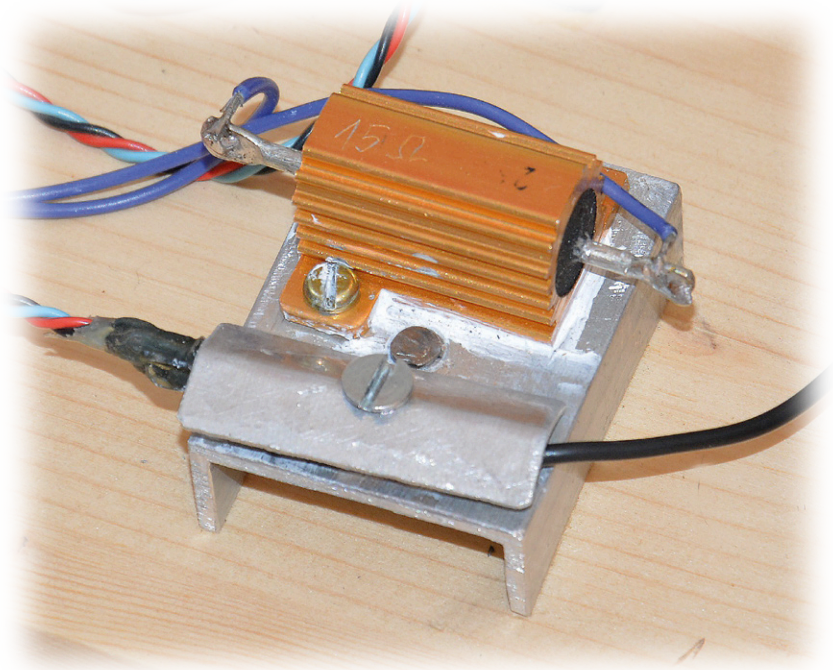
Test Jig for Temperature Probes & Sensors

Arduino Uno controlled



By
Pierre Commarmot
(France)

This Arduino-driven fixture enables the essential characteristics of many types of temperature sensor including silicon, PTC, and NTC to be measured for comparing to the factory specs. It also helps you identify a faulty sensor, or match a pair of them.

The project was designed after suffering a malfunction in the domestic solar water heater. The approach is based on the Joule effect: a small metal plate thermally coupled to the DUT (device under test) is heated to a constant, adjustable temperature. After stabilization you can measure the sensor's characteristics at different temperatures set with the help of your Arduino Uno.

Electronics and mechanics

The jig is made of a piece of aluminum profile, to which a 15-Ω power resistor is secured. As shown in the schematic in **Figure 1**, an N-channel power MOSFET driven by an Arduino PWM signal (Pulse Width Modulation) supplies power to this "heater" resistor. A signal level converter is required between the Arduino's TTL output and the MOSFET's gate. Even if it is possible — under certain circumstances — to drive a MOSFET gate directly from a TTL output, this option is not used here. Alternatively we use a pair of cheap, complementary small-signal transistors, T1 and T2, to drive our MOSFET's gate. Other types of complementary small-signal transistors should be okay for this function too, as gain and transition frequency are not really critical here. By the way, the software becomes easier because a TTL High level (logic 1) corresponds with maximum power delivered by the MOSFET. An external 12-V, 2-A DC supply is used to power the circuit including the heater resistor, simply because I had one available in my drawer!
The interface between the Arduino and the power governor was built on a piece of stripboard.
The two "probes" (LM35 and DUT/NTC) and

