Thermistor

Created by lady ada

https://learn.adafruit.com/thermistor

Last updated on 2023-08-29 02:08:20 PM EDT

Table of Contents

Overview

A thermistor is a thermal resistor - a resistor that changes its resistance with temperature. Technically, all resistors are thermistors - their resistance changes slightly with temperature - but the change is usually very very small and difficult to measure. Thermistors are made so that the resistance changes drastically with temperature so that it can be 100 ohms or more of change per degree!

There are two kinds of of thermistors, NTC (negative temperature coefficient) and PTC (positive temperature coefficient). In general, you will see NTC sensors used for temperature measurement. PTC's are often used as resettable fuses - an increase in temperature increases the resistance which means that as more current passes through them, they heat up and 'choke back' the current, quite handy for protecting circuits!

Thermistors have some benefits over other kinds of temperature sensors such as analog output chips (LM35/TMP36 ()) or digital temperature sensor chips (DS18B20) or thermocouples ().

- First off, they are much much cheaper than all the above! A bare 5% thermistor is only 10 cents in bulk.
- They are also much easier to waterproof since its just a resistor.
- They work at any voltage (digital sensors require 3 or 5V logic).
- Compared to a thermocouple, they don't require an amplifier to read the minute voltages - you can use any microcontroller to read a thermistor.

- They can also be incredibly accurate for the price. For example, the 10K 1% thermistor in the shop is good for measuring with ±0.25°C accuracy! (Assuming you have an accurate enough analog converter)
- They are difficult to break or damage - they are much simpler and more reliable

On the other hand, they require a little more work to interpret readings, and they dont work at very high temperatures like thermocouples. Without a digital-to-analog converter on board, you might be better off with a digital temperature sensor.

Their simplicity makes them incredibly popular for basic temperature feedback control. For example, lets say you wanted to have a fan that turns on when the temperature gets high. You could use a microcontroller, a digital sensor, and have that control the relay. Or you could use the thermistor to feed the base of a transistor, as the temperature rises, the resistance goes down, feeding more current into the transistor until it turns on. (This is a rough idea, you would need a few more components to make it work)

Even if you do use a microcontroller or complex system, for the price you can't beat 'em!

You can pick up a 10K 1% waterproof thermistor in the Adafruit shop (http://adafru.it/372)

Some Stats

Here are technical details for the thermistor in our shop (http://adafru.it/372)

- Resistance at 25°C: 10K ±1%
- B25/50: 3950 ±1%
- Thermal time constant ? 15 seconds
- Thermistor temperature range -55°C to 125°C
- Wire temperature range -55°C to 105°C
- 28 AWG PVC Wire
- Diameter: 3.5mm/0.13in
- Length: 18in/45cm
- Resistance/Temperature table ()

Note that even though the thermistor can go up to 125°C the cable itself maxes out at 105°C so this thermistor is not good for measuring very very hot liquids

Testing a Thermistor

Because thermistors are simply resistors, its easy to test it out. Simply measure the resistance using a multimeter:

