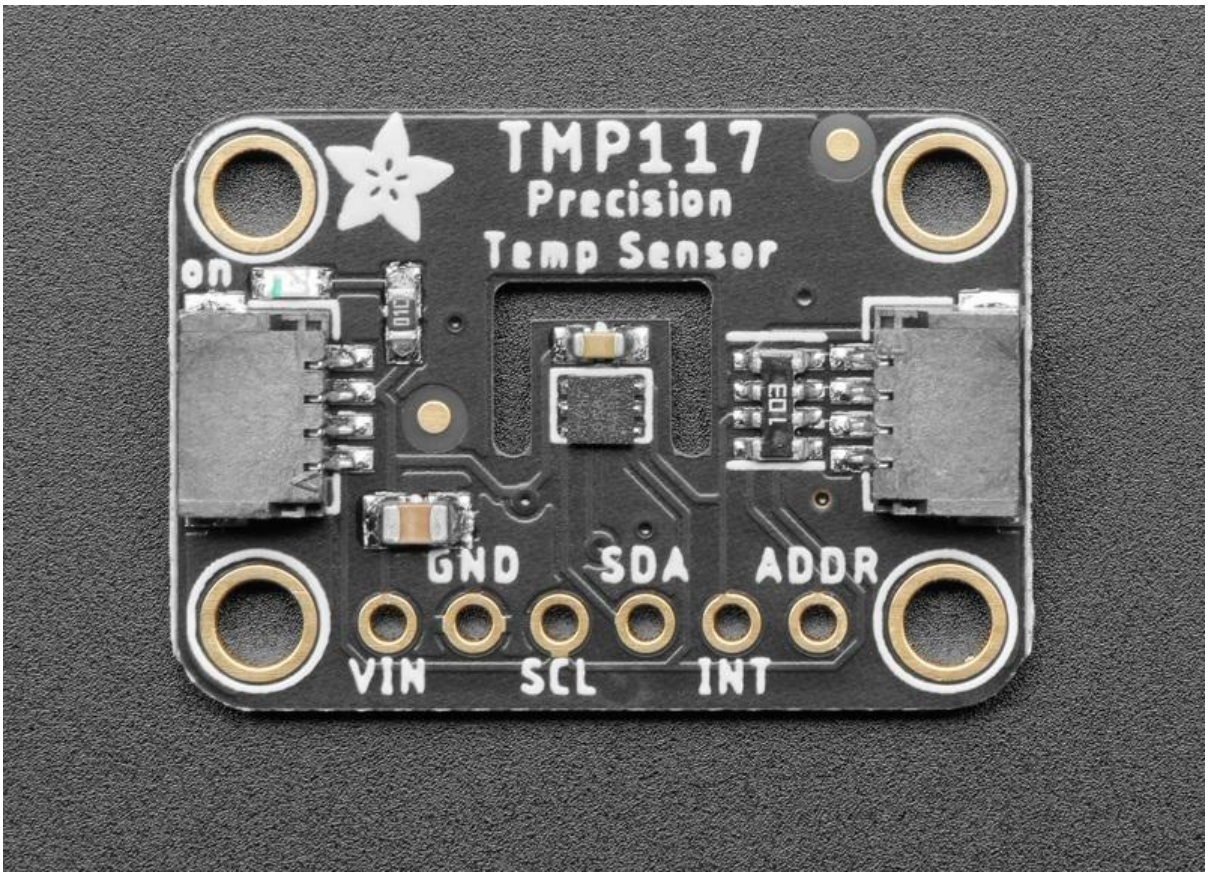
# Adafruit TMP117 High Accuracy I2C Temperature Monitor

Created by Bryan Siepert




https://learn.adafruit.com/adafruit-tmp117-high-accuracy-i2c-temperature-monitor

Last updated on 2023-01-10 02:39:04 PM EST

# Table of Contents


Overview 3

Featured Features 5

- Temperature offset management
- Temperature thresholds
- Threshold behavior
- Measurement Delay
- Sample Count and Averaging

Pinouts 9

- Power Pins
- Other Pins

Arduino 11

- I2C Wiring
- Library Installation
- Load Basic Example
- Basic Example Code
- Load Alerts Example
- Alerts Example Code

Arduino Docs 17

Python & CircuitPython 18

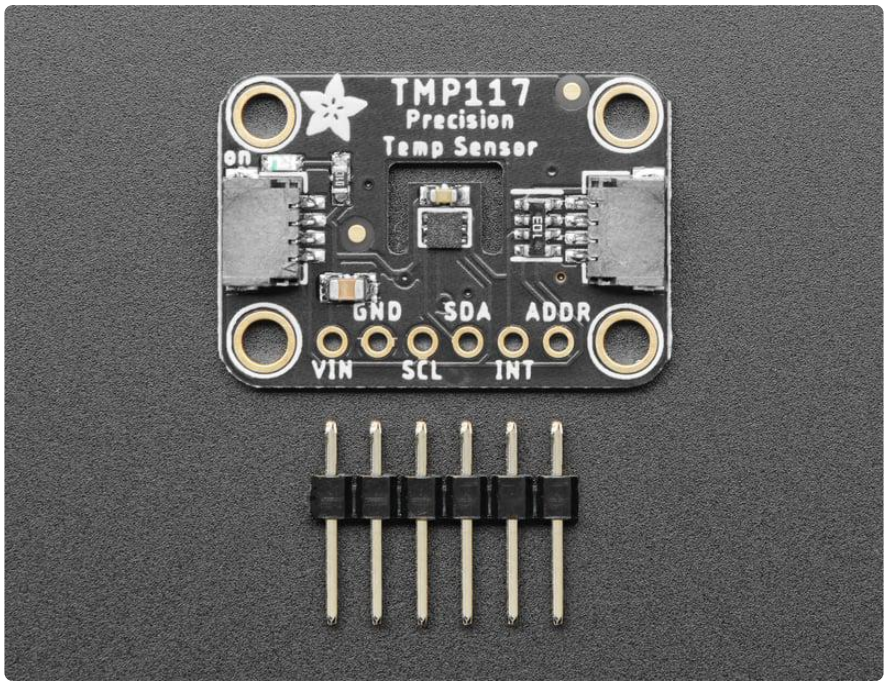
- CircuitPython Microcontroller Wiring
- Python Computer Wiring
- CircuitPython Installation of TMP117 Library
- Python Installation of TMP117 Library
- Basic CircuitPython & Python Usage
- Basic Example Code
- Temperature Alerts Example
- Temperature Alerts Example Code

