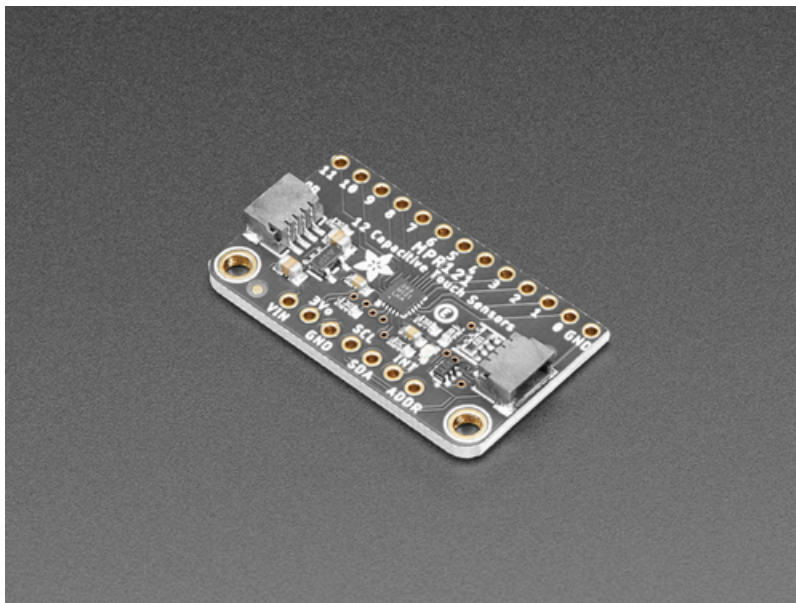




# Adafruit MPR121 12-Key Capacitive Touch Sensor Breakout Tutorial

Created by lady ada

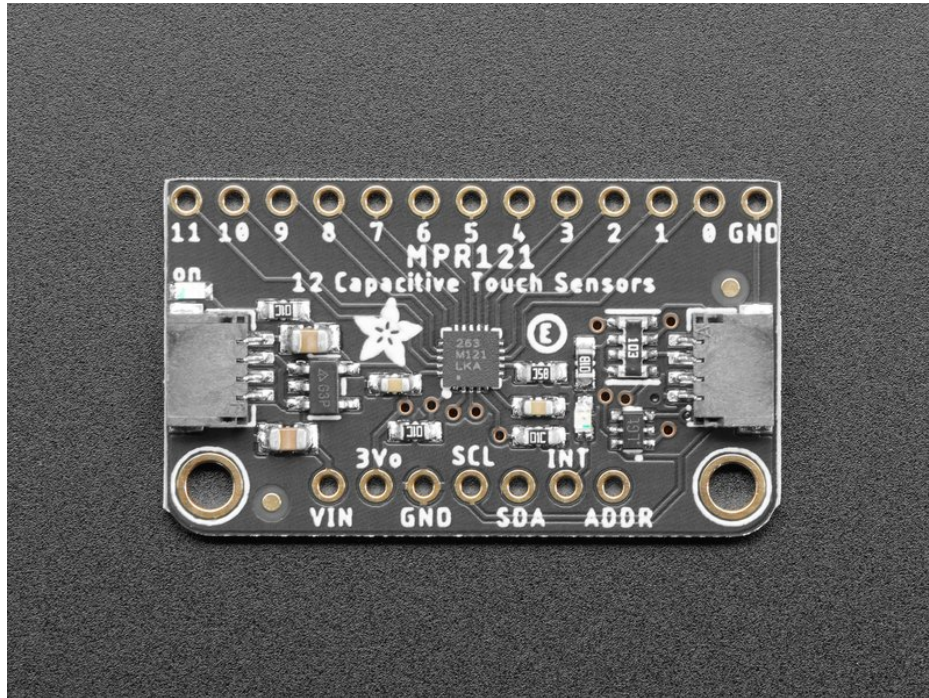


Last updated on 2021-01-26 02:47:51 PM EST

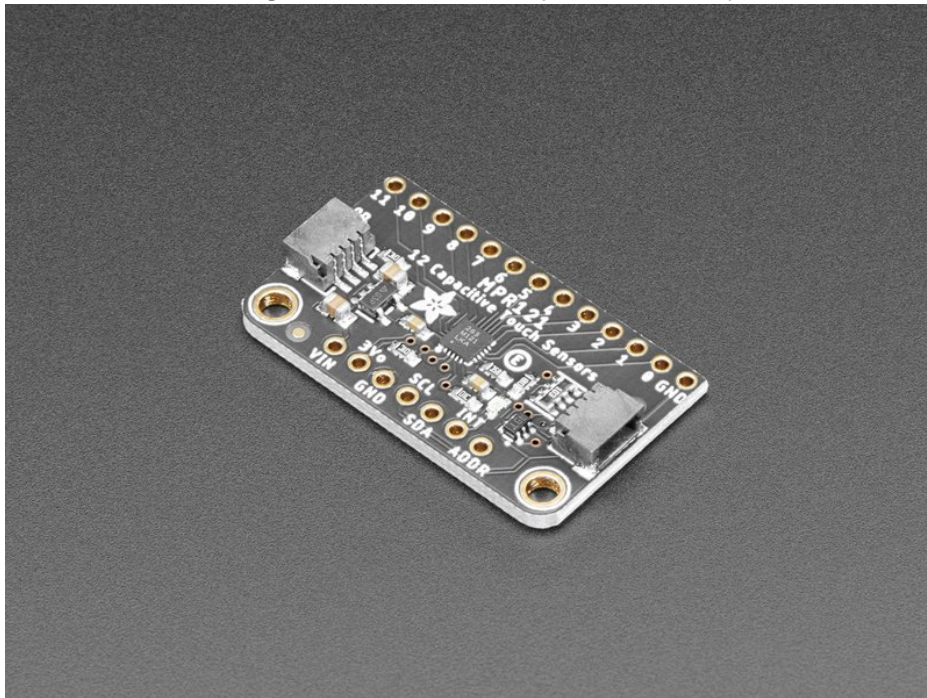
## Guide Contents

Guide Contents	2
Overview	3
Pinouts	6
Power Pins	7
I2C Pins	7
IRQ and ADDR Pins	7
Assembly	8
Prepare the header strip:	8
Add the breakout board:	8
And Solder!	9
Arduino	10
Download Adafruit_MPR121	11
Load Demo	11
Library Reference	14
Touch detection	14
Raw Data	14
Python & CircuitPython	16
CircuitPython Microcontroller Wiring	16
Python Computer Wiring	18
CircuitPython Installation of MPR121 Library	20
Python Installation of MPR121 Library	21
CircuitPython & Python Usage	21
Full Example Code	22
Python Docs	23
Raspberry Pi Virtual Keyboard	24
Wiring	24
Dependencies	24
Configuration	24
Usage	25
Launch In Background	26
Stop Background Process	26
Electrodes	28
Downloads	29
Datasheets & Files	29
STEMMA QT Revision Schematic and Fab Print	29
Original Breakout Board Schematic and Fab Print	29

# Overview

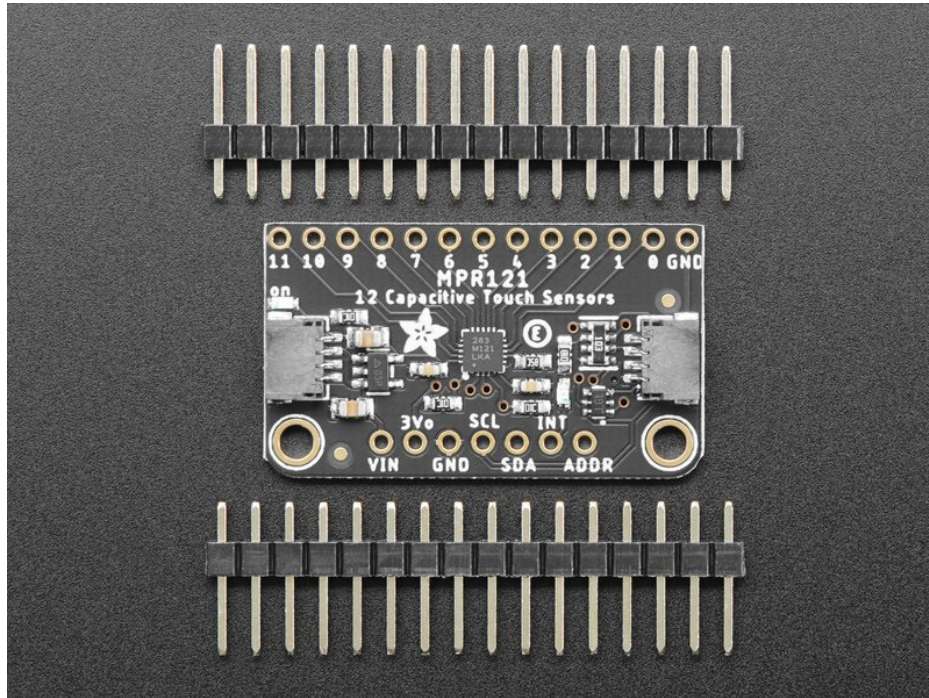


Add lots of touch sensors to your next microcontroller project with this easy-to-use 12-channel capacitive touch sensor breakout board, starring the MPR121. This chip can handle up to 12 individual touch pads.

