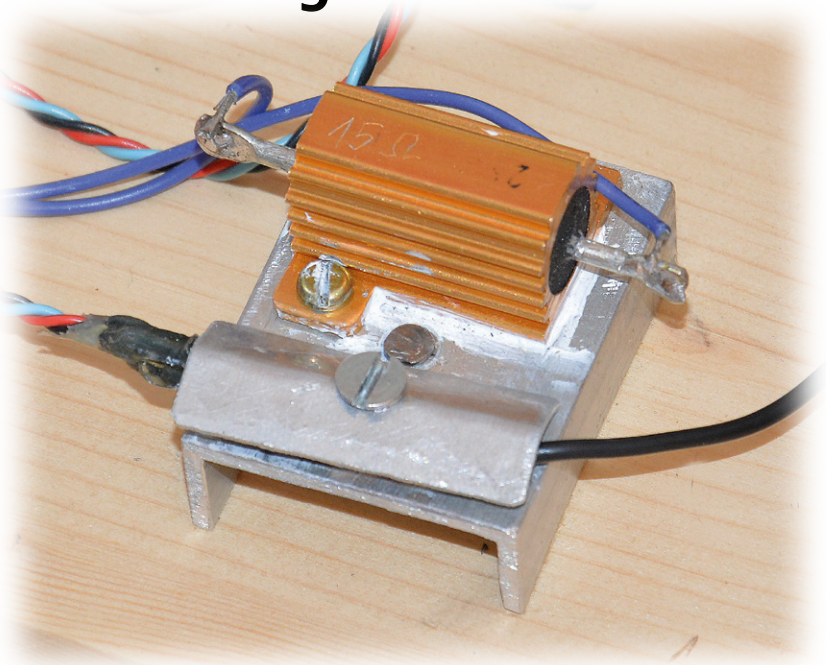
# Tester für Temperatur-Sensoren

## Arduino-Uno-gesteuert




Von
**Pierre Commarmot**
(Frankreich)

Dieser von einem Arduino gesteuerte Tester ermittelt die wesentlichen Kenndaten vieler Arten von Temperatursensoren, ob Silizium, PTC oder NTC, zum Vergleich mit den Datenblattangaben. Er hilft auch einen fehlerhaften Sensor zu identifizieren oder zwei Sensoren zu einem Paar zusammenzustellen.

Das Projekt entstand, nachdem in der heimischen Solaranlage eine Fehlfunktion auftrat. Es basiert auf dem Joule-Effekt: Eine Metallplatte, die thermisch mit dem DUT (device under test) und einem Referenzsensor gekoppelt ist, wird auf eine konstante, einstellbare Temperatur erhitzt. So können Sie die Eigenschaften des Sensors bei unterschiedlichen Temperaturen messen. Ein Arduino Uno steuert die Temperatur.

### Elektronik und Mechanik

Die Heizplatte besteht aus einem Stück Aluminiumprofil, an dem ein 15-Ω-Leistungswiderstand befestigt ist. Wie in der Schaltung in **Bild 1** zu sehen, fließt ein Strom aus dem 12-V-Netzteil über einen N-Kanal-Leistungs-MOSFET durch den Heiz-Widerstand R5. Der MOSFET wird vom Arduino Uno mit einem PWM-Signal (Pulse Width Modulation) angesteuert. Zwischen Arduino und MOSFET-Gate muss der Signalpegel von TTL auf 12 V angehoben werden. Es gibt zwar spezielle MOSFETs, die direkt mit TTL-Spannung angesteuert werden können, wir aber machen es billiger und setzen die beiden komplementären Kleinsignal-Transistoren T1 und T2 zu diesem Zweck ein. Neben den angegebenen Typen können auch andere komplementäre Kleinsignal-Transistoren verwendet werden, Verstärkung und Übergangsfrequenz sind hier nicht kritisch. Übrigens wird so auch die Software verständlicher, weil ein TTL-High-Pegel (logische 1) der maximalen Leistung am Widerstand entspricht. Ein externes 12-V-Netzteil, das 2 A Gleichstrom liefert, dient nur deswegen als Energiequelle, weil ich ein solches Netzgerät gerade zur Verfügung hatte!

