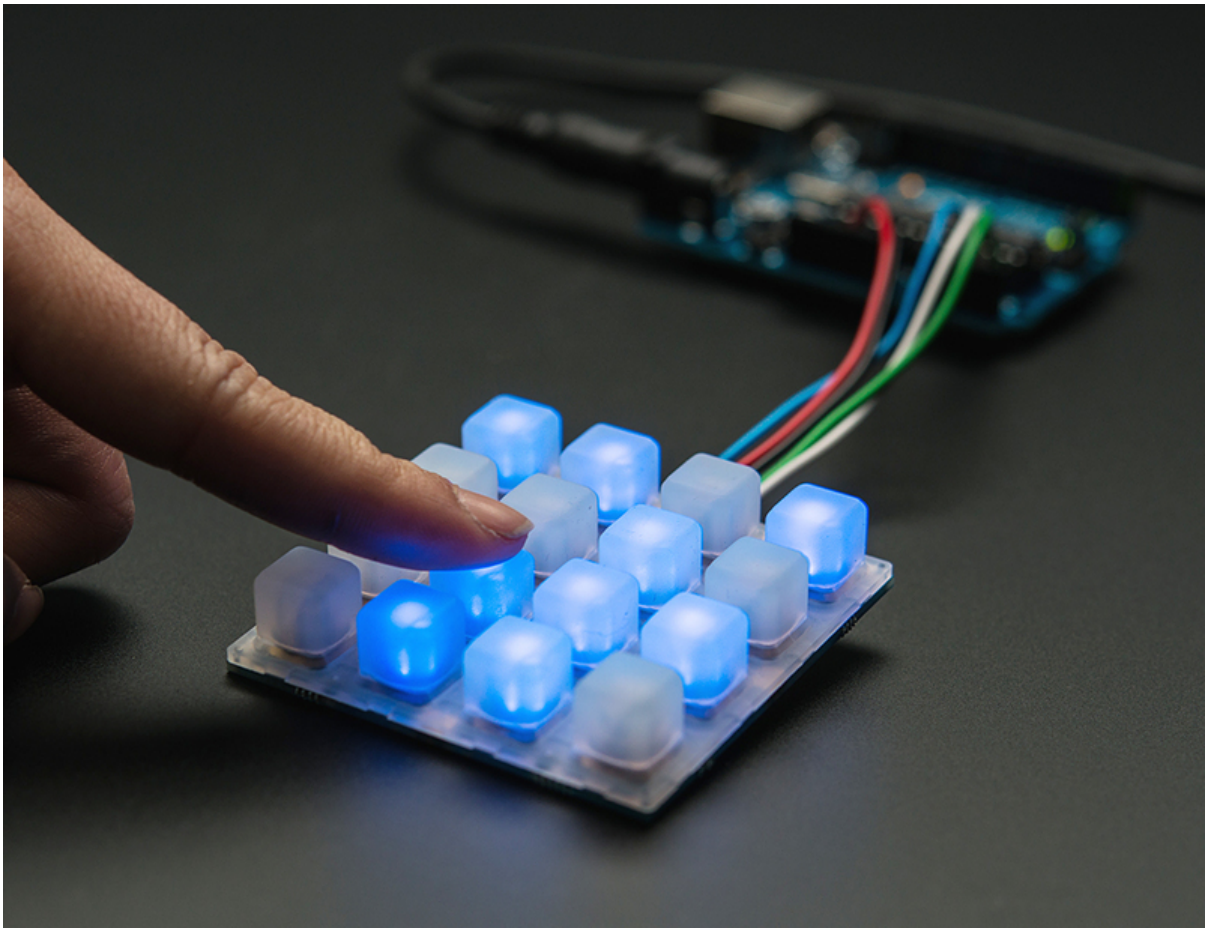
# Introducing Adafruit Trellis

Created by lady ada



https://learn.adafruit.com/adafruit-trellis-diy-open-source-led-keypad

Last updated on 2024-03-08 01:54:12 PM EST

# Table of Contents


Overview 3

Adding LEDs 6

Connecting 11

- Library reference
- Reading Switches
- Creating the objects
- Controlling LEDs
- Adding support for more tiles
- Make more objects
- Make a bigger set
- Say the number

Tiling 15

- Addressing
- Changing Addresses

Python & CircuitPython 23

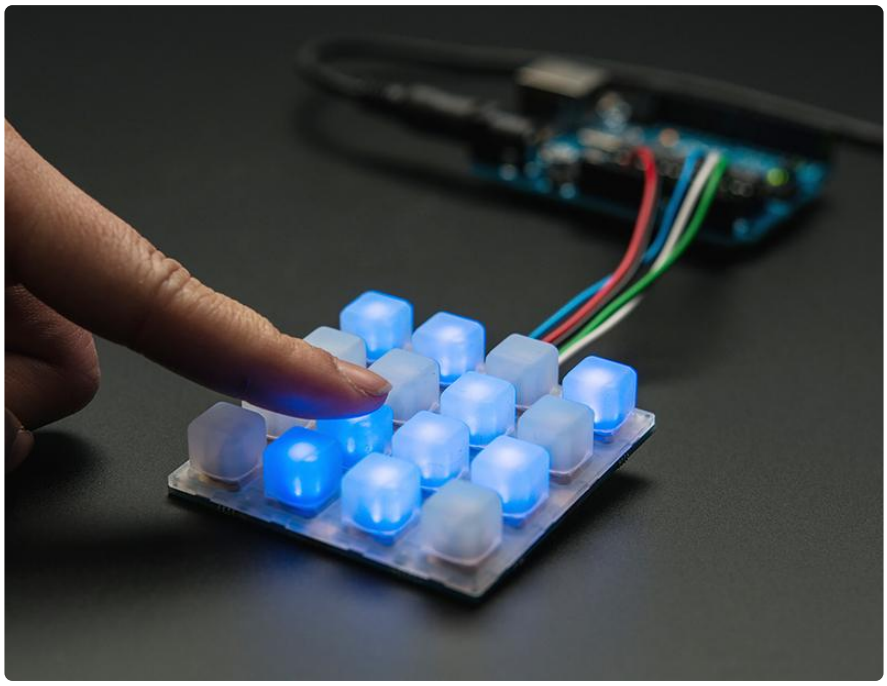
- CircuitPython Microcontroller Wiring
- Python Computer Wiring
- CircuitPython Installation of Trellis Library
- Python Installation of Trellis Library
- CircuitPython & Python Usage
- Full Example Code

