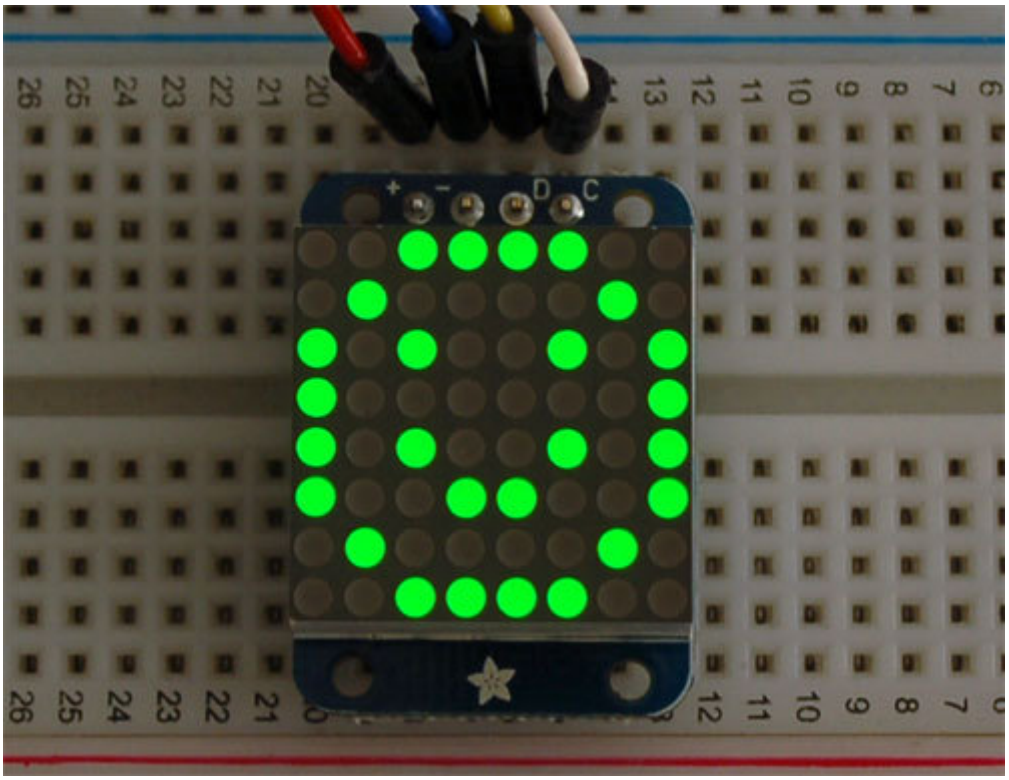
Adafruit LED Backpacks

Created by Melissa LeBlanc-Williams



https://learn.adafruit.com/adafruit-led-backpack

Last updated on 2025-06-11 04:11:58 PM EDT

Table of Contents

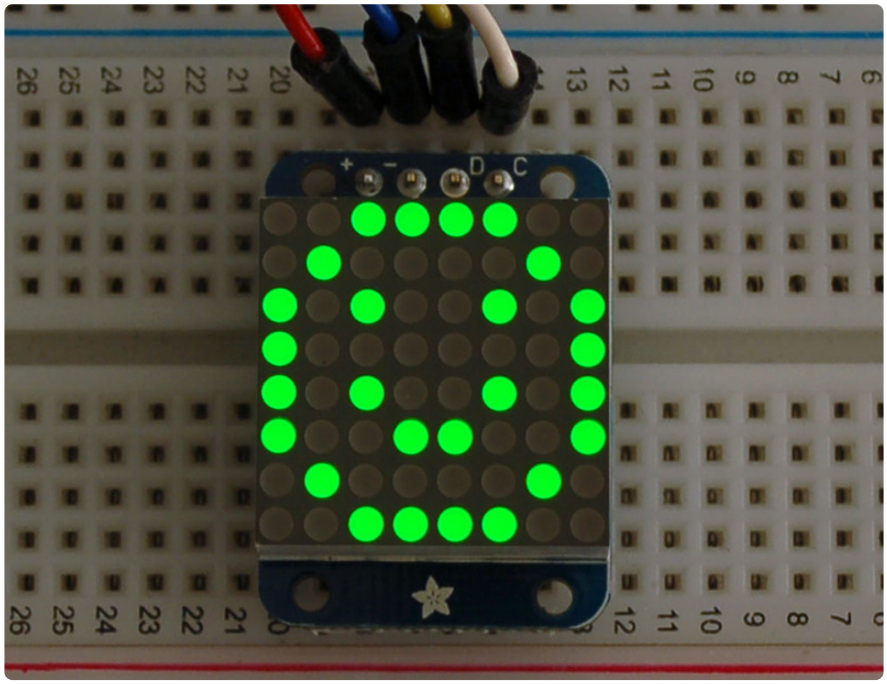
- Bicolor 24-Bargraph

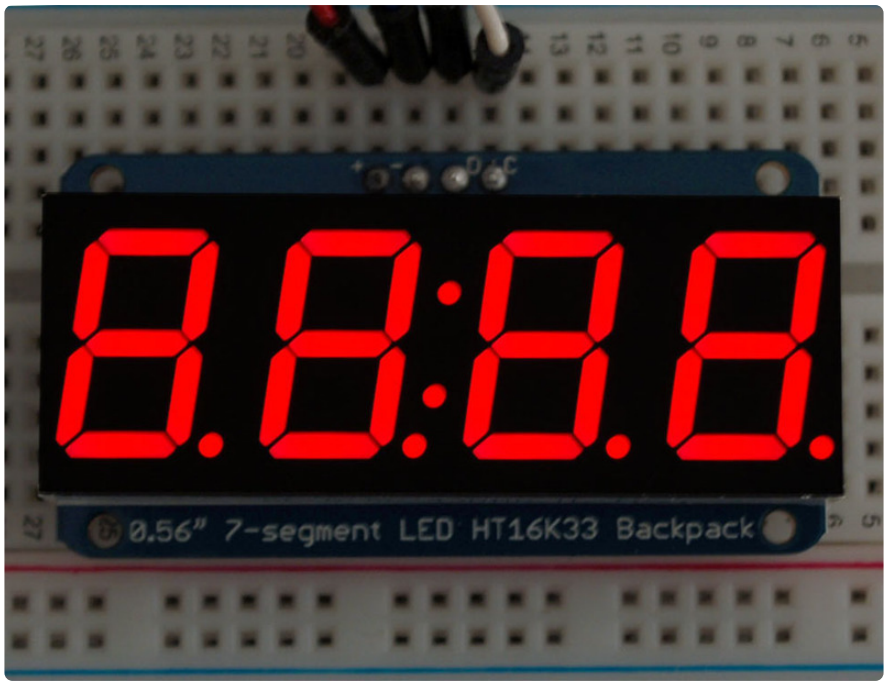
Overview

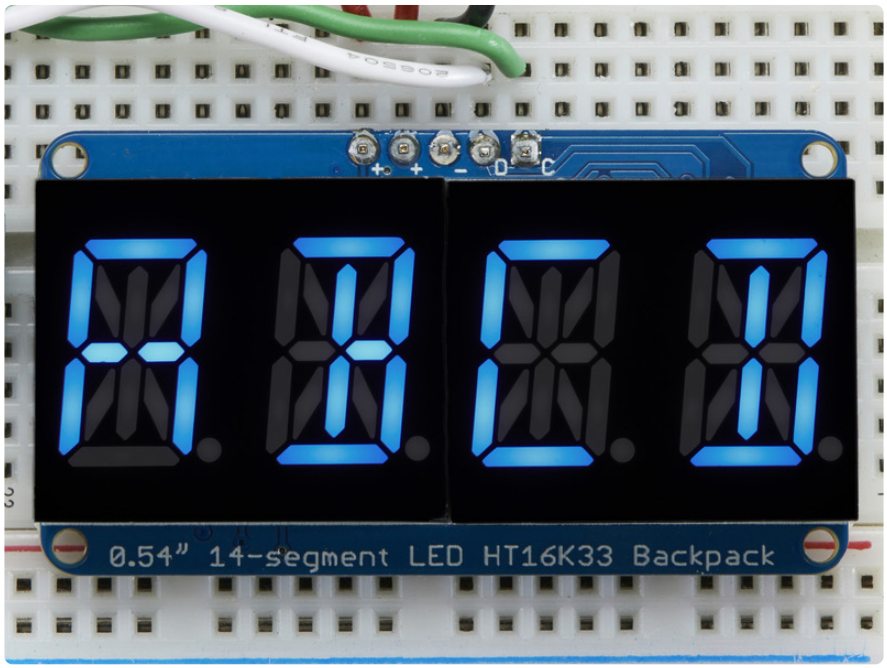
What's better than a single LED? Lots of LEDs! A fun way to make a small display is to use an 8x8 matrix (https://adafru.it/aLG) or a 4-digit 7-segment display (https://adafru.it/aLH). Matrices like these are 'multiplexed' - so to control 64 LEDs you need 16 pins. That's a lot of pins, and there are driver chips like the MAX7219 (http://adafru.it/453) that can control a matrix for you but there's a lot of wiring to set up and they take up a ton of space. Here at Adafruit we feel your pain! After all, wouldn't it be awesome if you could control a matrix without tons of wiring? That's where these adorable LED matrix backpacks come in.