Python Docs 22

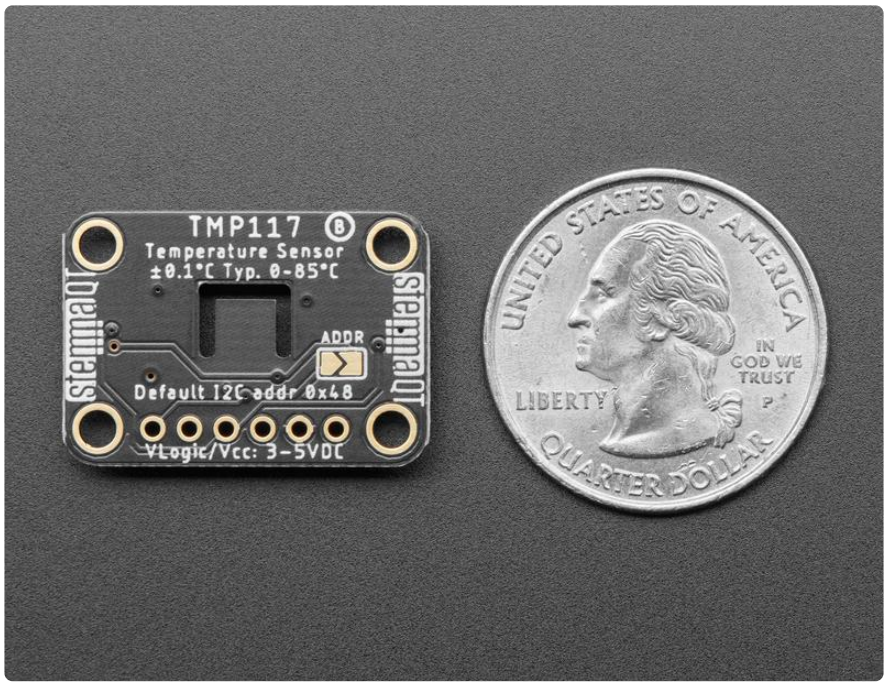
Downloads 23

- Files
- Schematic
- Fab Print

# Overview

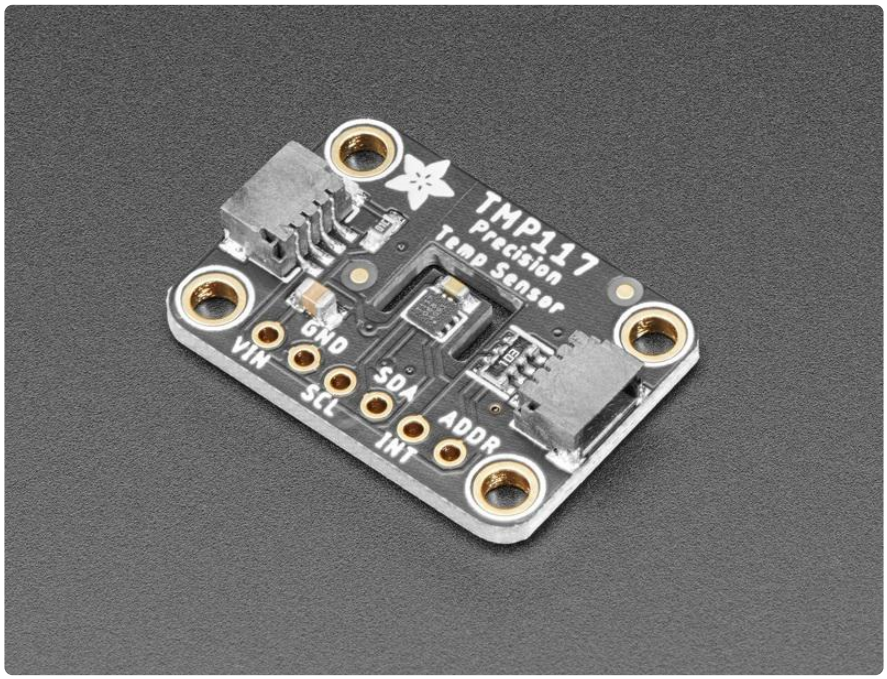



The TMP117 Precision Temperature Sensor is an I2C temperature sensor that will help you easily add temperature measurement and adjustment to your project. In addition to the obvious support for reading the temperature, the TMP117 can also monitor the temperature and alert you when corrective action needs to be taken.




With 16-bit measurement resolution and up to ±0.1°C accuracy, as well as high and low temperature alerts and interrupt support, and hardware support required for NIST traceability, this temperature sensor is perfect for applications where you need to

keep a close eye on temperature. The manufacturer, Texas Instruments, even suggest it for use in sensitive applications like thermostats and cold chain asset tracking or even gas and heat meters!




To make using it as easy as possible, we've put the TMP117 on a breakout PCB  in our Stemma QT form factor () with a sprinkle of support circuitry to give you options when testing. You can either use a breadboard or the SparkFun qwiic () compatible STEMM A QT () connectors, and compatibility with 5V voltage levels as commonly found on Arduinos (), as well as 3.3V logic used by many other boards like the Raspberry Pi or our Feathers.

To get started, all you need to do is look over the Pinouts page to familiarize yourself with the board, and then use the Arduino or Python & CircuitPython pages for instructions on how to wire up the TMP117 to your board, as well as libraries and example code to get you started.

# Featured Features


The TMP117 is not merely a temperature sensor, any hunk of silicon can do that. It comes with a handful of relevant features that help you make best use of this information.

## Temperature offset management

When you're trying to get an accurate measurement, despite your best efforts it's common that the temperature reported by your measurement is different than the actual value. You might be sitting in your kitchen, surrounded by thermometers telling you that the room is in fact 40 degrees Celcius, however due to specifics of the immediate environment of the sensor, it may declare with the upmost certainty that it is actually 43 degrees C.

