Zylinderkopftemeraturanzeige mittels Ardunio-Microcontroller

Stephan Voß – forum.bulli.org
voss.stephan@googlemail.com

Vorwort:

Dies ist keine genaue Schritt-für-Schriff Bauanleitung denn hierzu müsste man immer zu 100% identische Bauteile verbauen. Dies ist aber grade auf dem sich schnell wandelnden Markt der Arduino-Kompatiblen Bauteile nicht möglich, denn Komponenten wie der z.B. hier verwendete TFT-Bildschirm wird von verschiedensten Herstellern angeboten, welche aber unterschiedliche Bibliotheken vorraussetzen und ggf. Änderungen benötigen. Daher ist alles, was ich hier schreibe als Beispiel zu sehen. Jeder hat die Möglichkeit hier eine eigenen Ideen einzubringen und die Anzeige z.B. durch zusätzliche Sensoren und Anzeigen zu erweitern. (...)

Differentielle Zylinderkopftemperaturüberwachung:

Generell erstmal ein paar Worte, warum die Überwachung der Zylinderkopftemperatur sinnvoll ist: Beim luftgekühlten VW Motor ist der Zylinderkopf die eigentliche Achillesferse. Grade beim Typ4 Motor ist der Rest des Motors eigentlich sehr robust und verzeiht viel. Die Blöcke dieser Motoren halten bei normaler Pflege locker ihre 150.000 km – 200.000 km, wobei sie dann auch am Ende ihrer Lebensdauer sind. Bei den Zylinderköpfen ist dies nicht immer der Fall. Schäden durch Überhitzung zeigen sich am ehesten hier, sei es durch Risse oder durch Undichtigkeiten aufgrund von Verzug oder durchgeblasenen Dichtungen. Auch das eine oder andere Ventil soll schon in der Vergangen abhanden und den Rest des Motors meist mit ins Verderben gerissen haben. Da ich oft und gerne auch längere Touren mit dem Bus unternehme (ist immerhin als Westfalia Camper ein Urlaubsfahrzeug) und dies meist mit langen Autobahnetappen verbunden ist, war es mir wichtig hier ein besonderes Augenmerk drauf legen zu können. Zwar habe ich auch die obligatorische Öltemperaturanzeige, meiner Meinung nach ist aber die Zylinderkopftemperatur deutlich aussagekräftiger da viel dynamischer. So reagiert die Kopftemperatur sehr schnell auf Änderungen im Betriebszustand. Da der Typ4 Motor im T2 standardmäßig mit zwei Einzelvergasern daherkommt, fand ich es wichtig, dass man nicht nur eine Seite des Motors im Auge hat, sondern sowohl links als auch rechts misst. Hierdurch lassen sich Probleme an einem einzelnen Vergaser sehr gut feststellen, denn im Betrieb fällt es tatsächlich gar nicht so sehr auf wenn eine Gemischaufbereitung pltzlich nicht mehr mitspielen will.
Doch wo genau messen? Am verbreitetsten ist die Messung mittels eines Ringes unterhalb der Zündkerze. Vorteil ist, das es hier sehr viele Vergleichswerte gibt, da diese Methode relativ häufig angewendet wird. Nachteil ist, dass der zusätzliche Ring häufig für Undichtigkeiten sorgt und spätestens beim zweiten oder dritten Versuch die Kerzen auszubauen zerstört ist. Ein weiterer relativ häufig verwendete Messpunkt befindet sich beim Typ4 Zylinderkopf am Gewinde für die Befestigungsschrauben der Verblechung. Hier haben US-Fahrzeuge, welche mit einer Einspritzung ausgestattet sind bereits einen Messfühler für die Zylinderkopftemperatur für das Steuergerät. Neue AMC-Zylinderköpfe besitzen die entsprechenden Gewinde und hier habe ich auch meine Messfühler installiert. Bezüglich der maximalen zulässigen Temperatur am Zylinderkopf gibt es so viele unterschiedliche Meinungen und Werte wie es Messpunkte gibt. Im Normalbetrieb messe ich hier Temperaturen zwischen 120 und 150°C, je nach Betriebsbedingungen. Wer schon in den Quellcode geschaut hat wird sehen, dass ab 180°C der Temperaturwert in rot ausgegeben wird. Dies ist meine persönliche Grenze für Dauerbetrieb. Welche Temperaturen normal sind, hängt natürlich auch sehr vom jeweiligen Setup und der Nutzung ab. Viel wichtiger als die eigentliche

