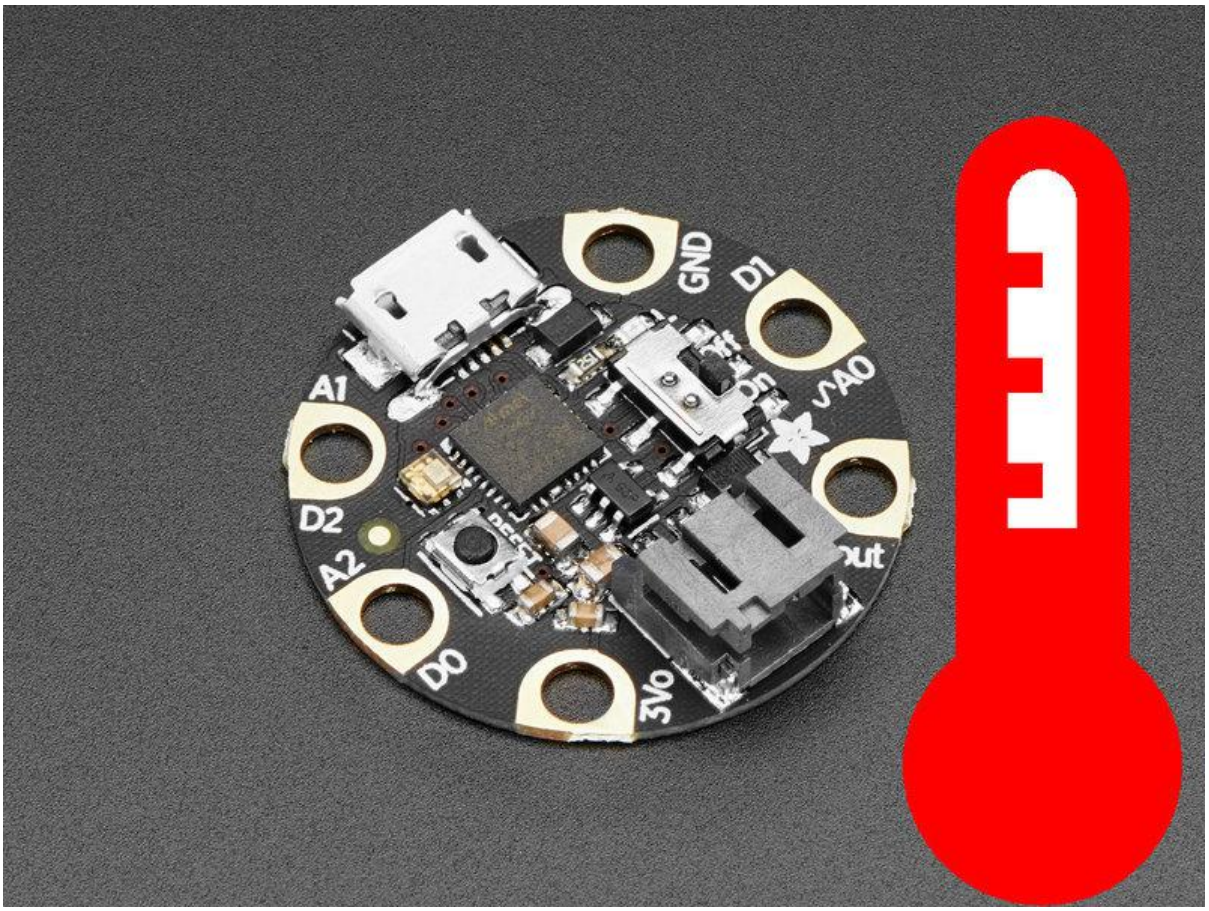
CPU Temperature Logging with CircuitPython

