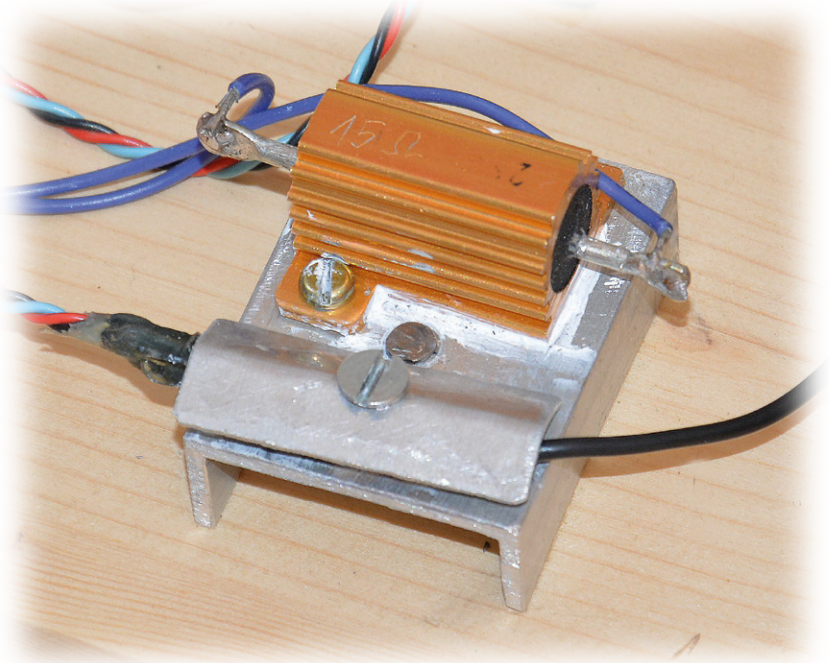
Meetopstelling voor temperatuursensors

Met Arduino Uno



Pierre Commarmot
(Frankrijk))

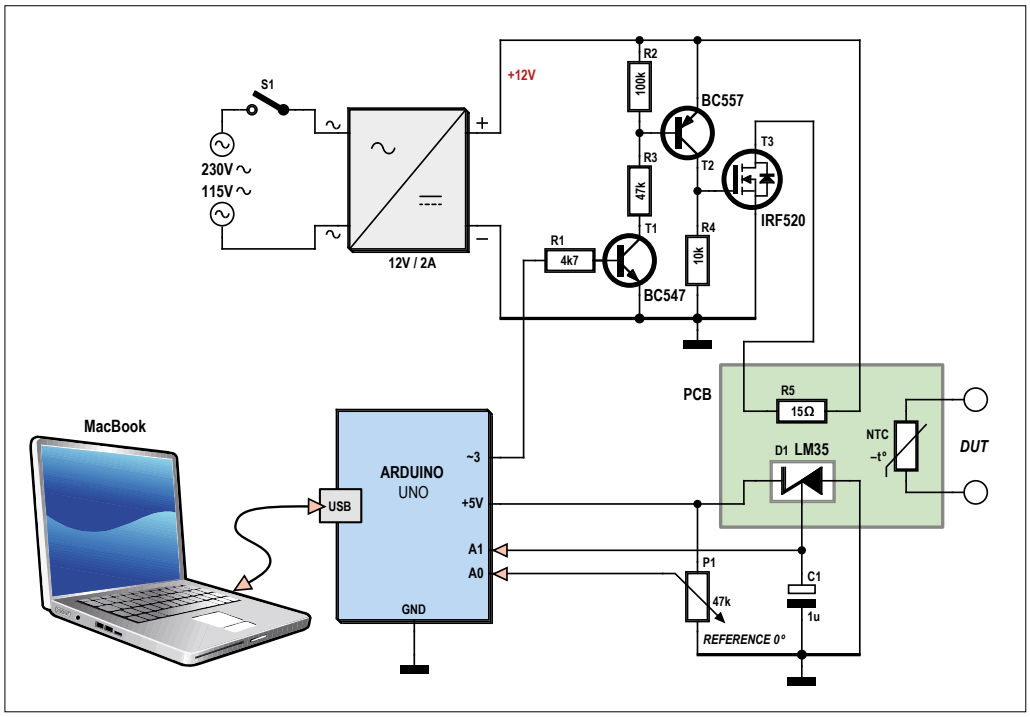
Met deze door een Arduino Uno bestuurde meetopstelling kunt u de belangrijkste kenmerken van verschillende silicium-, PTC- en NTC-temperatuursensors bepalen en deze met de fabrieksspecificaties vergelijken. U kunt hiermee defecte sensors opsporen of 'matched pairs' selecteren.