We have them in quite a few flavors!

- Adorable Mini 8x8 (https://adafru.it/dxb)
- Classic 1.2" 8x8 (round and square dots) (https://adafru.it/18dB)
- Mini 1.2" 16x8 (round and square dots) (https://adafru.it/18dC)
- 4-digit 0.56" 7-segment (https://adafru.it/dxd)
- 4-digit 1.2" 7-segment (https://adafru.it/18dD)
- 4-digit 0.54" 14-segment Alphanumeric (https://adafru.it/dxf)
- Bi-color 8x8 (http://adafru.it/902)
- Bi-color Bargraph (http://adafru.it/1721)







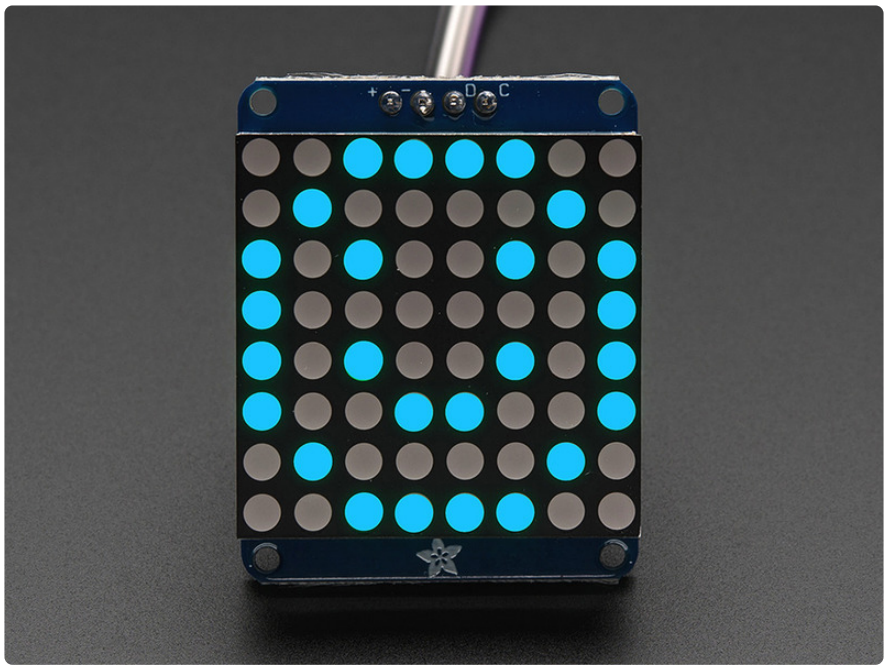
The matrices use a driver chip that does all the heavy lifting for you: They have a built in clock so they multiplex the display. They use constant-current drivers for ultra-bright, consistent color (the images above are photographed at the dimmest setting to avoid overloading our camera!), 1/16 step display dimming, all via a simple I2C interface. The backpacks come with address-selection jumpers so you can connect up to four mini 8x8's or eight 7-segments (or a combination, such as four mini 8x8's and four 7-segments, etc) on a single I2C bus.

The product kit comes with a fully tested and assembled LED backpack, a 4-pin header and the matrix of your choice. A bit of soldering is required to attach the matrix onto the backpack but it's very easy to do and only takes about 5 minutes.

Of course, in classic Adafruit fashion, we also have a detailed tutorial showing you how to solder, wire and control the display. We even wrote a very nice library for the backpacks so you can get running in under half an hour, displaying images on the matrix or numbers on the 7-segment. If you've been eyeing matrix displays but hesitated because of the complexity, his is the solution you've been looking for!