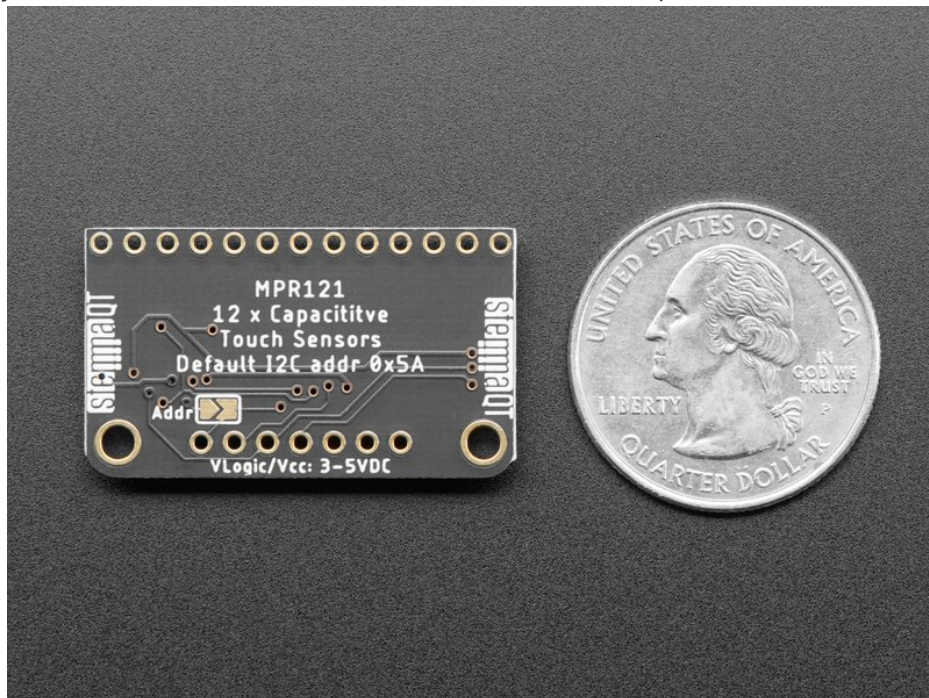


The MPR121 has support for only I2C, which can be implemented with nearly any microcontroller. You can select one of 4 addresses with the ADDR pin, for a total of 48 capacitive touch pads on one I2C 2-wire bus. Using this chip is a lot easier than doing the capacitive sensing with analog inputs: it handles all the filtering for you and can be configured for more/less sensitivity.





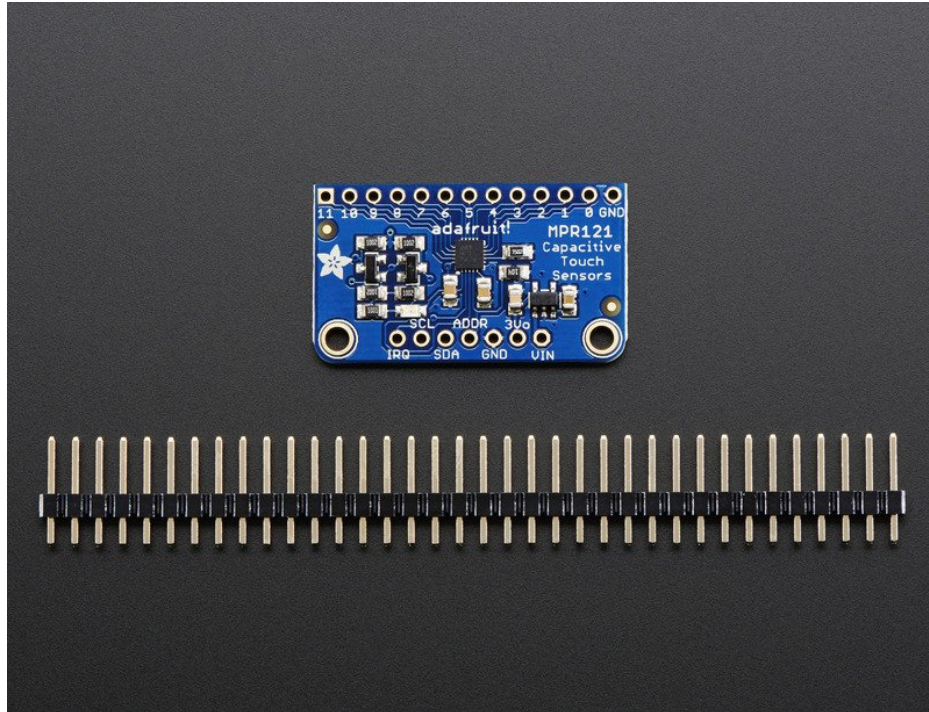
This sensor comes as a tiny hard-to-solder chip so we put it onto a breakout board for you. Since it's a 3V-only chip, we added a 3V regulator and I2C level shifting so its safe to use with any 3V or 5V microcontroller/processor like Arduino. We even added an LED onto the IRQ line so it will blink when touches are detected, making debugging by sight a bit easier on you. Comes with a fully assembled board, and a stick of 0.1" header so you can plug it into a breadboard. For contacts, we suggest using copper foil or pyralux, then solder a wire that connects from the foil pad to the breakout.



As if that weren't enough, we've now also added [SparkFun qwiic](https://adafruit.com/products/3082) (<https://adafruit.it/Fpw>) compatible [STEMMA QT](https://adafruit.com/products/3082) (<https://adafruit.it/Ft4>) connectors for the I2C bus so you don't even need to solder the I2C and power lines. Just wire up to your favorite micro using a [STEMMA QT adapter cable](https://adafruit.com/products/3082). (<https://adafruit.it/JnB>) The Stemma QT connectors also mean the MPR121

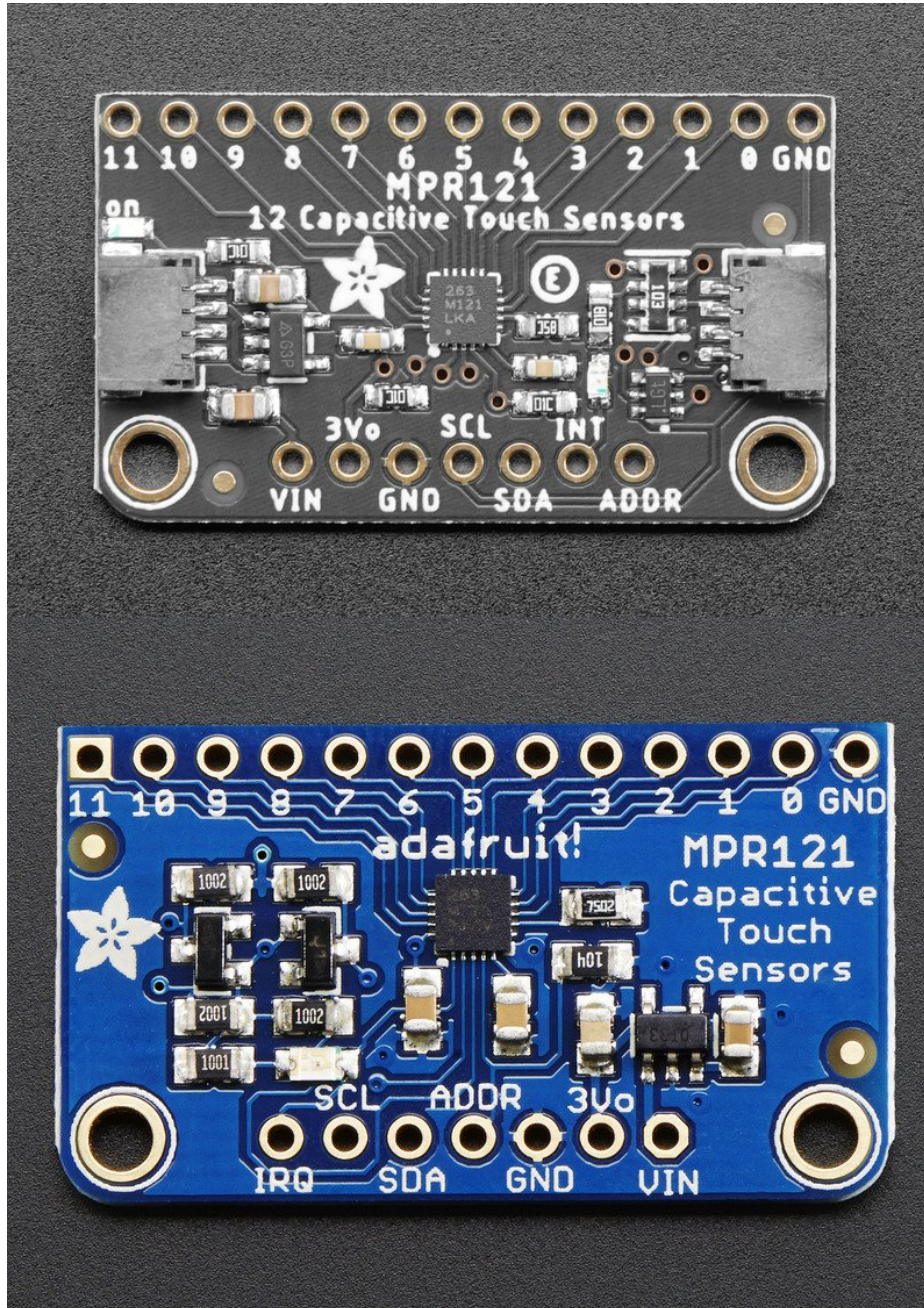
can be used with our [various associated accessories](https://adafru.it/Ft6). (<https://adafru.it/Ft6>) QT Cable is not included, but we have a variety in the shop (<https://adafru.it/JnB>).