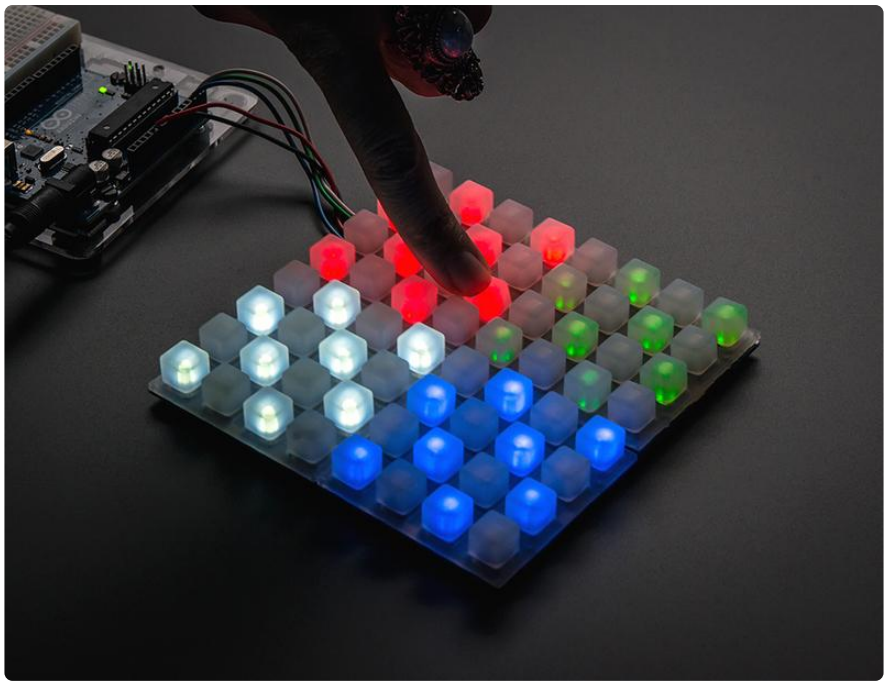
Python Docs 28

Downloads 29

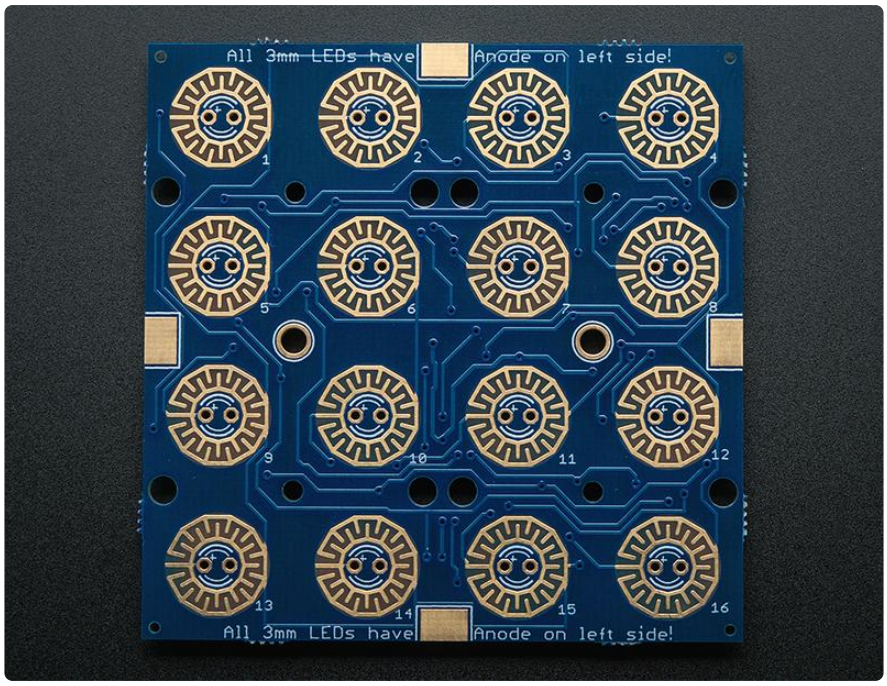
- Schematic
- Fabrication Print

# Overview

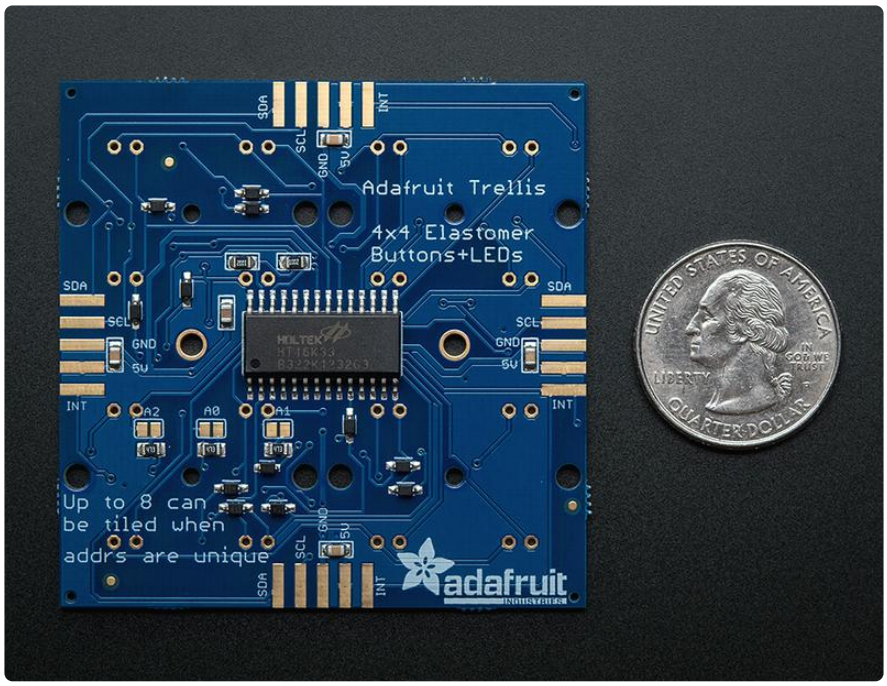




Trellis is an open source backlight keypad driver system. It is easy to use, works with any 3mm LEDs and eight tiles can be tiled together on a shared I2C bus.

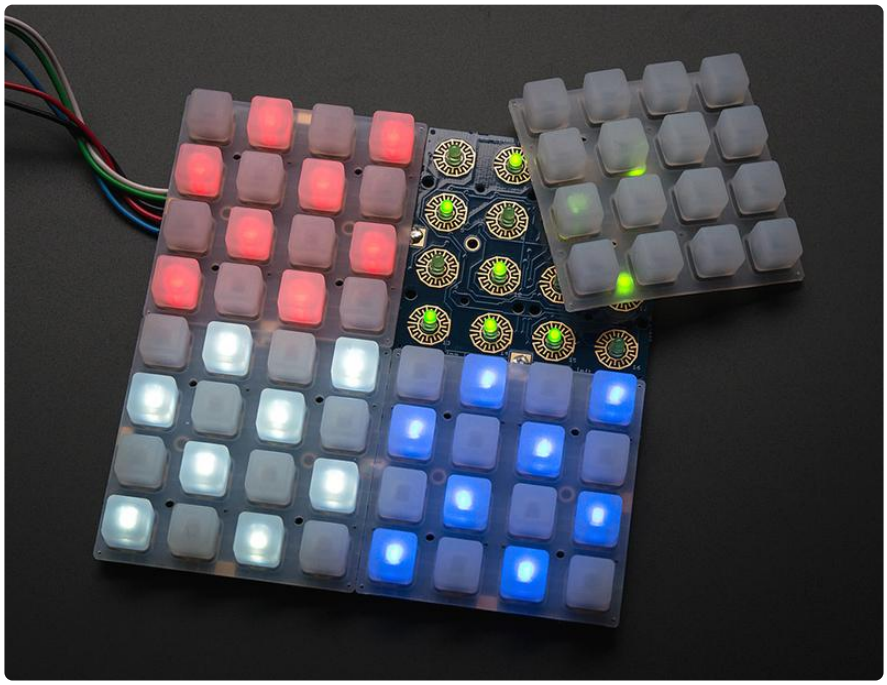



The Trellis PCB is specially made to match the Adafruit 4x4 elastomer keypad. Each Trellis PCB has 4x4 pads and 4x4 matching spots for 3mm LEDs. The circuitry on-board handles the background key-presses and LED lighting for the 4x4 tile. However, it does not have any microcontroller or other 'brains' - an Arduino (or similar microcontroller) is required to control the Trellis to read the keypress data and let it know when to light up LEDs as desired.

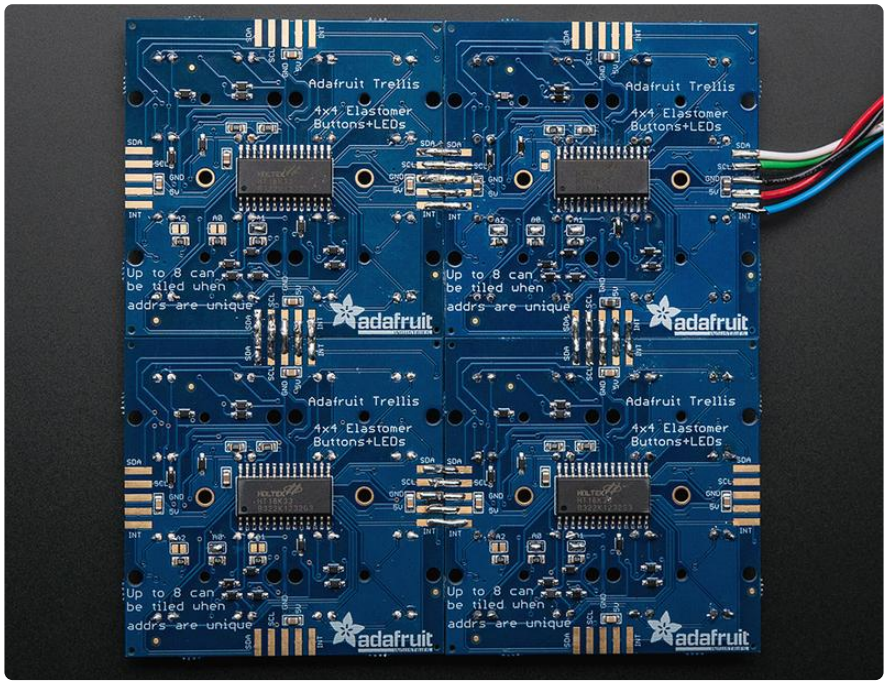



Each tile has an I2C-controlled LED sequencer and keypad reader already on it. The chip can control all 16 LEDs individually, turning them on or off. It cannot do grayscale or dimming. The same chip also reads any keypresses made with the rubber keypad. The connections are 'diode multiplexed' so you do not have to worry about "ghosting"

when pressing multiple keys, each key is uniquely addressed.



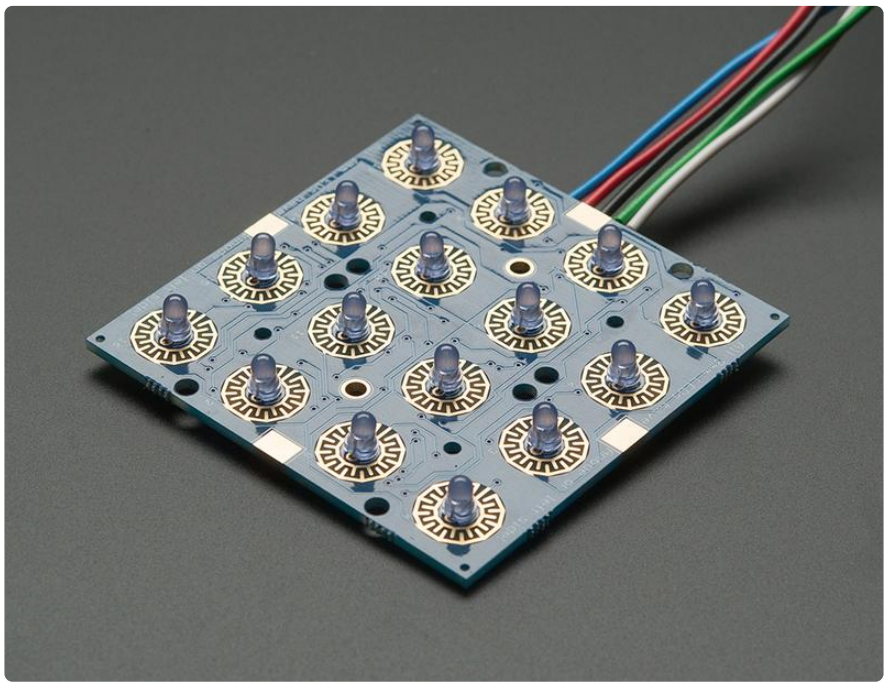
The tiles have 3 address jumpers. You can tile up to 8 PCBs together (for a total of 4x32 or 16x8=128 buttons/leds) on a single I2C bus, as long as each one has a unique address. All the tiles connect by the edges with solder, and share the same power, ground, interrupt, and i2c clock/data pins. So, you can easily set up to 128 LEDs and read up to 128 buttons using only 2 I2C wires! The tiles can be arranged in any configuration they want as long as each tile is connected to another with the 5 edge-fingers.