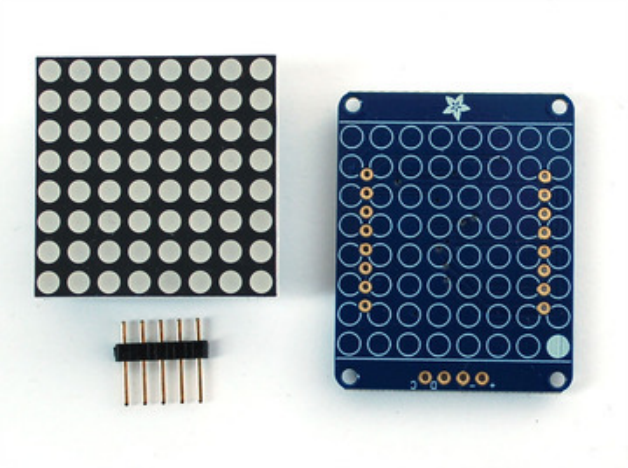
1.2" 8x8 Matrix

This version of the LED backpack is designed for the 1.2" 8x8 matrices. They measure only 1.2"x1.2" so its a shame to use a massive array of chips to control it. This backpack solves the annoyance of using 16 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all 64 LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are two address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.



Assembly

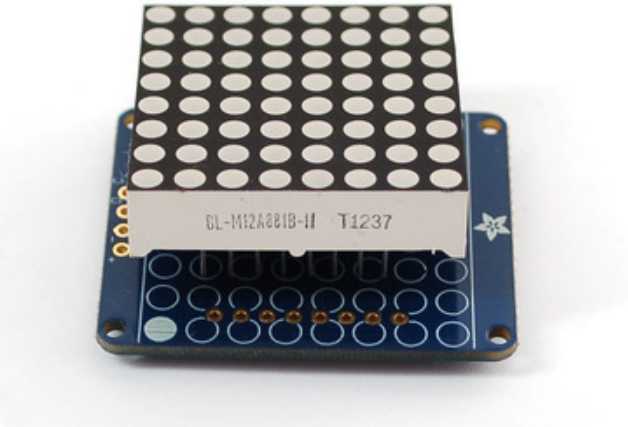
These instruction apply to the 1.2" Matrix only! If you have a Bi-Color or 0.8" square matrix, follow the links on the left side of the page.



When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as a 8x8 matrix in one of the colors we provide (say, red, yellow or green). You'll need to solder the matrix onto the backpack but its an easy task.

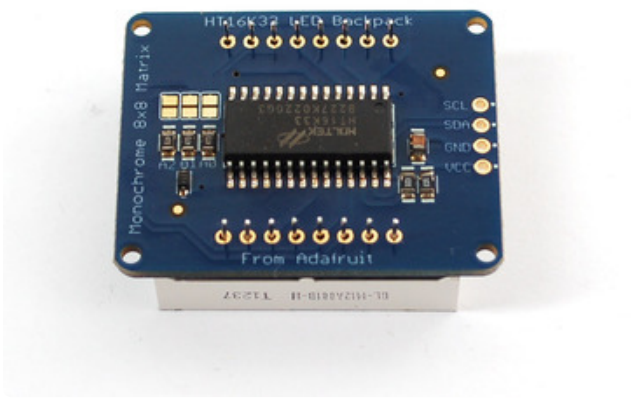
WATCH OUT! THE MATRIX MUST BE INSTALLED THE RIGHT WAY!

First look for the line of text on the side of the LED matrix



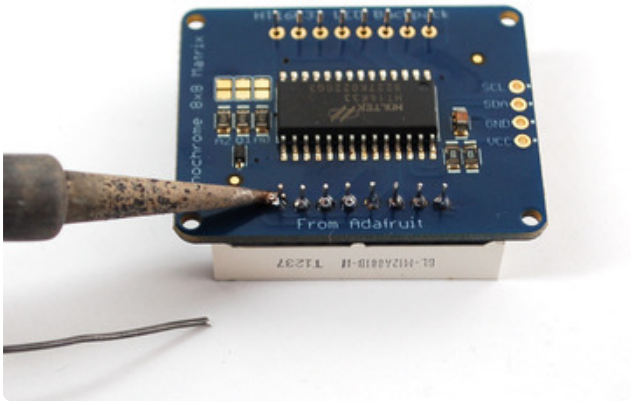
WATCH OUT! THE MATRIX MUST BE INSTALLED THE RIGHT WAY!