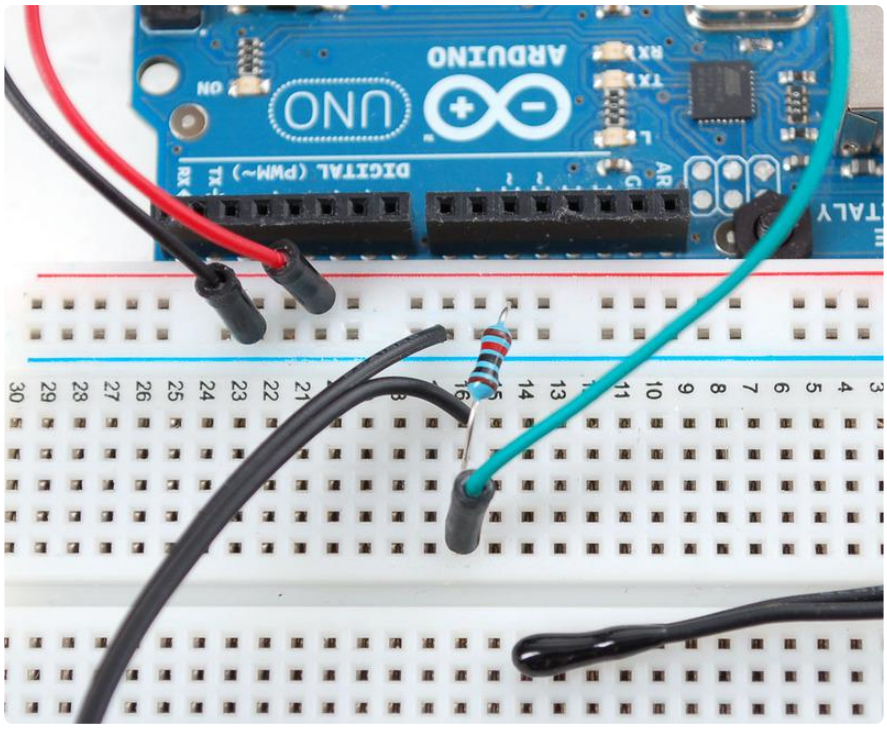
You should read about 10Kohm assuming its room temperature where you're sitting. The resistance of course may be higher or lower depending on the room temperature. Also, your hands may end up touching the contacts so your body-resistance will lower the value a bit too. But you should still get about 10 Kohm

For example, its warm here in an un-airconditioned room in the middle of the summer, so we read 8Kohm (30°C - 86°F!)

Using a Thermistor

Connecting to a Thermistor

These thermistors are pretty hardy, you can strip the PVC insulation and stick the wires into a breadboard or solder to them directly. Of course you can cut or extend the wires. Since the resistance is pretty high (10Kohm) the wire resistance won't make a huge difference.



Analog Voltage Reading Method

To measure the temperature, we need to measure the resistance. However, a microcontroller does not have a resistance-meter built in. Instead, it only has a voltage reader known as a analog-digital-converter. So what we have to do is convert the resistance into a voltage, and we'll do that by adding another resistor and connecting them in series. Now you just measure the voltage in the middle, as the resistance changes, the voltage changes too, according to the simple voltage-divider equation. We just need to keep one resistor fixed

Say the fixed resistor is 10K and the variable resistor is called R - the voltage output (V o) is:

Vo = R / (R + 10K) * Vcc

Where Vcc is the power supply voltage (3.3V or 5V)

Now we want to connect it up to a microcontroller. Remember that when you measure a voltage (Vi) into an Arduino ADC, you'll get a number.

ADC value = Vi * 1023 / Varef

So now we combine the two (Vo = Vi) and get:

ADC value = R / (R + 10K) * Vcc * 1023 / Varef

What is nice is that if you notice, if Vcc (logic voltage) is the same as the ARef, analog reference voltage, the values cancel out!

ADC value = R / (R + 10K) * 1023

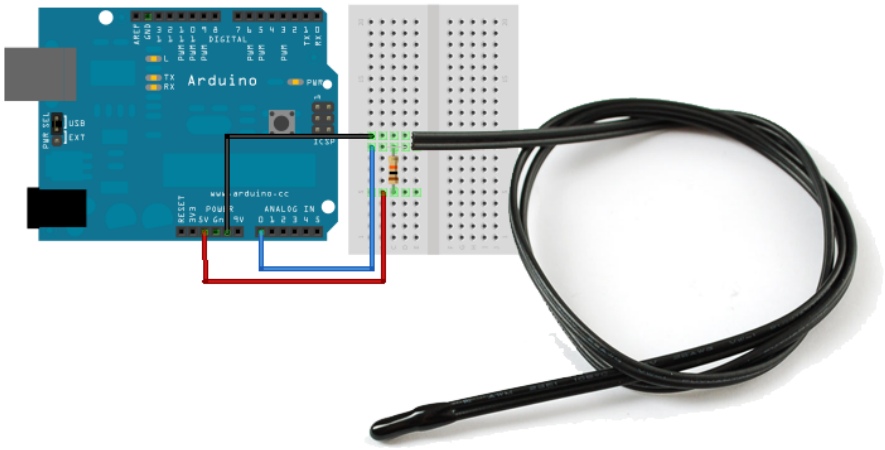
It doesn't matter what voltage you're running under. Handy!