This type of change from the actual value will most times stay the same or close to it as the sensor finds itself in different locations with different temperature. This type of deviation from the known value is called an offset. No matter where the sensor is, the sensor will measure the temperature, but because of how air flows around the sensor, or how heat moves through the circuit board, or one of a thousand other things, it then takes this temperature and sets it off to the side, always the same amount. The offset remains relatively constant because the factors causing it are not changing, so their effect on the temperature also does not change.


Because they're constant, a temperature offset is easy to fix by adding or subtracting it from the reported temperature. The TMP117 will manage this for you by allowing you to set the offset amount which is then added to the measured temperature before it is reported to you. If the sensor says itself 43 degrees but you know from other sources that it is actually 40 degrees, setting an offset of -3 will account for it. After setting the offset, the TMP117 will take the measured 43 degrees, add -3 to it to account for the offset before reporting the adjusted value to you.

## Temperature thresholds

Measuring temperature is all well and good, everyone needs to know the temperature at some point, but usually you want to know the temperature not out of pure curiosity but because you want to do something differently depending on what the temperature is. If it's colder than 60°F degrees outside, you probably want to wear a hoodie. If instead it's -10°C and you're in Saskatoon, you probably want to think about putting on an extra pair of dungarees.

The clever people at Texas Instruments were smart enough to predict this important decision point and they added high and low temperature thresholds to the TMP117. This allows you to delegate this decision to it and let it make decisions on your behalf. Most people may be hesitant to allow a machine to make such an important pants-related decision, however unlike your friend Joe who is well meaning but inattentive, the TMP117 won't get distracted by other stuff. All it's doing is sitting there, doing its part as a cog in the machine, measuring, reporting, and if appropriate, acting on the temperature.

By setting the high and low temperature thresholds, the TMP117 can then tell you when either threshold is exceeded. If you know you always wear pants when the

temperature is below 60 degrees, set the low temperature threshold to 60 degrees and let the TMP117 tell you when it's time for pants.

If like me you also believe that wearing pants when the temperature is above 75 degrees is tepidly tyrannical, you may then also wish to set a high temperature threshold that well tell you when it's time to release the bonds of your pants and be free of their stifling embrace.

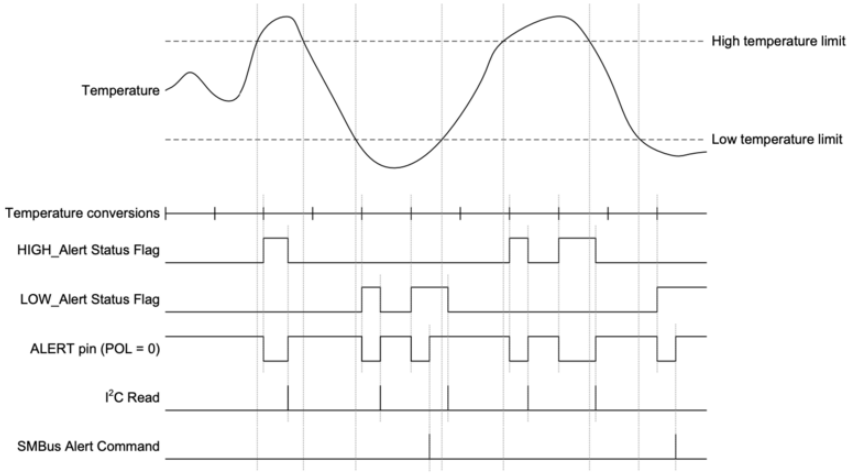



**Figure 18. Alert Mode Timing Diagram**

## Threshold behavior

This behavior essentially boils down to "tell me when it's hot" and "tell me when it's cold". You still have to decide what that _means_. Alternatively, you could ask the TMP117 to answer the question: "Should I be wearing a skirt"?. In this case, you don't care if the temperature is high or low exactly, you want to know if the temperature is in a range where wearing a skirt is warranted. In this case, obviously if the temperature is over a certain amount, you're going to be wearing a skirt. But if the temperature goes below 40 degrees, are you going to immediately take off your skirt and put on some capris? No, you're going to keep your skirt on as long as possible, so you want to wait not until it's OK to wear capris, but when must you wear capris.

In this case, you would want to use the TMP117's "THERM" mode. When configured this way, only the high threshold alert will be used. Instead of the low threshold being the "it's cold" temperature, it is instead used as a "it's not hot anymore" value.

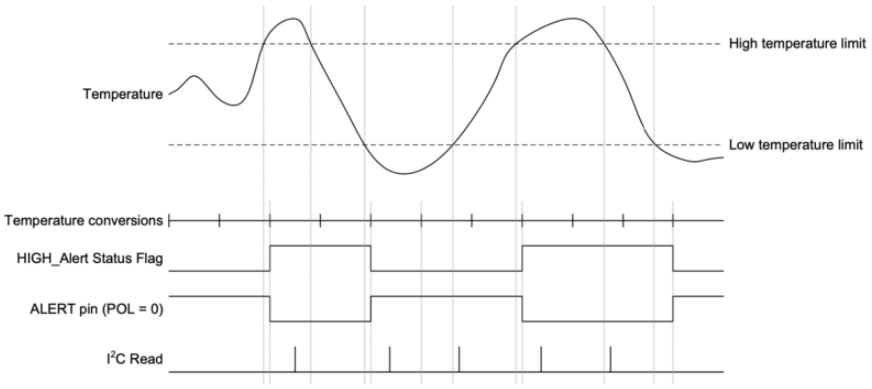



**Figure 19. Therm Mode Timing Diagram**

## Measurement Delay

The TMP117 also allows you to made adjustments to the measurement behavior. Unlike things like "speed" or "fashion", temperature doesn't change all that fast. If it's too cold for a skirt one second, it's going to still be too cold 15 seconds later, barring unexpected lava.

To make use of this, the TMP117 allows you to change how often it calculates the temperature. By specifying a read delay, the TMP117 can go into a low power mode when continuing to read new values would be redundant.