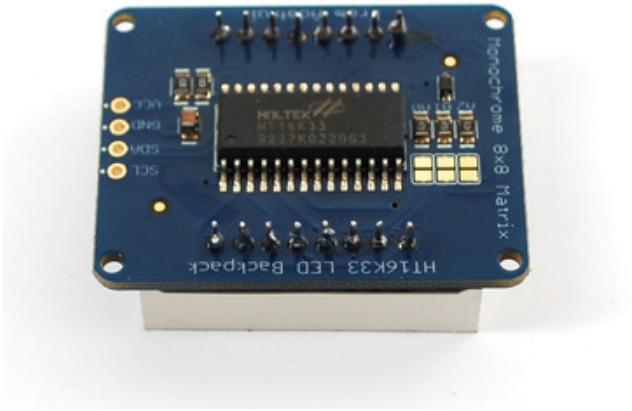
Find the corner of the backpack with a filled in dot. Make sure that the text on the side of the matrix is on the same side as the filled dot



WATCH OUT! THE MATRIX MUST BE INSTALLED THE RIGHT WAY!

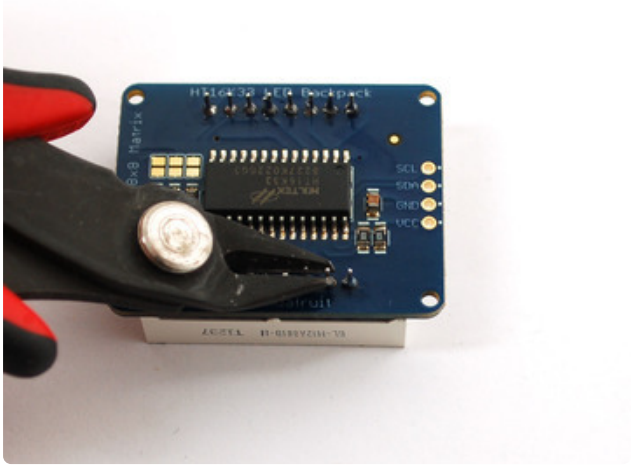
Slide the matrix into the backpack and flip it over. Triple check that the text is on the same side as the **From Adafruit** text