Finally, what we really want to do is get that R (the unknown resistance). So we do a little math to move the R to one side:

R = 10K / (1023/ADC - 1)

Lots of people have emailed me to tell me the above equation is wrong and the correct calculation is R = 10K*ADC / (1023 - ADC). Their equivalence is left as an exercise for the reader! ;)

Great, lets try it out. Connect up the thermistor as shown:



Connect one end of the 10K resistor to 5V, connect the other end of the 10K 1% resistor to one pin of the thermistor and the other pin of the thermistor to ground. Then connect Analog 0 pin to the 'center' of the two.

Now run the following sketch:

```
// SPDX-FileCopyrightText: 2011 Limor Fried/ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// thermistor-1.ino Simple test program for a thermistor for Adafruit Learning
System
// https://learn.adafruit.com/thermistor/using-a-thermistor by Limor Fried,
Adafruit Industries
// MIT License - please keep attribution and consider buying parts from Adafruit

// the value of the 'other' resistor
#define SERIESRESISTOR 10000
```

```
// What pin to connect the sensor to
#define THERMISTORPIN A0

void setup(void) {
  Serial.begin(9600);
}

void loop(void) {
  float reading;

  reading = analogRead(THERMISTORPIN);

  Serial.print("Analog reading ");
  Serial.println(reading);

  // convert the value to resistance
  reading = (1023 / reading)  - 1;     // (1023/ADC - 1)
  reading = SERIESRESISTOR / reading;  // 10K / (1023/ADC - 1)
  Serial.print("Thermistor resistance ");
  Serial.println(reading);

  delay(1000);
}
```

You should get responses that correspond to the resistance of the thermistor as measured with a multimeter

If you are not getting correct readings, check that the 10K resistor is placed between VCC and A0, and the thermistor is between A0 and ground. Check you have a 10K Thermistor and that you are using a 'standard' NTC thermistor. On a "5V" microcontroller like classic Arduino or Metro 328, use 5V for the VCC pin. On 3.3V microcontrollers like Feather or Arduino Zero, use 3.3V for the VCC pin.

If, when you heat up the thermistor, the temperature reading goes down, check that you don't have the two resistors swapped and check that you are using an NTC not PTC thermistor.

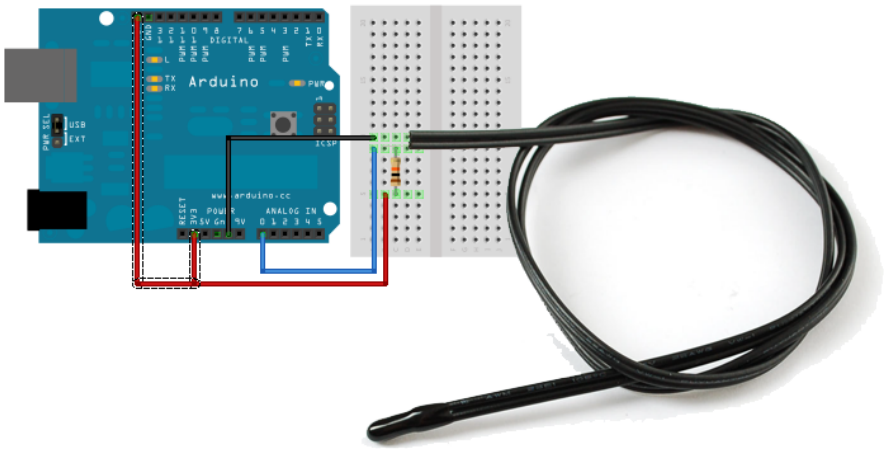
Better Readings

When doing analog readings, especially with a 'noisy' board like the arduino, we suggest two tricks to improve results. One is to use the 3.3V voltage pin as an analog reference and the other is to take a bunch of readings in a row and average them.