Each LED is multiplexed with a constant-current driver, so you can mix and match any colors you like. You don't need it to be all blue, all red, etc. Mix it up! Any 3mm LED can be used, although we find that diffused LEDs with 250mcd+ brightness look best.

---

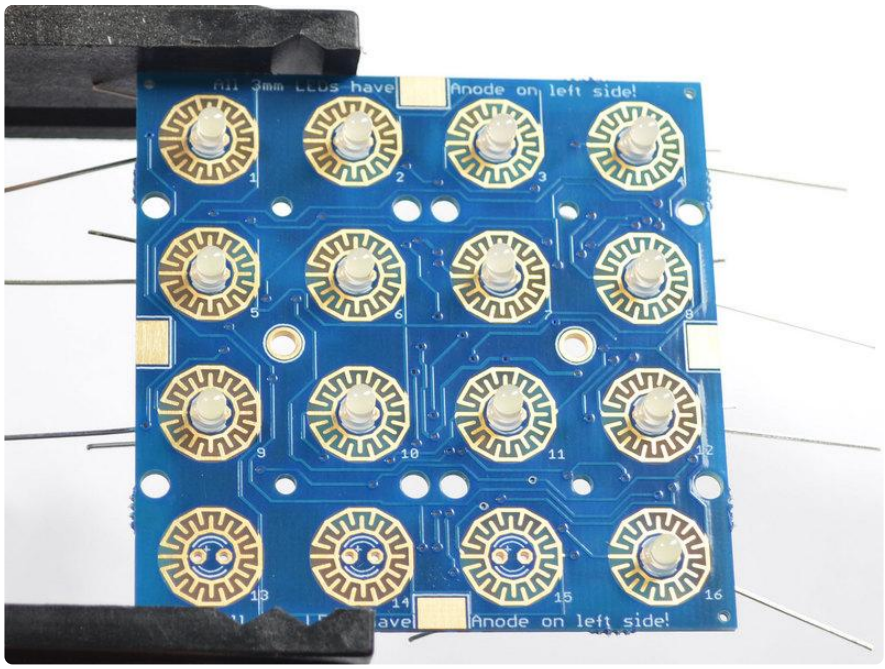
# Adding LEDs



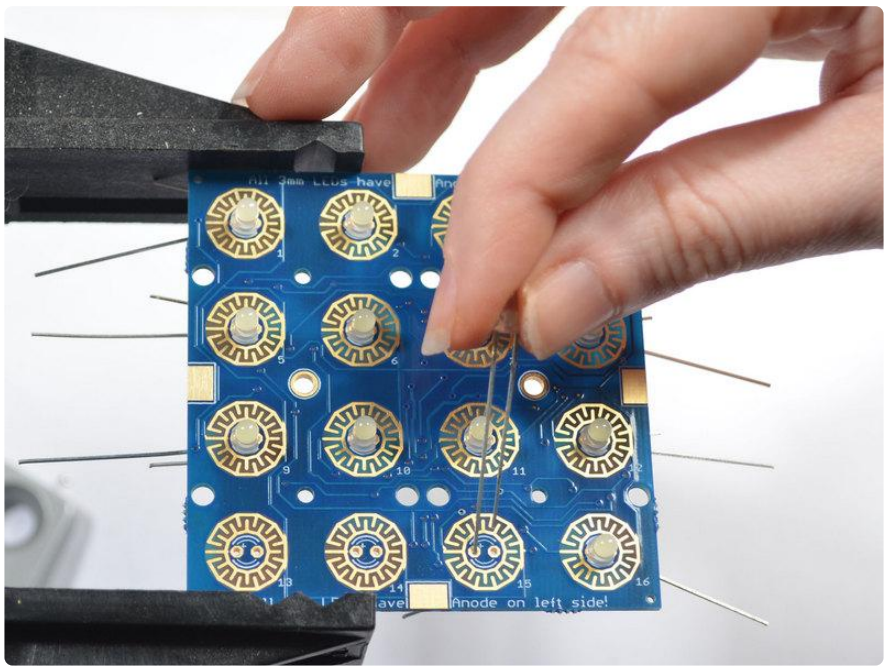
The Trellis PCB comes with all the driver circuitry **but doesn't come with LEDs installed!** This is because we know people want to get creative with the LED colors - choose any color you like! Diffused 3mm LEDs with 250mcd+ brightness look best.

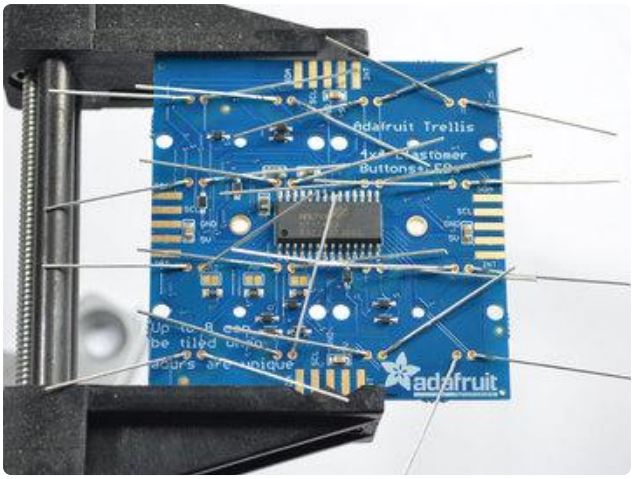
As you can see here, the LEDs sit inside the button cavity and the two button contacts surround it. This allows for the nice frosted LED effect.

The LEDs are seperately controlled from the buttons - an Arduino or similar is required to read data from the buttons and then write out what LEDs to light up.



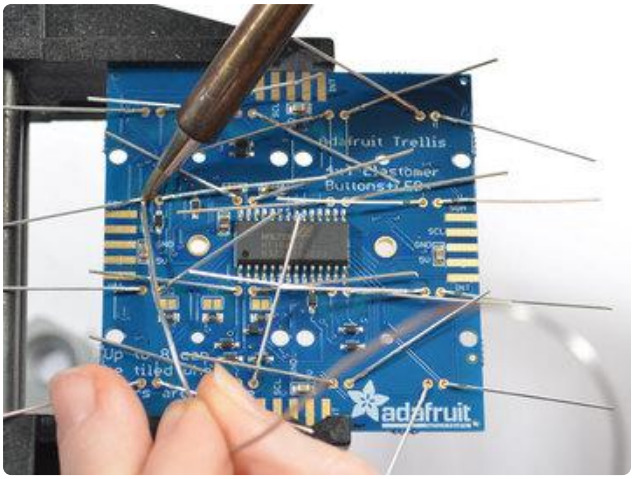

LEDs have **polarity** so they have to be installed the right way or they won't work! There's a little + symbol on the left hand LED pads, thats the anode/positive pin. LEDs have a longer leg on the + pin so make sure that pin goes into the + hole.

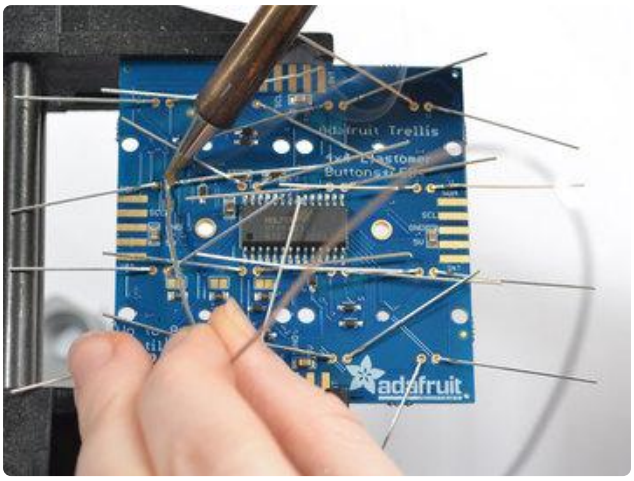




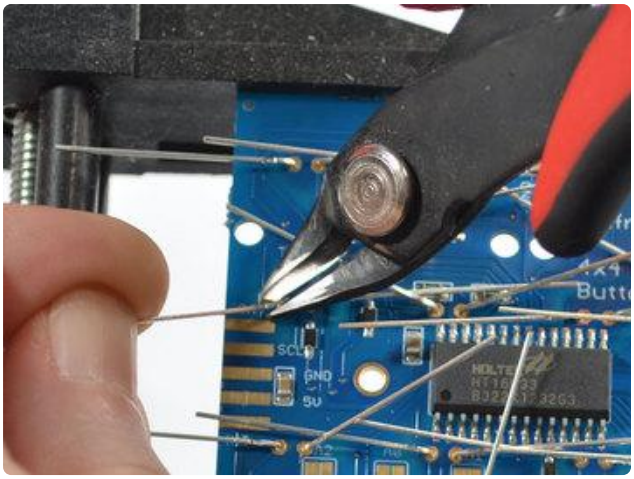

You can do one LED at a time, or all at once, its up to you and how experienced you are in soldering LEDs.