Absoluttemperaur ist meiner Meinung nach der Trend. Zeigt sich, dass die Temperatur plötzlich stark ansteigt oder absinkt, sollten im Kopf (des Fahrers) die Alarmglocken läuten, denn dies ist ein untrügliches Zeichen das etwas nicht stimm. Sind die Temperaturen bei euren Motoren dauerhaft etwas höher und niedriger als die, die ich hier für meinen Motor genannt habe, dann ist das halt so. Wenn der Motor sonst gut läuft und das vieleicht auch schon länger, sollte man sich erstmal keine Sorgen machen.
Mit einer differentiellen Anzeige bietet sich die Möglichkeit, dass beide Seiten des Motors gleichzeitig betrachtet werden können und damit eine gewisse Referenz immer vorhanden ist. So kann man immer die andere Seite als Vergleich heranziehen. Bedingt durch den vorgegebenen Punkt am Zylinderkopf (es gibt nur ein Gewinde pro Kopf) messe ich einmal am dritten Zylinder und einmal am zweiten Zylinder. Zwischen diesen beiden Messpunkten können gut- und gerne mal Temperaturunterschiede von 25 °C vorkommen. Bedingt ist dies dadurch, dass der Messpunkt vom dritten Zylinder im Luftstrom von Zylinder vier und damit schon mit heißer Luft eines Zylinders beaufschlagt ist. Der Messpunkt an Zylinder zwei hingegen liegt auf einer Seite, auf die direkt frische Kühlluft geleitet wird. Interessant ist, welche Veränderungen sich bei der Differenz im Betrieb ergeben. Laufen bei der Warmfahrt zunächst die Temperaturen noch sehr synchron, driften sie im weiteren Verlauf immer weiter auseinander, bis sich irgendwann ein stationärer Betrieb einstellt und die Differenz relativ konstant bleibt.

Thermoelemente oder PT100 Widerstände – Warum die hier beschriebene Lösung?

Hier möchte ich zunächst ein paar Worte über die verschiedenen Möglichkeiten verlieren, wie man eine Zylinderkopftemperaturanzeige realiseren kann. In einer ersten Version dieser Anzeige habe ich Thermoelemente verwendet. Hierbei handelt es sich um die einfachste Form von Thermofühlern, denn sie bestehen im Prinzip nur aus zwei verschiedenen Metalldrähten, welche an einem Ende miteinander verschweißt sind. Wird die Verbindungsstelle erwärmt, bildet sich zwischen beiden Drähten eine kleine Spannung, die sogn. Thermospannung. Hierbei handelt es sich nur um wenige mV, sie ist also recht klein und für eine Auswertung wird ist es notwendig diese Spannung durch eine Vertsärkerschaltung zu vergrößern. Im Gegensatz dazu handelt es sich beim PT100 um einen Widerstand, welcher seinen Widerstandswert bei Temperaturändeurngen verändert. Der PT100 hat bei einer Temperatur von 0 °C einen Widerstandwert von 100 Ω, daher der Name. Auch hierfür wird eine Schaltung benötigt, um den Widerstandswert für den Mikrocontroller verständlich zu machen. Für beide Systeme gibt es entsprechende Schaltungen zu kaufen, daher müssen wir uns hier keine Gedanken darüber machen.

Die in der ersten Version der Anzeige verwendeten Thermoelemente (TypK) werden auch bei vielen kommerziellen Anzeigen verwendet, denn sie sind sehr günstig und einfach in der Anwendung. Beim VW T2 bzw. den meisten Fahrzeugen mit Heckmotor ist dies aber aufgrund der langen Leitungen zwischen der Anzeige (üblicherweise im Cockpit) und dem Motor aber etwas problematisch. Da nur wenige mV als Spannungsdifferenz erfasst werden sollen, ist diese Konstellation sehr empfindlich was Übergangswiderstände angeht. Ich hatte damals Stecker im Motorraum vorgesehen, um nicht jedes Mal die Sensoren abschrauben zu müssen, wenn z.B. der Motor ausgebaut werden sollte (was ja zum Glück nicht so häufig vorkommt). Mit dem Resultat, dass mit der ersten (billigen) China-Anzeige keine verlässliche Messung mehr möglich war, da die Stecker zu hohe Widerstände hatten.

Aus der Lust heraus, mal wieder etwas mit Mikrocontrollern zu basteln, hatte ich dann die Idee diese Anzeige grundlegend anders zu gestallten. Um die langen Kabelwege zu überwinden, sollte ein Mikrocontroller in der Nähe des Motors die Messwerte erfassen und dann über eine Datenverbindung diese an einen zweiten Controller zur Anzeige senden. Mit dem Ardunio-System

ist des über eine Serielle-Verbindung tatsächlich relativ einfach zu implementieren. Der Messaufnehmer landete, für möglichst kurze Kabellängen, direkt im Motorraum. Dieses System hat tatsächlich mehrere Jahre funktioniert, es hatte aber auch einige entscheidene Nachteile, welche mich dazu veranlassten nochmal alles gründlich zu verändern. So zeigte sich relativ schnell, dass die verwendeten Messverstärker für Thermoelemente sehr empfindlich auf induziere Spannungen reagierten. Davon gibt es leider im Motorraum, wegen der Zündanlage sehr viel. Außerdem hatte der Mikrocontroller im Motorraum große Probleme mit der Temperatur. Im T2 wird es hinten gut und gerne mal 80 °C was für den Speicher-Eeprom manchmal zu viel wurde und der Mikrocontroller seine programmierung verlor. Entsprechend musste dieser desöfteren neu programmiert werden, was insbesondere im Urlaub irgendwann nervig wird.

