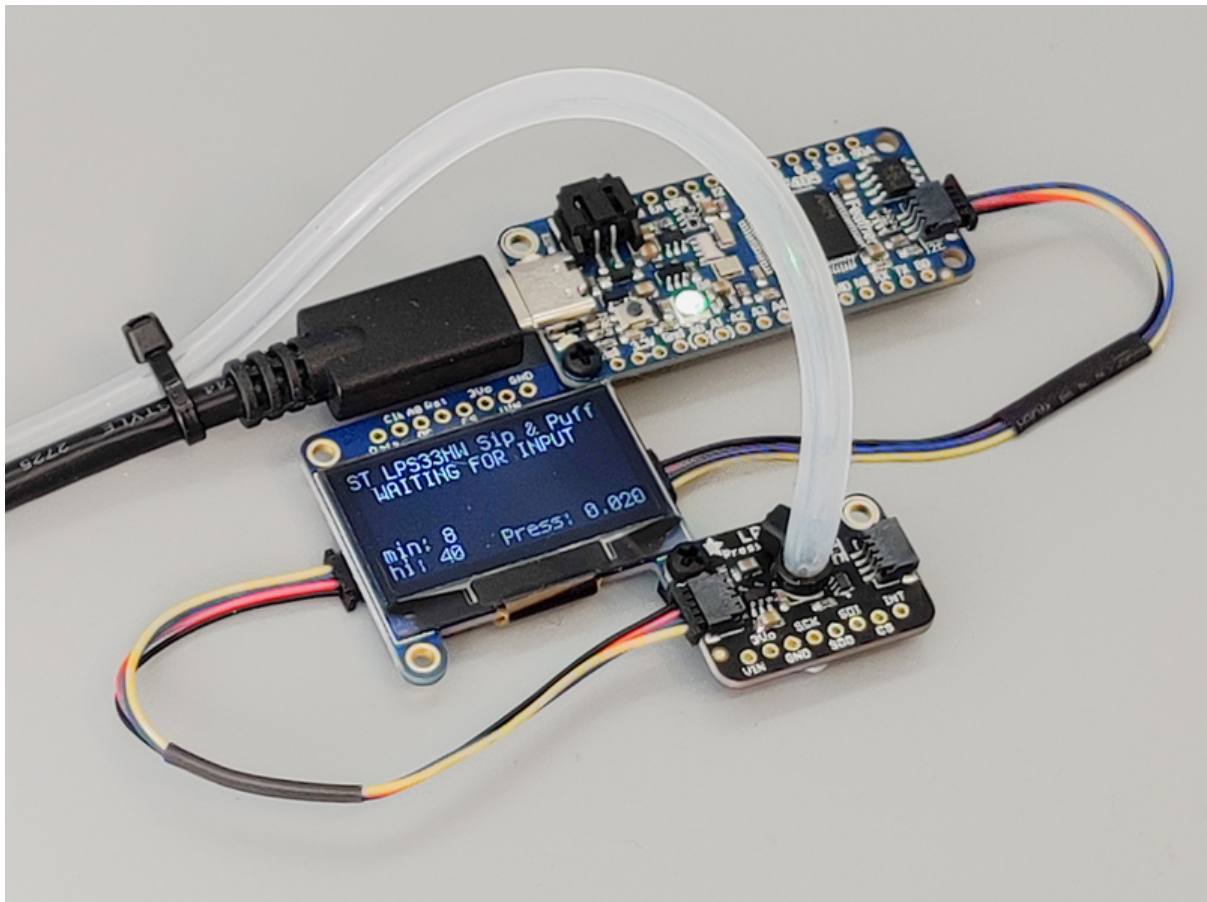




CircuitPython Powered Sip & Puff with ST LPS33HW Pressure Sensor

Created by Bryan Siepert



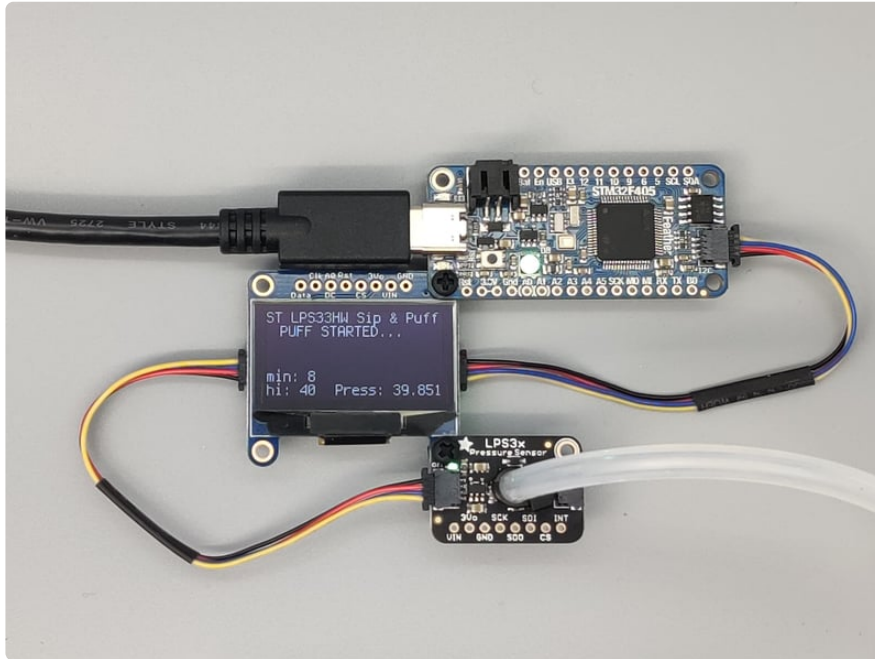
<https://learn.adafruit.com/st-lps33-and-circuitpython-sip-and-puff>

Last updated on 2026-02-03 10:52:44 PM UTC

Table of Contents

Overview	3
<ul style="list-style-type: none">• Parts and Materials• Pick a Cable• Optional Tool: Flush Cutters• Optional Fasteners:	
Assemble the Sensor	8
Plug it all Together	12
<ul style="list-style-type: none">• Screwing the Boards Together• Plug and Play• Optional but Suggested	
CircuitPython Setup	15
<ul style="list-style-type: none">• CircuitPython Notes	
Installing the Mu Editor	17
<ul style="list-style-type: none">• Download and Install Mu• Starting Up Mu• Using Mu	
CircuitPython Code	20
<ul style="list-style-type: none">• Installing Project Code• Sip and Puff code on GitHub	
Usage	22
<ul style="list-style-type: none">• Using the Sip and Puff Code• Detecting Sips• Detecting Puffs• Configuring the Pressure Thresholds	
Extending the Capabilites	25
<ul style="list-style-type: none">• Call Me Maybe?• Make it Your Own	

Overview



Sip and Puff (SNP) devices are one example of Assistive Technology that allow one to interact with things by applying negative or positive pressure to a straw or tube in their mouth. This is done by either sipping on the tube, like a straw, or puffing into it, like when you blow bubbles in your milk until your parental unit (or partner) makes you stop.

Sip and Puff devices have been around for a while. They have helped many people who aren't able to use things like keyboards, mice or other input devices that were designed to be used with hands. There are a number of commercial options, however a newer trend is towards people and groups developing open source designs. These can be made with off the shelf and 3D printed parts by gathering the materials, assembling them, and programming them with open source software.