Created by Dan Conley

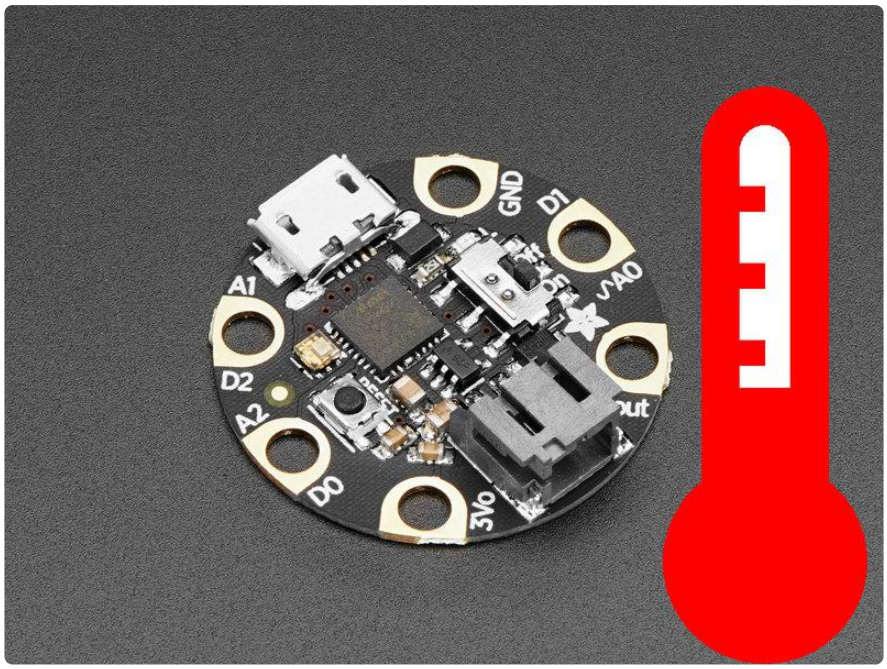


https://learn.adafruit.com/cpu-temperature-logging-with-circuit-python

Last updated on 2023-08-29 03:26:34 PM EDT

Table of Contents

Introduction



CircuitPython () (a derivative of MicroPython ()) is designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads.

Version 2.0.0 introduced a lot of features, among which were the `storage` module (allows CircuitPython to save data to the internal filesystem) and `microcontroller.cpu.temperature` object. So, let's put the two together and create a very tiny temperature logger!

Be sure you have CircuitPython 2.0.0 or above on your device!

2.0.0 was released September 12, 2017, so if you bought your board before or around that time there's a good chance it needs updating. Download the update and follow the instructions on the 2.0.0 release page ().

We'll be using an Adafruit Gemma M0 () for this guide but any CircuitPython-capable device that uses the SAMD21 can run this code. If you are not using SAMD21 you will be missing the cpu temperature capability so just wire up an external temp sensor

Getting the temperature

One of the new packages included in 2.0.0 is `microcontroller.cpu.temperature`: this gives you the temperature (in Celsius) of the board's CPU. It isn't an accurate

room temperature sensor like the TMP36 () or DHT22 (), since it's the temperature of the CPU, not the ambient air, but it's still a data point and it is very accurate!

Getting the temperature is as simple as can be. In the REPL ():

```
Adafruit CircuitPython 2.0.0 on 2017-09-12; Adafruit Gemma M0 with samd21e18
&gt;&gt;&gt; import microcontroller
&gt;&gt;&gt; microcontroller.cpu.temperature
29.6556
```

If you'd like it in Fahrenheit, you can create a function for that.

```
&gt;&gt;&gt; def fahrenheit(celsius):
...     return (celsius * 9 / 5) + 32
...
&gt;&gt;&gt; fahrenheit(microcontroller.cpu.temperature)
90.1077
```

Once you've got all that working, we can go on to the next step, storage!

Writing to the filesystem

Getting the temperature in the REPL is all well and good, but requires you to type the command every time.

Thankfully, CircuitPython 2.0.0 also introduced the `storage` module, which lets you access the internal storage of the board. There isn't a lot of it, but for storing a few readings it should be plenty. By default you can write to the system via USB (ie saving code.py) and not in code, so first you'll need to change that.

Setting readonly to False on boot

You can only use this in boot.py, which is executed before the USB connection is made. If boot.py doesn't exist, create it with this:

Warning: once you start this you need to continue until the end of the section or you'll have problems writing code to the board. If you'd prefer to be safe, just read this and skip to the next section, "Selectively setting readonly to False on boot".

```
import storage

storage.remount("/", False)
```

On every boot the root filesystem will be mounted so that CircuitPython can write to it.