There are two versions of this board - the STEMMA QT version shown above, and the original header-only version shown below. Code works the same on both!





# Pinouts



The pins are in a slightly different order on the original board from the STEMMA QT version. They function the same!

The little chip in the middle of the PCB is the actual MPR121 sensor that does all the capacitive sensing and filtering. We add all the extra components you need to get started, and 'break out' all the other pins you may want to connect to onto the PCB. For more details you can check out the schematics in the Downloads page.

# Power Pins

The sensor on the breakout requires 3V power. Since many customers have 5V microcontrollers like Arduino, we tossed a 3.3V regulator on the board. Its ultra-low dropout so you can power it from 3.3V-5V just fine.

- **Vin** - this is the power pin. Since the chip uses 3 VDC, we have included a voltage regulator on board that will take 3-5VDC and safely convert it down. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
- **3Vo** - this is the 3.3V output from the voltage regulator, you can grab up to 100mA from this if you like
- **GND** - common ground for power and logic

# I2C Pins

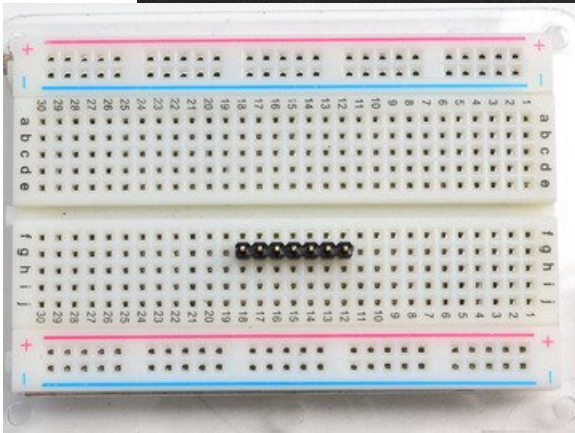
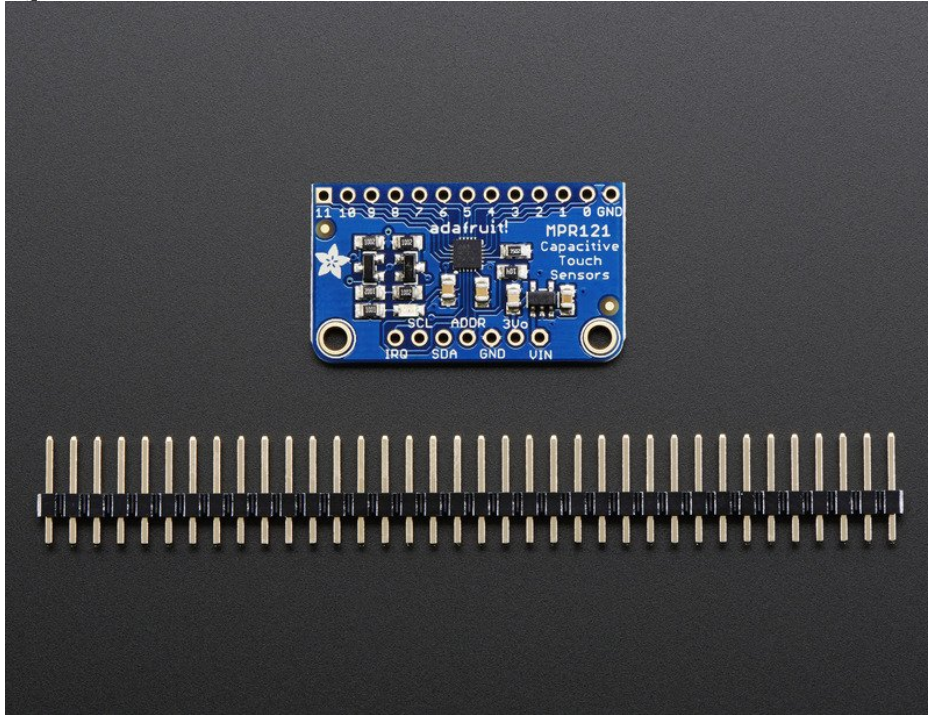
- **SCL** - I2C clock pin, connect to your microcontroller's I2C clock line.
- **SDA** - I2C data pin, connect to your microcontroller's I2C data line.

# IRQ and ADDR Pins

- **ADDR** is the I2C address select pin. By default this is pulled down to ground with a 100K resistor, for an I2C address of 0x5A. You can also connect it to the 3Vo pin for an address of 0x5B, the SDA pin for 0x5C or SCL for address 0x5D
- **INT** (IRQ on original version) is the Interrupt Request signal pin. It is pulled up to 3.3V on the breakout and when the sensor chip detects a change in the touch sense switches, the pin goes to 0V until the data is read over i2c

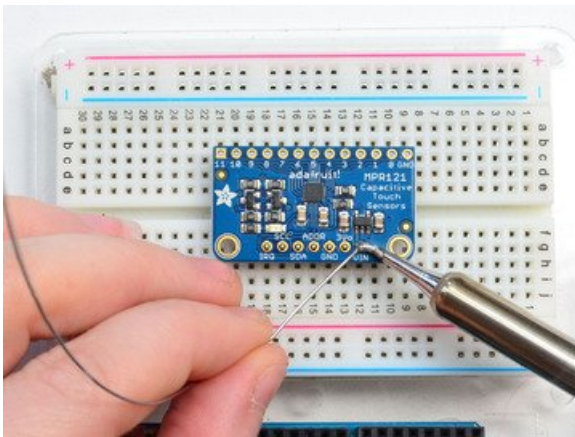


# Assembly



- **Prepare the header strip:**

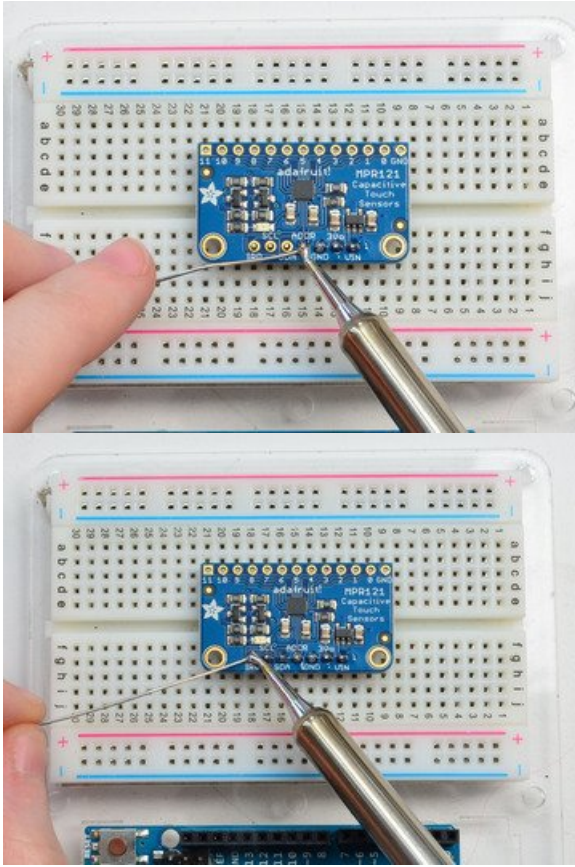
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**



