delay_ms(100);     // Relaisklappern vermeiden

    // Abfrage der Taste: Soll ein neuer Schwellwert eingestellt werden?
    if (!(BtnInport & (1 << BtnPin))){  // wenn Taste gedrückt war (auf GND)
        soll = ist - 1;         // neuer Sollwert ist um Eins kleiner als Istwert
        // ins EEPROM damit
        EEPROM_write(EEadr, soll);  // "soll"-Byte speichern
        _delay_ms(500);         // sichtbar lange Pause nach dem Brennen
    }

    // alles von vorn, neuer Zyklus
    ist = 0;            // Zähler reset
    DisCharge;          // Messkondensator Cm entladen
    _delay_ms(100);     // für Entladung Zeit lassen

  } // while(1)
}
```