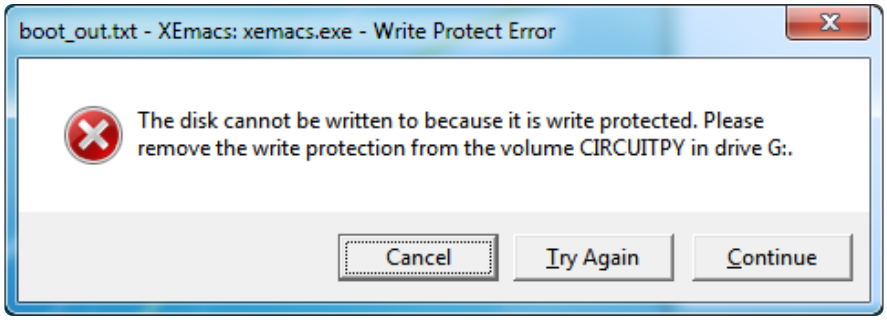
boot.py only runs on first boot of the device, not if you re-start the REPL with ^D or if you save the file, so you must EJECT the USB drive, then physically press the reset button!

You can now write to the internal storage via the REPL:

```
&gt;&gt;&gt; with open("/tmp.txt", "a") as fp:
...     fp.write("hello, world!")
...
```

You might need to reboot the board before you see the file, but it will be there in the file explorer.

Only one thing can have write access at a time, though, so by allowing your Python code to write to the device you've disabled USB write access. This means updating c ode.py will no longer work! Even worse, you can't edit boot.py either, so at first you might think you're stuck like this forever.



Luckily, you can edit and remove/rename files via the REPL:

```
&gt;&gt;&gt; import os
&gt;&gt;&gt; os.listdir("/")
&gt;&gt;&gt; os.rename("/boot.py", "/boot.bak")
```

Then reboot the device and you'll be able to edit via USB as normal.

Selectively setting readonly to False on boot

Recreate boot.py:

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT
```

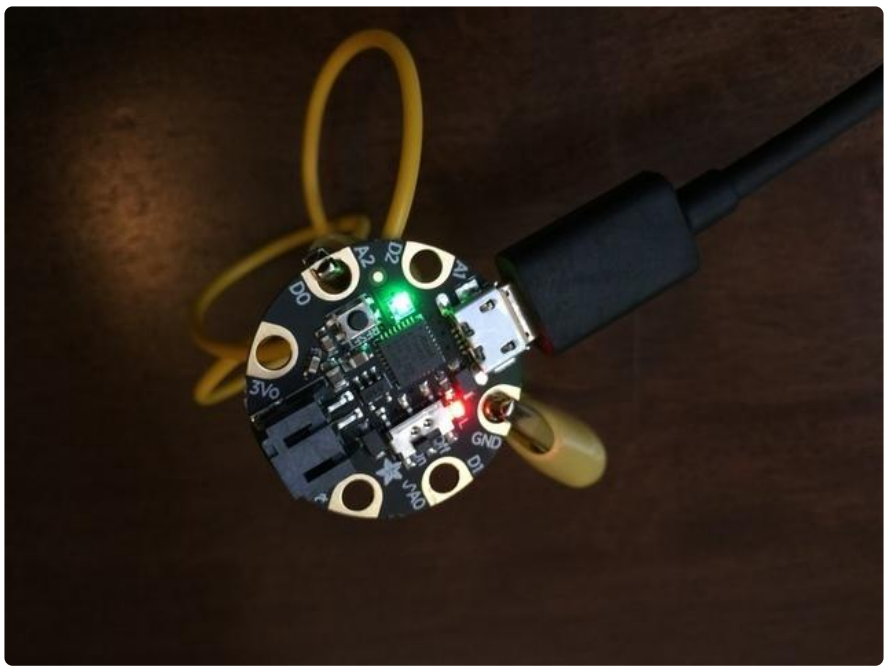
```
import board
import digitalio
import storage

switch = digitalio.DigitalInOut(board.D0)
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