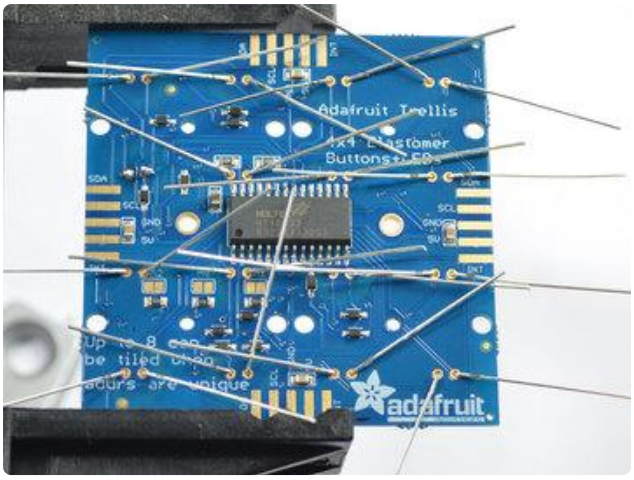
Bend the LED legs out so that the LEDs sit nice and flat against the PCB. Flip over the PCB so you can solder them in



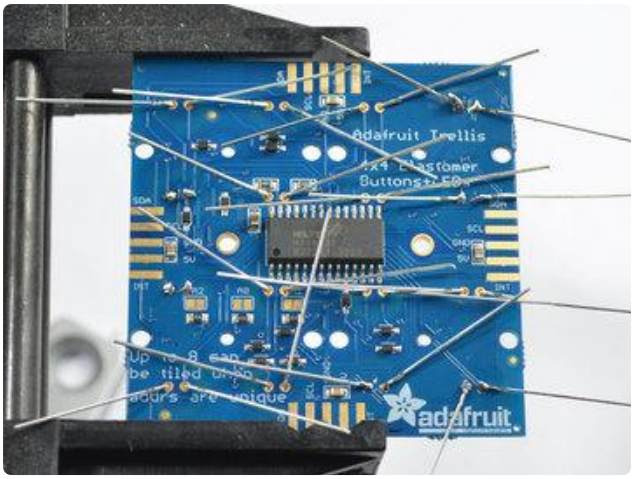



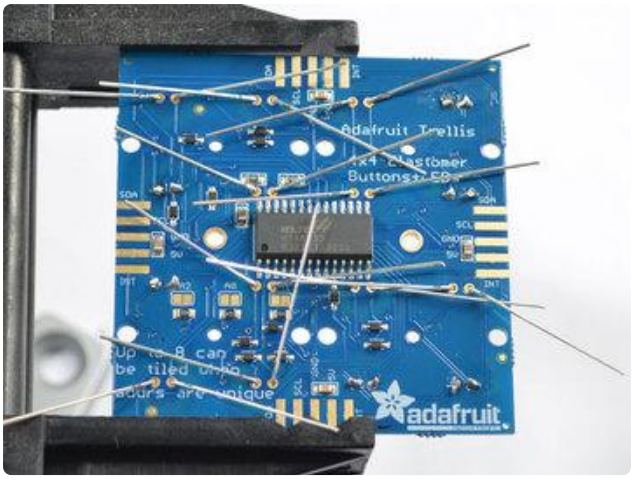

Solder each and every LED leg. I like to do one LED at a time.



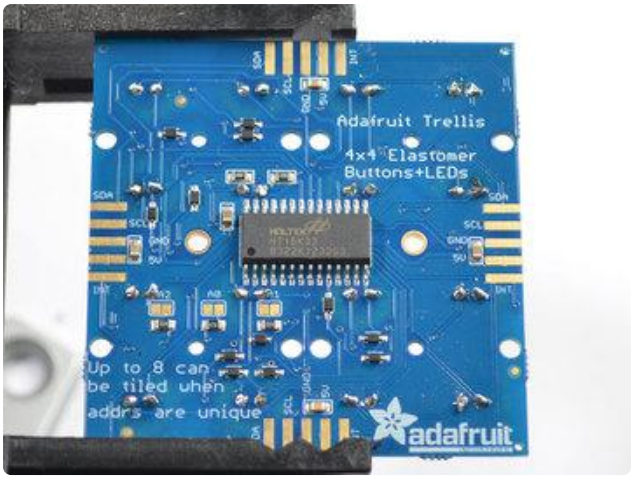




Clip the legs about 2-3mm from the PCB so the leads aren't flying everywhere



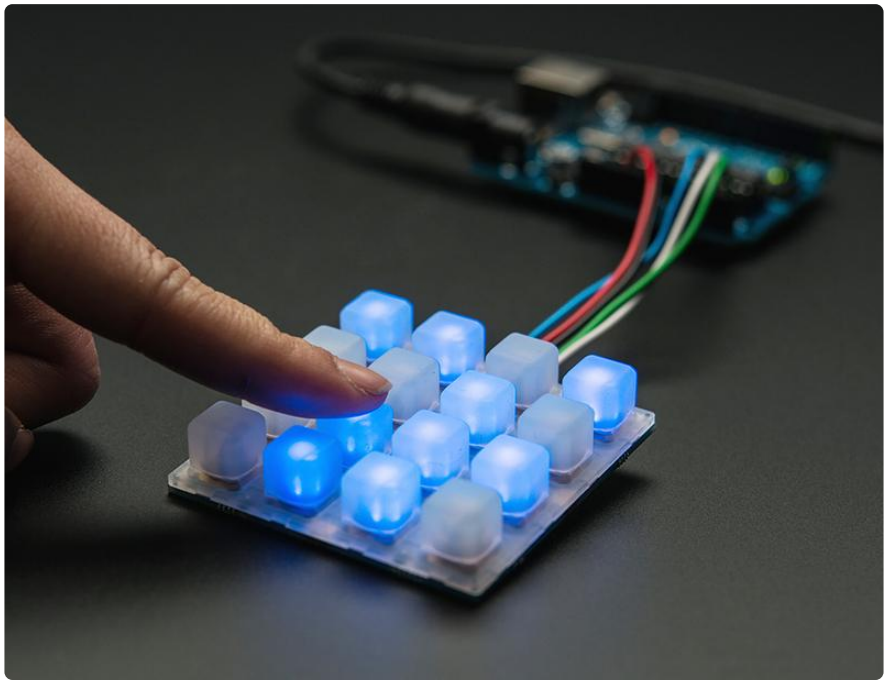




Keep soldering!




Till you are done with all of them

# Connecting



As shown here, Trellis is just an LED/keypad controller. You still need an Arduino or similar to do the work of reading the keypad data and then signifying when to light LEDs.

Luckily, we wrote a library that handles all the hard work for you, making it all very easy!

Download the Trellis Arduino library from our github repository (https://adafru.it/cZf) by clicking this shiny button

**Download Trellis Arduino library**

https://adafru.it/cZg

Rename the uncompressed folder **Adafruit_Trellis**. Check that the **Adafruit_Trellis** folder contains **Adafruit_Trellis.cpp** and **Adafruit_Trellis.h**, and an **examples** folder

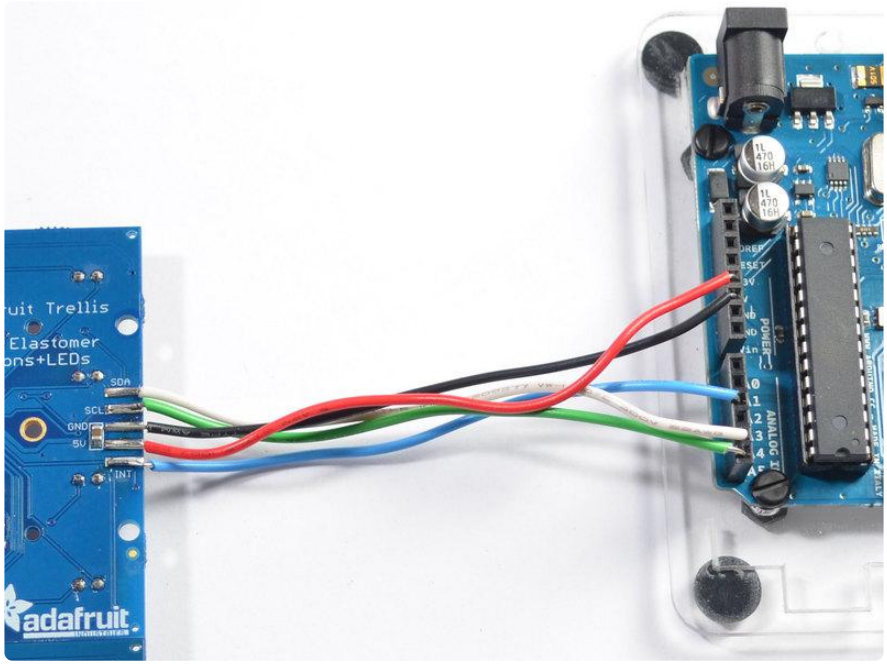
Place the **Adafruit_Trellis** library folder your **sketchbookfolder/libraries/** folder. You may need to create the libraries subfolder if its your first library. Restart the IDE. You can figure out your **sketchbookfolder** by opening up the Preferences tab in the Arduino IDE.

If you're not familiar with installing Arduino libraries, please visit our tutorial: All About Arduino Libraries (https://adafru.it/aYM)!

Next we will connect up the PCBs, we soldered solid-core wires from the 5 connection fingers and plug them into an Arduino. We suggest starting with an Uno since its guaranteed to work

Connect the wires:

- **5V** goes to the 5V power pin on the Arduino
- **GND** goes to and GND ground pin
- **SCL** goes to the I2C clock pin, on an Uno this is also known as A5
- **SDA** goes to the I2C data pin, on an Uno this is also known as A4
- We connect the **INT** interrupt pin to A2 - this pin isn't used in our demo code so you can leave it unconnected if you wish.




Now open up the Arduino IDE and open up the **File->Examples->Adafruit_Trellis->TrellisTest** example sketch and upload it to your Arduino

This sketch tests a single tile, with the default 0x70 address. It will light up all the LEDs in order and then turn them off. Then you can place the elastomer on top & press buttons to toggle buttons

# Library reference

The trellis example sketch shows you just about everything you can do with the Trellis library.