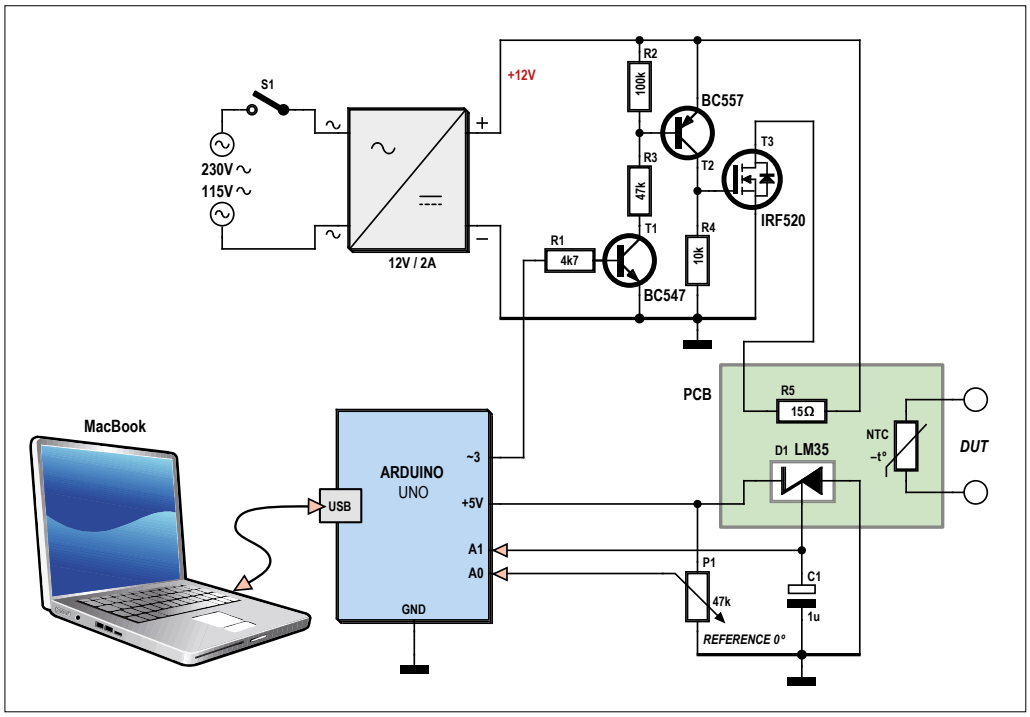



Bild 1.
Schaltung des Testers für Temperatursensoren.

Die Schnittstelle zwischen Arduino und Leistungsregler wurde einfach auf ein Stück Rasterplatine aufgebaut. Auf dem Aluminiumprofil mit dem 15-Ω-Widerstand sind der zu überprüfende Sensor und der Referenzsensor LM35 sehr eng beieinander angeordnet und mit einem kleinen Alustreifen thermisch optimal gekoppelt. So ist der Fehler aufgrund von Temperaturgradienten in der Vorrichtung minimal. Der 1-µF-Kondensator C1 filtert das Ausgangssignal und reduziert so das Rauschen. Die Solltemperatur wird durch ein Potentiometer am 10-bit-ADC-Eingang des Arduino eingestellt.

## Software

Die Software für den Tester muss keine Wunder vollbringen, es werden nur ein paar Pins des Arduino verwendet. Wenn Sie ein anderes Arduino-Modell verwenden möchten, achten Sie darauf, dass die Schaltung entsprechend angepasst wird.