- **Add the breakout board:**

Place the breakout board over the pins so that the short pins poke through the breakout pads





## And Solder!

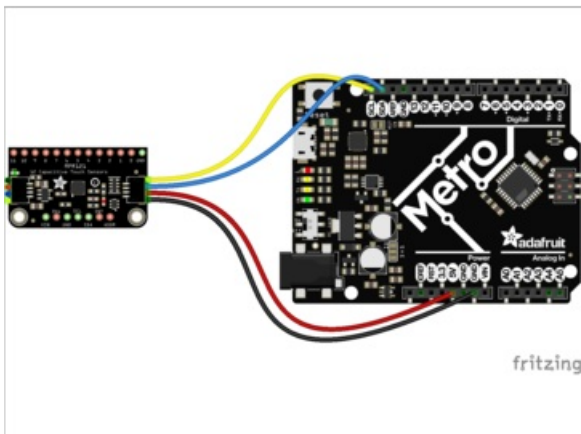
Be sure to solder all pins for reliable electrical contact.

*(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](https://adafru.it/aTk) (<https://adafru.it/aTk>)).*

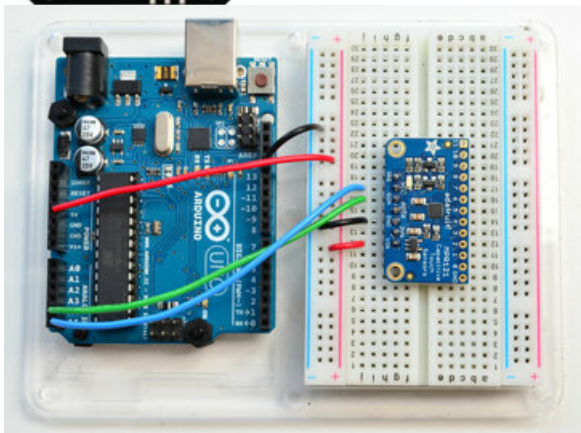
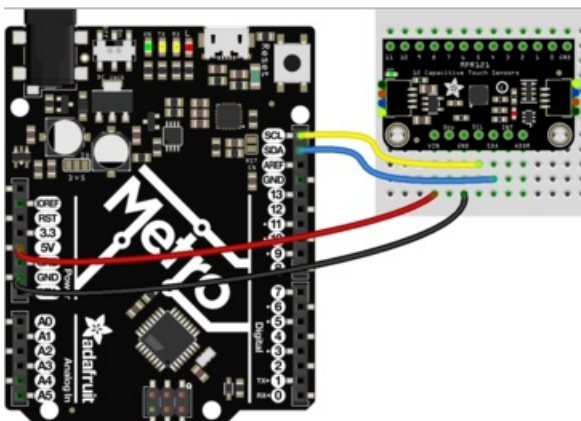
You're done! Check your solder joints visually and continue onto the next steps

# Arduino

You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C, then port the code - its pretty simple stuff!



fritzing



- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
- Connect the **SDA** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**

The MPR121 **ADDR** pin is pulled to ground and has a default I2C address of **0x5A**

You can adjust the I2C address by connecting **ADDR** to other pins:

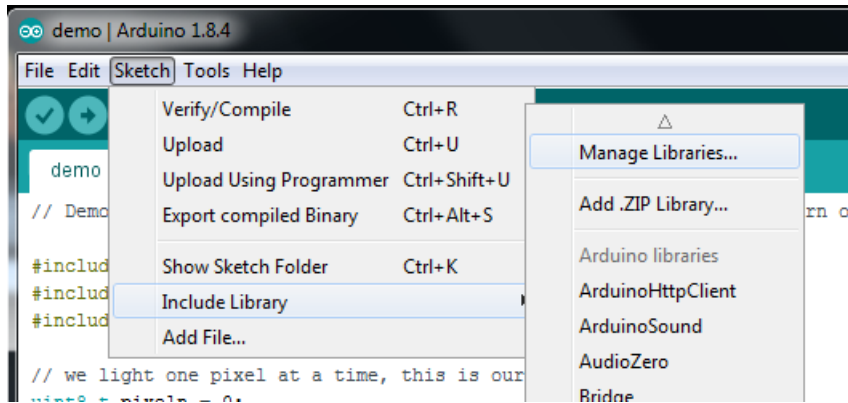
- ADDR not connected: 0x5A
- ADDR tied to 3V: 0x5B
- ADDR tied to SDA: 0x5C
- ADDR tied to SCL: 0x5D

We suggest sticking with the default for the test demo, you can always change it later.

# Download Adafruit\_MPR121

To begin reading sensor data, you will need to download the **Adafruit\_MPR121** library from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit MPR121** library and install it

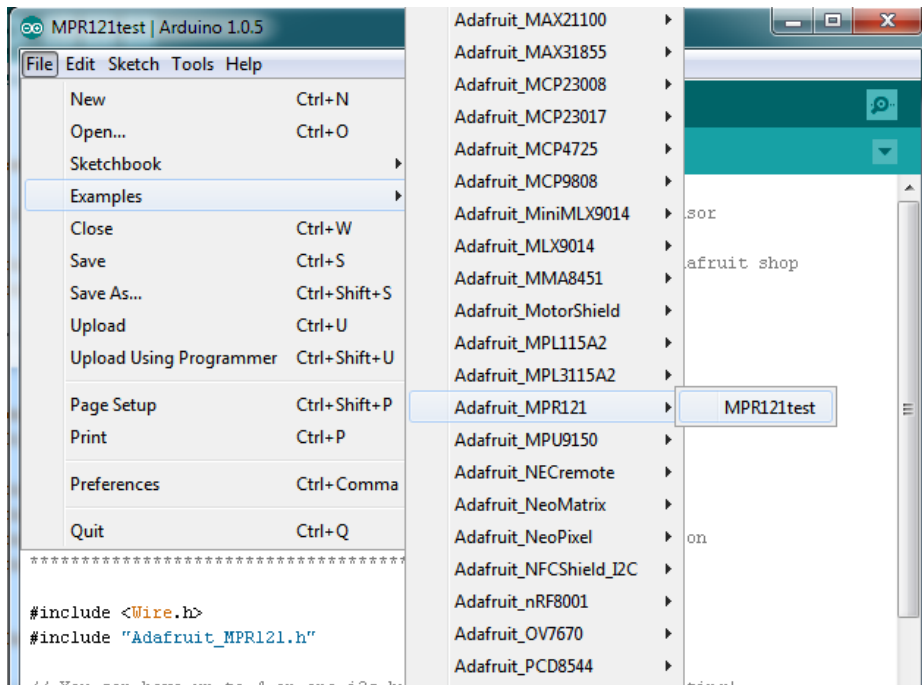


We also have a great tutorial on Arduino library installation at:

<http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use> (<https://adafru.it/aYM>)

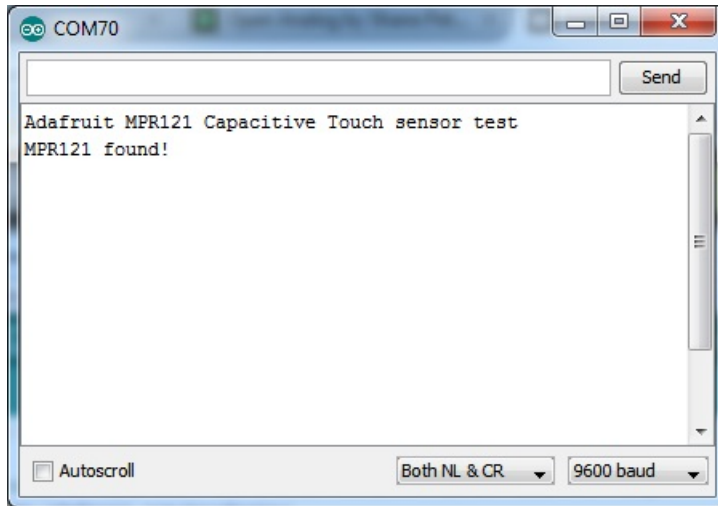
## Load Demo

Open up **File->Examples->Adafruit\_MPR121->MPR121test** and upload to your Arduino wired up to the sensor