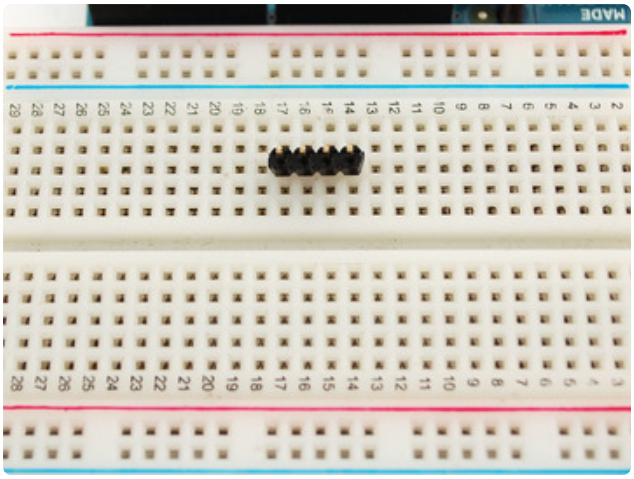




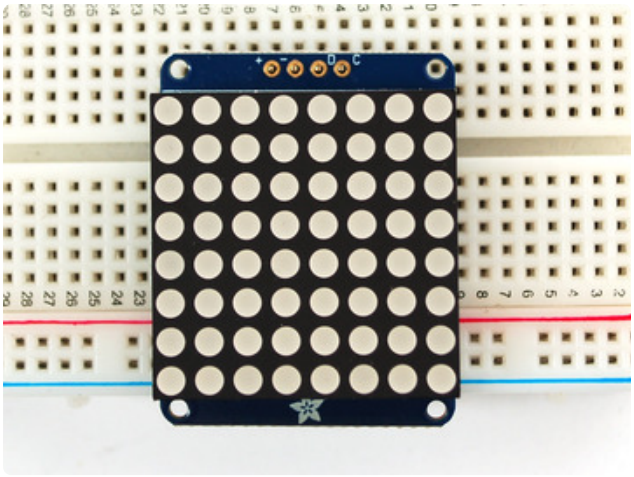
Solder in all 16 pins

Then clip the matrix leads short

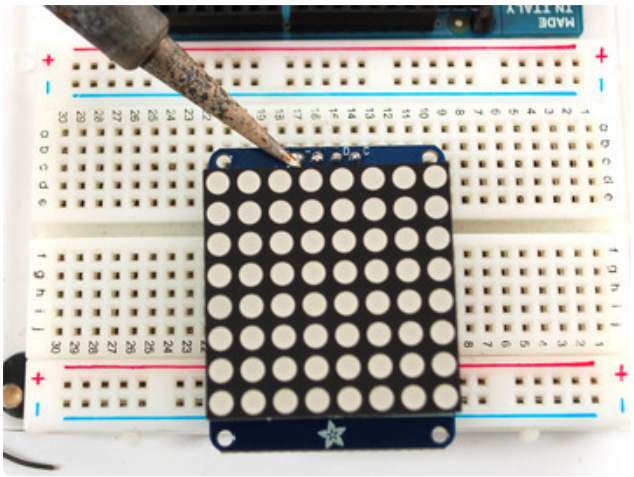




Now you're ready to wire it up to a microcontroller. We'll assume you want to use a 4pin header. You can also of course solder wires directly. Place a 4-pin piece of header with the LONG pins down into the breadboard.



Place the soldered backpack on top of the header.



Solder the four pins

That's it! now you're ready to run the firmware!

Arduino Setup

You can use these with a 3.3v or 5v microcontroller. Just connect the VCC+ pin is the same voltage as the logic on your microcontroller.

Mini 8x8 Matrix Software

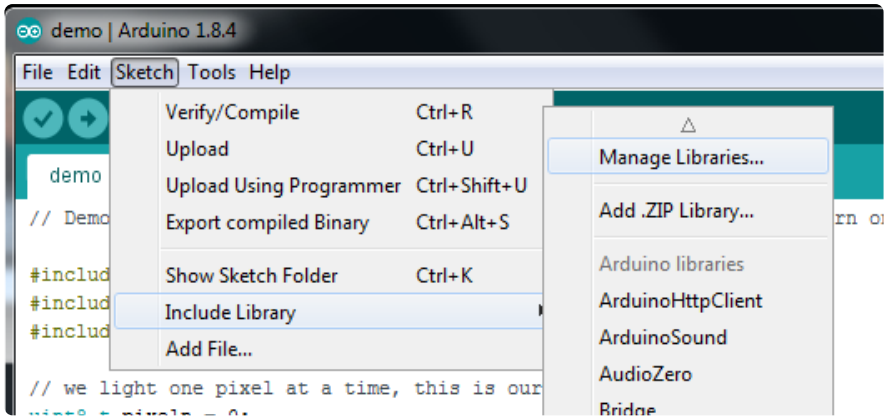
We wrote a basic library to help you work with the mini 8x8 matrix backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

Wiring to the matrix is really easy

- Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo its Digital #3, on the Mega its digital #21
- Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo its Digital #2, on the Mega its digital #20
- Connect **GND** to common ground
- Connect **VCC+** to power - 5V is best but 3V also seems to work for 3V microcontrollers.

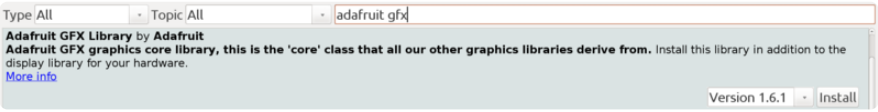
Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it

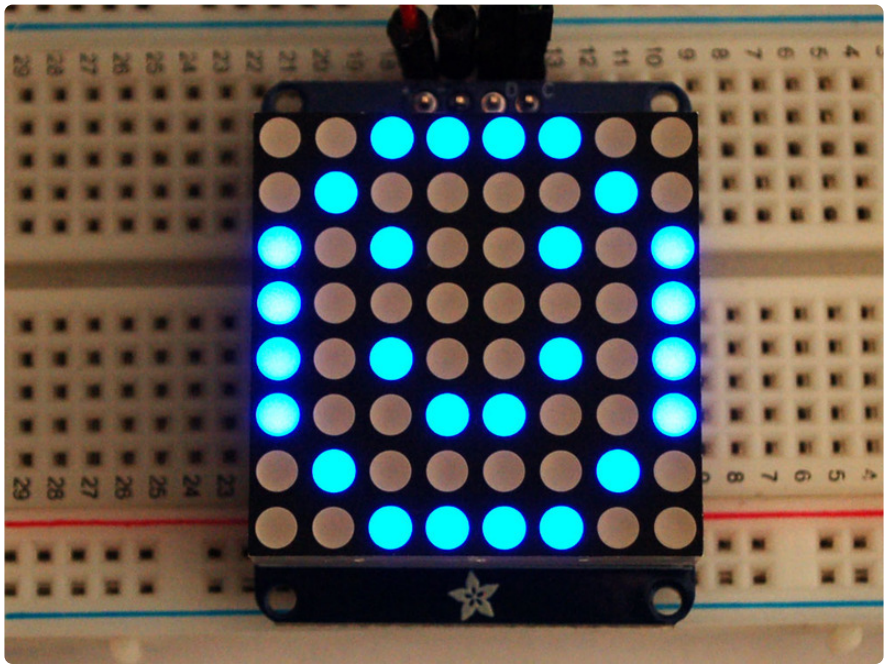
Search for the **Adafruit GFX** library and install it