The first trick relies on the fact that the 5V power supply that comes straight from your computer's USB does a lot of stuff on the Arduino, and is almost always much noisier than the 3.3V line (which goes through a secondary filter/regulator stage!) It's easy to use, simply connect 3.3V to AREF and use that as the VCC voltage. Because our calcuations don't include the VCC voltage, you don't have to change your equation. You do have to set the analog reference but that's a single line of code

Taking multiple readings to average out the result helps get slightly better results as well, since you may have noise or fluctuations, we suggest about 5 samples.

Rewire as shown, the 10K resistor is still connected to the higher voltage, and the thermistor to ground



This sketch takes those two improvements and integrates them into the demo, you will have better, more precise readings.

Note that this code specifies an EXTERNAL voltage reference. To work properly, you must make the additional connection to the AREF pin as shown in the diagram above.

```
// SPDX-FileCopyrightText: 2011 Limor Fried/ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// thermistor-2.ino Intermediate test program for a thermistor. Adafruit Learning
System Tutorial
// https://learn.adafruit.com/thermistor/using-a-thermistor by Limor Fried,
Adafruit Industries
// MIT License - please keep attribution and please consider buying parts from
Adafruit

// which analog pin to connect
#define THERMISTORPIN A0
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define NUMSAMPLES 5
// the value of the 'other' resistor
#define SERIESRESISTOR 10000

int samples[NUMSAMPLES];

void setup(void) {
  Serial.begin(9600);
  // connect AREF to 3.3V and use that as VCC, less noisy!
  analogReference(EXTERNAL);
}

void loop(void) {
```

```
  uint8_t i;
  float average;

  // take N samples in a row, with a slight delay
  for (i=0; i< NUMSAMPLES; i++) {
   samples[i] = analogRead(THERMISTORPIN);
   delay(10);
  }

  // average all the samples out
  average = 0;
  for (i=0; i< NUMSAMPLES; i++) {
     average += samples[i];
  }
  average /= NUMSAMPLES;

  Serial.print("Average analog reading ");
  Serial.println(average);
  // convert the value to resistance
  average = 1023 / average - 1;
  average = SERIESRESISTOR / average;

  Serial.print("Thermistor resistance ");
  Serial.println(average);

  delay(1000);
}
```

Converting to Temperature

Finally, of course, we want to have the temperature reading, not just a resistance! If you just need to do a quick comparison circuit (if temperature is below X do this, if its above Y do that), you can simply use the temperature/resistance table which correlates the resistance of the thermistor to the temperature.

However, you probably want actual temperature values. To do that we'll use the Steinhart-Hart equation () , which lets us do a good approximation of converting values. Its not as exact as the thermistor table (it is an approximation) but its pretty good around the temperatures that this thermistor is used.

$$\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3$$

However, this equation is fairly complex, and requires knowing a lot of variables that we don't have for this thermistor. Instead we will use the simplified B parameter equation ().

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B}\ln\left(\frac{R}{R_0}\right)$$