Die Software besteht aus zwei Teilen: einem, der mit der freien *Arduino-IDE* [1] geschrieben wurde und dem anderen, der mit *Processing* [2] entwickelt wurde, um die Daten dazustellen. Beide Entwicklungsumgebungen habe ich in den Versionen von September 2014 benutzt. Da ich Mac-User bin, muss die *Processing*-Software für Linux oder Windows vermutlich angepasst werden (platform porting). Der **Arduino-Sketch** für das Projekt ist in **Listing 1** zu sehen und kann von [3] heruntergeladen werden. Zuerst ermittelt der Arduino die gewünschte Referenztemperatur für die Messung, indem er das Potentiometer abfragt. Als nächstes misst der Arduino mit dem LM35-Sensor, der mit einem anderen ADC-Kanal verbunden ist, die Ist-Temperatur der Testvorrichtung. Der LM35 liefert eine Spannung, die proportional ist zur Temperatur (10 mV/°C). Der Fehler ist mit ±0,5 °C gering. Die Leistung, die zum Erreichen der gewünschten Temperatur erforderlich ist, wird von einem einfachen Software-PID-Regler berechnet. Dabei gibt es zwei Modi: „aggressiv“, um sich der Solltemperatur schnell anzunähern und „konservativ“, wenn die Ist- nahe an die Soll-Temperatur kommt. Der konservative Modus vermeidet langsame, langwellige Schwingungen um den PID-Gleichgewichtspunkt. Ich hatte im Sinn, auch den Prüfling automatisch zu messen, aber diese Funktion bisher nicht in der Software implementiert – das DUT wird mit einem gewöhnlichen Digitalmultimeter ausgelesen.

Arduino sendet etwa alle 500 ms die folgenden Parameter, getrennt durch Kommas (,) und beendet mit einem Wagenrücklauf (CR):

- Soll-Temperatur (Gleitkomma; Bereich 0...100 °C)
- Ist-Temperatur von LM35 (Integer; Bereich 0...100 °C)

- Berechnung der Heizleistung (Integer; Bereich 0...100 %)
- DUT-Temperatur (Gleitkomma, Bereich 0...100 °C)

sprechend eingestellt.
Danke an die hervorragende G4P-Website für die Vermittlung des umfassenden Wissens über Processing!


Bild 2.
Der Tester für Temperatursensoren auf dem Labortisch des Autors in Aktion.

Das USB-Kabel zwischen dem Host-Computer und dem Arduino (Uno) liefert die Versorgungsspannung für den Arduino und für den LM35; dient aber auch zur Übertragung der Daten. Die kleine Processing-Software zeichnet nur ein Rechteck mit vier Slidern auf den Schirm, jeder für die Anzeige eines Parameters. Wenn ein String eintrifft, wird jeder Parameter extrahiert und der Slider ent-

**In der Praxis**

Schließen Sie das Uno–Board an den (MacBook-)Computer an und starten Sie die Arduino-IDE. Stellen Sie den Board-Typ und die verwendete serielle Schnittstelle ein, öffnen Sie dann die Datei „TestSondeTemp.ino“ und übertragen Sie sie auf den Uno.
Nun müssen Sie die Display-Anwendung auf Ihrem Host-Computer erstellen. *Processing*

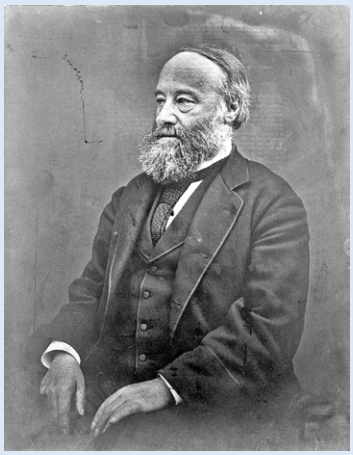
## Das Erste Joulesche Gesetz

Das Erste Joulesche Gesetz von 1840 (Wärmestromgesetz) besagt, dass die Wärmeproduktion $Q$, die von einem Strom $I$ in einem Leiter mit dem Widerstand $R$ verursacht wird, gleich

$Q = P\,t = U\,I\,t = R\,I^2\,t$ [Joule/Sekunde] ist.