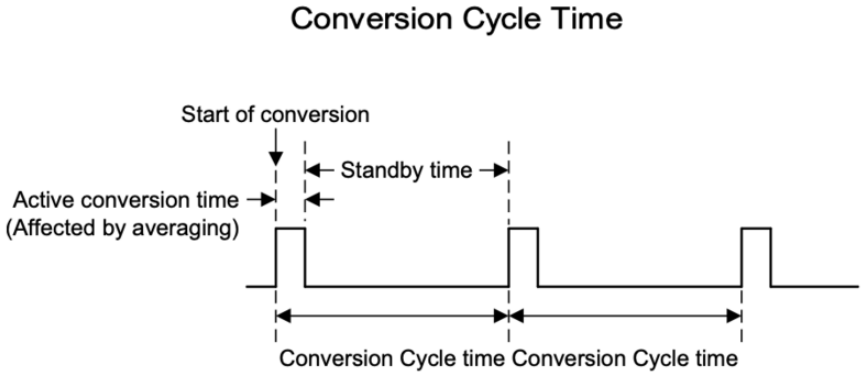
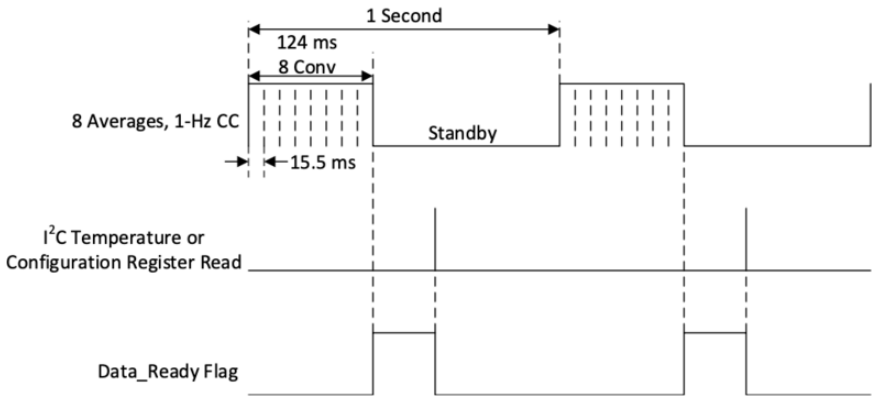



**Figure 16. Conversion Cycle Timing Diagram**

## Sample Count and Averaging

As we've established, from a human perspective, temperature doesn't change very quickly. Sensor are not humans however; they don't "have a feeling" about what the temperature is, they _measure_ it. That means that even if just one second ago the temperature was exactly 36.42 degrees C, if moments later a draft or paranormal

phenomenon causes a micro-current of cold air to wisp across the sensor, the temperature *for the sensor* will drop, if only for a fraction of a second.

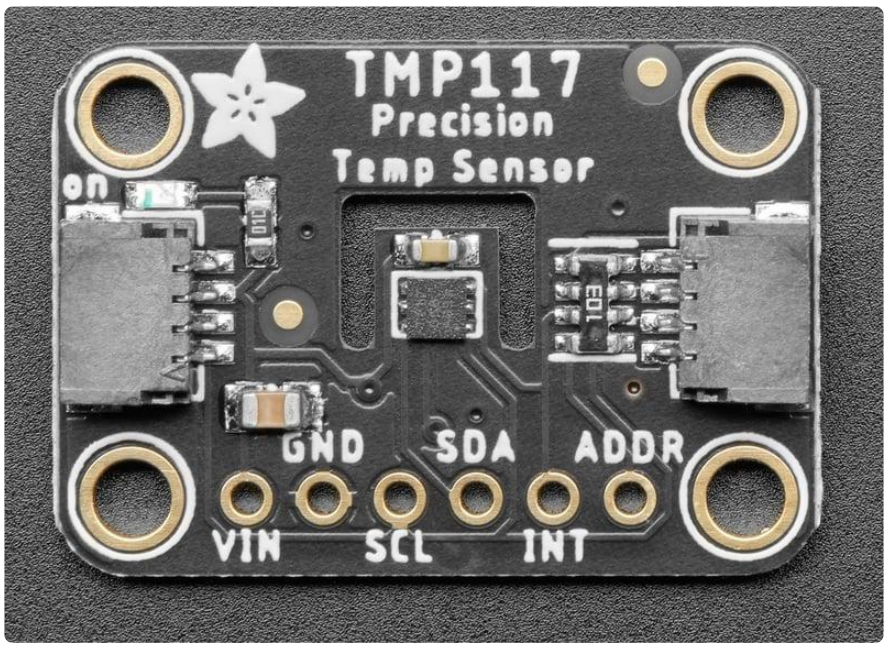



The result of this is that if you were to take that single measurement that occurred while the sensor was momentarily in a cold spot, while technically accurate it won't be representative of the overall temperature. Like good scientists, we know that to account for this inevitable fluctuation we should instead take several measurements and average them to get a more representative value.

The TMP117 also allows you to specify the number of readings that are taken and averaged. While they only take approximately 15.5 milliseconds, they will add up, so the more you average, the longer it will take to return a result.

---

# Pinouts



## Power Pins

- VCC - this is the power pin. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V microcontroller like Arduino, use 5V
- GND - common ground for power and logic

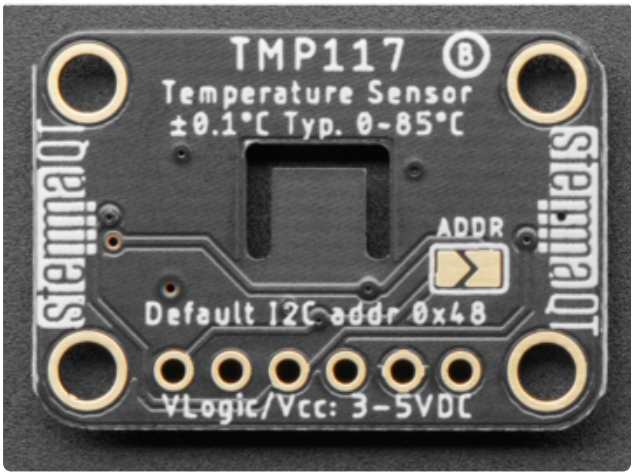
I2C Logic Pins

- SCL - I2C clock pin, connect to your microcontroller I2C clock line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin.
- SDA - I2C data pin, connect to your microcontroller I2C data line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin.
- ADDR/AD0 Jumper - I2C Address pin. Pulling this pin high or bridging the solder jumper on the back will change the I2C address from 0x48 to 0x49

- STEMMA QT () - These connectors allow you to connectors to dev boards with S TEMMA QT connectors or to other things with various associated accessories ()