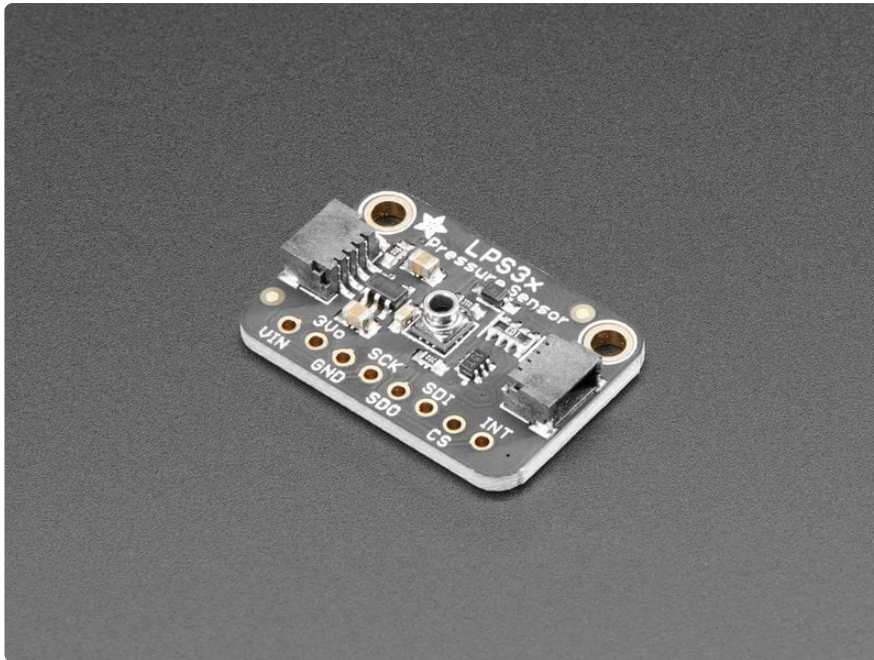


[There are many open source sip and puff devices around \(https://adafru.it/19uc\)](https://adafru.it/19uc) A well-known example is [LipSync \(https://adafru.it/J2D\)](https://adafru.it/J2D) which was developed by [Makers Making Change \(https://adafru.it/J2E\)](https://adafru.it/J2E), and released by the [Neil Squire Society \(https://adafru.it/J2F\)](https://adafru.it/J2F) with support from [Google.org \(https://adafru.it/J3a\)](https://adafru.it/J3a). Lots of technical detail can be found on the project's [GitHub page \(https://adafru.it/J3b\)](https://adafru.it/J3b). The LipSync also integrates a joystick and mount for attaching to wheelchairs, and works as an input device to allow people operate a touchscreen device.

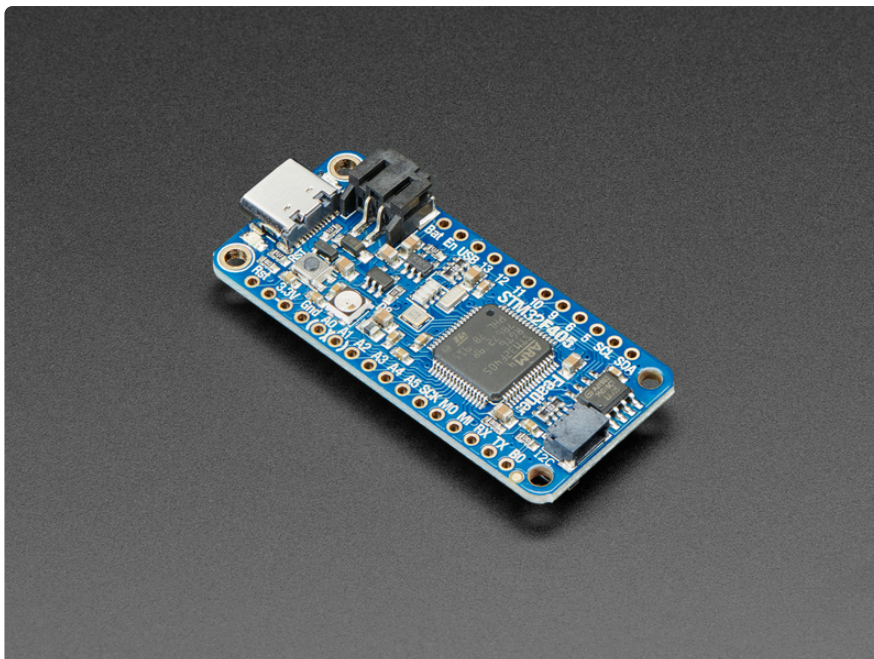
Here, we show how with the right sensors, microcontrollers, and software, it can be relatively easy to make a very basic but expandable Sip and Puff device. It is a fun, easy project that you can use to learn a bit more about them and how they might be used.



EMMA QT sensors/cables are used, no soldering is required for this project - it's completely plug and play!



The heart of this project is a STMicroelectronics (ST) LPS33HW MEMS pressure sensor mounted to an Adafruit breakout board with built in STEMMA QT connectors and support circuitry. Importantly the LPS33HW has a metal port on top like a hat and a groove for fitting an o-ring. With 24-bit pressure data output, an absolute pressure range of 260-1260 hPa ([hectopascals \(https://adafru.it/J3c\)](https://adafru.it/J3c)), and the ability to work in absolute or relative modes, this sensor is the perfect tool for the job.



The brains of the operation is a ST STM32F405 based express Feather running CircuitPython. The STM32F405 is a fast chip with plenty of memory so it's great for

CircuitPython use, and the STEMMA QT connector on the end makes this a plug-n-play project



Many thanks are due to Bill Binko of [ATMakers.org](https://adafru.it/udY) (<https://adafru.it/udY>) for his work making Assistive Technology more widely accessible, and for taking the time to educate me on Sip and Puffs and Assistive Technology in general. Bill has his own CircuitPython powered Sip and Puff, the [AirTalker](https://adafru.it/J3d) (<https://adafru.it/J3d>), developed with his friend Jim to meet his needs. It even includes [Morse Code](https://adafru.it/Chp) (<https://adafru.it/Chp>) detection!



Parts and Materials

To follow along and make your own ST LPS33HW powered Sip and Puff, you'll need the items below. Additionally if you are inspired to take it further, check out the bits on the Extending the Capabilities page for more ideas.

1 x [Adafruit Feather STM32F405 Express](https://www.adafruit.com/product/4382)

<https://www.adafruit.com/product/4382>

The STM32 Feather that powers the project. Includes a SparkFun QWIIC/ Stemma QT socket

1 x [Adafruit LPS33HW Water Resistant Pressure Sensor - STEMMA QT](https://www.adafruit.com/product/4414) <https://www.adafruit.com/product/4414>

The ST pressure sensor with integrated port to attach the tubing

1 x [Monochrome 1.3" 128x64 OLED graphic display - STEMMA QT / Qwiic](https://www.adafruit.com/product/938) <https://www.adafruit.com/product/938>

Small OLED Display with STEMMA QT Connectors

2 x [STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long](https://www.adafruit.com/product/4210) <https://www.adafruit.com/product/4210>

STEMMA QT Cables to connect the Feather, Sensor and OLED screen

1 x [Silicone Tubing - 1 Meter](https://www.adafruit.com/product/3659) <https://www.adafruit.com/product/3659>

Food Safe Silicone Tubing - The ultimate silly straw!

1 x [Small cable ties](https://www.mcmaster.com/7130k101) <https://www.mcmaster.com/7130k101>

Small plastic cable ties to keep the tubing attached to the sensor

Pick a Cable

The Feather STM32F405 Express requires a USB C cable which you may not have. Pick the one below that fits your computer. Most PCs will have USB Type A. Macs made in the last few years will likely have a USB C port which is also referred to as Thunderbolt.

If you're not sure, take a look at the pictures for each cable and compare them to the ports on your computer before ordering

1 x [USB C to USB C Cable](https://www.adafruit.com/product/4199) <https://www.adafruit.com/product/4199>

A USB C Cable for the Feather STM32F405 for computers with USB C / Thunderbolt ports

1 x [USB Type A to Type C Cable - approx 1 meter](https://www.adafruit.com/product/4474) <https://www.adafruit.com/product/4474>

A USB C Cable for the Feather STM32F405 for computers with traditional USB Type A ports

Optional Tool: Flush Cutters

1 x Flush Diagonal Cutters

<https://www.adafruit.com/product/152>

A very useful tool you'll use all the time. It'll come in handy to cut cable ties and tubing if needed

Optional Fasteners:

The guide shows using these nylon screws, nuts, and standoffs to hold the boards together. Pick these up if you want to take that approach, or find a different M2.5 fastener if you wish to take a different approach.