# If the D0 is connected to ground with a wire
# CircuitPython can write to the drive
storage.remount("/", switch.value)
```

This will read the value of the D0 pin, which has been set to a pullup: it reads `True` (`HIGH`, `1`, etc in Arduino) if it has not been grounded, but if connected to ground it reads `False`. Since we want it to be readonly False when the board should be written by the code and not USB, you only need to connect the D0 pin to ground when you want the board to be able to write via the code.

boot.py only runs on first boot of the device, not if you re-start the REPL with ^D or if you save the file, so you must EJECT the USB drive, then physically press the reset button!

The Circuit Playground makes this easy: the D7 pin is the toggle switch.

On other boards, like the Gemma M0, you'll need to use wires or alligator clips like so:



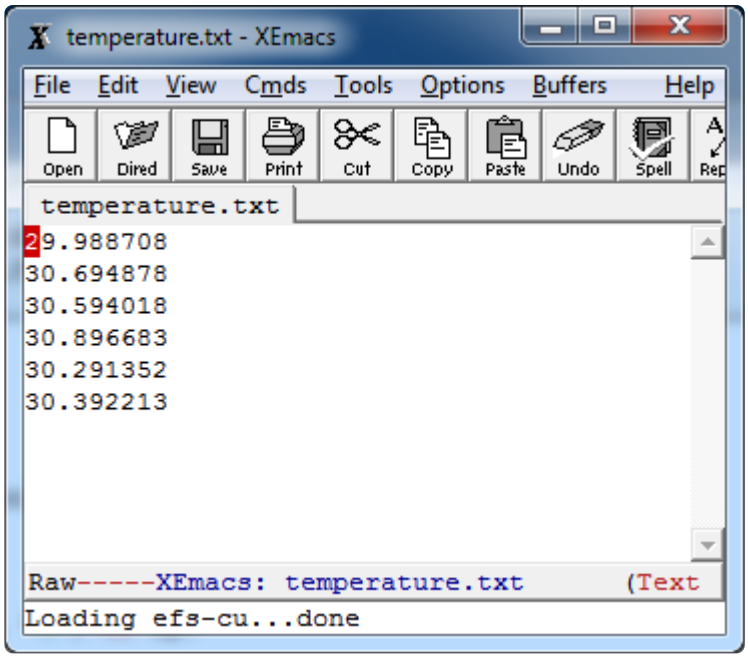
Logging the temperature

Let's put it all together: make this your code.py:

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time

import board
import digitalio
import microcontroller

led = digitalio.DigitalInOut(board.D13)
led.switch_to_output()

try:
    with open("/temperature.txt", "a") as fp:
        while True:
            temp = microcontroller.cpu.temperature
            # do the C-to-F conversion here if you would like
            fp.write('{0:f}\n'.format(temp))
            fp.flush()
            led.value = not led.value
            time.sleep(1)
except OSError as e:
    delay = 0.5
    if e.args[0] == 28:
        delay = 0.25
    while True:
        led.value = not led.value
        time.sleep(delay)
```

This code creates a `led` variable, sets it to the D13 pin and configures it for output (this is the built-in LED pin). Then it opens the temperature.txt file for appending (so future reboots add to the end of the file instead of overwriting it), gets the temperature and writes it to the file with a line break after each reading (on Windows, some editors like Notepad won't recognize the line ending). The LED blinks on and off in a two second loop: each change indicates a value has been written to the file.

There could be an error opening the file for writing, or for writing the file: maybe your board doesn't have D0 pulled low to enable writing. Maybe your internal storage is out of space. The `except` block handles an `OSError` exception. If the error code is 28 that means the device is out of space. The LED will blink four times a second to indicate this. Otherwise the "issue" is probably that the board set to read-only (which is probably by design!) and will blink twice a second.



Keep going

You now have code that will change the read/write status of the drive depending on a pin state, log the temperature once a second a provide three different status indicators. But there could be more to do, if you'd like.

Maybe you want to add better error reporting (the OSError error code for a read only device is 30). Or log differently, or log to a different file. This was only a starting point, so make it your own!

Acknowledgements

The code from this guide is based on code from @jerryneedell () and @edgecollective ().