## Creating the objects

Each panel has its own named object called an Adafruit_Trellis, created like this:

```
Adafruit_Trellis matrix = Adafruit_Trellis();
```

when you have many Adafruit_Trellis objects, we suggest creating a TrellisSet which will read all the buttons at once, write all the LEDs at once, etc. Each TrellisSet is given the names of the Adafruit_Trellis objects you created, up to 8.

```
Adafruit_TrellisSet trellis = Adafruit_TrellisSet(&matrix0, &matrix1,
&matrix2, &matrix3);
```

When you call **begin** to start the **Adafruit_TrellisSet** object, pass in the addresses that correspond to your PCBs (see the next page on how to set addresses). The addresses range from 0x70 to 0x77

```
trellis.begin(0x70, 0x71, 0x72, 0x73); // or four!
```

## Controlling LEDs

You can set or clear LEDs with **trellis.setLED(n)** and **trellis.clrLED(n)** where **n** is the LED # from 0 to (number of Trellis')*16-1. So if you have 4 Trellis's in a set, thats 0 to 63

You can only turn LEDs on or off, there is no grayscale or PWM on this chip

When you are done setting and clearing LEDs you **must** call writeDisplay() to send the data to all the boards: **trellis.writeDisplay()** will write all Trellis PCBs in the set at once

You can also test if an LED is lit with **trellis.isLED(n)** - will return true if the LED is lit, and false if it isn't

# Reading Switches

You can read buttons by calling

**trellis.readSwitches()**

It will return **true** if there has been any **change** in switches since the last time you called readSwitches(). So if some buttons were pressed and now aren't or vice versa, it will return **true**. If nothing's changed, it will return **false**
Once you've read the switches, you can query the TrellisSet about them.
If you'd like to know if a key #**k** (k is 0..(number of Trellis')*16-1) is currently pressed, call

**isKeyPressed(k)**

If you want to know if there was a change in the button, you can ask if it's been pressed or released since the last call to readSwitches()

**trellis.justReleased(k)**
**trellis.justPressed(k)**

# Adding support for more tiles

You can tile up to 8 Trellis PCBs together (see the next page for the mechanical connections of doing so)

Make sure each Trellis has a unique address ID!

Open up the TrellisTest sketch and change the following

## Make more objects

After
**Adafruit_Trellis matrix0 = Adafruit_Trellis();**
add as many matrices as you like, each with a unique name, e.g.
**Adafruit_Trellis matrix1 = Adafruit_Trellis();**
**Adafruit_Trellis matrix2 = Adafruit_Trellis();**
etc...

## Make a bigger set

Next we will make a set of matrices. Instead of
**Adafruit_TrellisSet trellis = Adafruit_TrellisSet(&matrix0);**
update it to add up to 8 matrix names you defined. For example, 4 panels looks like:
**Adafruit_TrellisSet trellis = Adafruit_TrellisSet(&matrix0, &matrix1, &matrix2, &matrix3);**

## Say the number

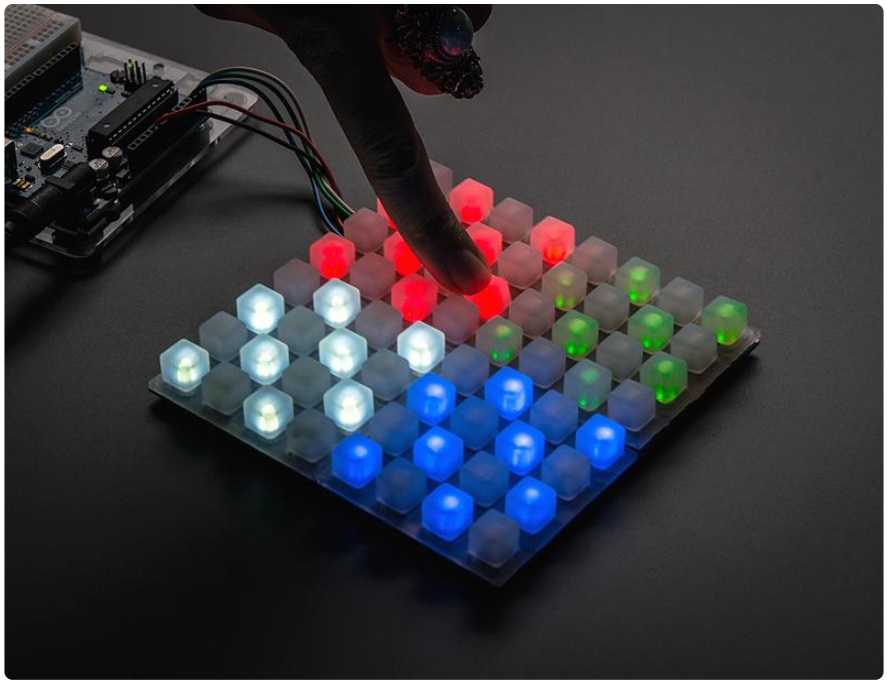
Change this number from 1 to whatever # you are addressing
// set to however many you're working with here, up to 8
**#define NUMTRELLIS 1**

Begin again
Change the begin() call to add more addresses. Originally we only have the default 0x70 address:
**trellis.begin(0x70); // only one**
Change this to add all the addresses you are using:
**trellis.begin(0x70, 0x71, 0x72, 0x73);** // four!

That's it! Now your TrellisSet will know and control up to 8 panels.

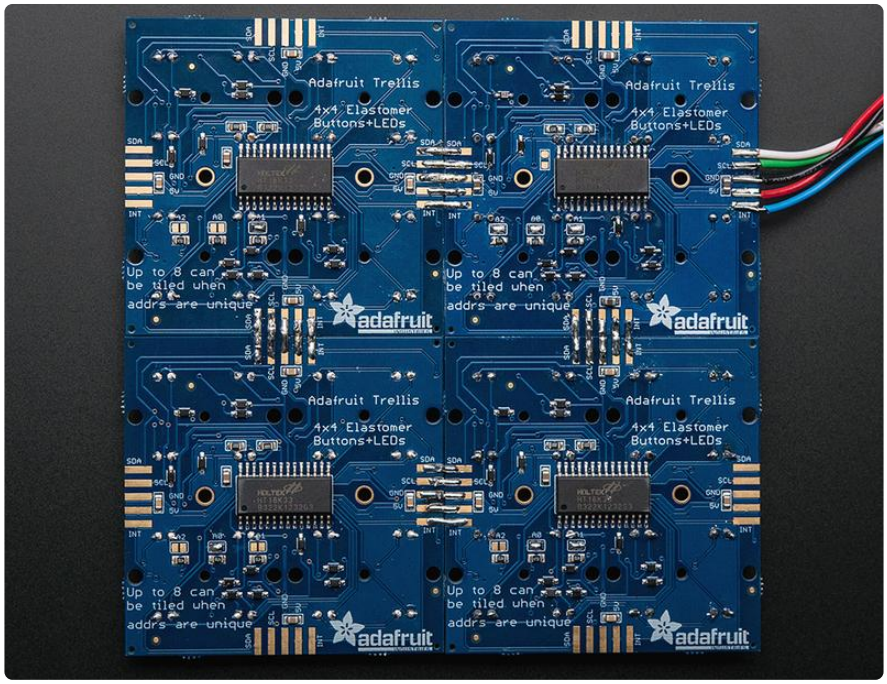
---

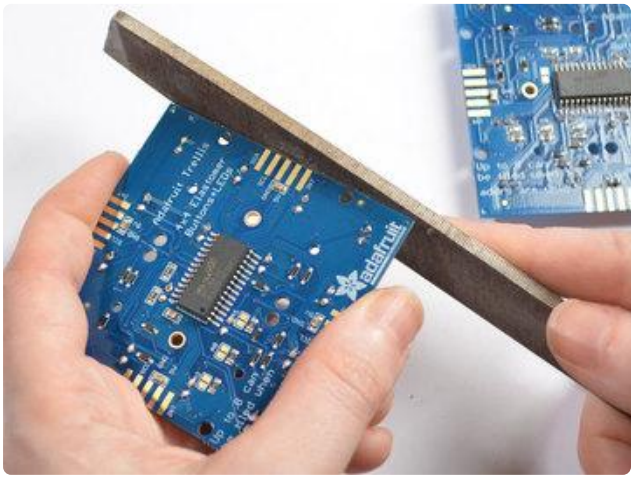
# Tiling



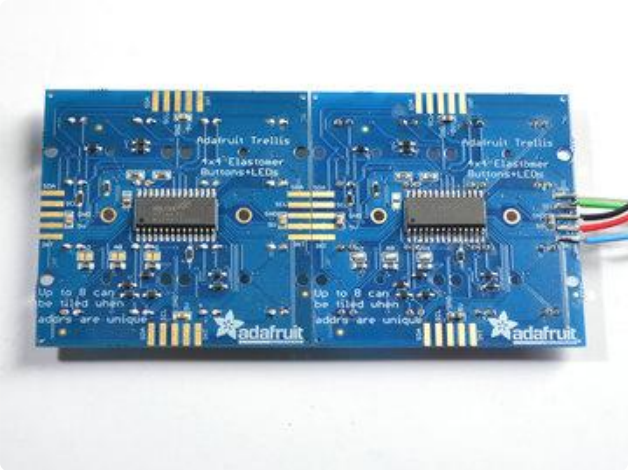
You can tile up to eight Trellis PCBs on a single 2-wire I2C bus. This allows you to easily build up to 8x16 or 4x32 panels which can be lots of fun!

To start with, its a good idea to assemble and test each individually so you know each Trellis works individually.