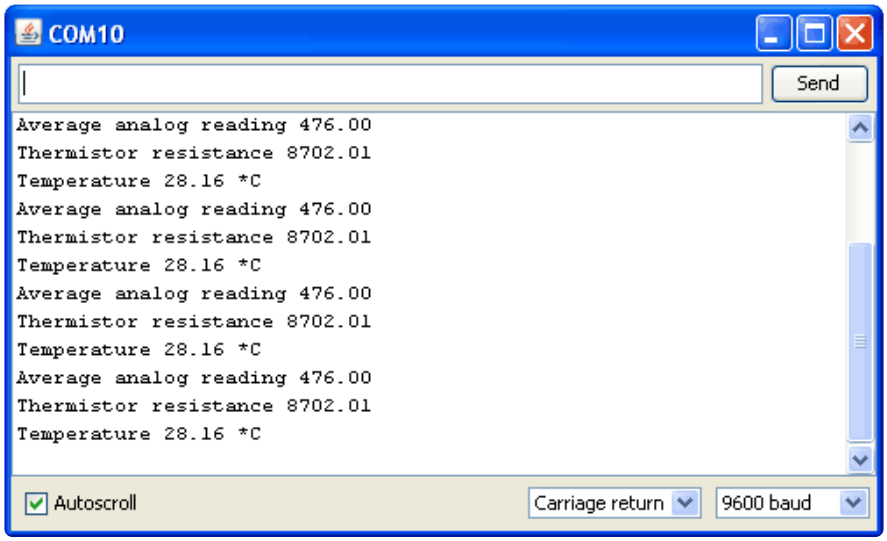
For this one we only need to know To (which is room temperature, 25 °C = 298.15 K) B (in this case 3950, the coefficient of the thermistor), and Ro (the resistance at room

temp, in this case 10Kohm). We plug in R (resistance measured) and get out T (temper ature in Kelvin) which is easy to convert to °C

The following sketch will calculate °C for you

```
// SPDX-FileCopyrightText: 2011 Limor Fried/ladyada for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Thermistor Example #3 from the Adafruit Learning System guide on Thermistors
// https://learn.adafruit.com/thermistor/overview by Limor Fried, Adafruit
Industries
// MIT License - please keep attribution and consider buying parts from Adafruit

// which analog pin to connect
#define THERMISTORPIN A0
// resistance at 25 degrees C
#define THERMISTORNOMINAL 10000
// temp. for nominal resistance (almost always 25 C)
#define TEMPERATURENOMINAL 25
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define NUMSAMPLES 5
// The beta coefficient of the thermistor (usually 3000-4000)
#define BCOEFFICIENT 3950
// the value of the 'other' resistor
#define SERIESRESISTOR 10000

int samples[NUMSAMPLES];

void setup(void) {
  Serial.begin(9600);
  analogReference(EXTERNAL);
}

void loop(void) {
  uint8_t i;
  float average;

  // take N samples in a row, with a slight delay
  for (i=0; i< NUMSAMPLES; i++) {
   samples[i] = analogRead(THERMISTORPIN);
   delay(10);
  }

  // average all the samples out
  average = 0;
  for (i=0; i< NUMSAMPLES; i++) {
     average += samples[i];
  }
  average /= NUMSAMPLES;

  Serial.print("Average analog reading ");
  Serial.println(average);

  // convert the value to resistance
  average = 1023 / average - 1;
  average = SERIESRESISTOR / average;
  Serial.print("Thermistor resistance ");
  Serial.println(average);

  float steinhart;
  steinhart = average / THERMISTORNOMINAL;     // (R/Ro)
  steinhart = log(steinhart);                  // ln(R/Ro)
  steinhart /= BCOEFFICIENT;                   // 1/B * ln(R/Ro)
  steinhart += 1.0 / (TEMPERATURENOMINAL + 273.15); // + (1/To)
```

```
  steinhart = 1.0 / steinhart;                  // Invert
  steinhart -= 273.15;                          // convert absolute temp to C

  Serial.print("Temperature ");
  Serial.print(steinhart);
  Serial.println(" *C");

  delay(1000);
}
```



For better precision, we suggest reading the exact value of the 'series 10K' it should be nearly exactly 10K but if you can get a better reading that will reduce your error.

How Accurate is the Reading?

You may notice that above, the temperature reading is 28.16°C - does that mean we have 0.01°C accuracy? Unfortunately no! The thermistor has error and the analog reading circuitry has error.