Aus diesem Grund musste eine neue Lösung gefunden werden, in der die anfälligen Thermoelemente durch deutlich robustere Sensoren auf Basis eines PT100 ersetzt wurden. Die dazu notwendigen Messumwandler fanden nun ihren Platz im Heck-Schrank des Westfaliaausbaus. Sie geben jeweils eine Spannung zwischen 0-5V für einen Temperaturbereich von 20 – 230 °C aus, welche nun von einem einzelnen Mikrocontroller an der Anzeige ausgewertet und dargestellt wird.

Ardunio-Microkontroller:

Für die Anzeige wurde ein Ardunio-Microcontroller. Vorteil des Ardunio ist die sehr große Verbreitung und die große Menge an mehr oder weniger kompatiblen Zubehörs. So gibt es für fats jedes Problem mit etwas Suchen im Internet eine Lösung und man kann auch als unerfahrener Programmierer hier sehr schnell ein kleines Projekt auf die Beine stellen.
Ein weiterer Vorteil, die Boards gibt es sehr günstig und auch in den verschiedensten Bauformen. Für die Anzeige habe ich die Micro-Bauform gewählt, welche denselben Umfang bietet, aber z.B. nicht mit einem USB-Seriell-Interface ausgestattet ist. D.h. für die Programmierung muss dieses Interfacer temporär angeschlossen werden. Dafür ist die Grundplatine sehr klein und kompakt und für wenige Euros erhältlich.

TFT Display:

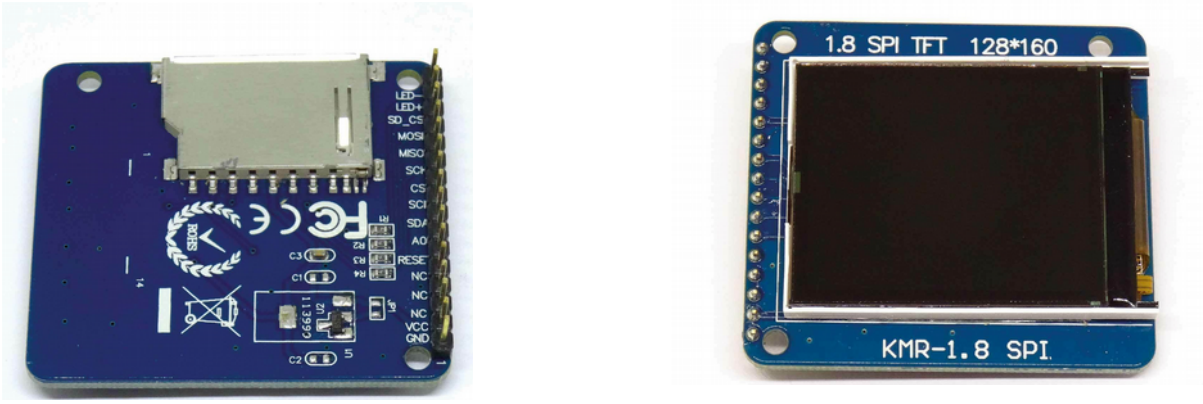


Abb 2.: Das verwendete 1.8Zoll TFT Display. Für die Verwendung mit dem mitgelieferten code bitte auf die Beschriftung “KMR-1.8 SPI” achten. Das Display hat eine Auflösung von 128x160 Pixel.

Als Display habe ich ein 1,8 Zoll TFT-Bildschirm verwendet. Diese gibt es für weniger Euros z.B. bei eBay, jedoch haben diese keine einheitliche Ansteuerung. Meines verwendet einen ILI9163

kompatiblen Chip für die Ansteuerrung des Panels. Es sind aber auch Modelle verfügbar mit anderen Treibern. Wichtig ist, dass man die zum Treiber passende Bibliothek verwendet da ansonsten z.B. die Farben falsch dargestellt werden oder auch gar nichts angezeigt wird. Obwohl das Display mit dem Treiber funktionierte, musste ich die entsprechende Konfigurationsdatei noch anpassen, da zum einen das Bild gespiegelt aber auch die Farben falsch dargestellt wurden da die Werte für Rot, Grün und Blau vertauscht waren. Die entsprechend angepasste Header-Datei findet sich im Anhang.

Ein weiterer wichtiger Punkt, die verschiedenen Displays unterscheiden sich auch hinsichtlich der Spannung der Logik-Ein- und Ausgänge. Der Arduino arbeitet standardmäßig mit 5V-Logik. Das von mit verwendete Display tolleriert (entgegen der Angeben in der Beschreibung) nur 3.3V-Logik. Entsprechend müssen die Signale vom Ardunio über einen Spannungsteiler bestehend aus 1 kΩ und 2 kΩ Widerständen auf 3.3V gebracht werden.
Gleiches gilt auch für die Hintergrundbeleuchtung, wobei es hier ausreicht die Versorgung über einen in Reihe geschalteten 22 Ω Widerstand auf den gewünschten Wert zu bringen. Alternativ kann hier auch ein entsprechendes Potentionmeter eingesetzt werden, um die Display-Helligkeit nach belieben zu variieren. Der am Display befindliche SD-Kartenleser wurde nicht genutzt. Es ist aber einfach möglich hier eine Speicherung der Messwerte zu realisieren.