Available in Black or White

1 x Black Nylon Screw and Stand-off Set – M2.5 Thread

<https://www.adafruit.com/product/3299>

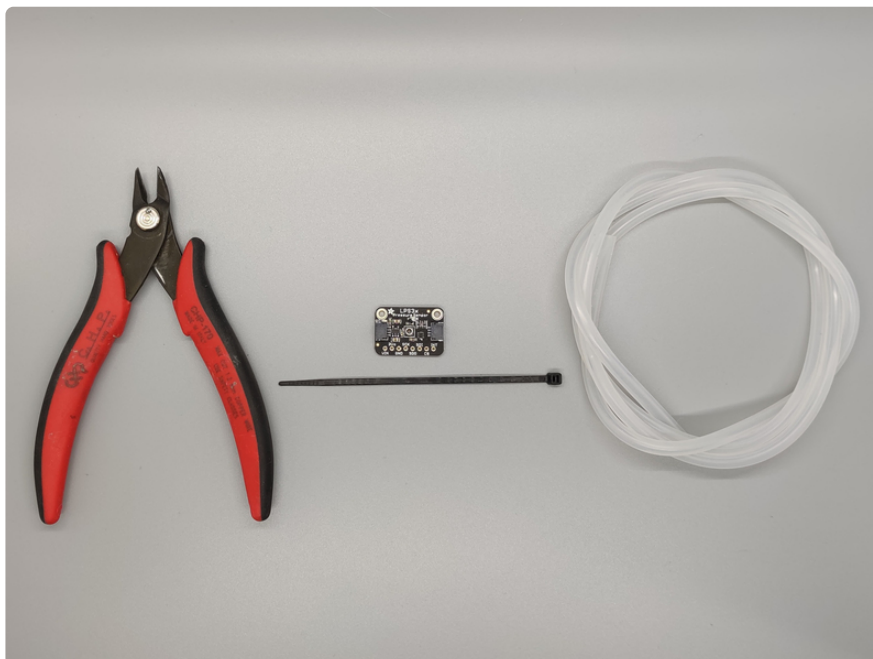
A set of black M2.5 nylon screws, standoffs and nuts.

1 x White Nylon Screw and Stand-off Set – M2.5 Thread

<https://www.adafruit.com/product/3658>

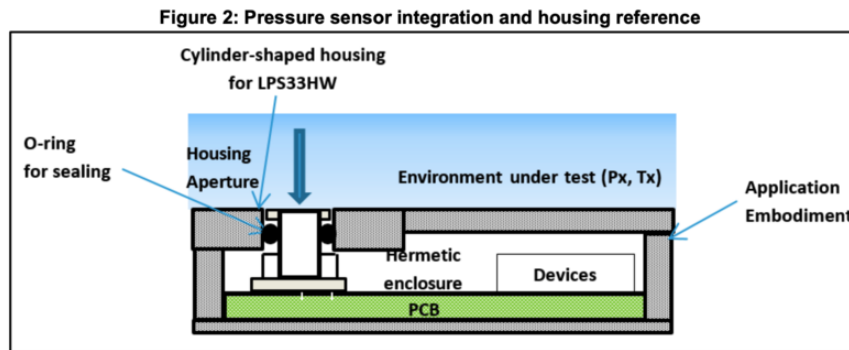
A set of white M2.5 nylon screws, standoffs and nuts.

Assemble the Sensor

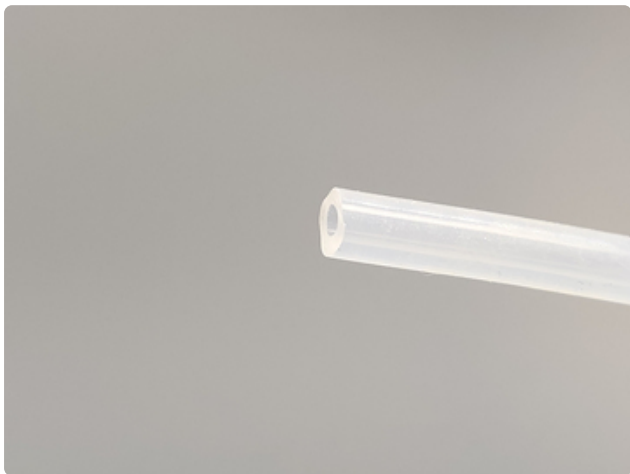


It's assembly time! First off, we're going to affix the tubing to the sensor in order to sip and puff. The port on the LPS33 isn't really meant to have tubing attached directly to

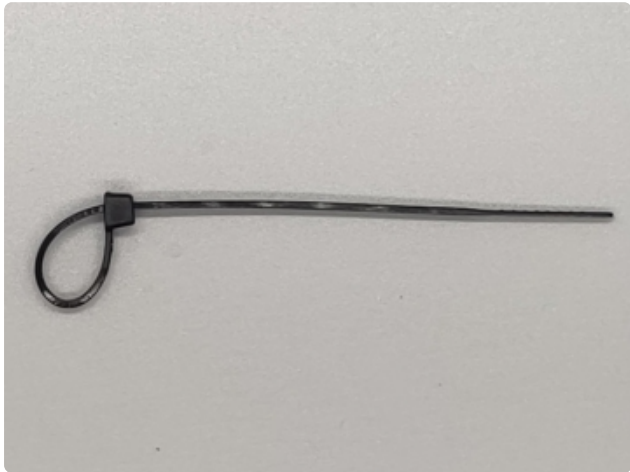
it, but if we take care, we can get a surprisingly robust attachment by using a cable tie to keep the tubing in place and sealed against the port. This is definitely an off label use of the sensor port, as it is meant to be sealed against an enclosure by installing an o-ring in the little groove and fitting the port into a precisely sized hole.



Before we start assembly, you'll want to make sure that the end of your tubing is cut nice and straight. We want the end of the tubing to be flat, at a right angle to tube itself. Having a flat end will allow the tubing to seal properly against the sensor's port and will allow us to get the tubing as close to the circuit board as possible.



You may use flush cutters to cut the end of the tubing nice and straight. If yours are dull, or you don't have any, you can use a nice sharp hobby knife to make a clean cut. Don't saw it, try to make the cut in one stroke.



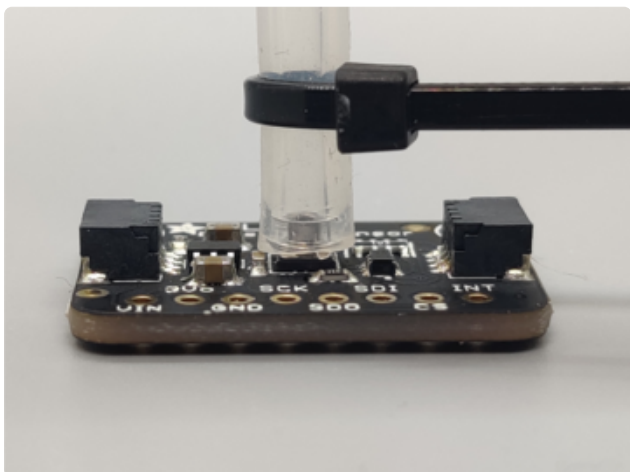
Next, prepare your cable tie by inserting the thin tip into the rectangular end and close the loop nearly all the way but not all the way. You want to leave room to insert the tubing.

If the cable tie doesn't make a zipping sound as you close it, you may have it backwards. When you are done it should have this shape:



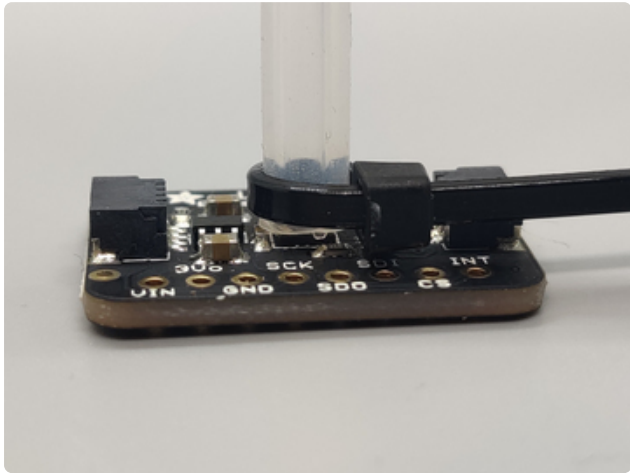
Next, place the cable tie over the assuredly straight end and carefully close it a bit more until the tie is just big enough for the tubing to fit through unimpeded.