Bij dit project wordt gebruik gemaakt van het Joule-effect: een metalen plaatje dat thermisch is gekoppeld met het testobject (DUT, Device Under Test) wordt verwarmd tot een bepaalde temperatuur. Met de Arduino Uno kan deze temperatuur worden ingesteld, waardoor de eigenschappen van de sensor bij verschillende temperaturen kunnen worden gemeten.

Schakeling en mechanische opbouw

De meetopstelling is gemaakt van een stukje aluminiumprofiel waarop een vermogensweerstand van 15 Ω is bevestigd. Zoals in het schema van **figuur 1** is te zien wordt dit 'verwarmingselement' van energie voorzien door een N-kanaal vermogens-MOSFET die door de Arduino met een PWM-signaal wordt aangestuurd. Tussen de TTL-uitgang van de Arduino en de gate van de MOSFET bevindt zich een niveau-omzetter die bestaat uit een complementair transistorpaar (T1 en T2). Omdat versterking en afsnijfrequentie hier niet van belang zijn, kunnen ook andere complementaire types worden gebruikt. Door deze schakeling wordt de software vereenvoudigd omdat een TTL-'hoog'-niveau (logische '1') nu overeenkomt met maximaal door de MOSFET geleverd vermogen. Een externe 12V/2A-gelijkstroomvoeding levert de voedingsspanning voor de schakeling en de verwarmingsweerstand. De interface tussen de Arduino en de vermogensregelaar is gemonteerd op experimenteerprint.

De twee 'probes' (LM35 en testobject/NTC) zijn bij de 15Ω-vermogensweerstand op het aluminiumprofiel gemonteerd dat als warmhoudplaat fungeert. Op deze manier worden



Figuur 1.
Schema van de meetopstelling voor temperatuursensors.

fouten door temperatuurverschillen in de opstelling tot een minimum beperkt. Het testobject en de LM35-referentiesensor moeten zo dicht mogelijk bij elkaar worden geplaatst. Eventuele stoorsignalen worden met een condensator van 1 µF (C1) uit het uitgangssignaal van de LM35 gefilterd. De temperatuur van de opstelling wordt ingesteld met een potmeter die direct met de 10-bits ADC-aansluiting van de Arduino is verbonden.

Software

De software is niet echt ingewikkeld, want er worden maar een paar pennen van de Uno gebruikt. Als u een ander Arduino-model wilt gebruiken, controleer dan of de bedrading overeenkomt met de ingangen en uitgangen van uw Arduino-board.

De software bestaat uit twee delen: het eerste is ontwikkeld met de gratis *Arduino-IDE* [1] en het andere, dat de data op de hostcomputer weergeeft, is ontwikkeld met *Processing* [2]. Ik heb hiervoor de *Arduino IDE* en *Processing* versies van september 2014 gebruikt. Als ervaren Mac-gebruiker verliep de ontwikkeling met *Processing* voor mij heel natuurlijk, maar voor Linux of Windows zijn er misschien enkele aanpassingen (‘platform porting’) nodig.