We can approximate the expected error by first taking into account the thermistor resistance error. The thermistor is correct to 1%, which means that at 25°C it can read 10,100 to 9900 ohms. At around 25°C a difference of 450 ohms represents 1°C so 1% error means about +-0.25°C (you may be able to calibrate this away by determining the resistance of the thermistor in a 0°C ice bath and removing any offset). You can also spring for a 0.1% thermistor which will reduce the possible resistance error down to +-0.03°C

Then there is the error of the ADC, for every bit that it is wrong the resistance (around 25°C) can be off by about 50 ohms. This isn't too bad, and is a smaller error than the thermistor error itself +-(0.1°C) but there is no way to calibrate it 'away' - a higher precision ADC (12-16 bits instead of 10) will give you more precise readings

In general, we think thermistors are higher precision than thermocouples, or most low cost digital sensors, but you will not get better than +-0.1°C accuracy on an Arduino with a 1% thermistor and we would suggest assuming no better than +-0.5°C.

Self-Heating

If you have a 10K thermistor + 10K resistor connected between 5V and ground, you'll get about 5V / (10K + 10K) = 0.25mA flowing at all times. While this isn't a lot of current, it will heat up your thermistor as the 10K thermistor will be dissipating about 0.25mA * 2.5V = 0.625 mW.

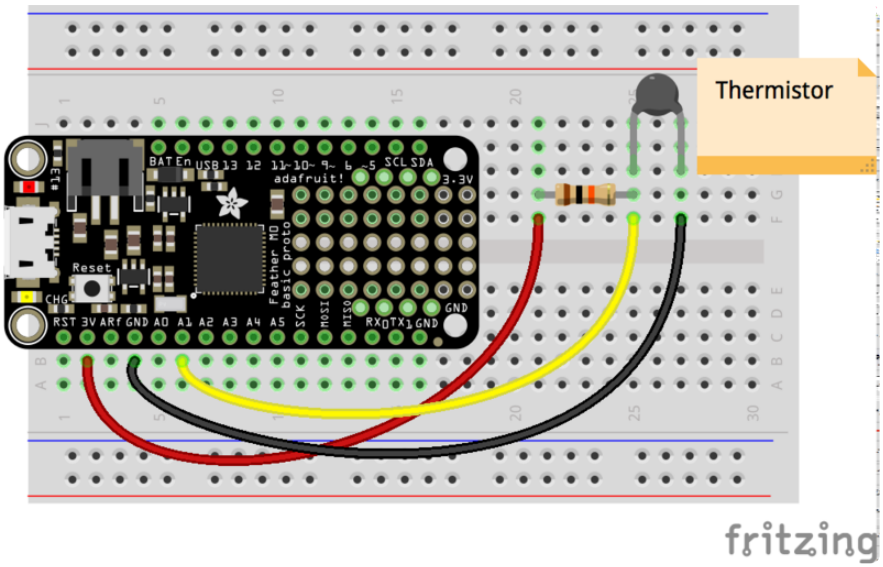
To avoid this heating, you can try connecting the 'top' of the resistor divider to a GPIO pin and set that pin HIGH when you want to read (thus creating the divider) and then LOW when you are in low power mode (no current will flow from 0V to ground)

CircuitPython

It's easy to use a thermistor with CircuitPython and your board's built-in analog to digital converters. Just like with the Arduino example on the previous page you can connect the thermistor to a board's analog input and read the resistance. As the temperature changes the resistance changes and you can convert that resistance into a precise temperature value with Python code!

Before you get started it will help to familiarize yourself with analog inputs in CircuitPython ().

Next wire up a thermistor to your board exactly as shown on the previous page. You need a fixed resistor (typically 10 kilo-ohms) connected from an analog input up to 3.3V. Then connect one pin from the thermistor to the same analog input and the other pin to your board's ground. In this example we'll use analog input A1 on a Feather M0 basic.



Fritzing Source

Next connect to the board's serial REPL ()so you are at the CircuitPython >>> prompt.