Now we'll scoot the tie up the tubing a teeny bit and place the very straight end of the tubing over the little sensor port and carefully push the tubing down as close to the circuit board as you can. It may help to twist the tubing back and forth slightly as you go, kinda like turning a screwdriver back and forth.



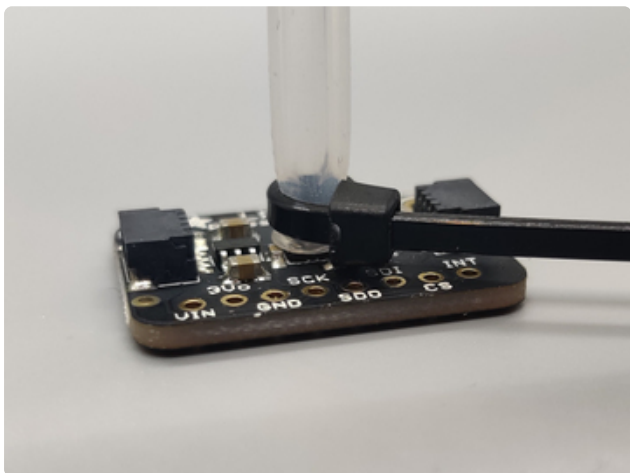
You probably won't be able to get the tubing all the way against the PCB, but you want to make sure there is enough of the port sticking into the tubing that when you tighten the cable tie, you will squish the tubing against the port, not just squeeze the tubing shut.

Once you've got the tubing as far down as you can, hold the tubing in place with one hand while scooting the cable tie back down the tubing towards the board.



While the cable tie is still relatively loose, you may wish to rotate it around the tubing to make sure that you'll be able to pull it tight without bumping into the STEMMA QT connectors or something else on the board. Cable ties will rotate as you tighten them, so keep that in mind.

Using one hand to keep the tube in place, use the other to start tightening the cable tie and get it pretty tight, but not too tight. Remember that this is not how these sensors are meant to be used so we don't want to crank down on it too hard.



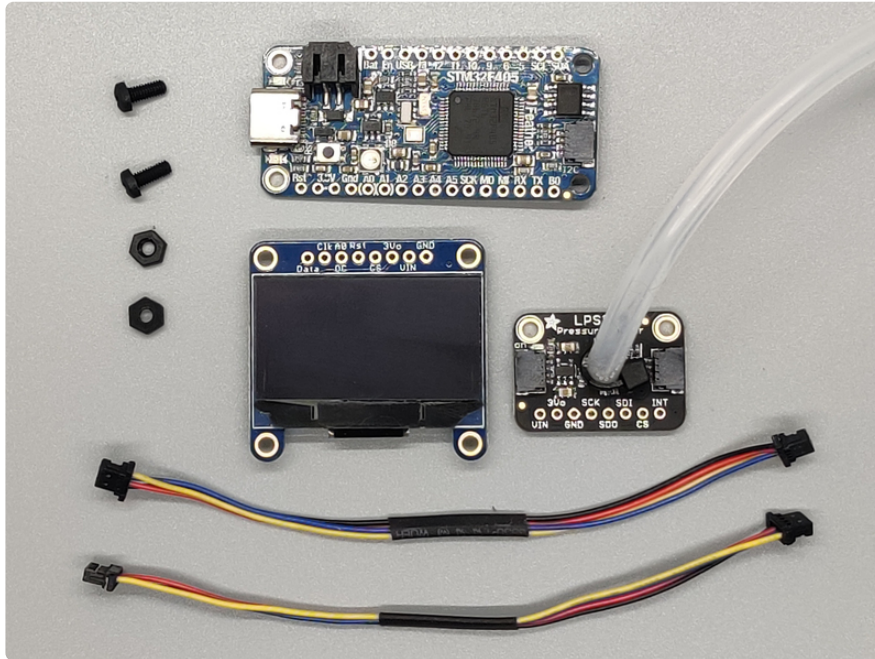
When you are done, viewed from the side the tubing should be squished a bit by the cable tie

Give the tubing a little tug (not a yank) and make sure that it doesn't come off. If it does, pull or cut the cable tie off the end of the tubing and try again. Fortunately, small cable ties like these usually come in packs of 100, so you'll have plenty of times to try.

Once you're happy with it, cut the floppy end of the cable tie off with your flush cutters. I highly recommend using flush cutters for this part because diagonal cutters or other tools will often leave a surprisingly sharp nub at the end. I speak from experience :!

Huzzah! You've successfully assembled the business end of your Sip and Puff.

Plug it all Together



With the tubing on the sensor, next is to assemble the pieces. We'll primarily be plugging together the Feather, LPS33, and OLED screen, however you will want to attach the pieces to something, otherwise moving the tubing will move everything else around, because the other pieces are so light.

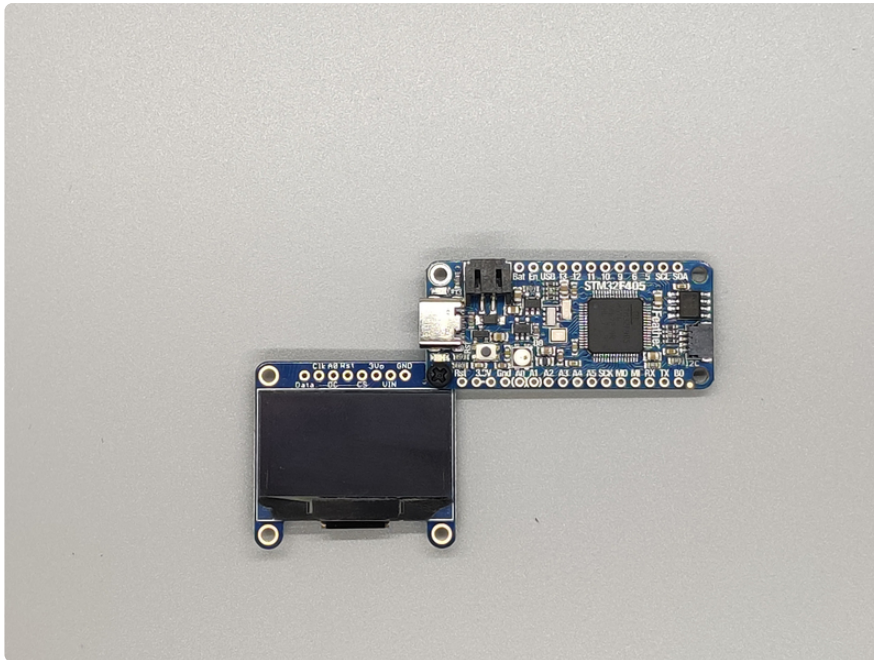
For this simple project, we will use some handy dandy [nylon screws and nuts \(http://adafruit.it/3299\)](http://adafruit.it/3299) to attach the pieces together using the mounting holes. You may choose to mount them all to something else like a small piece of wood, ideally something with a bit of heft to keep it in place.

Screwing the Boards Together

First, place a **6mm M2.5 screw** through the bottom left mounting hole of the Feather.

Next, using your thumb or finger to hold it in place, put the screw through the top right mounting hole on the OLED screen. **Make sure the screen is under the Feather**, otherwise it won't have the clearance needed.