PT100 Messumformer:

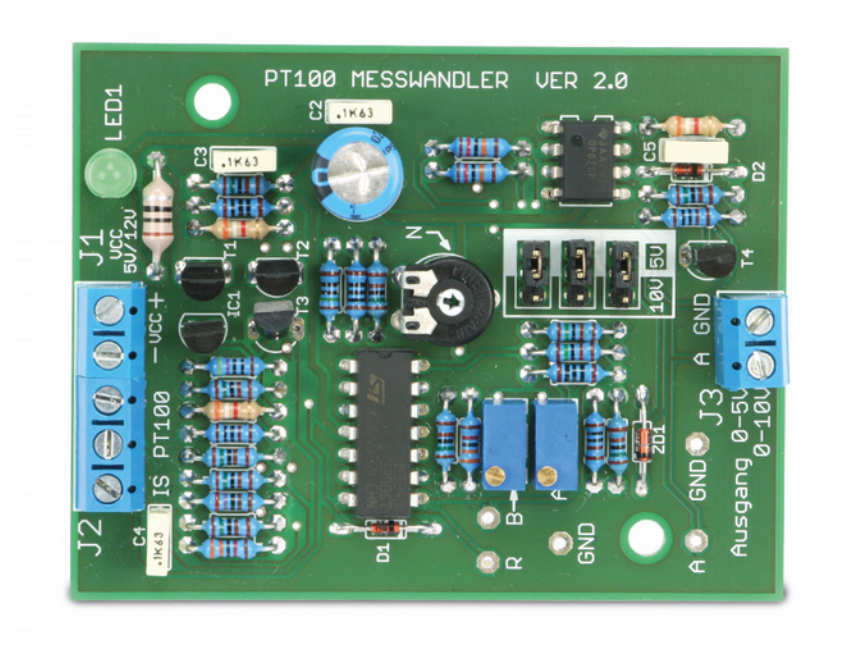


Abb. 3: PT100 Messumformerbausatz von Pollin.

Die verwendeten PT100 Messumformer sind z.B. über Pollin mit der Bestellnummer 810272 als Bausatz erhältlich. Diese Messumformer bieten eine sehr gute Glättung des Signals und zeigen sich als sehr störungsunempfindlich im laufenden Betrieb. Der Zusammenbau ist relativ einfach und die beigelegte Anleitung gut. Für einen der halbwegs geübt ist im Umgang mit dem Lötkolben ist das Board nach etwa einer Stunde fertig. Wichtig ist, dass nach Fertigstellung zunächst eine Kalibrierung auf den entsprechenden Messbereich erfolgen muss. Dieser Schritt ist in der beiliegenden Anleitung detailiert beschrieben. Grob gesagt, stellt man den Messbereich ein, den man mit einem Spannungsbereich 0-5V abdecken will. Die Diskretisierung der Spannung und damit

die Messgenauigkeit wird über den verwendeten Microkontroller bestimmt. Da es sich beim verwendeten Ardunio um einen 8bit µC handelt, können wir also unseren gewählten Messbereich auf 256 Messschritte aufteilen. Es macht also keinen Sinn, dass wir einen möglichst großen Temperaturbereich abdecken. Für die Zylinderkopftemperatur habe ich einen Messbereich von 20°C bis 230°C gewählt, wobei 20°C in etwa 0 V und 230 V entspricht. Darüber und darunter werden keine weiteren Temperaturen vom Microkontroller mehr erfasst. Da wir für die Kalibrierung logischerweise den Sensor entsprechend auf die Temperaturwerte bringen müssen (alternativ kann man auch anhand einer Widerstandstabelle die korrespondierenden Widerstände anschließen) macht es durchaus Sinn einen Messbereich zu wählen den man mit Hausmitteln noch erreichen kann. 20°C ist Raumtemperatur und 230°C entspricht der höchsten Stellung eines Backofens in der Küche. Beide Temperaturen wurden mit einem anderen (geeichten) Messgerät überprüft. Vertständlicherweise werden beide Messumformer gleich und auch gleichzeit kalibriert, so das beide dieselben Spannungen für die jeweiligen Temperaturen ausgeben. Es wird in der Anleitung eine Grob- und eine Feinjustierung beschrieben. Mir reichte bisher die Grobjustierung.