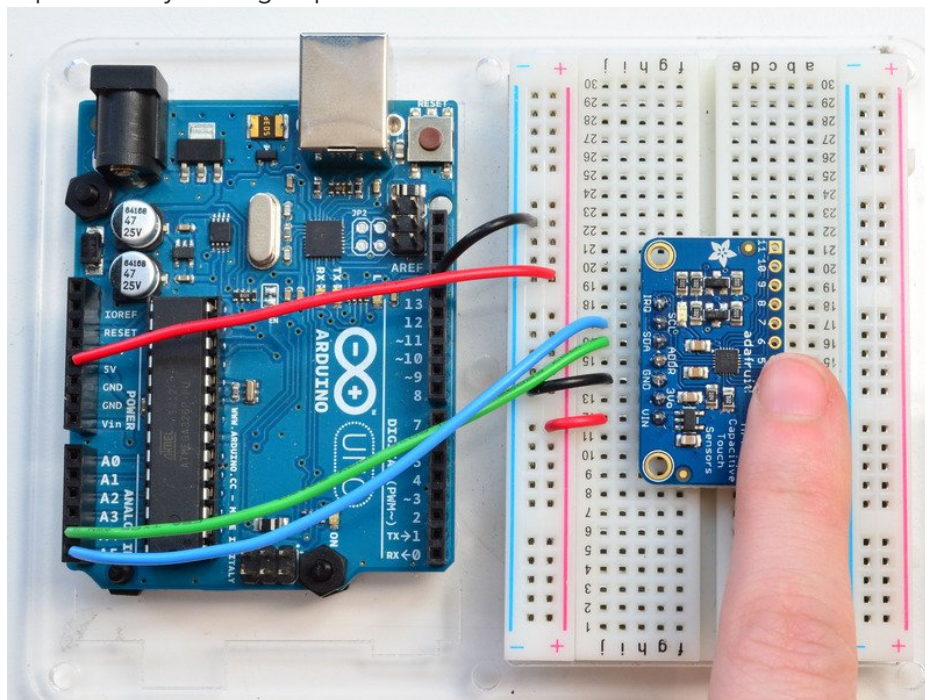
That's it! Now open up the serial terminal window at 9600 speed to begin the test.

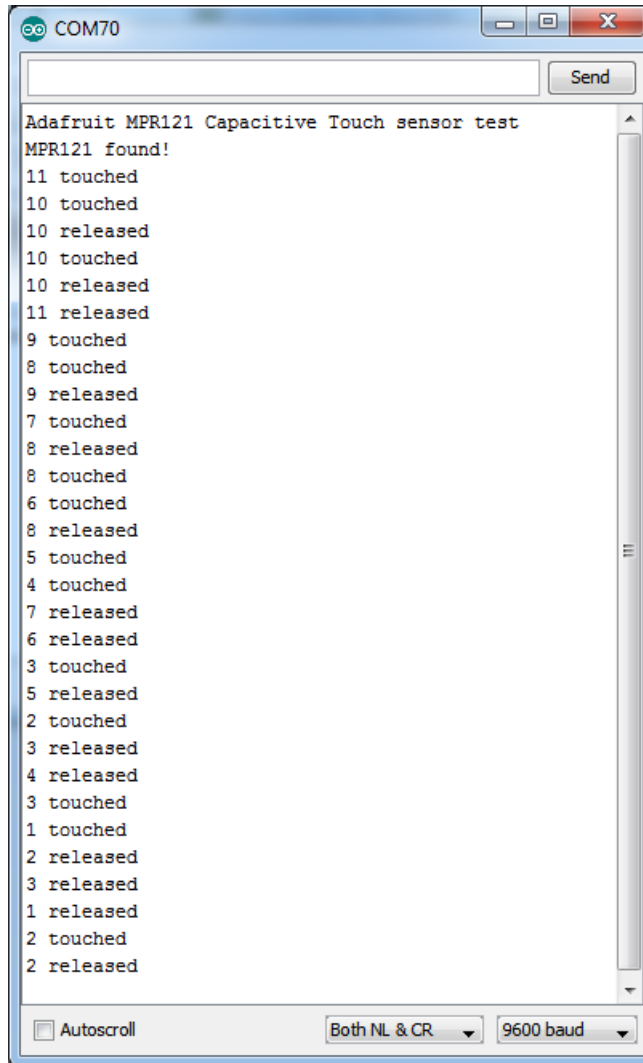




Make sure you see the "MPR121 found!" text which lets you know that the sensor is wired correctly.

Now touch the 12 pads with your fingertip to activate the touch-detection





For most people, that's all you'll need! Our code keeps track of the 12 'bits' for each touch and has logic to let you know when a contact is touched or released.

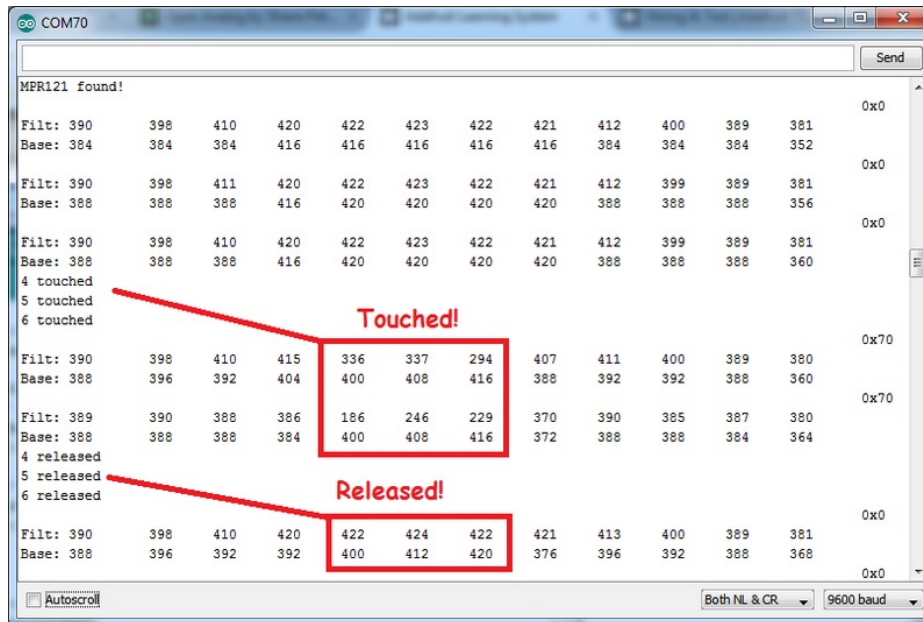
If you're feeling more advanced, you can see the 'raw' data from the chip. Basically, what it does it keep track of the capacitance it sees with "counts". There's some baseline count number that depends on the temperature, humidity, PCB, wire length etc. Where's a dramatic change in number, its considered that a person touched or released the wire.

Comment this "return" line to activate that mode:

```
// comment out this line for detailed data from the sensor!  
return;
```

Then reupload. Open up the serial console again - you'll see way more text

Each reading has 12 columns. One for each sensor, #0 to #11. There's two rows, one for the 'baseline' and one for the current filtered data reading. When the current reading is within about 12 counts of the baseline, that's considered untouched. When the reading is more than 12 counts smaller than the baseline, the chip reports a touch.



Most people don't need raw data too much, but it can be handy if doing intense debugging. It can be helpful if you are tweaking your sensors to get good responsivity.

## Library Reference

Since the sensors use I2C, there's no pins to be defined during instantiation. You can just use:

```
Adafruit_MPR121 cap = Adafruit_MPR121();
```

When you initialize the sensor, pass in the I2C address. It can range from 0x5A (default) to 0x5D

```
cap.begin(0x5A)
```

begin() returns true if the sensor was found on the I2C bus, and false if not.

## Touch detection

99% of users will be perfectly happy just querying what sensors are currently touched. You can read all at once with

```
cap.touched()
```

Which returns a 16 bit value. Each of the bottom 12 bits refers to one sensor. So if you want to test if the #4 is touched, you can use

```
if (cap.touched() & (1 << 4)) { do something }
```

You can check its not touched with:

```
if (! (cap.touched() & (1 << 4)) ) { do something }
```

## Raw Data

You can grab the current baseline and filtered data for each sensor with



```
filteredData(sensornumber);  
baselineData(sensornumber);
```

It returns a 16-bit number which is the number of counts, there's no unit like "mg" or "capacitance". The baseline is initialized to the current ambient readings when the sensor `begin()` is called - you can always reinitialize by re-calling `begin()`! The baseline will drift a bit, that's normal! It is trying to compensate for humidity and other environmental changes.