Finally thread a **M2.5 nylon nut** onto the end of the screw and tighten it down to keep it in place. When you're done it should look something like this:



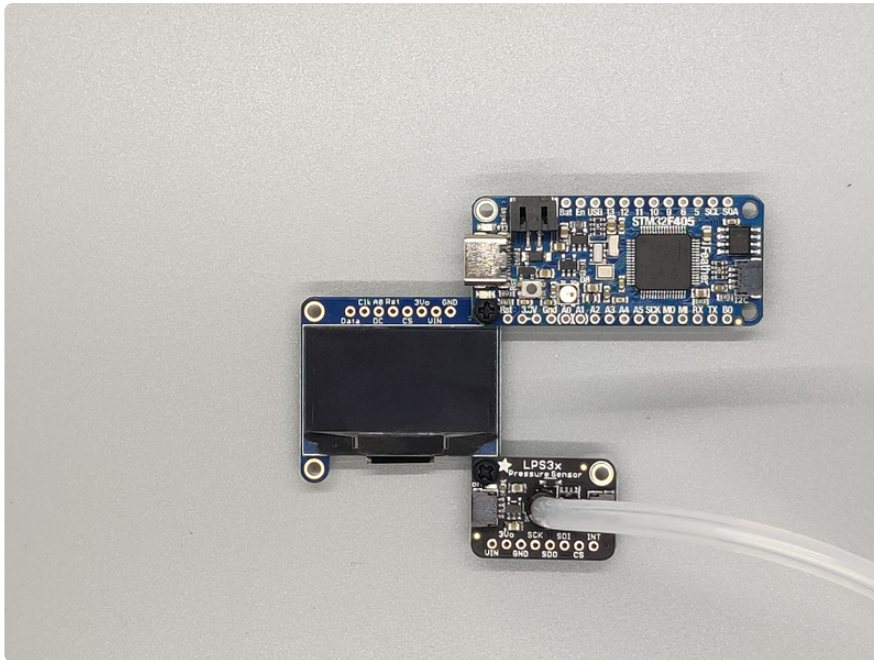
Next we'll repeat the process to attach the sensor to the screen.

Place another M2.5 6mm screw through the top left mounting hole of the **LPS33**. Hold the screw in place with a finger or [tacky putty \(https://adafruit.it/J1F\)](https://adafruit.it/J1F) and then put the screw through the bottom right mounting hole on the OLED.

Keep it all together by screwing your second M2.5 nylon nut onto the screw and tighten it down to keep it all together.

Similar to attaching the screen to the Feather, the order of the boards matters when attaching the sensor to the OLED. **The LPS33 must be on top** or you'll run into clearance issues.

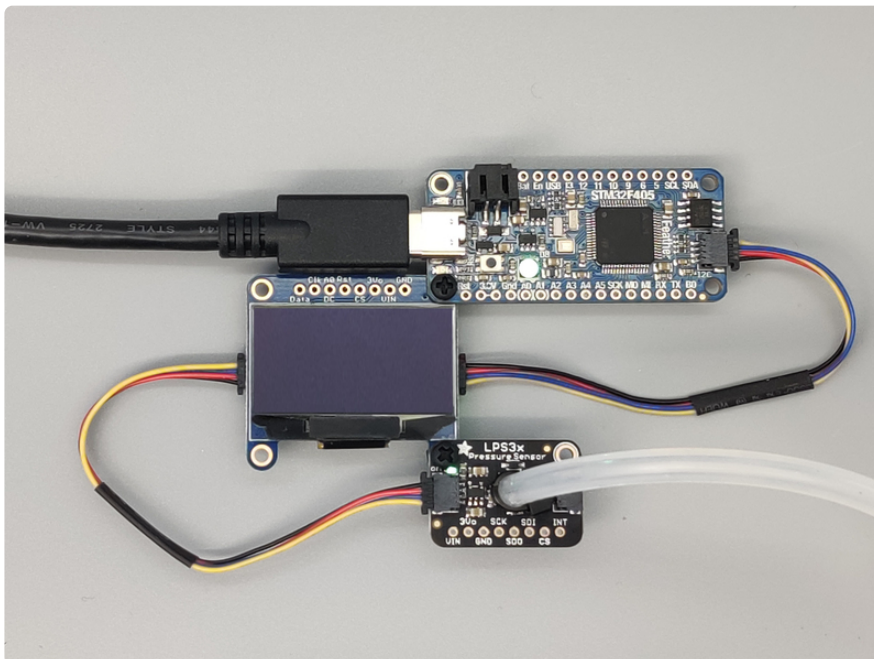
Now you should have something that looks like this:



Plug and Play

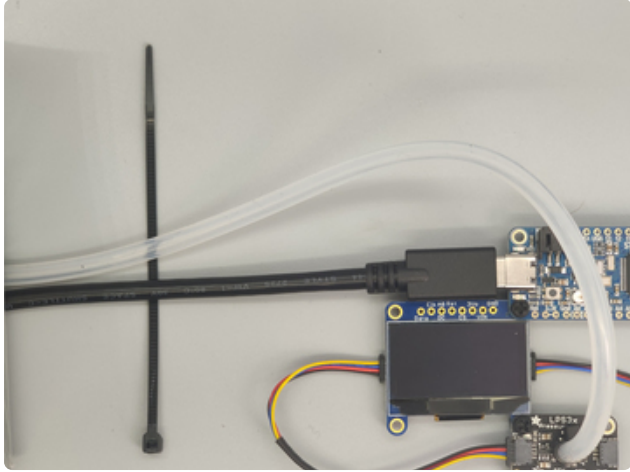
The last step is the easiest: connect the boards together using [SparkFun Qwiic](https://adafruit.com/products/244) (<https://adafruit.com/products/244>) compatible [STEMMA QT](https://adafruit.com/products/244) (<https://adafruit.com/products/244>) cables. As long as everything is connected, the order doesn't matter, but I found the routing below aesthetically pleasing.

Wired together it should look like this:

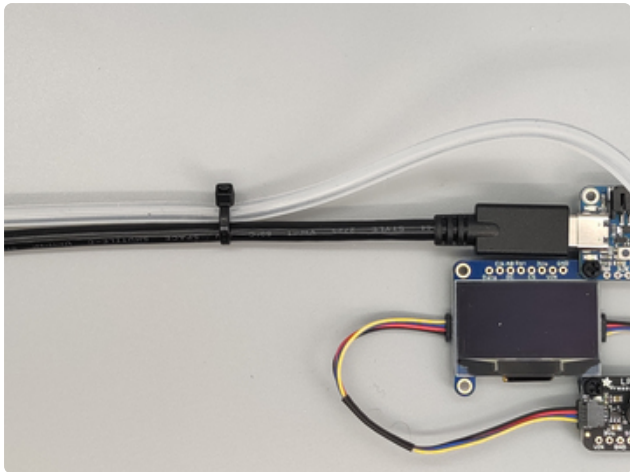


Optional but Suggested

As assembled, everything will work just fine, however I would suggest using another one of your cable ties to hold the tubing against the [USB-C cable \(https://adafruit.it/J2a\)](https://adafruit.it/J2a) that you will use to connect the Feather to your computer.



Simply plug the cable in and fold the tubing over against the cable and use a cable tie to hold the tubing and cable together, making sure to have a gentle bend to the tubing so the airflow isn't blocked.



Giving the tubing a bit of a twist will help curve it out of the way of the screen.

Like when you attached the tubing to the sensor, you want to hold the cable and tubing together snugly but not so hard that you squish the tubing shut (a teeny bit of squish is OK)

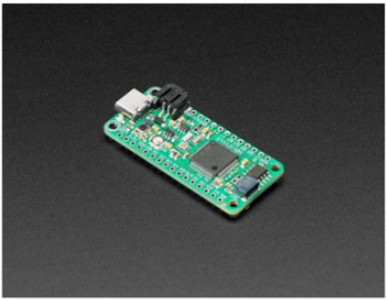
If you are successful, now when your dog or toddler decides to play tug of war with the tubing, it will pull on the comparatively robust cable and not the sensor. This type of connection is called [strain relief \(https://adafruit.it/J2b\)](https://adafruit.it/J2b) and they are commonly used with things like cables that will be tugged on and moved about.

CircuitPython Setup

To load CircuitPython, follow the **DFU Bootloader** instructions to get the board into bootloader mode

Visit https://circuitpython.org/board/feather_stm32f405_express/ (<https://adafruit.it/GDY>) To get the latest firmware available

Feather STM32F405 Express
by Adafruit



ST takes flight in this upcoming Feather board. The new STM32F405 Feather (idea) that we

CircuitPython 5.0.0-alpha.5

This is the latest unstable release of CircuitPython that will work with the Feather STM32F405 Express.

Unstable builds have the latest features but are more likely to have critical bugs.