You can import the necessary board and analogio modules by running:

```
import board
import analogio
```

Now create an analog input using the pin you've connected to the thermistor:

```
thermistor = analogio.AnalogIn(board.A1)
```

At this point you can read the raw ADC value from the thermistor:

```
thermistor.value
```

```
>>> thermistor.value
32753
>>>
```

The raw value isn't very interesting to us though, we really want to convert it to a resistance and temperature value. One thing to note though is that this raw value will always be in the range from 0 to 65535, unlike in Arduino where it ranges from 0 to 1023. As the resistance of the thermistor changes (based on temperature changes) this raw value will change too.

Using the same equation as the previous page you can calculate the resistance of the thermistor:

```
R = 10000 / (65535/thermistor.value - 1)
print('Thermistor resistance: {} ohms'.format(R))
```

```
>>> R = 10000 / (65535/thermistor.value - 1)
>>> print('Thermistor resistance: {} ohms'.format(R))
Thermistor resistance: 10912.3 ohms
>>>
```

Remember if you're using a different size resistor your might need to change the equation above and the 10000 value inside it!

Convert to Temperature

Converting the thermistor's resistance to temperature is just like you saw on the previous page with Arduino. You can use a special equation and some known parameters about the thermistor to perform this conversion in Python code.

First make sure you know these values for your thermistor (check its datasheet if available):

- Ro - The resistance at a nominal temperature value. This is typically 10,000 ohms.
- To - The temperature (in Celsius) at the nominal resistance value above. This is typically 25.0 degrees C.
- Beta - The beta coefficient value for the thermistor. Typically this is a value in the range of 3000-4000, for example 3950.

We can now solve the simplified B coefficient Steinhart-Hart equation mentioned on the previous page. Here's a function you can define to perform this math:

```
def steinhart_temperature_C(r, Ro=10000.0, To=25.0, beta=3950.0):
    import math
    steinhart = math.log(r / Ro) / beta      # log(R/Ro) / beta
    steinhart += 1.0 / (To + 273.15)         # log(R/Ro) / beta + 1/To
    steinhart = (1.0 / steinhart) - 273.15   # Invert, convert to C
    return steinhart
```

Now call the function and pass it the thermistor resistance that you calculated. You can pass in explicit Ro, To, and beta parameters too or use the defaults (10k, 25.0C, 3950):

```
R = 10000 / (65535/thermistor.value - 1)
steinhart_temperature_C(R)
```

```
>>> def steinhart_temperature_C(r, Ro=10000.0, To=25.0, beta=3950.0):
...     import math
...     steinhart = math.log(r / Ro) / beta
...     steinhart += 1.0 / (To + 273.15)
...     steinhart = (1.0 / steinhart) - 273.15
...     return steinhart
...
>>> R = 10000 / (65535/thermistor.value - 1)
>>> steinhart_temperature_C(R)
22.7711
>>>
```

Or if you're passing in explicit Ro, To, beta parameters:

```
steinhart_temperature_C(R, Ro=10000.0, To=25.0, beta=3950)
```

```
>>> steinhart_temperature_C(R, Ro=10000.0, To=25.0, beta=3950)
22.7711
>>>
```

Now you have the temperature from the thermistor as a value in degrees Celsius!

Thermistor Module

If you just want to read the value of a thermistor you can actually use a handy CircuitPython module to perform all the above calculations for you automatically. To use the thermistor module sensor with your Adafruit CircuitPython () board you'll need to install the Adafruit_CircuitPython_Thermistor () module on your board. Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!