In **Listing 1** vindt u de Arduino-sketch die bij dit project hoort. Deze kan van [3] worden gedownload.
Eerst bepaalt de Arduino de temperatuurinstelling voor de meting door de stand van de potmeter uit te lezen. Vervolgens wordt de actuele temperatuur gemeten met de LM35-sensor die met een ander ADC-kanaal is verbonden. De LM35 levert een spanning die recht evenredig is met de temperatuur (10 mV/°C) met een nauwkeurigheid van ±0.5 °C. Het vermogen (de warmte) die nodig is om deze temperatuur te bereiken wordt dan berekend door een eenvoudige software-PID-regelaar (Proportioneel-Integrerend-Differentiërend). Er worden twee modi gebruikt: de ‘aggressieve’ voor snelle convergentie en de ‘conservatieve’ als de actuele temperatuur dicht bij de doelwaarde komt. De conservatieve modus voorkomt langzame schommelingen rondom het PID-balanspunt. Vervolgens verzendt de Arduino ongeveer iedere 500 ms de volgende gegevens, gescheiden door een komma (,) en afgesloten met een carriage return (CR):

- Ingestelde referentietemperatuur (floating point; bereik 0 tot 100 °C)
- Actuele temperatuur van de opstelling (LM35) (integer; bereik 0 tot 100 °C)
- Berekend verwarmingsvermogen (integer; bereik 0 tot 100%)

- Temperatuur testobject (floating point, bereik 0 tot 100°C)

Ik was van plan om de meting van het testobject automatisch uit te laten voeren, maar die functie is in deze softwareversie niet geïmplementeerd. Het testobject moet met een digitale multimeter worden gemeten.
De USB-kabel tussen uw computer en de Arduino (Uno) levert de voeding voor de Arduino en de LM35 en zorgt voor de gegevensoverdracht. Het (kleine) verwerkingsprogramma tekent een rechthoek met vier wijzers met maatverdeling die ieder een parameter weergeven. Na ontvangst van een gegevensreeks worden de parameters 'uitgepakt' en met de bijbehorende wijzer weergegeven. Daarna wordt een speciale taak gestart die computer-events verwerkt. Met dank aan de uitstekende 'G4P'-website met veel informatie over Processing!

Figuur 2.
De opstelling in gebruik op mijn labtafel.

Meetpraktijk
Verbind het Uno-board met uw (MacBook) computer en start de Arduino IDE. Definieer het type board en de gebruikte seriële poort. Open vervolgens het bestand 'TestSondeTemp.ino' en laad het in de Uno.
Verbind de Uno met de rest van de schakeling. Nu moet u het displayprogramma op uw computer installeren. Start *Processing* en open 'TestSondeTemp.pde'; u kunt hiervoor

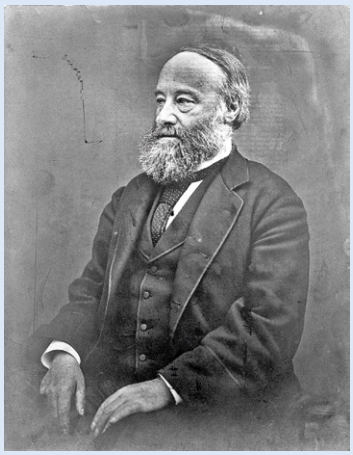
De wet van Joule

James Prescott Joule beschreef in 1840 als eerste dat een stroom I die vloeit in een weerstand R en daarover een spanningsval U veroorzaakt een hoeveelheid warmte Q produceert die gelijk is aan:

$Q = P\,t = U\,I\,t = R\,I^2\,t$ [joule/seconde]

Hetgeen in de elektronica overeenkomt met:

$P = I^2 \times R$ [watt]



een autonome toepassing creëren of u kunt het direct in de Processing-IDE starten. Het font 'ArialMT-20.vlw' moet aanwezig zijn in een 'Data'-bestand op hetzelfde niveau als het bronbestand 'TestSondeTemp.pde'.
Na controle van de bedrading en de montage van de onderdelen kan de voeding van de meetopstelling worden ingeschakeld. Wacht ongeveer twee minuten totdat thermische stabilisatie is bereikt en meet dan het testobject met een multimeter.

De grafische weergave geeft u snel inzicht in de eigenschappen van het testobject.
Veel succes!

(150062)

Weblinks

[1] Arduino IDE: www.arduino.cc/en/Main/Software
[2] Processing: https://processing.org/download/?processing
[3] Project software: www.elektormagazine.com/articles

Listing 1. Arduinosketch voor temperatuursensor-meetopstelling [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);
}
```