If you need to change the thresholds for touch detection, you can do that with

```
setThresholds(uint8_t touch, uint8_t release)
```

By default, the touch threshold is 12 counts, and the release is 6 counts. It's reset to these values whenever you call `begin()` by the way.

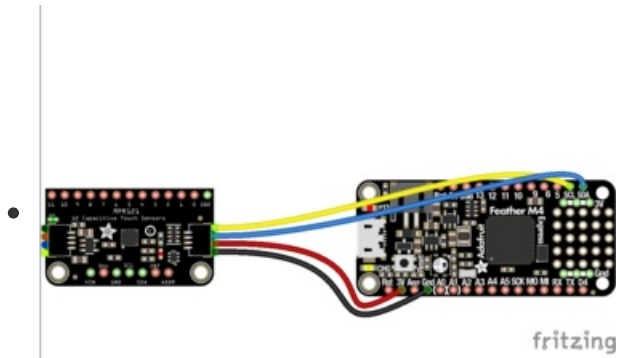
# Python & CircuitPython

It's easy to use the MPR121 sensor with Python or CircuitPython and the [Adafruit CircuitPython MPR121 \(https://adafru.it/v5e\)](https://adafruit.com/product/285) module. This module allows you to easily write Python code that reads capacitive touch from the sensor.

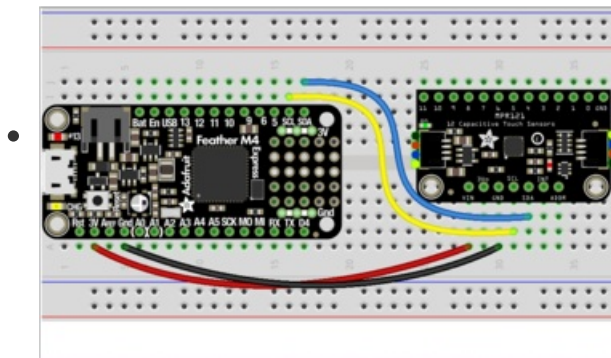
You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to [Adafruit\\_Blinka](https://adafruit.com/product/285), our [CircuitPython-for-Python compatibility library \(https://adafru.it/BSN\)](https://adafruit.com/product/285).

## CircuitPython Microcontroller Wiring

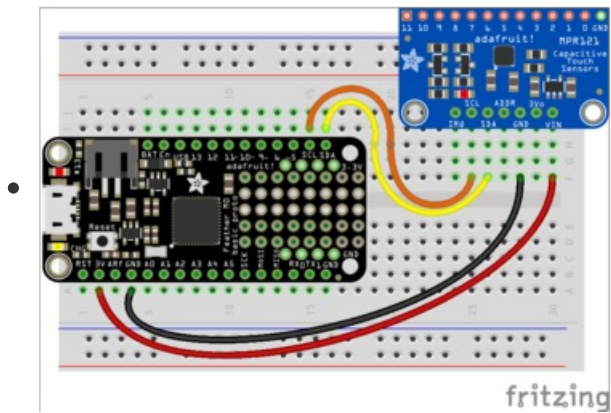
First wire up a MPR121 to your board exactly as shown on the previous pages for Arduino. Here's an example of wiring a Feather M0 to the sensor with I2C:



fritzing



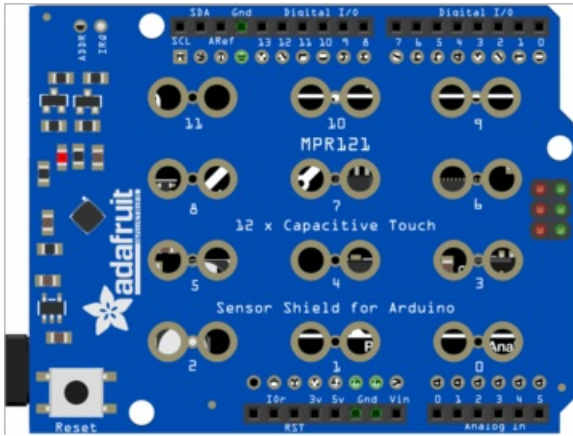
- Board 3V to sensor VIN
- Board GND to sensor GND
- Board SCL to sensor SCL
- Board SDA to sensor SDA



fritzing

Here's an example of the MPR121 shield connected to a Metro M0:





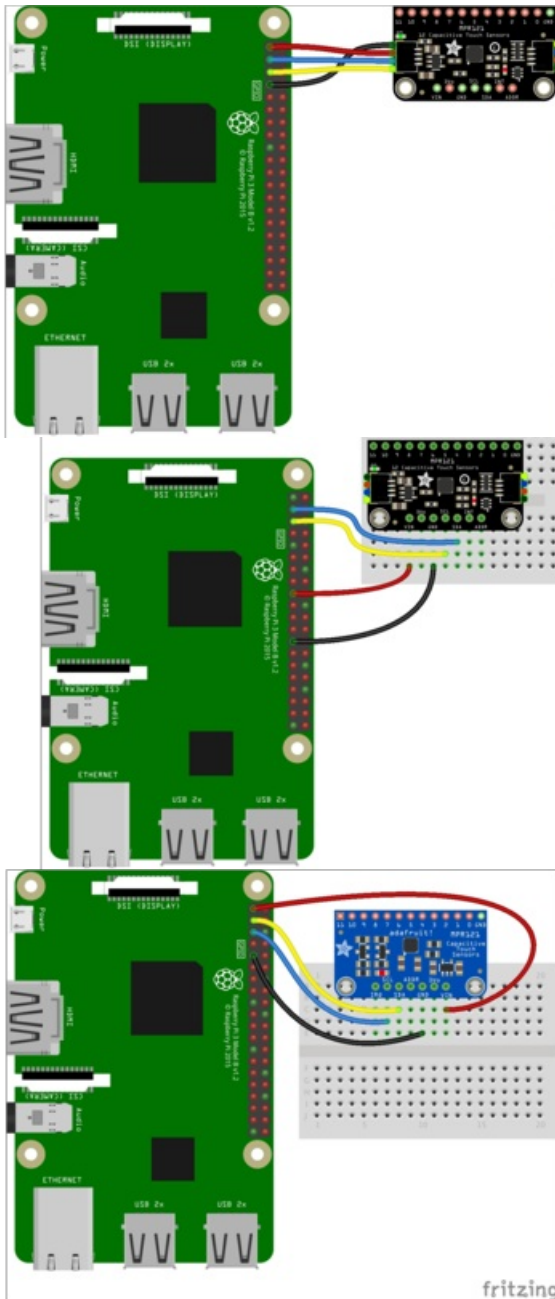
Solder the headers onto the shield, then simply connect it to your Metro M0.

This shield may not work with the Metro M4! It is not designed to work with it and may cause undesired behavior from the M4 microcontroller board. If you want to use this shield with CircuitPython for a project, use the Metro M0!

## Python Computer Wiring

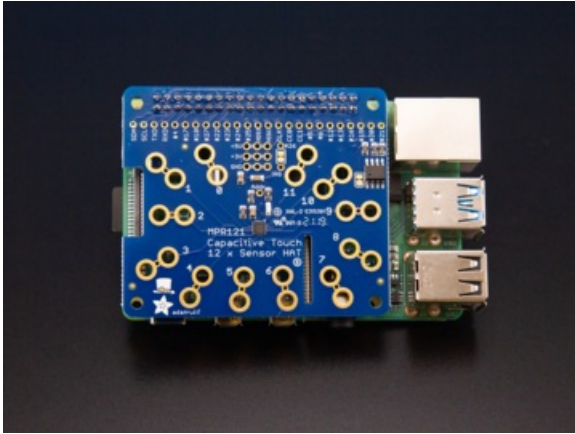
Since there's *dozens* of Linux computers/boards you can use, we will show wiring for Raspberry Pi. For other platforms, [please visit the guide for CircuitPython on Linux to see whether your platform is supported \(https://adafru.it/BSN\)](https://adafru.it/BSN).