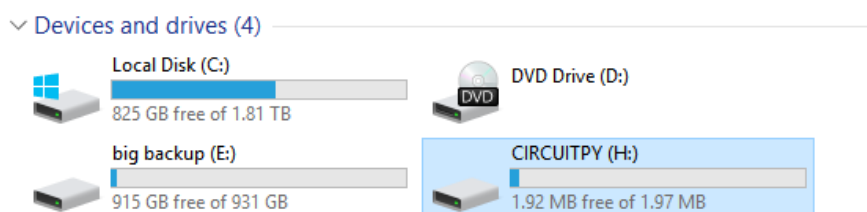
[Release Notes for 5.0.0-alpha.5](#)

ENGLISH

DOWNLOAD .BIN NOW

Download the **bin** file, and then program it using [dfu-util](#) or [STM32CubeProgrammer](https://adafru.it/HOB) (<https://adafru.it/HOB>)

Upon success, reset the board without the **BOOT0** jumper and you will see after a few seconds the **CIRCUITPY** disk drive appear



Next you can visit <https://learn.adafruit.com/welcome-to-circuitpython> (<https://adafru.it/cpy-welcome>) and <https://learn.adafruit.com/circuitpython-essentials/> (<https://adafru.it/BX8>) to learn more about CircuitPython

CircuitPython Notes

If you are intending to start a project that is very RAM intensive, note you cannot access the full 196KB of RAM that listed on the F405 datasheet and website - only 128KB is available to Circuitpython programs for system reasons. You'll find the same limitation on Micropython and most other F405 devices.

STM32F4 support is new compared to the SAMD and nRF boards, but is now considered stable. Working modules on this board include:

- Digital IO (LEDs/buttons)
- analog input
- analog output (DAC)
- PWM output on timer pins
- I2C
- SPI
- [NeoPixel Support \(https://adafru.it/GDZ\)](https://adafru.it/GDZ)
- [UART Support \(https://adafru.it/GD-\)](https://adafru.it/GD-)
- DisplayIO
- PulseIO

To come:

- I2S
- Audio
- TouchIO
- many others!

[If you find something missing or flawed, please open an issue in circuitpython \(https://adafru.it/19sa\)](https://adafru.it/19sa)

Installing the Mu Editor

Mu is a simple code editor that works with the Adafruit CircuitPython boards. It's written in Python and works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in so you get immediate feedback from your board's serial output!

Mu is our recommended editor - please use it (unless you are an experienced user with a favorite editor already). While it has been announced end of life in 2026, it still works fine.

You are free to use whatever text editor you wish along with a terminal program to connect to the CircuitPython REPL. Thonny is one such editor.



Download and Install Mu



Download Mu from <https://codewith.mu> (<https://adafru.it/Be6>).

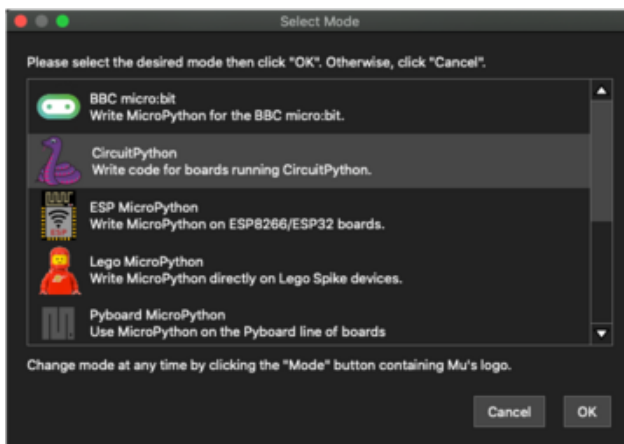
Click the **Download** link for downloads and installation instructions.

Click **Start Here** to find a wealth of other information, including extensive tutorials and and how-to's.



Shows users: due to the nature of MSI installers, please remove old versions of Mu before installing the latest version.

Starting Up Mu



The first time you start Mu, you will be prompted to select your 'mode' - you can always change your mind later. For now please select **CircuitPython!**

The current mode is displayed in the lower right corner of the window, next to the "gear" icon. If the mode says "Microbit" or something else, click the **Mode** button in the upper left, and then choose "CircuitPython" in the dialog box that appears.

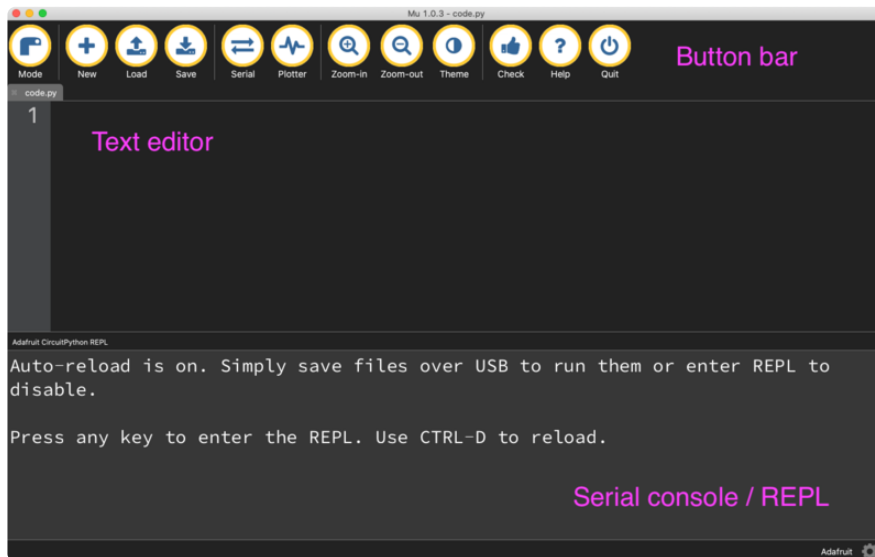


Mu attempts to auto-detect your board on startup, so if you do not have a CircuitPython board plugged in with a **CIRCUITPY** drive available, Mu will inform you where it will store any code you save until you plug in a board.

To avoid this warning, plug in a board and ensure that the **CIRCUITPY** drive is mounted before starting Mu.

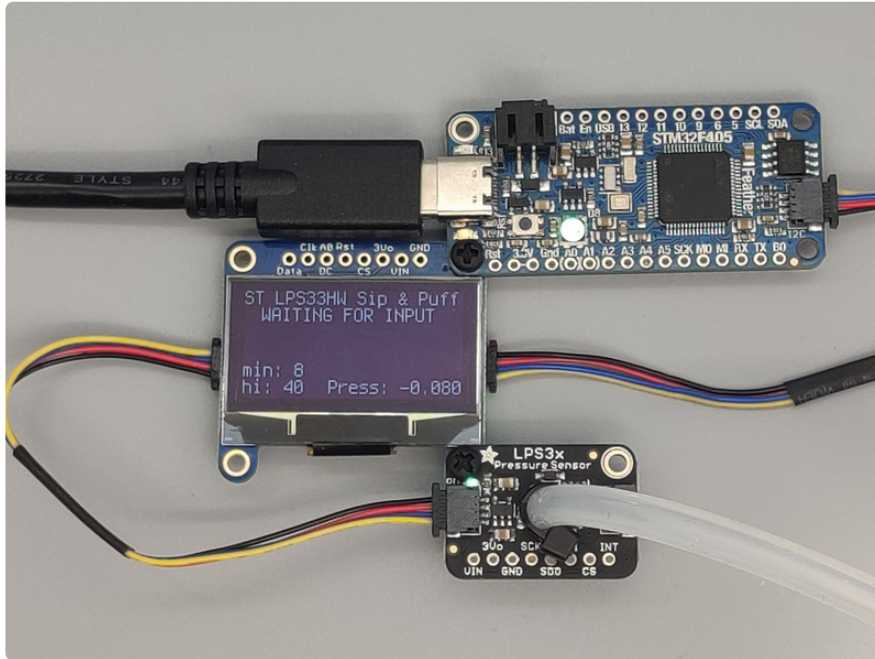
Using Mu

You can now explore Mu! The three main sections of the window are labeled below; the button bar, the text editor, and the serial console / REPL.



Now you're ready to code! Let's keep going...