## Other Pins

- INT -This is the primary interrupt pin. You can setup the TMP117 to pull this low when certain conditions are met such as new measurement data being available, or if high or low temperature alerts are triggered.
- ADDR - The address pin can be used to change the I2C address from its default value of 0x48 to another value depending on the voltage it is tied to, according to the table below




You can solder the back jumper closed to change the address from 0x48 to 0x49

| ADDR Pin Voltage | 7-bit I2C address (hex) |
|---|---|
| Ground (Default) | 0x48 |
| VCC (Solder Jumper) | 0x49 |
| SDA | 0x4A |
| SCL | 0x4B |

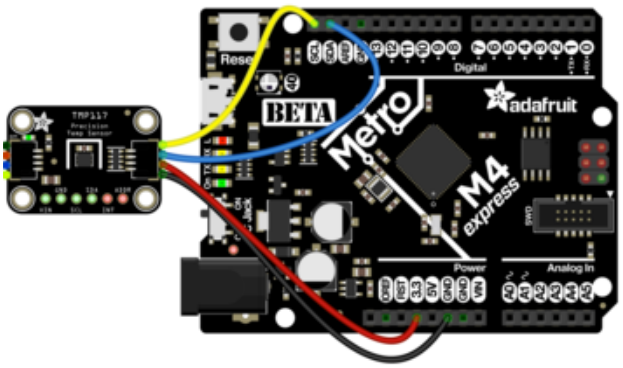
# Arduino

Using the TMP117 with Arduino is a simple matter of wiring up the sensor to your Arduino-compatible microcontroller, installing the Adafruit TMP117 () library we've written, and running the provided example code.

## I2C Wiring

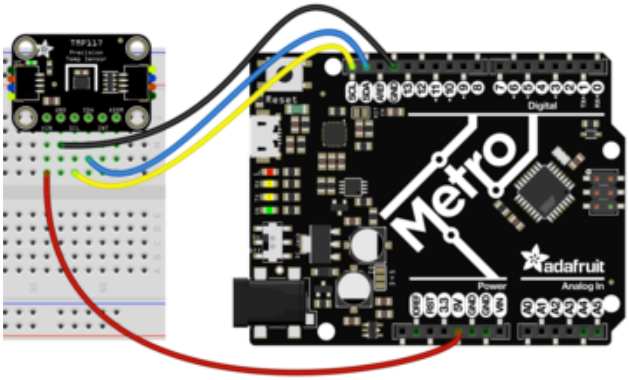
Use this wiring if you want to connect via I2C interface. The default I2C address for the TMP117 is 0x48 but it can be switched to 0x49 by pulling the address pin high to VCC.

Here is how to wire up the sensor using one of the STEMMA QT () connectors. The examples show a Metro but wiring will work the same for an Arduino or other compatible board.




Connect board VIN (red wire) to Arduino 5V if you are running a 5V board Arduino (Uno, etc.). If your board is 3V, connect to that instead.
Connect board GND (black wire) to Arduino GND
Connect board SCL (yellow wire) to Arduino SCL
Connect board SDA (blue wire) to Arduino SDA

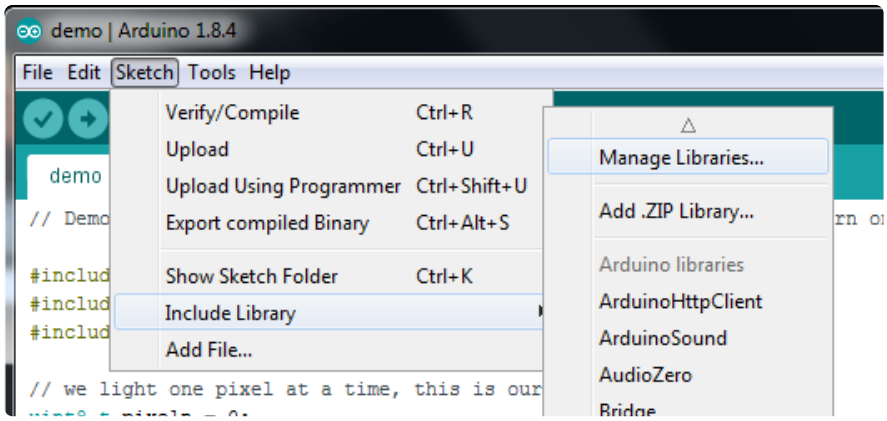
Here is how to wire the sensor to a board using a solderless breadboard:



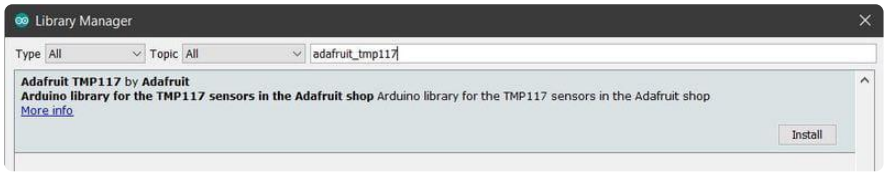

Connect board VIN (red wire) to Arduino 5V if you are running a 5V board Arduino (Uno, etc.). If your board is 3V, connect to that instead.
Connect board GND (black wire) to Arduino GND
Connect board SCL (yellow wire) to Arduino SCL
Connect board SDA (blue wire) to Arduino SDA

## Library Installation

You can install the Adafruit TMP117 library for Arduino using the Library Manager in the Arduino IDE.