Here's the Raspberry Pi wired with I2C:



- Pi 3V3 to sensor VIN
- Pi GND to sensor GND
- Pi SCL to sensor SCL
- Pi SDA to sensor SDA

You can also use the MPR121 Pi HAT with a Raspberry Pi:



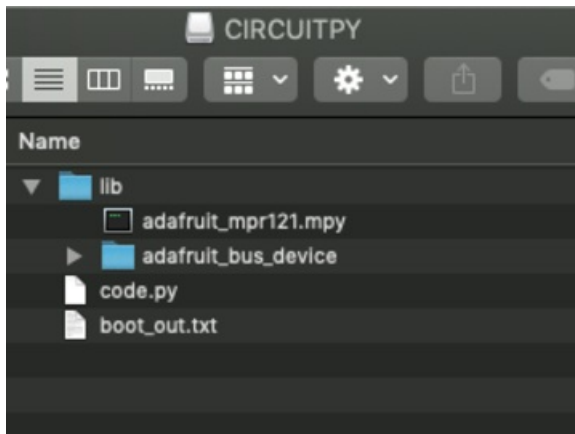
- Simply [solder](https://adafru.it/drl) the header provided with the HAT onto the HAT, and attach the HAT to the Pi.

## CircuitPython Installation of MPR121 Library

You'll need to install the [Adafruit CircuitPython MPR121](https://adafru.it/v5e) library on your CircuitPython board.

First make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) 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](https://adafru.it/uap). Our CircuitPython starter guide has a [great page on how to install the library bundle](https://adafru.it/ABU).



For non-Express boards like the Trinket M0 or Gemma M0, you'll need to manually install the necessary libraries from the bundle:

- `adafruit_mpr121.mpy`
- `adafruit_bus_device`

Before continuing, make sure your board's `lib` folder has the `adafruit_mpr121.mpy` and `adafruit_bus_device` files and folders copied over.

Next [connect to the board's serial REPL](https://adafru.it/Awz) so you are at the CircuitPython `>>>` prompt.



# Python Installation of MPR121 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 \(https://adafru.it/BSN\)](#)!

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

- `sudo pip3 install adafruit-circuitpython-mpr121`

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!

## CircuitPython & Python Usage

To demonstrate the usage of the sensor, we'll initialize it and read capacitive touch from the board's Python REPL.

If you're using an I2C connection, run the following code to import the necessary modules and initialize the I2C connection with the sensor:

```
import time
import board
import busio
import adafruit_mpr121
i2c = busio.I2C(board.SCL, board.SDA)
mpr121 = adafruit_mpr121.MPR121(i2c)
```

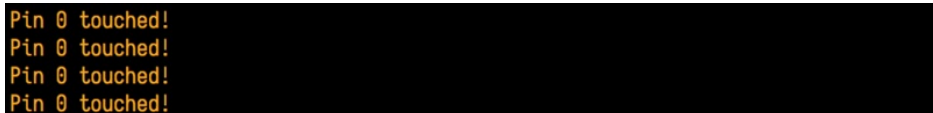
Now you're ready to read capacitive touch from the sensor. Use the following syntax to check a specific pin.

- `mpr121[i].value` - Return `True` if the specified pin is being touched, otherwise returns `False`.

Use a value 0 to 11 for [i] and it will return a boolean `True` or `False` value depending on if the input is currently being touched or not.

For example, to print when pin 0 is touched, run the following code, and then touch pin 0:

```
while True:
    if mpr121[0].value:
        print("Pin 0 touched!")
```



```
Pin 0 touched!
Pin 0 touched!
Pin 0 touched!
Pin 0 touched!
```

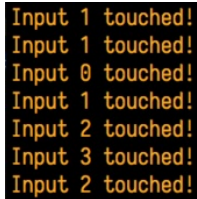
If you don't see any messages when you touch the inputs, you might need to ground yourself to the board by touching the GND pin on the board with one finger and then touching the input pads with another finger.

Also make sure nothing is touching the pins when you first run the code, or else it might confuse the MPR121's touch detection (unmount the board's file system from your operating system, then press the board's reset button to reset the script and run it again with nothing touching the pins). The pins are

calibrated on start-up, and will not react properly if you're touching the pins when the board starts up.

To print when any pin is touched, run the following code and then touch any capacitive touch pin:

```
while True:
    for i in range(12):
        if mpr121[i].value:
            print('Input {} touched!'.format(i))
```

A terminal window with a black background and yellow text showing the output of the code above. The output consists of seven lines: 'Input 1 touched!', 'Input 1 touched!', 'Input 0 touched!', 'Input 1 touched!', 'Input 2 touched!', 'Input 3 touched!', and 'Input 2 touched!'.

The example doesn't show its usage, but if you want to check all of the inputs at once you can use `touched_pins`. This function returns a 12 member tuple of the current state for each of the 12 pins. `True` is touched and `False` is not touched. For example, to test if pin 0 and 11 are being touched with one call you could run code like:

```
# Use touched_pins to get current state of all pins.
touched = mpr121.touched_pins
# Test if 0 and 11 are touched.
if touched[0] and touched[11]:
    print('Input 0 and 11 touched!')
```

That's all there is to using the MPR121 module with CircuitPython!

## Full Example Code

Temporarily unable to load content:

# Python Docs

[Python Docs \(https://adafru.it/Cbi\)](https://adafru.it/Cbi)



# Raspberry Pi Virtual Keyboard

One great use for the MPR121 is as a capacitive touch keyboard, where pressing a touch input causes a key to be pressed on a Raspberry Pi. This is kind of like a [MaKey MaKey \(http://adafru.it/1068\)](http://adafru.it/1068), but built right into the Pi using just the MPR121 and some special software. You could for example configure the MPR121 to act as a giant gamepad that controls games on the Raspberry Pi!

## Wiring

To use the MPR121 as a virtual keyboard you'll first want to make sure you've followed the earlier pages in this guide to connect the MPR121 to the Raspberry Pi and install the software.

## Dependencies

Now open a terminal on the Raspberry Pi using SSH and execute the following commands to install a few dependencies required by the virtual keyboard script:

```
sudo apt-get update
sudo apt-get install libudev-dev
sudo pip3 install python-uinput
sudo pip3 install adafruit-circuitpython-mpr121
```

## Configuration

After the dependencies are installed navigate to the MPR121 library **examples** folder again. Open the **pi\_keyboard.py** script in a text editor such as nano by executing:

```
nano pi_keyboard.py
```

Now scroll down to the key configuration near the top of the file:

```
KEY_MAPPING = {

    0: uinput.KEY_UP,

    1: uinput.KEY_DOWN,

    2: uinput.KEY_LEFT,

    3: uinput.KEY_RIGHT,

    4: uinput.KEY_B,

    5: uinput.KEY_A,

    6: uinput.KEY_ENTER,

    7: uinput.KEY_SPACE,

}
```

The `KEY_MAPPING` variable is a dictionary that maps an input number on the MPR121 to a keyboard button that will be sent when the input is pressed.

For example the code above configures input 0 to the UP key, input 1 to the DOWN key, input 2 to the LEFT key, etc.

Adjust the inputs and key codes depending on your needs. Most of the key codes are self explanatory (i.e. the key code for the letter Q is `uinput.KEY_Q`), but if you are unsure of an input you can find the name of a keycode in the [Linux input header here \(https://adafru.it/eik\)](https://adafru.it/eik). Take the key name and add `uinput.` to the front of it to get the key code that should be in the configuration above.

If you need to add more inputs you can add them as new lines after input 7 above. **Be careful to make sure each new line ends in a comma so the python dictionary is defined correctly.**

After you've configured your key mapping save the file by pressing **Ctrl-O** and **Enter**, then quit by pressing **Ctrl-X**.

## Usage

Now run the program by executing:

```
sudo python3 pi_keyboard.py
```

After a moment you should see a message displayed that tells you to press **Ctrl-C** to quit the program. If you press inputs to the MPR121 they should send the keys you've configured to the Raspberry Pi!

Note that you won't see any output from the program when keys are pressed, unless you first enable logging by un-commenting the following line:

```
# Uncomment to enable debug message logging (might slow down key detection).  
  
logging.basicConfig(level=logging.DEBUG)
```

Quit the program by pressing **Ctrl-C**.

## Launch In Background

Running the program by itself is great, but you probably want to run the program in the background while a game or other application runs and takes input from the MPR121 key presses. To do this you can launch the program into the background by executing at the terminal:

```
sudo python keyboard.py &
```

You should see a response such as:

```
[1] 2251
```

This tells you the program is launched in the background and is currently running under the process ID 2251. Try to remember the process ID as it will help you shut down the program later (but don't worry, I'll show you how to shut down the program even if you forget the ID).

Now run a game or other program that relies on keyboard input. Try pressing inputs on the MPR121 and you should see them register as keyboard presses!