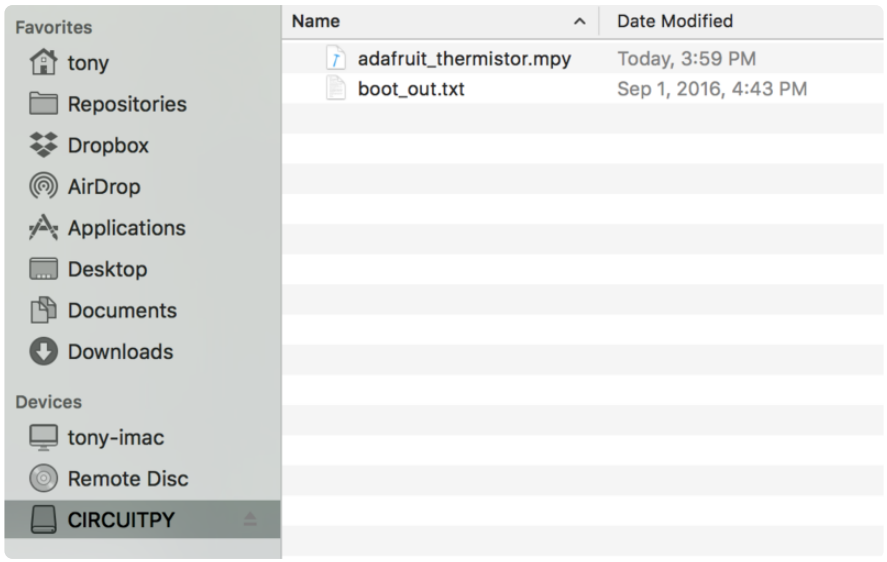
First make sure you are running the latest version of Adafruit CircuitPython () for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (). For example the Circuit Playground Express guide has a great page on how to install the library bundle () for both express and non-express boards.

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- adafruit_thermistor.mpy

Before continuing make sure your board's lib folder or root filesystem has the adafruit_thermistor.mpy module copied over.



Usage

To demonstrate the usage of the thermistor module you can connect to your board's serial REPL and run Python code to read the temperature and humidity.

First connect to the board's serial REPL ()so you are at the CircuitPython >>> prompt.

Next import the board and adafruit_thermistor modules, these are necessary modules to initialize and access the sensor:

```
import board
import adafruit_thermistor
```

Now create an instance of the Thermistor class from the module. You'll need to know the Ro, To, and beta parameters for your thermistor just like when performing the math yourself. For example with the same thermistor setup as before you would run:

```
thermistor = adafruit_thermistor.Thermistor(board.A1, 10000.0, 10000.0, 25.0,
3950.0, high_side=False)
```

Let's break down all the parameters sent to the Thermistor initializer:

- Analog input - The first parameter is the name of the analog input connected to the thermistor, board pin A1 in this case.
- Series resistance - The second parameter is the value of the series resistor connected to the thermistor. If you're following this guide you want a value of 10,000 ohms.

- Nominal resistance (Ro) - The third parameter is the value of the thermistor's resistance at a nominal temperature. For the thermistor in this guide use the same 10,000 ohms value.
- Nominal temperature (To) - The fourth parameter is the value of the thermistor's temperature (in degrees Celsius) at a nominal resistance value. For the thermistor in this guide use the same 25.0 degree value.
- Beta coefficient - The fifth parameter is the beta coefficient for your thermistor, 3950 in this case.
- High_side boolean - The sixth parameter is optional and indicates if the thermistor is connected on the high side or low side of the resistor voltage divider. For this guide we've actually wired up the thermistor on the low side, or from the ADC input down to ground. However for other boards and usage you might wire the thermistor the opposite way from the high side, or from ADC input up to 3.3V or 5V. The default value for this high_side parameter is true but for the wiring in this guide we need to tell it that we're using low side wiring by setting high_side to false.

Once the thermistor instance is created you can read the temperature property to get the temperature value in degrees Celsius:

```
thermistor.temperature
```

```
>>> thermistor.temperature
25.8647
```

That's all you need to read the temperature of a thermistor with the thermistor module! Internally the module will do all the necessary Steinhart-Hart equation math for you. You can grab the temperature result and use it in your own programs to add temperature sensing!

Python Docs

Python Docs ()

Buy a Thermistor

Buy a Thermistor (http://adafru.it/372)