Gesamt-Verschaltungsplan:

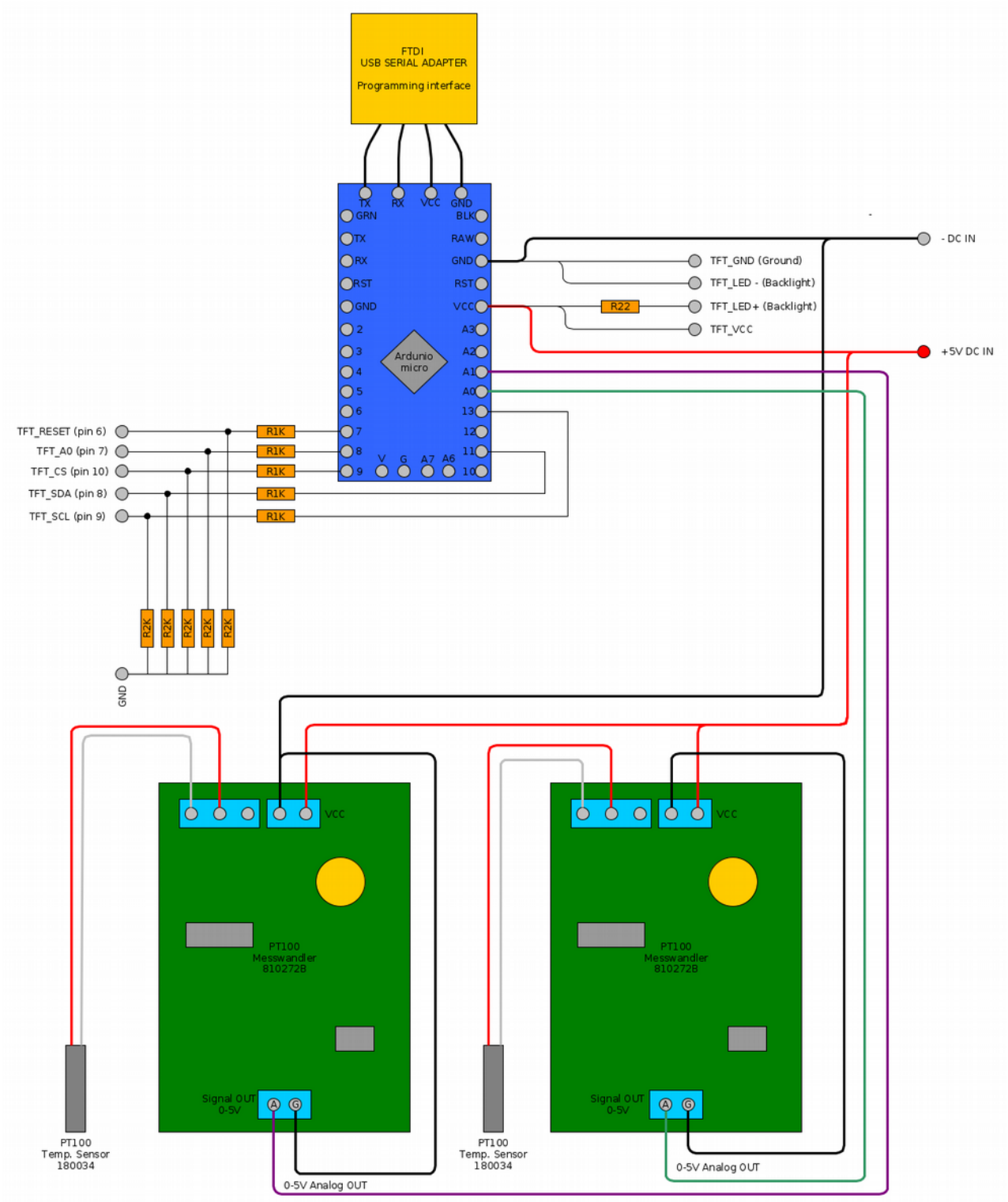


Abb.3: Verschaltung des Messsystems. Die Display-Ports sind durch die grauen Anschlüsse gekennzeichnet und entsprechend beschrieben. In grün, die beiden PT100 Messumformer.

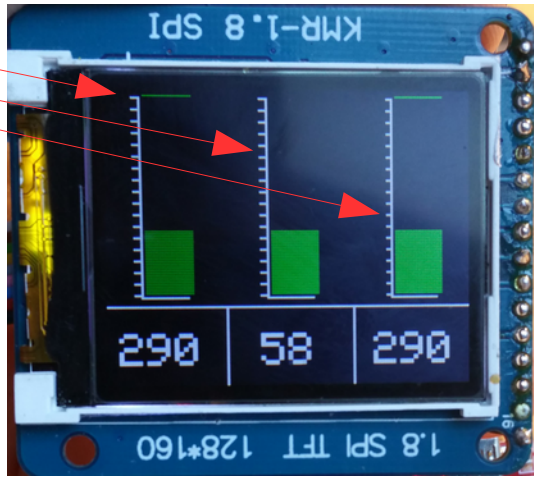
Programmierung:

```
#include <TFT_ILI9163.h> // Graphics and font library for ILI9163 driver chip
#include <SPI.h>

TFT_ILI9163 myGLCD = TFT_ILI9163(); // Invoke library, pins defined in
User_Setup.h

#define TFT_GREY 0x7BEF
#define TFT_W 160 // display width in px
#define TFT_H 128 // display height in px

// integer definition
int analogPin1 = A0;
int analogPin2 = A1;
int temp1 = 0;
int temp2 = 0;
int bar1 = 0;
int bar2 = 0;
int difftemp = 0;
int bardiff = 0;
int temp1_old;
int temp2_old;
int difftemp_old;

// double definition
double Temp_11; // zyl 1 -- meas 1
double Temp_12; // zyl 1 -- meas 2
double Temp_21; // zyl 2 -- meas 1double Drift_1 = 0.0;
double Drift_1 = 0.0;
double Drift_2 = 0.0;
double Drift_diff = 0.0;
double Temp_22; // zyl 2 -- meas 2

void setup()
{
  Serial.begin(9600);              //  setup serial

//resetting display
  pinMode(7, OUTPUT);
  digitalWrite(7, LOW);
  delay(10);
  digitalWrite(7, HIGH);

// Setup the LCD
  myGLCD.init();
  myGLCD.setRotation(1);  //Rotation/Spiegeln des Bildes vom TFT
  delay(1000);

// Splash-Screen (also avoids graphical errors)
  myGLCD.fillRect(0,0,160,128,TFT_BLACK);
  myGLCD.setTextColor(TFT_WHITE,TFT_BLACK);
  myGLCD.setCursor(0,0);
  myGLCD.println("Cylinder head temperature v.2.2 TFT");
  myGLCD.println("P100 Sensors");
  delay(500);
  myGLCD.setCursor(0,40);
```

```
  myGLCD.println("(c)2018 by Stephan Voss");
  delay(2000);
  myGLCD.fillRect(0,0,160,128,TFT_BLACK);
```