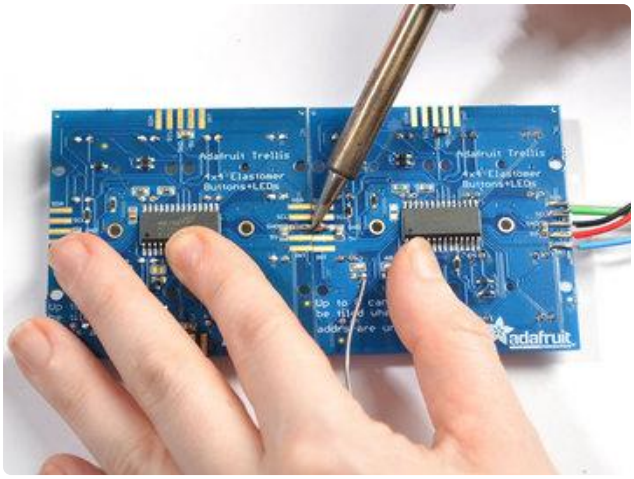



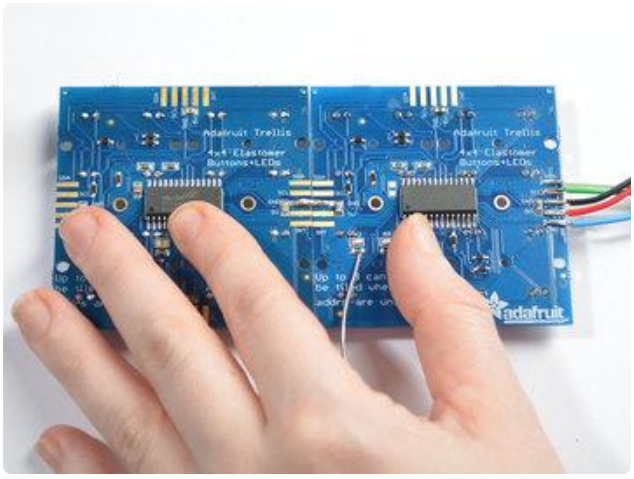
There's little nubs on the sides of some PCBs that keep them on the assembly panel, you can file them off with any file or sandpaper.




Arrange the tiles up the way you want, we'll start with two. Make sure the Adafruit logo is lined up the same.

Solder two blobs of solder on two adjacent finger pads.

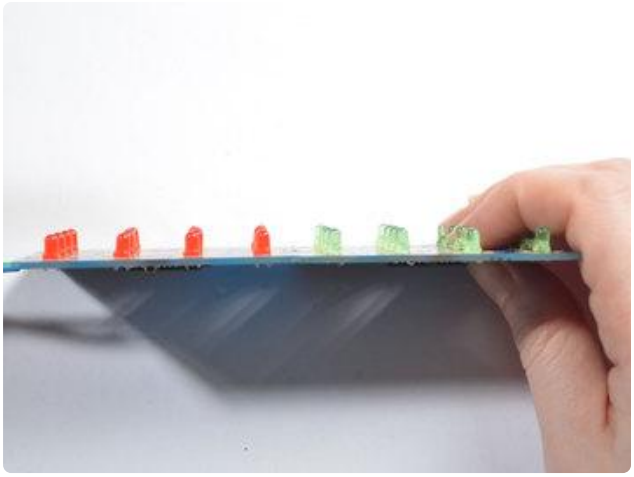


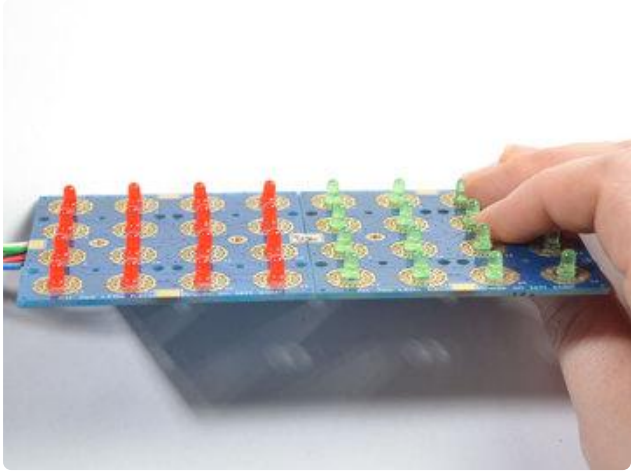


Use your soldering iron to drag solder from one pad to another, with a little effort they'll stick together and make a connection. You can add more solder to make the connection stronger. Its still not mechanically strong - so be careful not to bend or shake the arrangement

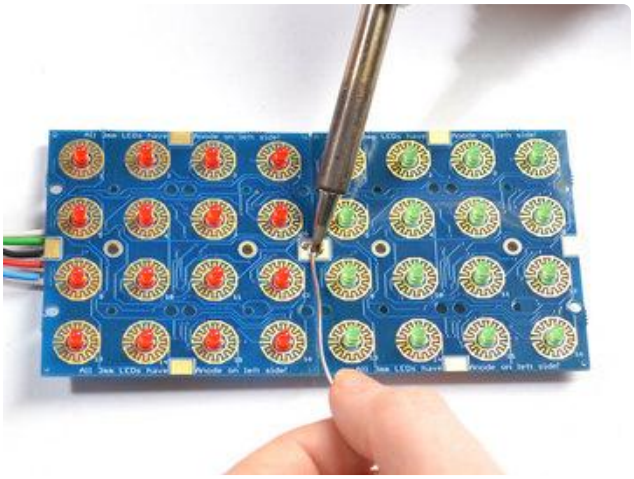
For a more robust connection, use a small bare wire across the pads instead of just solder. The legs trimmed off the LEDs are perfect for this!

Repeat for the other 4 fingers





Check that the panels are aligned by looking from the side, gently bend/reheat until they are nice and straight



You can add a little more mechanical stability by soldering the large front tabs as well

Repeat for up to 8 panels connected together, in any arrangement you like

# Addressing

Each Trellis tile must have a unique address. You can set the addresses on the back of each panel using a little solder over the address jumpers.

The HT16K33 driver chip on the Trellis has a default I2C address of **0x70**. Since each device on an I2C bus must have a unique address, its important to avoid collisions or you'll get a lot of strange responses from your electronic devices!

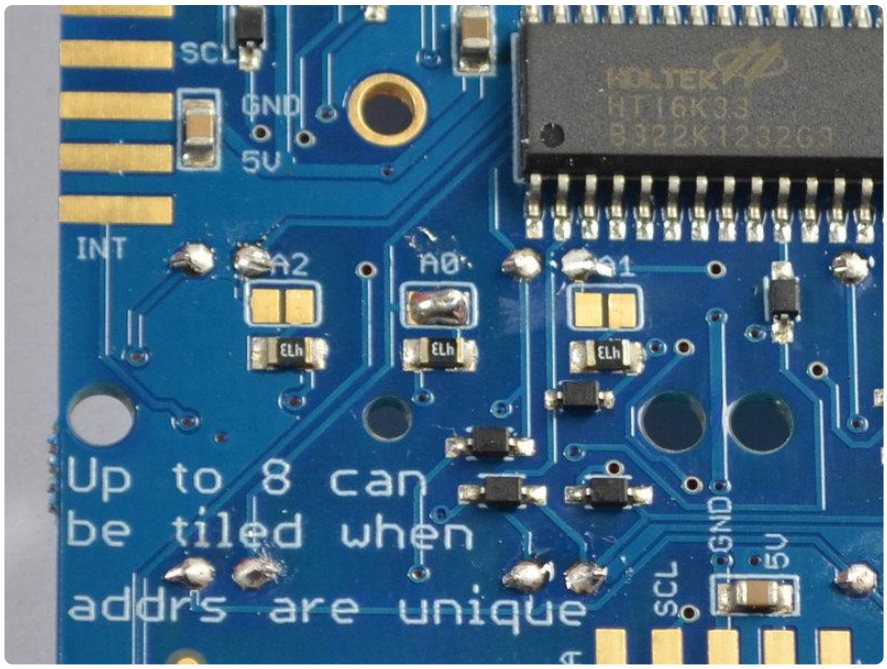
Luckily, the HT16K33 has 3 address adjust pins, so that the address can be changed. Each pin changes one binary bit of the address, so you can set the address to any of the following (in hex) 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77

The panels don't have to have consecutive address #'s, they just have to be unique.

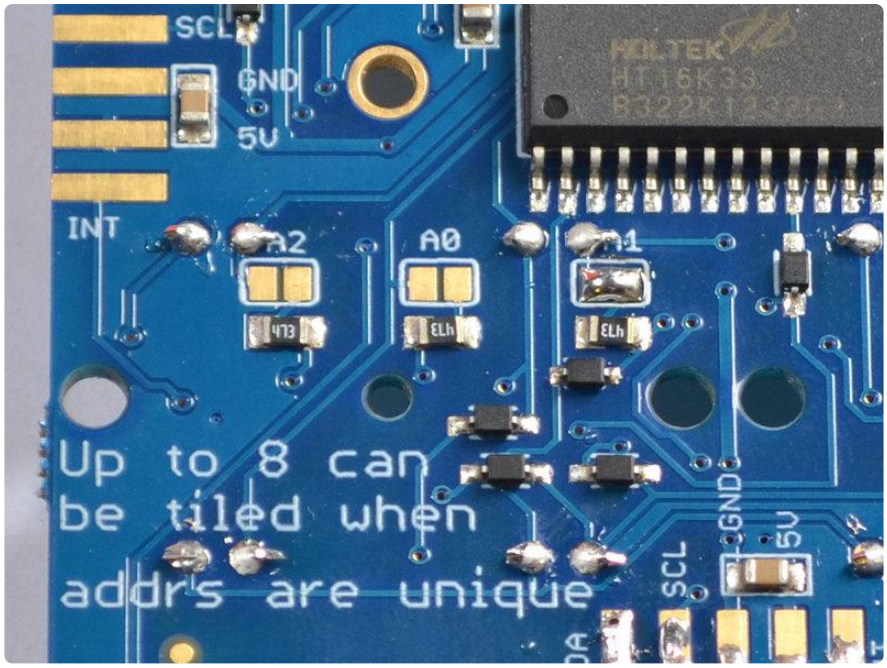
## (https://adafru.it/cZd)Changing Addresses

You can change the address of very easily. Look on the back to find the three **A0, A1 or A2** solder jumpers. Each one of these is used to hardcode in the address. If a jumper is shorted with solder, that sets the address. **A0** sets the lowest bit with a value of **1**, **A1** sets the middle bit with a value of **2** and **A2** sets the high bit with a value of **4**. The final address is **0x70** + **A2** + **A1** + **A0**. So for example if **A2** is shorted and **A0** is shorted, the address is **0x70** + **4** + **1** = **0x75.** If only A1 is shorted, the address is **0x70** + **2** = **0x72**