CircuitPython Code



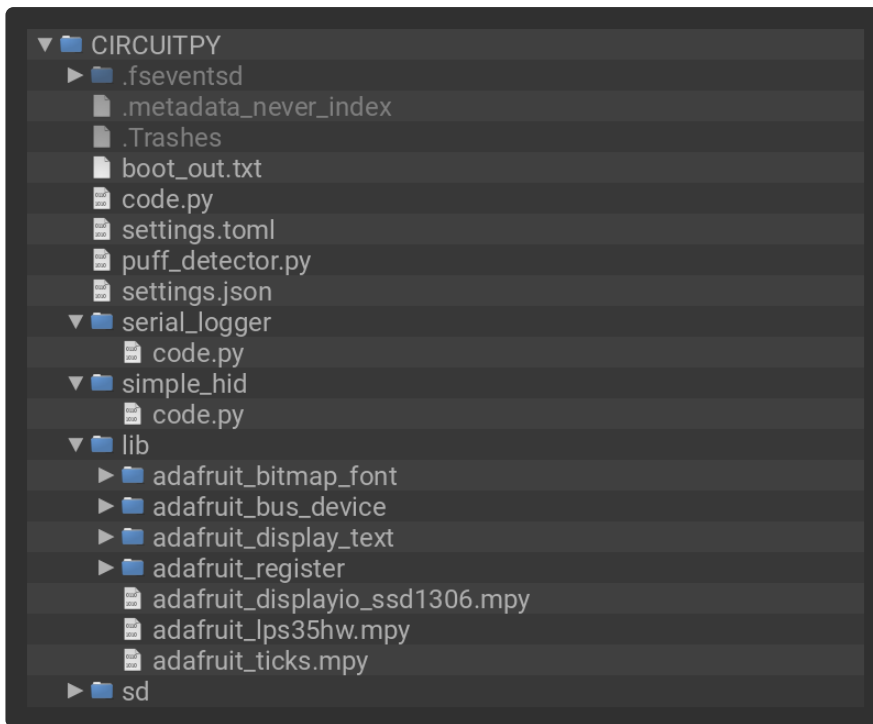
Now that you've got your pieces assembled and have CircuitPython installed on the Feather STM32F405 Express, all you have to do is copy over a bit of code we've written to make the whole thing go. Before you continue, you'll want to make sure that your Feather is plugged into your machine and your operating system file explorer/finder has the board showing up as a drive named **CIRCUITPY**.

Installing Project Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **REPLACE/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:



```
# SPDX-FileCopyrightText: 2020 Bryan Siepert for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import puff_detector

detector = puff_detector.PuffDetector()

@detector.on_sip
def on_sip(strength, duration):
    if strength == puff_detector.STRONG:
        print("GOT STRONG SIP")
    if strength == puff_detector.SOFT:
        print("GOT SOFT SIP")
    print("%.2f long" % duration)

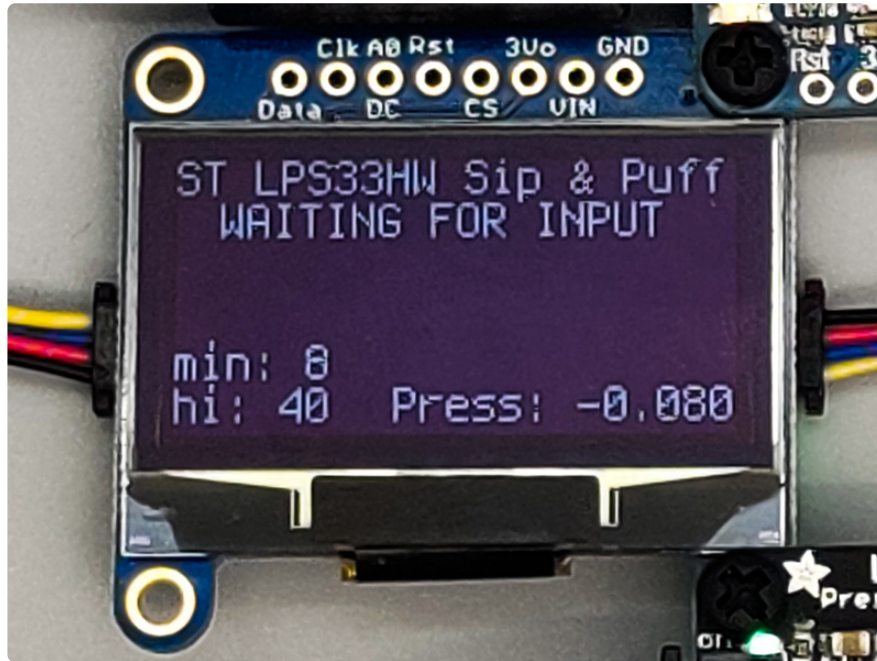
@detector.on_puff
def on_puff(strength, duration):
    if strength == puff_detector.STRONG:
        print("GOT STRONG PUFF")
    if strength == puff_detector.SOFT:
        print("GOT SOFT PUFF")
    print("%.2f long" % duration)

detector.run()
```

Sip and Puff code on GitHub

If you wish to access the code on GitHub, it can be found [in this directory \(https://adafru.it/Jb0\)](https://adafru.it/Jb0)

The board should come up with the starting screen! If you see any errors, check the Mu editor's REPL for warnings or failures



Usage

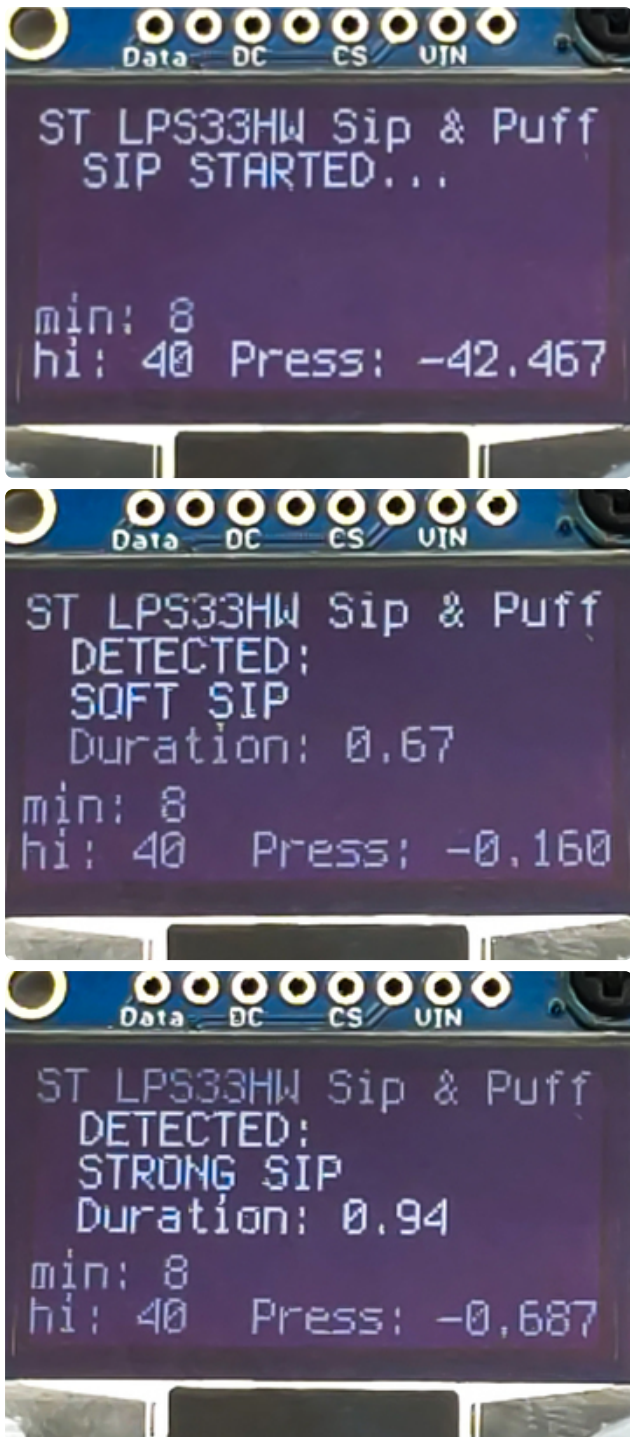
Using the Sip and Puff Code

Once installed and running, the code is pretty simple. When the program starts it will set the currently measured barometric air pressure and set it as **0 hPa** so that the following measurements are made relative to it. While idle the screen will display the following:

- **WAITING FOR INPUT** - This is the current measurement state. This will be updated during use to include the current measurement state to be one of **WAITING FOR INPUT**, **SIP/PUFF STARTED...**, or **DETECTED**:
- **min** - The low pressure threshold in hPa. Any measurement negative or positive will have to be this many hPa above or below the ambient pressure. This helps prevent incidental variations in the air pressure from triggering events when not intended. You may wish to tune this value to your particular needs
- **hi** - The high pressure threshold in hPa. Measurements that are this many hPa above or below ambient pressure will be registered as a **Strong** event. Anything below it but above the low pressure threshold will be registered as a **Soft** event
- **Press** - The current pressure measurement in hPa. This value will vary in real time as negative or positive pressure is applied to the sensor. You can use these

measurements to determine what your min and hi thresholds should be. If you watch this measurement while the sensor is idle, you will be able to see the regular variations in ambient pressure that make the low threshold necessary.