If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

You should now be able to select the **File→Examples→Adafruit_LEDBackpack→matrix88** example sketch. Upload it to your Arduino as usual. You should see a basic test program that goes through a bunch of different drawing routine

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)

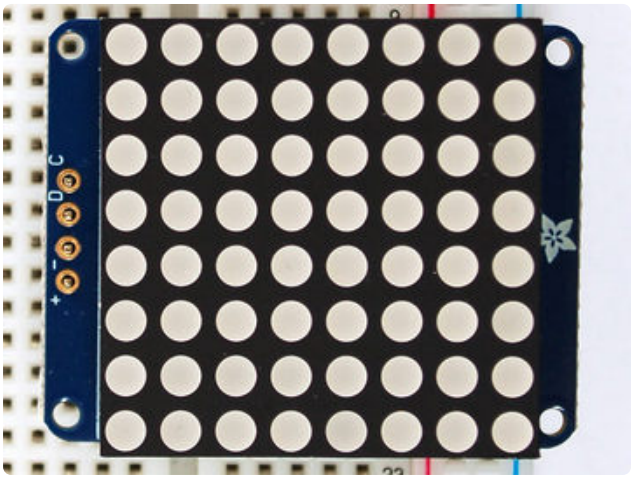


Once you're happy that the matrix works, you can write your own sketches. The 8x8 matrix supports everything the Adafruit GFX library - drawing pixels, lines, rectangles, circles, triangles, roundrects, and small bitmaps. For more details check out the GFX page which will detail all of the GFX routines (https://adafru.it/aPx).

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the matrix:

- **setBrightness(brighness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start. You can call this function at any time to change the brightness of the -entire- display
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.You can call this function at any time to change the blink rate of the -entire- display



The default orientation for graphics commands on this display places pixel (0,0) at the top-left when the header is at the left and Adafruit logo at the right. To use the matrix as shown above (header at top, logo at bottom), call matrix.setRotation(3) before issuing graphics commands.

CircuitPython Wiring and Setup

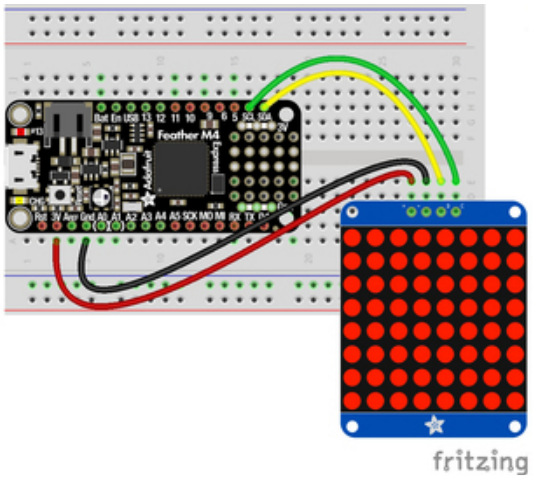
Wiring

It's easy to use LED Matrices with CircuitPython and the [Adafruit CircuitPython HT16K33 (https://adafru.it/u1E)](https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the LED Matrix to your CircuitPython microcontroller board. First assemble your LED Matrix.

Connect the LED Matrix to your microcontroller board as shown below.