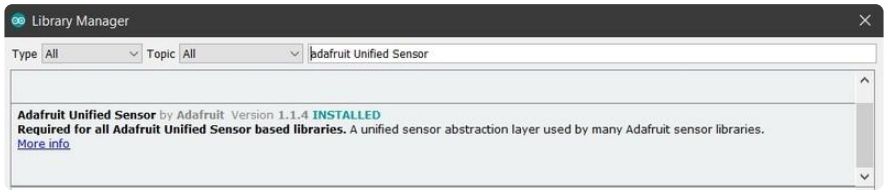


Click the Manage Libraries ... menu item, search for Adafruit TMP117 , and select the A dafruit TMP117 library:



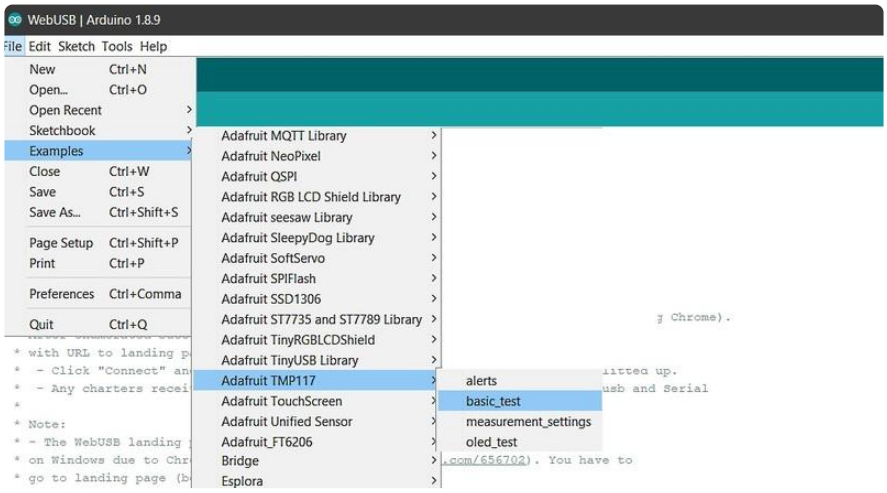

Follow the same process for the Adafruit BusIO library.


Finally follow the same process for the Adafruit Unified Sensor library:




## Load Basic Example

Open up File -> Examples ->Adafruit TMP117 -> basic_test




After opening the demo file, upload to your Arduino wired up to the sensor. Once you upload the code, you will see the Temperature values being printed when you open the Serial Monitor (Tools->Serial Monitor) at 115200 baud, similar to this:

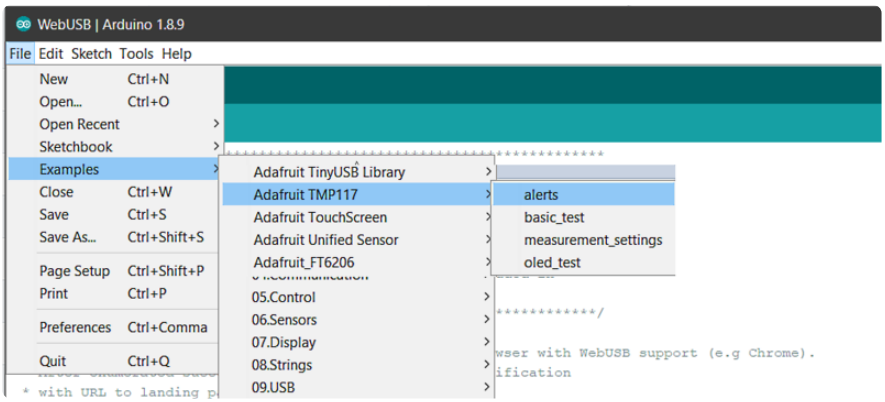
## Basic Example Code


```
/**
 * @file basic_test.ino
 * @author Bryan Siepert for Adafruit Industries
 * @brief Shows how to specify a
 * @date 2020-11-10
 *
 * @copyright Copyright (c) 2020
 *
 */
#include <Wire.h>
#include <Adafruit_TMP117.h>
#include <Adafruit_Sensor.h>

Adafruit_TMP117  tmp117;
void setup(void) {
  Serial.begin(115200);
  while (!Serial) delay(10);     // will pause Zero, Leonardo, etc until serial
console opens
  Serial.println("Adafruit TMP117 test!");

  // Try to initialize!
  if (!tmp117.begin()) {
    Serial.println("Failed to find TMP117 chip");
    while (1) { delay(10); }
  }
  Serial.println("TMP117 Found!");

}
void loop() {

  sensors_event_t temp; // create an empty event to be filled
  tmp117.getEvent(&temp); //fill the empty event object with the current
measurements
  Serial.print("Temperature  "); Serial.print(temp.temperature);Serial.println("
degrees C");
  Serial.println("");

  delay(1000);
}
```

In addition to the basic examples, we have a few other options in the examples directory, including one for using the build in alerts.

## Load Alerts Example

Open up File -> Examples -> Adafruit TMP117 -> alerts




You may want to adjust the high and low temperature limits to fit your conditions to see how they trigger. If the default values don't work for your setup, changing the argument for `setLowThreshold` and `setHighThreshold` will specify a new temperature to use.

```
// You may need to adjust these thresholds to fit the temperature range of where
the test is
// being run to be able to see the alert status change.
tmp117.setHighThreshold(35.0);
Serial.print("High threshold: "); Serial.println(tmp117.getHighThreshold(), 1);
tmp117.setLowThreshold(28.5);
Serial.print("Low threshold: "); Serial.println(tmp117.getLowThreshold(), 1);
```

Once any adjustments are made, the file should be compiled and uploaded to your connected board. Once finished, you can see the temperature being reported along with the states of the two alerts.

```
Adafruit TMP117 test!
TMP117 Found!
Therm mode enabled: False
High threshold: 35.0
Low threshold: 28.5
Alerts are active when the INT pin is HIGH


Temperature: 29.10 degrees C
High temperature alert active:False
Low temperature alert active:False
```

Once you've got the alerts triggering, try changing the code below to enable or disable "Therm" mode which makes the thresholds work more like a target temperature and hysteresis value to designate the lower bounds of the acceptable temperature range.

```
// Set the enable flag below to see how the low temp limit can be used as a
// hysteresis value that defines the acceptable range for the temperature values
where
// the high temp alert is not active
tmp117.thermAlertModeEnabled(true);
```