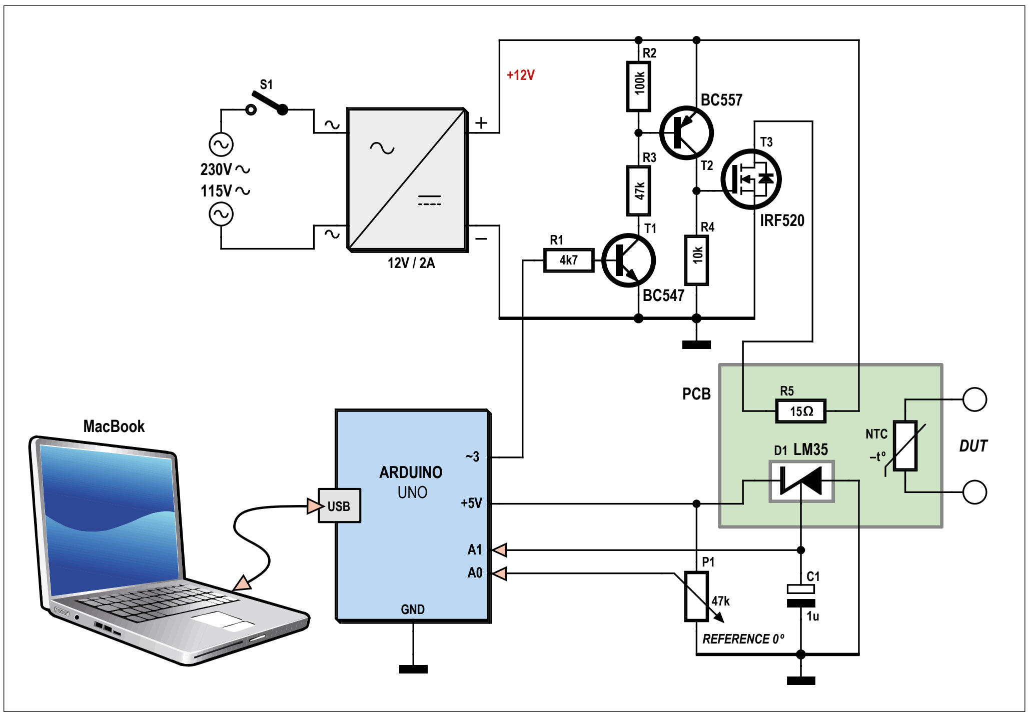


Figure 1.
Schematic of the test jig for temperature sensors.

the 15-Ω power resistor are jointly secured to a small piece of aluminum, which acts as a hotplate. This assembly method reduces errors due to temperature gradients of the jig to a minimum. The sensor under test and the LM35 reference sensor must be mounted very closely together. In order to reduce noise, a 1-μF capacitor, C1, filters the LM35's output signal. The target temperature of the jig is set by a potentiometer directly connected to the Arduino's 10-bit ADC input.

Software

In terms of software there's absolutely no need here for muscle flexing, and we use a few pins of the Uno only. If you want to use a different Arduino model, just check that the wiring is correct according to input/output function of your Arduino board.
The software is composed of two parts: one written with the free *Arduino IDE* [1] and the other, developed with *Processing* [2] for the host computer to display data. Please note that I worked with releases of *Arduino IDE* and *Processing* current as of September 2014. Being a long time Mac user, the *Processing* software came naturally; I suppose some adjustments ('platform porting') should be done for Linux or Windows.

The **Arduino sketch** developed for the project is appended to this **.**POST article as **Listing 1**, and can be downloaded from [3].
Let's see what Arduino is doing. First, it gets the reference temperature requested for the measurement by reading a potentiometer. Next, using the LM35 sensor connected on another ADC channel, it measures the actual temperature of the test jig. The LM35 yields a voltage that's proportional to temperature at a 10 mV/°C rate with good precision of ±0.5 °C. The power (i.e. heat) required to achieve this temperature is then computed by means of a rudimentary software-PID controller (proportional-integral-derivative). Two modes are used: the 'aggressive' one to converge rapidly, and the 'conservative' one when the actual temperature gets very close to the target level. Conservative mode avoids slow long wave oscillations around the PID balance point. I did plan to automatically measure the device under test, but eventually the function was not implemented in this software version — you have to read out the DUT with a digital multimeter.
Next! Arduino sends the following parameters roughly every 500 ms, separated by a comma (,) and terminated with a carriage return (CR):

- Reference temperature requested (floating point; range 0 to 100 °C)
- Actual jig temperature from LM35 (inte-

ger; range 0 to 100 °C)
- Computed heating power (integer; range 0 to 100%)
- DUT temperature (floating point, range 0 to 100°C)