Detecting Sips



While the program and sensor are idle, applying positive pressure above the **min** amount will trigger the start of a puff event.

If you don't exceed the **hi** threshold, releasing the pressure will result in a **SOFT SIP** event being triggered.

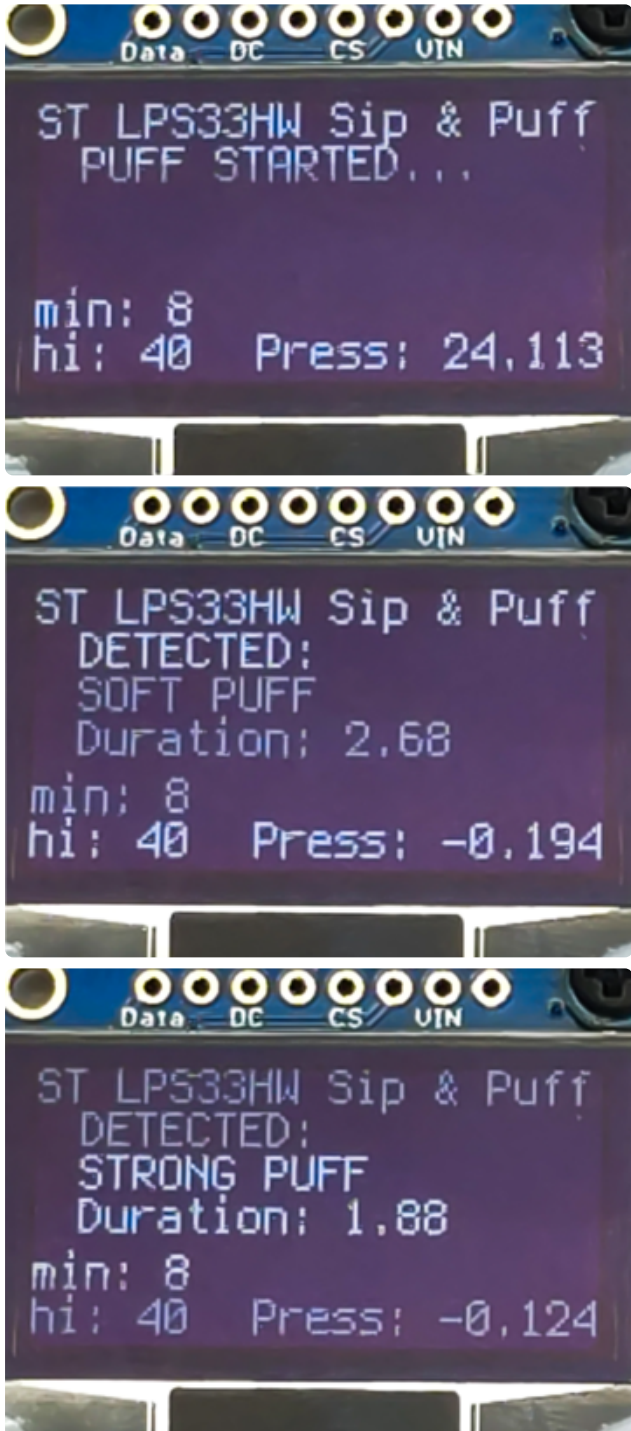
If you do exceed the **hi** threshold, when you release the pressure it will trigger a **STRONG PUFF** event.

Along with the type of event detected, the **Duration** of the event in seconds will be displayed.

Along with the information displayed on the OLED screen, the pressure events will also be printed to the serial console:

```
GOT SOFT SIP
0.27 Long
GOT STRONG SIP
1.61 Long
```

Detecting Puffs



While the program and sensor are idle, applying positive pressure above the **min** amount will trigger the start of a puff event.

If you don't exceed the **hi** threshold, releasing the pressure will result in a **SOFT PUFF** event being triggered.

If you do exceed the **hi** threshold, when you release the pressure it will trigger a **STRONG PUFF** event.

Along with the type of event detected, the **Duration** of the event in seconds will be displayed.

Like with sips, puff events will be printed to the serial log as well:

```
1.61 long
GOT SOFT PUFF
0.43 long
GOT STRONG PUFF
1.21 long
```

Configuring the Pressure Thresholds

Since people will have their own needs when it comes to how much or little pressure they will want to trigger events, we've made it possible for you to easily change these thresholds by adding a simple JSON file to the root of the **CIRCUITPY** drive along with the code. Edit the JSON below to have your desired pressures and then save it to the **CIRCUITPY** drive, making to preserve the filename of **settings.json**

```
{
  "MIN_PRESSURE": 10,
  "HIGH_PRESSURE": 60,
  "DISPLAY_TIMEOUT": 1
}
```

Extending the Capabilites



While an interesting demonstration of how it works, the demo on the previous page isn't all that useful. Currently all that the code in **code.py** is doing when an event is triggered is printing the event type and duration to the serial console as it's only meant to demonstrate the basic functioning of the puff detector.

By modifying the basic structure and content of the demonstration code and combining it with bits and pieces from our many CircuitPython demos, you can make the Sip and Puff trigger all sorts of things.

Call Me Maybe?

The [sine qua non](https://adafru.it/Jb1) (<https://adafru.it/Jb1>) of the demo code is the pair of `on_` functions, `on_sip` and `on_puff` that get called when the corresponding pressure event is detected. This type of function is usually referred to as a **callback** and is very common when writing code that reacts to user input. We will take a look at the `on_sip` function as an example as the two functions work the same but for different pressure events.

```
@detector.on_sip
def on_sip(strength, duration):
    if strength == puff_detector.STRONG:
        print("GOT STRONG SIP")
    if strength == puff_detector.SOFT:
        print("GOT SOFT SIP")
    print("%.2f long" % duration)
```

`on_sip` is a simple function that you provide to define the behavior that you want to happen when a sip event is detected. When a sip event happens, it gets called with two parameters,

- `strength` indicates if the event was `SOFT` or `STRONG` and can be compared to the constants of the same names in the `puff_detector` module
- `duration` is the duration of time that the sip event took from beginning to end.

You may wonder "how does the puff detector know what to call?". The answer to that question is the short statement above the `on_sip` function that starts with an `@`

```
@detector.on_sip
```

This statement tells the `detector` that the function that follows should be remembered or registered as a function that should be called when the corresponding event happens.

You might imagine leaving a voicemail for your friend where you say "call me on Friday to let me know if you will come to my party". Unfortunately they'll probably miss your party because no one checks their voice mail anymore, but none the less you

could think of leaving that message as registering a callback with your friend. If your friend was a Feather STM32F305 Express or other micro running CircuitPython, you might imagine programming them like so:

```
@friend.on_friday
def on_friday(rsvp):
    if rsvp:
        print("We're gonna need more guacamole")
    else:
        print("We're gonna need more tupperware for all this guacamole")
```

Make it Your Own

To get the sip and puff to do all manner of things, all you have to do is make your own functions that do what you want to happen when a pressure event happens, and then register it as a callback. That's all there is to it!