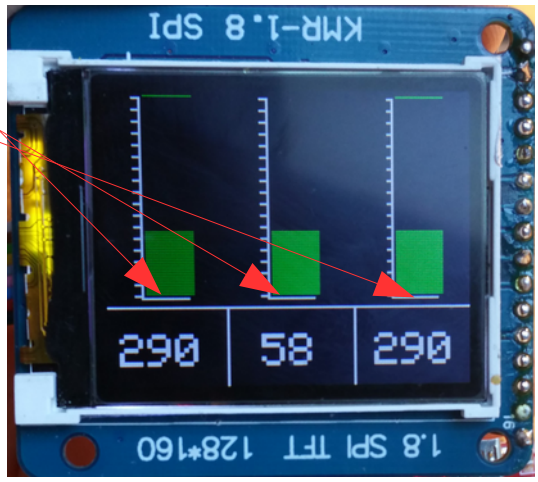
Vertikale Linien:

```
// draw vertical line
  myGLCD.drawLine(15,3,15,89,TFT_WHITE);
  myGLCD.drawLine(70,3,70,89,TFT_WHITE);
  myGLCD.drawLine(125,3,125,89,TFT_WHITE);
```



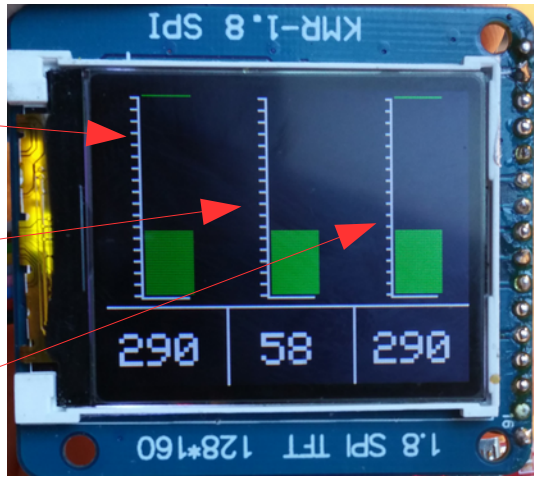
Horizontale Linen:

```
// draw horizontal lines
  myGLCD.drawLine(15,89,35,89,TFT_WHITE);
  myGLCD.drawLine(70,89,90,89,TFT_WHITE);
  myGLCD.drawLine(125,89,145,89,TFT_WHITE);
```

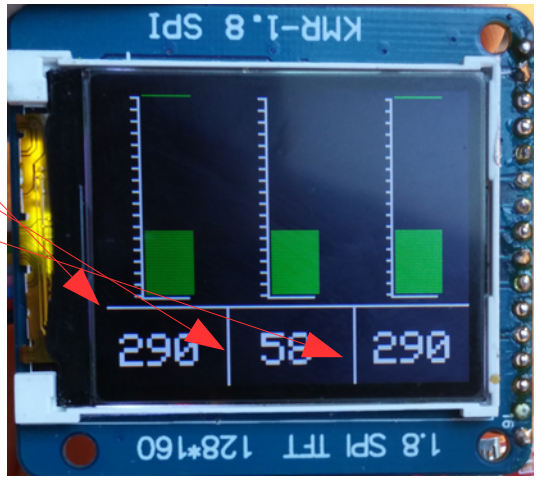


Skaleneinteilungen:

```
// draw ticks
  for(int i=3; i<89; i+=5)
  {
    myGLCD.drawLine(12,i,14,i,TFT_WHITE);
  }
  for(int i=3; i<89; i+=5)
  {
    myGLCD.drawLine(67,i,69,i,TFT_WHITE);
  }
  for(int i=3; i<89; i+=5)
  {
    myGLCD.drawLine(123,i,125,i,TFT_WHITE);
  }
```



```
// draw text fields

myGLCD.drawLine(0,93,159,93,TFT_WHITE);

myGLCD.drawLine(52,94,52,127,TFT_WHITE);

myGLCD.drawLine(108,94,108,127,TFT_WHITE);
```