## Stop Background Process

To stop the background process you'll need to tell Linux to kill the python **keyboard.py** process that was launched in the background earlier. If you remember the process ID number you can skip below to the kill command. However if you forgot the process ID number you can find it by executing a command like this to search all running processes for the **keyboard.py** script:

```
ps aux | grep keyboard.py
```

You should see a list of processes such as:

```
root      2251  0.5  0.3  5136  1488 pts/0    S   09:13   0:00 sudo python keyboard.py  
root      2252 13.2  1.4 18700  5524 pts/0    Sl  09:13   0:00 python keyboard.py  
pi        2294  0.0  0.2  4096   804 pts/0    S+  09:13   0:00 grep --color=auto keyboard.py
```

The first line with **sudo python keyboard.py** is the background process that was launched earlier. You can kill this process by running:

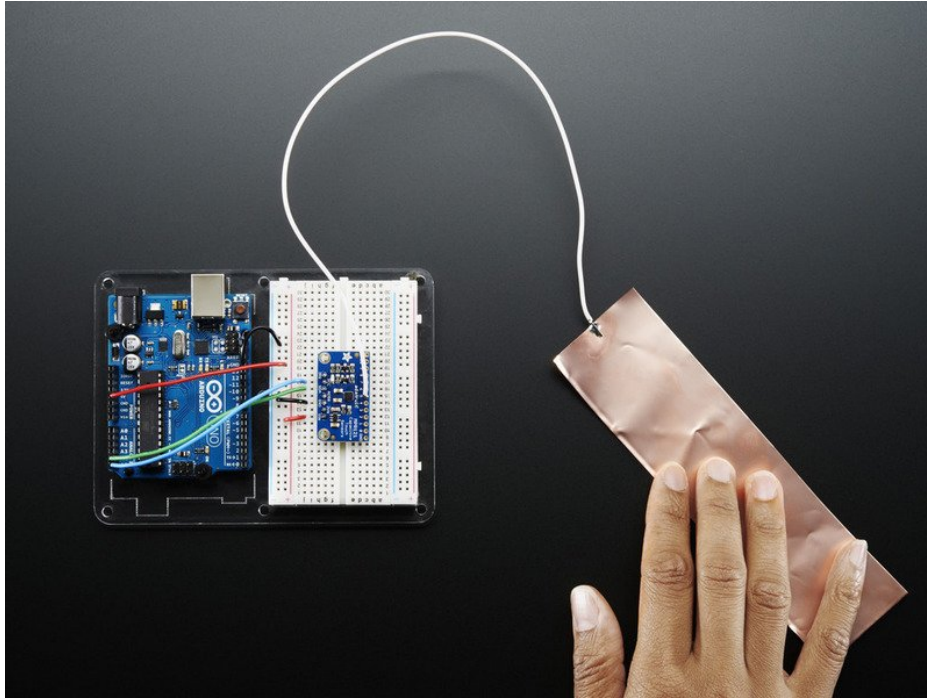
```
sudo kill 2251
```

If you run the **ps** command above again you should now see the Python **keyboard.py** processes have terminated.

That's all there is to using the MPR121 virtual keyboard on a Raspberry Pi. Have fun using the capacitive touch buttons to control your own games and programs!



# Electrodes



Once you have the MPR121 breakout working you'll want to construct electrodes. These are large conductive piece of copper, foil, paint, etc that will act as the "thing you touch"

Remember that electrodes must be electrically conductive! We suggest copper foil tape, conductive fabrics, ITO, pyralux flex PCB, etc. [We have tons of great conductive materials in our Materials category. Some can be soldered to, others can be clipped to with alligator chips. \(<https://adafru.it/dKI>\)](#)

Remember, it doesn't have to be metal to be electrically conductive. Other things that work are tap or salt water, many kinda of food, even fruit!

We suggest soldering a wire to the electrode pad on the breakout and then soldering or clipping it to whatever you want your electrode to be.

The wires and electrodes themselves have a certain amount of 'inherent capacitance'!

This means that whenever you attach an alligator clip, or a large piece of copper, or whatever your electrode is, the capacitive sense chip will detect it and may think you're touching it. What you have to do is *recalibrate* the sensor. The easiest way to do that is to restart the python sketch since calibration is done when the chip is initialized. So, basically...

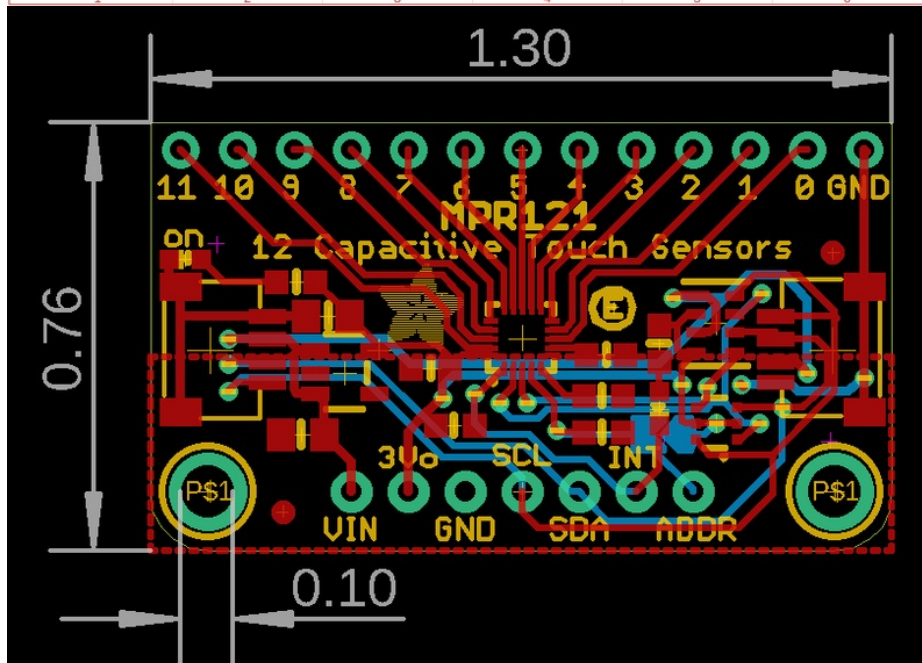
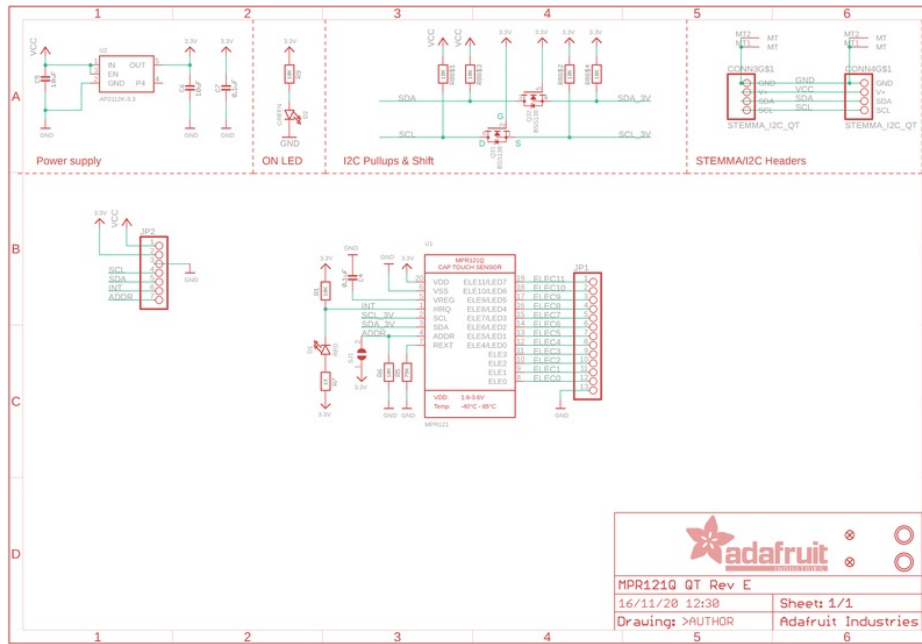
**connect all your wires, electrodes, fruit, etc... *then* start up the capacitive touch program!**

# Downloads

## Datasheets & Files

- [MPR121 Datasheet \(https://adafru.it/dKG\)](https://adafru.it/dKG)
- [EagleCAD PCB files on GitHub \(https://adafru.it/pJa\)](https://adafru.it/pJa)
- [Fritzing object in Adafruit Fritzing library \(https://adafru.it/c7M\)](https://adafru.it/c7M)

# STEMMA QT Revision Schematic and Fab Print



Original Breakout Board Schematic and Fab Print