## Alerts Example Code

```
/**
 * @file alerts.ino
 * @author Bryan Siepert for Adafruit Industries
 * @brief Show how to set adjust and use the sensor's included alert settings
 * @date 2020-11-10
 *
 * @copyright Copyright (c) 2020
 *
 */
#include <Adafruit_SSD1306.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TMP117.h>
#include <Wire.h>

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

Adafruit_TMP117 tmp117;
void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens
  Serial.println("Adafruit TMP117 test!");

  // Try to initialize!
  if (!tmp117.begin()) {
    Serial.println("Failed to find TMP117 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("TMP117 Found!");

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ; // Don't proceed, loop forever
  }
  display.clearDisplay();
  display.display();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setRotation(0);

  // Set the enable flag below to see how the low temp limit can be used as a
  // hysteresis value that defines the acceptable range for the temperature values
where
  // the high temp alert is not active
  // tmp117.thermAlertModeEnabled(true);
  Serial.print("Therm mode enabled: ");
  if (tmp117.thermAlertModeEnabled()) {
    Serial.println("True");
  } else {
    Serial.println("False");
  }

  // You may need to adjust these thresholds to fit the temperature range of where
the test is
  // being run to be able to see the alert status change.
  tmp117.setHighThreshold(35.0);
  Serial.print("High threshold: "); Serial.println(tmp117.getHighThreshold(), 1);
  tmp117.setLowThreshold(28.5);
  Serial.print("Low threshold: "); Serial.println(tmp117.getLowThreshold(), 1);

  // tmp117.interruptsActiveLow(false);
```

```
  if(tmp117.interruptsActiveLow()){
    Serial.println("Alerts are active when the INT pin is LOW");
  } else {
    Serial.println("Alerts are active when the INT pin is HIGH");
  }

  Serial.println("");
  Serial.println("");
}

void loop() {
  display.clearDisplay();
  display.setCursor(0, 0);

  tmp117_alerts_t alerts;
  sensors_event_t temp;
  // Reading temp clears alerts, so read alerts first
  tmp117.getAlerts(&alerts); // get the status of any alerts
  tmp117.getEvent(&temp);   // get temperature

  Serial.print("Temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degrees C");

  Serial.print("High temperature alert active:");
  if (alerts.high) {
    Serial.println("True");
  } else {
    Serial.println("False");
  }

  Serial.print("Low temperature alert active:");
  if (alerts.low) {
    Serial.println("True");
  } else {
    Serial.println("False");
  }
  Serial.println("");

  //       Print to OLED
  display.print("Tmp:");
  display.print(temp.temperature, 1);
  display.println(" C");

  display.print("HI:");
  if (alerts.high) {
    display.print("Y");
  } else {
    display.print("N");
  }

  display.print(" LOW:");
  if (alerts.low) {
    display.println("Y");
  } else {
    display.println("N");
  }

  display.display();
  delay(300);
}
```

# Arduino Docs

Arduino Docs ()

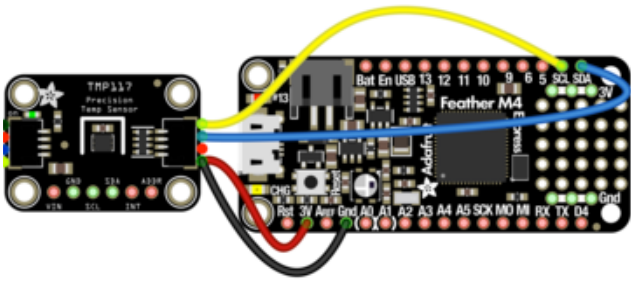
# Python & CircuitPython

It's easy to use the TMP117 with Python or CircuitPython, and the Adafruit CircuitPython TMP117 () module. This module allows you to easily write Python code that reads the temperature from the TMP117 sensor.

You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library ().

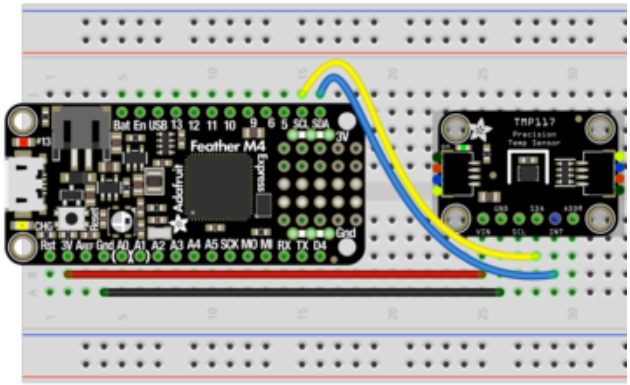
## CircuitPython Microcontroller Wiring

First wire up a TMP117 to your board exactly as shown below. Here's an example of wiring a Feather M4 to the sensor with I2C using one of the handy STEMMA QT () connectors:



Board 3V to sensor VIN (red wire)
Board GND to sensor GND (black wire)
Board SCL to sensor SCL (yellow wire)
Board SDA to sensor SDA (blue wire)

You can also use the standard 0.100" pitch headers to wire it up on a breadboard:

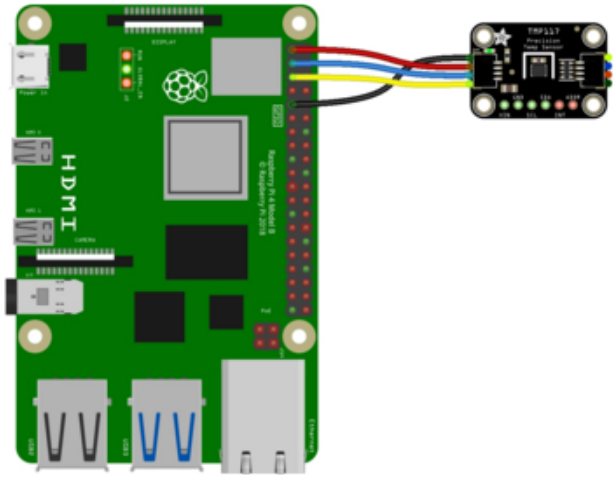


Board 3V to sensor VIN (red wire)
Board GND to sensor GND (black wire)
Board SCL to sensor SCL (yellow wire)
Board SDA to sensor SDA (blue wire)

## Python Computer Wiring

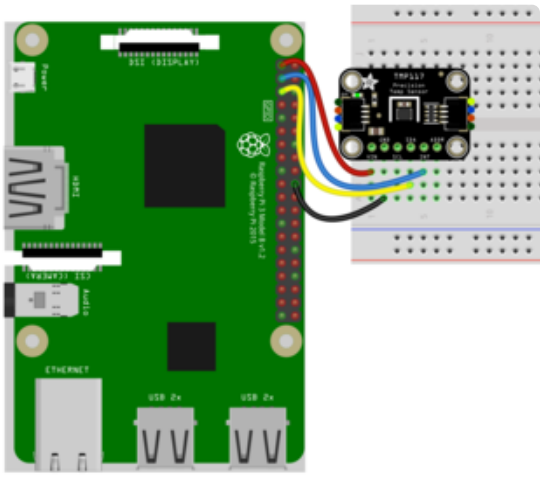
Since there's dozens of Linux computers/boards you can use, we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported ().