Nachfolgender Code wird innerhalb einer Endlosschleife ausgeführt und besteht im Wesentlichen aus der eigentlichen Messung, der Bestimmung der Differenztemperatur und der Visualisierung in Form der Zahlen und der Balken:

```
}
void loop() {
  // put your main code here, to run repeatedly
```

Die beiden Messumformer geben jeweils Spannungen von 0 - 4.9 V aus. Die Funktion “analogRead()” gibt hierfür Werte von 1 – 1024 aus (10bit). Die Umrechnung in Volt erfolgt durch U_{Volt} = (4.9/1024)*analogRead. Die Umrechnung von Spannung erfolgt mittels einer linearen Funktion $T_{meas}=U_{volt}(T)*U_{base} + T_{min}$
Wobei $U_{volt}(T)$ die Temperaturabhängige Spannung ist, U_{base} die bei der Kalibrierung festgestelle Steigung der Spannung. T_{min} ist die Minimaltemperatur die anzeigt werden kann. In diesem Fall 20°C. Für unsere Zwecke reicht dies als min. Temperatur. Man wird merken, dass der Motor recht schnell “auf Temperatur kommt”.

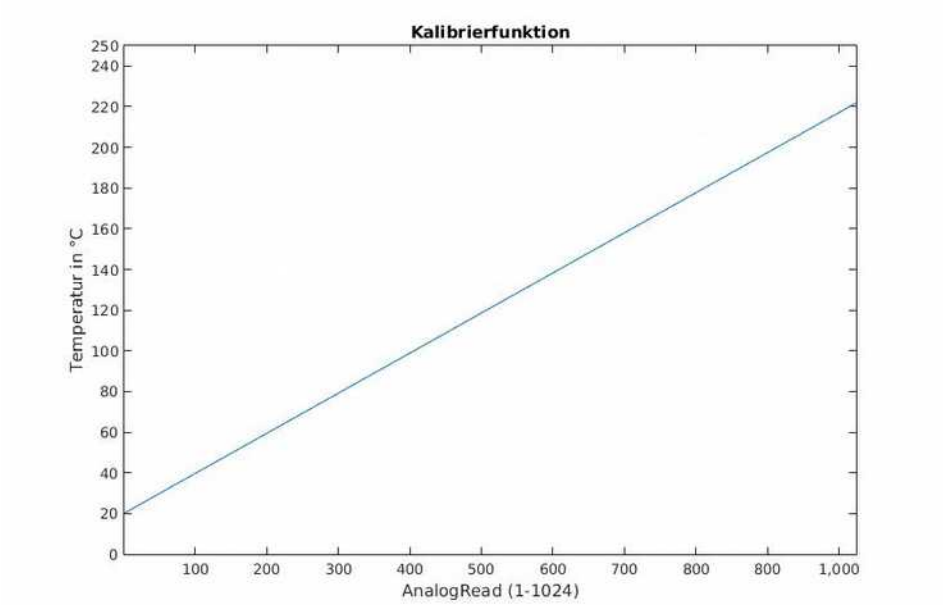


Abb.4: Kalibrierfunktion. Angezeigte Temperatur in Abhängigkei des Analog-Eingangs.

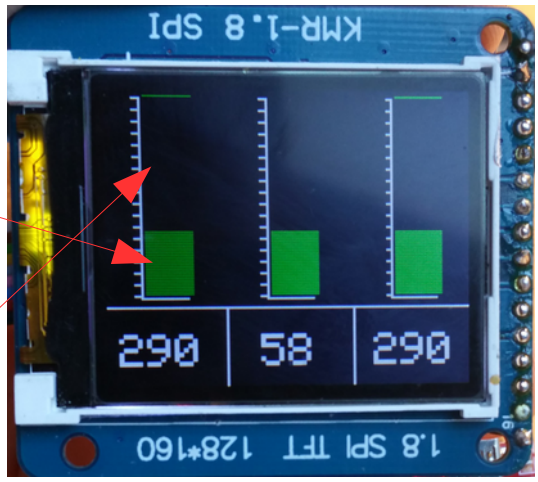
```
// Measuring temperature from analog input (A0) and (A1)
// +20: zero point for 0V input
// 41.20: Degree Celsius per Volt

Temp_11 = 41.20 * (4.90/1024.00) * analogRead(A0) + 20.00;
Temp_21 = 41.20 * (4.90/1024.00) * analogRead(A1) + 20.00;

// get temperature difference
temp1 = Temp_11;
temp2 = Temp_21;

difftemp = abs(temp1 - temp2);

// draw bars from analog values
      bar1 = 85 - temp1 / 2.5;
bar1 = abs(bar1);

bar2 = 85 - temp2 / 2.5;
bar2 = abs(bar2);

bardiff = 85 - difftemp;
bardiff = abs(bardiff);
```

Über die Serielle Schnittstelle können die gemessenen Temperaturen abgerufen werden. Diese Funktion wird im eingebauten Zustand nicht genutzt es kann hier aber eine externe Speicherung z.B. über einen RS232 Messaufnehmer oder einen PC realisiert werden.