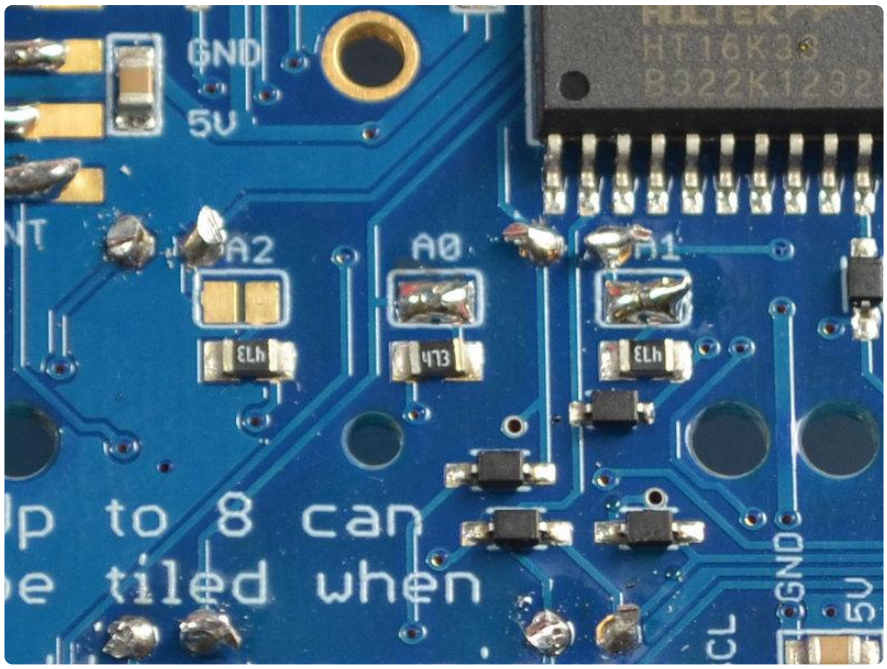
For example, this Trellis has **A0** shorted, the address is 0x71



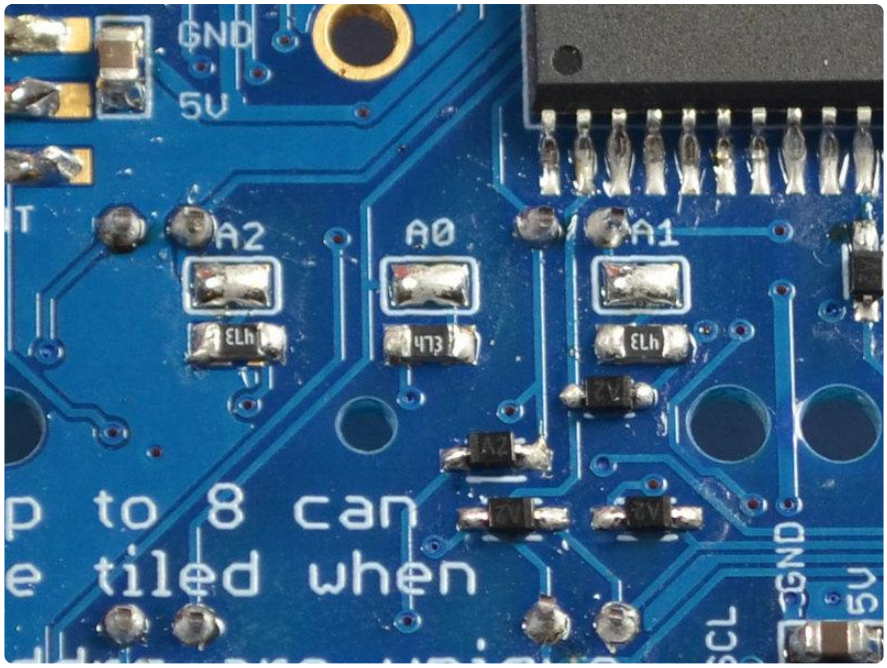

This one has **A1** shorted, the address is 0x72




If both **A0** and **A1** are shorted, the address is 0x73




If **A2**, **A0** and **A1** are shorted, the address is 0x77




Once you have set the unique addresses, you can set the addresses in the Arduino code, see the software page for details on how to do it.

---

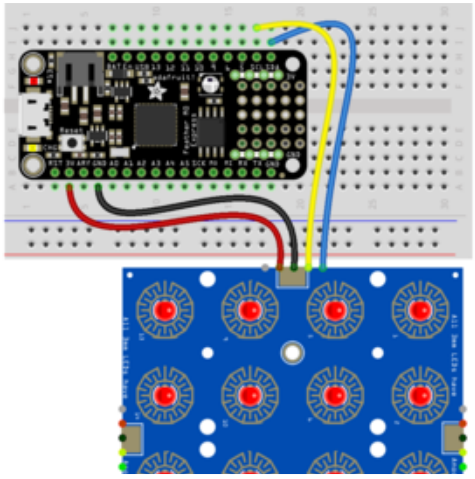
# Python & CircuitPython

It's easy to use the Adafruit Trellis with Python or CircuitPython, and the Adafruit CircuitPython Trellis module (https://adafru.it/F-C). This module enables you to write Python code that allows you to easily control the LEDs and read button presses on a single Trellis board, or with a matrix of up to eight Trellis boards.

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

## CircuitPython Microcontroller Wiring

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

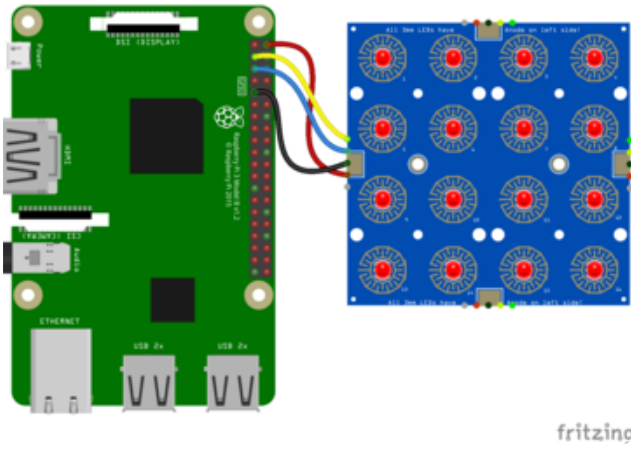


**Board 3V** to **Trellis 5V/VCC**
**Board GND** to **Trellis GND**
**Board SCL** to **Trellis SCL**
**Board SDA** to **Trellis SDA**

## 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).

Here's the Raspberry Pi wired to a Trellis:




**Pi 5V** to **Trellis 5V**
**Pi GND** to **Trellis GND**
**Pi SCL** to **Trellis SCL**
**Pi SDA** to **Trellis SDA**

## CircuitPython Installation of Trellis Library

Next you'll need to install the Adafruit CircuitPython Trellis (https://adafru.it/F-C) 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/ENC). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Copy the following libraries from the bundle to the **lib** folder on your **CIRCUITPY** drive:

- **adafruit_trellis.mpy**
- **adafruit_bus_device**

Before continuing make sure your board's **lib** folder or root filesystem has the **adafruit_trellis.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 Trellis 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-trellis`

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 Trellis, we'll initialize it, control the LEDs and read the button presses from the board's Python REPL.

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

```
import time
import busio
from board import SCL, SDA
from adafruit_trellis import Trellis

i2c = busio.I2C(SCL, SDA)
trellis = Trellis(i2c)
```

Now, you're ready to control the LEDs and read button presses with the following:

- **fill** - Turns on all of the LEDs
- **brightness** - Change the brightness of the LEDs
- **auto_show** - When set to True, LED changes update automatically. When set to False, you must call `show()` to update the LEDs. Defaults to True.
- **show()** - Required to update the LEDs if `auto_show=False`.
- **read_buttons** - read the button matrix on the Trellis board. Returns two lists: one for button presses and one for button releases.

To turn on all the LEDs:

```
trellis.led.fill(True)
```

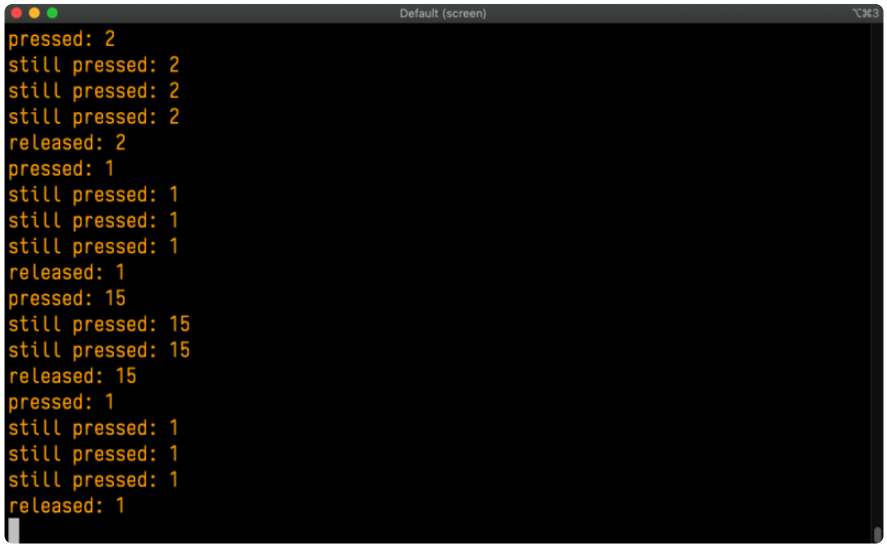
To turn off all the LEDs:

```
trellis.led.fill(False)
```

To read when the buttons are pressed and released:

```
pressed_buttons = set()
while True:
    time.sleep(.1)
    just_pressed, released = trellis.read_buttons()
    for b in just_pressed:
        print('pressed:', b)
        trellis.led[b] = True
    pressed_buttons.update(just_pressed)
    for b in released:
        print('released:', b)
        trellis.led[b] = False
    pressed_buttons.difference_update(released)
```

```
    for b in pressed_buttons:
        print('still pressed:', b)
        trellis.led[b] = True
```




That's all there is to using your Trellis with CircuitPython!