Dies entspricht dem in der Elektronik bekannten

$P = I^2\,R$ [Watt]



James Prescott Joule

wird gestartet und „TestSondeTemp.pde“ geöffnet. Sie können nun zwischen einer eigenständigen Applikation oder dem Start aus der Processing-IDE wählen. Die Schriftart „ArialMT-20.vlw“ muss in einer „Data“-Datei auf der gleichen Verzeichnis-Ebene wie das Source-File „TestSondeTemp.pde“ vorhanden sein.
Nach sorgfältiger Überprüfung der Bauteile und der Verkabelung können Sie nun das Gerät einschalten und die Temperatur wählen. In weniger als zwei Minuten sollte die Soll-Temperatur erreicht und stabil sein.

Dann messen Sie den Prüfling mit dem Multimeter. Viel Spaß beim Messen!

(150062)

## Weblinks

[1] Arduino-IDE: www.arduino.cc/en/Main/Software

[2] Processing: https://processing.org/download/?processing

[3] Projektsoftware: www.elektormagazine.de/artikel

**Listing 1. Arduino-Sketch für den Test von Temperatur-Sensoren [3]**

```
// Test bench PID temperature regulation
// Inputs:  potentiometer temperature reference (0..5V) input A0
//          bench temperature sensor LM35 (0..5V) input A1
// Outputs: heating PWM command out 3
//          USB port: data output every 500 ms

// Check Github website
#include <PID_v1.h>

double Consigne, Input, Output;

// Aggressive and conservative settings
double aggKp=4, aggKi=0.2, aggKd=1;
double consKp=1, consKi=0.05, consKd=0.25;

PID MonPID(&Input, &Output, &Consigne, consKp, consKi, consKd, DIRECT);

// Wiring
int BrocheRefTemp = 0;   // potentiometer on A0
int BrocheSondeRef = 1;  // LM35 on A1
int BrocheSondeTest = 2; // DUT on A2, not used
int BrocheMosFET = 3;    // MOSFET gate

// Variables initialisation
int PotValue;         // Potentiometer value
int ValPot1024 = 0;   // 0-1023 PotValue
int ValPWM256 = 0;    // PWM output value

// Defining inputs-outputs, initialisation
void setup() {
  Serial.begin(19200);
  pinMode(BrocheRefTemp, INPUT);
  pinMode(BrocheSondeRef, INPUT);
```

```
 pinMode(BrocheSondeTest, INPUT);
 pinMode(BrocheMosFET, OUTPUT);
 MonPID.SetMode(AUTOMATIC);
}

void loop() {

 // Getting reference temperature
 PotValue = analogRead(BrocheRefTemp);
 PotValue = map(PotValue, 0, 1023, 0, 100);
 Consigne = double(PotValue); // Target temperature
 Serial.print(float(PotValue));
 Serial.print(“,”);

 // LM35 probe reading
 ValPot1024 = analogRead(BrocheSondeRef);
 ValPot1024 = map(ValPot1024, 0, 205, 0, 100);
 Input = ValPot1024;
 Serial.print(float(ValPot1024));
 Serial.print(“,”);

 // Computing required power
 double gap = abs(Consigne-Input); // Target temp distance
 if(gap<10)
 {  // approaching target temp we use conservative parameters
   MonPID.SetTunings(consKp, consKi, consKd);
 }
 else
 {
    // far from target temp, use aggressive parameters
    MonPID.SetTunings(aggKp, aggKi, aggKd);
 }

 MonPID.Compute();
 ValPWM256 = int(Output);  // output = 1 -->Max power
 if (Consigne<21) {
 ValPWM256 = 0;};
 analogWrite(BrocheMosFET, ValPWM256);
 Serial.print(map(ValPWM256, 0, 255, 0, 100));
 Serial.print(“,”);


 // DUT reading
 ValPot1024 = analogRead(BrocheSondeTest);
 ValPot1024 = map(ValPot1024, 0,1023, 0, 100);
 Serial.print(float(ValPot1024));
 Serial.println(“”);

 delay(500);
}
```