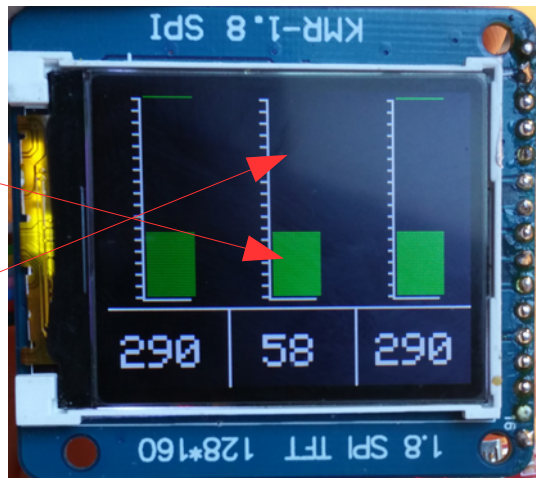
```
// sending serial output (for debugging)
Serial.println(temp1);
Serial.println(temp2);
Serial.println(difftemp);
```

Die Balken bestehen aus insgesamt 87 horizontalen Linien mit einer Höhe von einem Pixel und einer Breise von 20 Pixeln. Je nachdem wie hoch die Temperatur ist, werden Teile der Balken entweder grün oder rot dargestellt. Der rote Bereich kann durch die Veränderung des Wertes in der for-Schleife eingestellt werden. ACHTUNG: i- wird von oben nach unten gezählt!

```
for (int i = 87; i>bar1; i-=1 )
{
  if(i<25){
    myGLCD.drawLine(17,i,37,i,TFT_RED);}
  else{
     myGLCD.drawLine(17,i,37,i,TFT_DARKGREEN);
}
  }
for (int i = 0; i<bar1; i+=1){
      myGLCD.drawLine(17,i,37,i,TFT_BLACK);
      }
```



```
for (int i = 87; i>bardiff; i-=1 )
{
  if(i<65){
    myGLCD.drawLine(72,i,92,i,TFT_RED);}
  else{
     myGLCD.drawLine(72,i,92,i,TFT_DARKGREEN);
}
  }
for (int i = 0; i<bardiff; i+=1){
  myGLCD.drawLine(72,i,92,i,TFT_BLACK);
  }
```




```
for (int i = 87; i>bar2; i-=1 )
{
  if(i<25){
    myGLCD.drawLine(127,i,147,i,TFT_RED);}
  else{
     myGLCD.drawLine(127,i,147,i,TFT_DARKGREEN
);}
  }

for (int i = 0; i<bar2; i+=1){
  myGLCD.drawLine(127,i,147,i,TFT_BLACK);
  }
```

Ähnlich den Balken, werden die Zahlen aktualisiert indem diese einfach nochmal in schwarz dargestellt werden und somit “verschwinden”. Erst dann werden die neuen Werte dargestellt.

```
// delete old values from text fields
  myGLCD.setTextColor(TFT_BLACK,TFT_BLACK);
  myGLCD.setCursor(5,105);
  myGLCD.setTextSize(2);
  myGLCD.print(temp1_old);

  myGLCD.setCursor(80-13,105);
  myGLCD.setTextSize(2);
  myGLCD.print(difftemp_old);

  myGLCD.setCursor(117,105);
  myGLCD.setTextSize(2);
  myGLCD.print(temp2_old);
```

Die Temperaturwerte werden in die entsprechenden Felder geschrieben. Dargestellt wird weiße Schrift auf schwarzem Grund. Ab einer Temperatur von 180 °C werden die Zahlenwerte in rot auf schwarzem Grund dargestellt.

```
// write values in text fields
if (temp1 < 180){
  myGLCD.setTextColor(TFT_WHITE,TFT_BLACK);
  }
  else{
    myGLCD.setTextColor(TFT_RED,TFT_BLACK);
  }
  myGLCD.setCursor(5,105);
  myGLCD.setTextSize(2);
  myGLCD.print(temp1);

if (difftemp < 20){
  myGLCD.setTextColor(TFT_WHITE,TFT_BLACK);
  }
  else{
    myGLCD.setTextColor(TFT_RED,TFT_BLACK);
  }
  myGLCD.setCursor(80-13,105);
  myGLCD.setTextSize(2);
  myGLCD.print(difftemp);

if (temp2 < 180){
  myGLCD.setTextColor(TFT_WHITE,TFT_BLACK);
  }
  else{
    myGLCD.setTextColor(TFT_RED,TFT_BLACK);
  }
  myGLCD.setCursor(117,105);
  myGLCD.setTextSize(2);
  myGLCD.print(temp2);

  delay(700);
```

Die zuvor dargestellten Werte werden nun in eine neue Variable übertragen und die alten im nächsten Durchgang überschrieben. Mithilfe der hier zuvor gespeicherten Werte werden die Textfelder überschrieben.

```
  temp1_old = temp1;
  temp2_old = temp2;
  difftemp_old = difftemp;

}
```