Here's the Raspberry Pi wired to the sensor using I2C and a STEMMA QT () connector:



Pi 3V to sensor VCC (red wire)
Pi GND to sensor GND (black wire)
Pi SCL to sensor SCL (yellow wire)
Pi SDA to sensor SDA (blue wire)

Finally here is an example of how to wire up a Raspberry Pi to the sensor using a solderless breadboard



Pi 3V to sensor VCC (red wire)
Pi GND to sensor GND (black wire)
Pi SCL to sensor SCL (yellow wire)
Pi SDA to sensor SDA (blue wire)

## CircuitPython Installation of TMP117 Library

You'll need to install the Adafruit CircuitPython TMP117 () library on your CircuitPython board.

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 ().  Our CircuitPython starter guide has a great page on how to install the library bundle ().

Before continuing make sure your board's lib folder or root filesystem has the adafruit_TMP117.mpy file and adafruit_bus_device folder copied over.

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

## Python Installation of TMP117 Library

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready ()!

Once that's done, from your command line run the following command:

- sudo pip3 install adafruit-circuitpython-TMP117

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

## Basic CircuitPython & Python Usage

To demonstrate the usage of the sensor we'll initialize it and read the temperature measurements from the board's Python REPL.

Run the following code to import the necessary modules and initialize the I2C connection with the sensor:

```
import time
import board
import adafruit_tmp117

i2c = board.I2C()
tmp117 = adafruit_tmp117.TMP117(i2c)
```

```
>>> import time
>>> import board
>>> import adafruit_tmp117
>>> i2c = board.I2C()
>>> tmp117 = adafruit_tmp117.TMP117(i2c)
```

Now you're ready to read values from the sensor using the temperature property:

- temperature - The temperature measured by the sensor, a value in degrees Celsius.

```
print("Temperature: %.2f degrees C" % tmp117.temperature)
```

```
>>> print("Temperature: %.2f degrees C" % tmp117.temperature)
Temperature: 24.29 degrees C
>>>
```

## Basic Example Code

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
import adafruit_tmp117

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
tmp117 = adafruit_tmp117.TMP117(i2c)

while True:
    print("Temperature: %.2f degrees C" % tmp117.temperature)
    time.sleep(1)
```

## Temperature Alerts Example

Once you've tried out the basic example, you can play with setting and checking high and low temperature alerts. Use the code above to set up a TMP117 instance in the REPL and then use the `high_limit` and `low_limit` properties to set alerts for high and low temperatures:

```
tmp117.high_limit = 25
tmp117.low_limit = 10
```

The above will set the alert temperatures to 25 and 10 degrees C respectively. You may wish to change these values to suit your needs.

```
>>> tmp117.high_limit = 25
>>> tmp117.low_limit = 10
```

Once the limits are set, we can check them using the `alert_status` property to read the state of the temperature limits:

```
alert_status = tmp117.alert_status
print("High alert:", alert_status.high_alert)
print("Low alert:", alert_status.low_alert)
```

```
>>> alert_status = tmp117.alert_status
>>> print("High alert:", alert_status.high_alert)
High alert: True
>>> print("Low alert:", alert_status.low_alert)
Low alert: False
```

## Temperature Alerts Example Code

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
from adafruit_tmp117 import TMP117, AlertMode

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller

tmp117 = TMP117(i2c)

tmp117.high_limit = 25
tmp117.low_limit = 10

print("\nHigh limit", tmp117.high_limit)
print("Low limit", tmp117.low_limit)

# Try changing `alert_mode`  to see how it modifies the behavior of the alerts.
# tmp117.alert_mode = AlertMode.WINDOW #default
# tmp117.alert_mode = AlertMode.HYSTERESIS

print("Alert mode:", AlertMode.string[tmp117.alert_mode])
print("\n\n")
while True:
    print("Temperature: %.2f degrees C" % tmp117.temperature)
    alert_status = tmp117.alert_status
    print("High alert:", alert_status.high_alert)
    print("Low alert:", alert_status.low_alert)
    print("")
    time.sleep(1)
```

---

# Python Docs

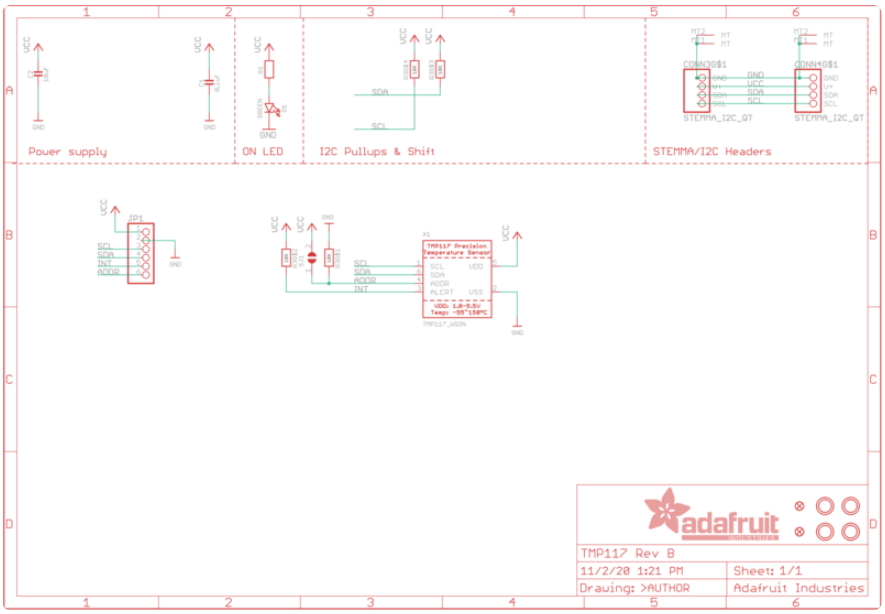
Python Docs ()

# Downloads

## Files

- TMP117 Datasheet ()
- EagleCAD files on GitHub ()
- Fritzing object in the Adafruit Fritzing Library ()

## Schematic




## Fab Print