## Full Example Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Basic example of turning on LEDs and handling Keypad
# button activity.

# This example uses only one Trellis board, so all loops assume
# a maximum of 16 LEDs (0-15). For use with multiple Trellis boards,
# see the documentation.

import time
import busio
from board import SCL, SDA
from adafruit_trellis import Trellis

# Create the I2C interface
i2c = busio.I2C(SCL, SDA)

# Create a Trellis object
trellis = Trellis(i2c)  # 0x70 when no I2C address is supplied

# 'auto_show' defaults to 'True', so anytime LED states change,
# the changes are automatically sent to the Trellis board. If you
# set 'auto_show' to 'False', you will have to call the 'show()'
# method afterwards to send updates to the Trellis board.

# Turn on every LED
print("Turning all LEDs on...")
trellis.led.fill(True)
time.sleep(2)

# Turn off every LED
print("Turning all LEDs off...")
trellis.led.fill(False)
time.sleep(2)
```

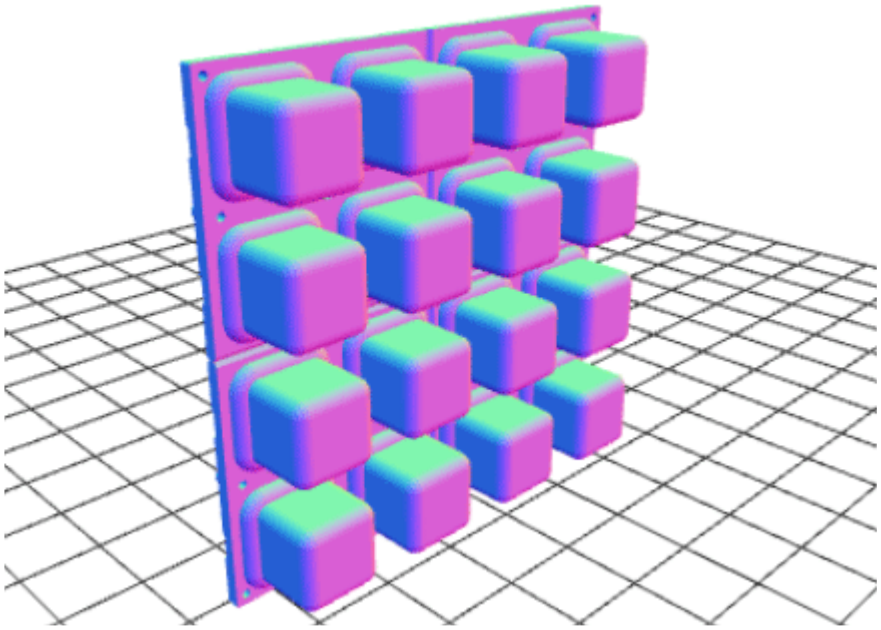
```
# Turn on every LED, one at a time
print("Turning on each LED, one at a time...")
for i in range(16):
    trellis.led[i] = True
    time.sleep(0.1)

# Turn off every LED, one at a time
print("Turning off each LED, one at a time...")
for i in range(15, 0, -1):
    trellis.led[i] = False
    time.sleep(0.1)

# Now start reading button activity
# - When a button is depressed (just_pressed),
#   the LED for that button will turn on.
# - When the button is relased (released),
#   the LED will turn off.
# - Any button that is still depressed (pressed_buttons),
#   the LED will remain on.
print("Starting button sensory loop...")
pressed_buttons = set()
while True:
    # Make sure to take a break during each trellis.read_buttons
    # cycle.
    time.sleep(0.1)

    just_pressed, released = trellis.read_buttons()
    for b in just_pressed:
        print("pressed:", b)
        trellis.led[b] = True
    pressed_buttons.update(just_pressed)
    for b in released:
        print("released:", b)
        trellis.led[b] = False
    pressed_buttons.difference_update(released)
    for b in pressed_buttons:
        print("still pressed:", b)
        trellis.led[b] = True
```

---

# Python Docs

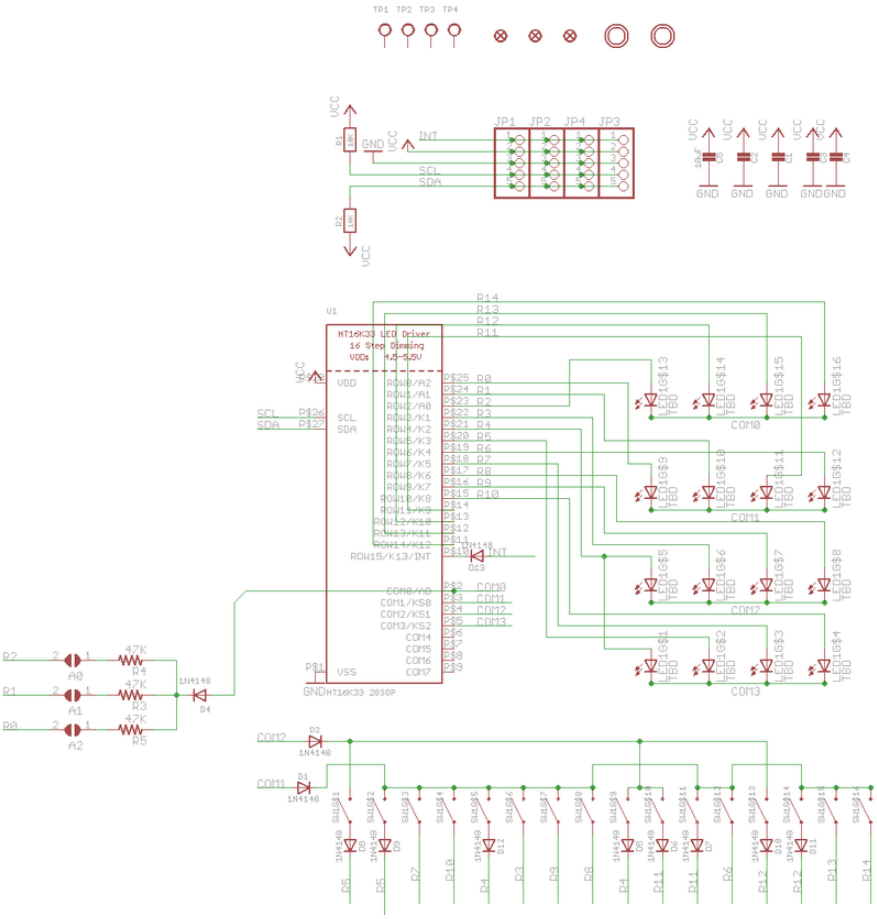
Python Docs (https://adafru.it/G0D)

# Downloads



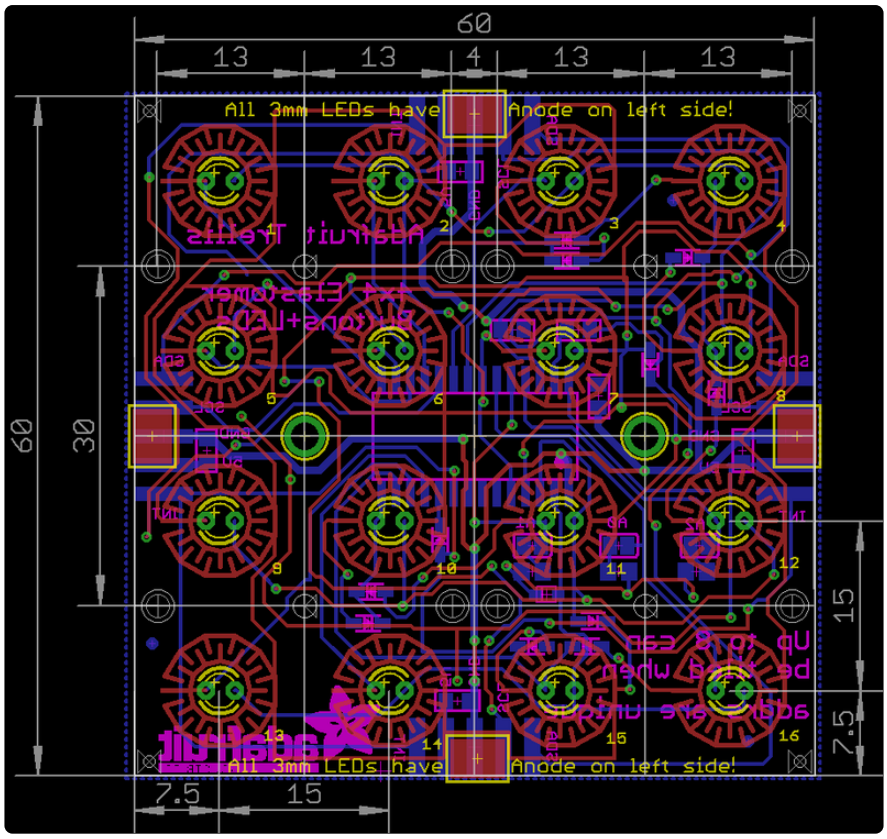
- EagleCAD PCB and CAD files for the Trellis buttons and PCB from the github repo here (https://adafru.it/cZh)
- You can get the Arduino library for Trellis from the github repo here (https://adafru.it/cZf)
- Fritzing object in the Adafruit Fritzing library (https://adafru.it/aP3)

# Schematic



# Fabrication Print

Dims in mm


60
13
13
4
13
13
All 3mm LEDs have
Anode on left side!
60
30
15
7.5
7.5
15
All 3mm LEDs have
Anode on left side!