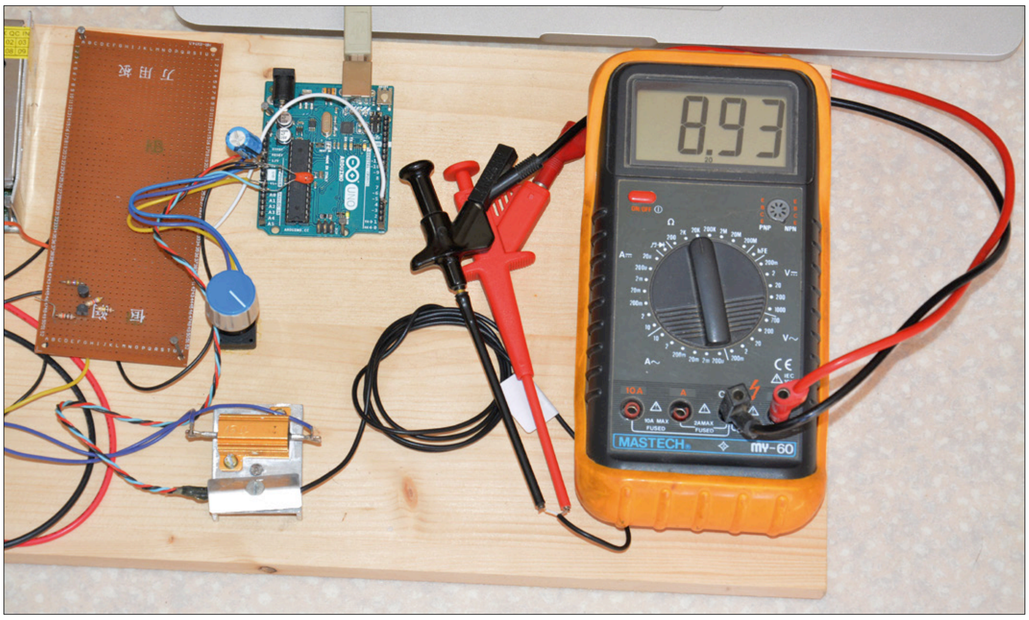


Figure 2.
The author's jig in use on his workbench.

The USB cable between host computer and the Arduino (Uno) carries supply power for the Arduino and the LM35, as well as the transmitted parameters. The small Processing software only draws a rectangle including four graded sliders, each one displaying a parameter. When a string is received, each parameter is extracted and tied to its slider; when done, a special task is launched which handles computer events. Thanks to the excellent "G4P" website for broadcasting a wide knowledge about Processing!

Using the jig

Connect the Uno board to your (MacBook) computer, and launch Arduino IDE. Define your board type and which serial port is used, then open the file 'TestSondeTemp.ino' and transfer it to your Uno.
Wire the Uno as indicated here. You now have to create the display application on your host computer. Launch *Processing* and open 'TestSondeTemp.pde'; you may create an autonomous application, or choose to launch it from Processing IDE.

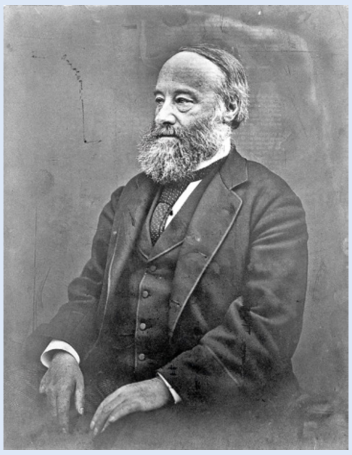
Joule's Law (was first)

says that around 1840, the heat production Q resulting from current I flowing in a conductor with resistance R and consequently dropping a voltage U was observed to equal

$Q = P\,t = U\,I\,t = R\,I^2\,t$ [joules/second]

which in the domain of electronics is the familiar

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



The ‘ArialMT-20.vlw’ font must be present in a ‘Data’ file at the same level as the source file ‘TestSondeTemp.pde’.
After carefully checking the wiring and the component connections, you can now power on the device, set a temperature, wait under two minutes for thermal stabilization to be reached and then measure the DUT with a multimeter.
A graphical representation offers a very quick glance at the characteristics of your DUT. Have fun!

(150062)

Web Links

[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. Arduino Sketch for Temperature Sensor Test Jig (Ref. [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);
}
```