Microcontroller 3V to **LED Matrix VIN**
Microcontroller GND to **LED Matrix GND**
Microcontroller SCL to **LED Matrix SCL**
Microcontroller SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/lfy

Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

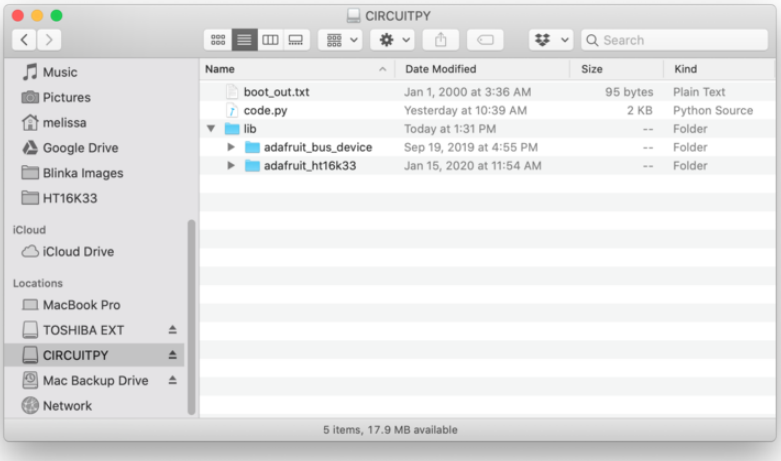
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

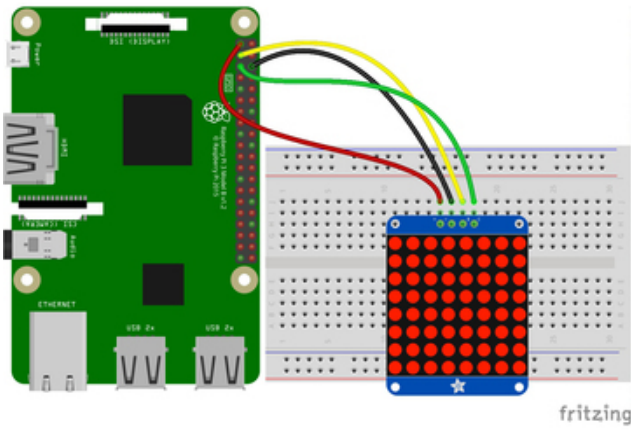
Wiring

It's easy to use LED Matrices with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the LED Matrix to your Raspberry Pi. First assemble your LED Matrix.

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).

Connect the LED Matrix as shown below to your Raspberry Pi.



Raspberry Pi 3.3V to **LED Matrix VIN**
Raspberry Pi GND to **LED Matrix GND**
Raspberry Pi SCL to **LED Matrix SCL**
Raspberry Pi SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/lfz

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `Matrix8x8`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the matrix, you just pass `i2c` in.

> When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.matrix import Matrix8x8

i2c = board.I2C()
matrix = Matrix8x8(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
matrix = Matrix8x8(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
matrix.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual pixels. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
matrix.blink_rate = 3
```

Setting Individual Pixels

To set individual pixels to on, you simply treat the `matrix` object as a multidimensional list and set it to 1.

```
matrix[0, 0] = 1
matrix[4, 4] = 1
matrix[7, 7] = 1
```

Filling the Entire Matrix

To fill the entire matrix, just use the fill() function and pass in either 0 or 1 depending on whether you want all pixels off or on. For instance, if you wanted to set everything to on, you would use:

```
matrix.fill(1)
```

Shifting the Matrix

To shift the pixels on the matrix, there are 5 functions you can use. The main function, called shift(), is used to shift the pixels, up, down, left, right, or even diagonally. By passing a positive number, it will shift the pixels right/up and passing a negative number will shift them left/down. For instance:

```
matrix.shift(2, 0)    # shift pixels to the right by 2
matrix.shift(-1, 0)    # shift pixels to the left by 1
matrix.shift(0, -3)    # shift pixels down by 3
matrix.shift(-2, 2)    # shift pixels left by 2 and up by 2
```

You can pass `True` as a third parameter to loop all the pixels that get shifted off over to the other side.

```
matrix.shift(2, 0, True)    # loop pixels to the right by 2
matrix.shift(-1, 0, True)    # loop pixels to the left by 1
matrix.shift(0, -3, True)    # loop pixels down by 3
matrix.shift(-2, 2, True)    # loop pixels left by 2 and up by 2
```

Additionally, there are a few convenience functions that will shift the pixels by one. These can also be passed a value of `True` to loop the pixels.

```
matrix.shift_up()        # Shift pixels up
matrix.shift_left()        # Shift pixels left
matrix.shift_down()        # Shift pixels down
matrix.shift_right()        # Shift pixels right
matrix.shift_up(True)        # Loop pixels up
matrix.shift_left(True)    # Loop pixels left
matrix.shift_down(True)    # Loop pixels down
matrix.shift_right(True)    # Loop pixels right
```

Displaying an Image (Pillow Only)

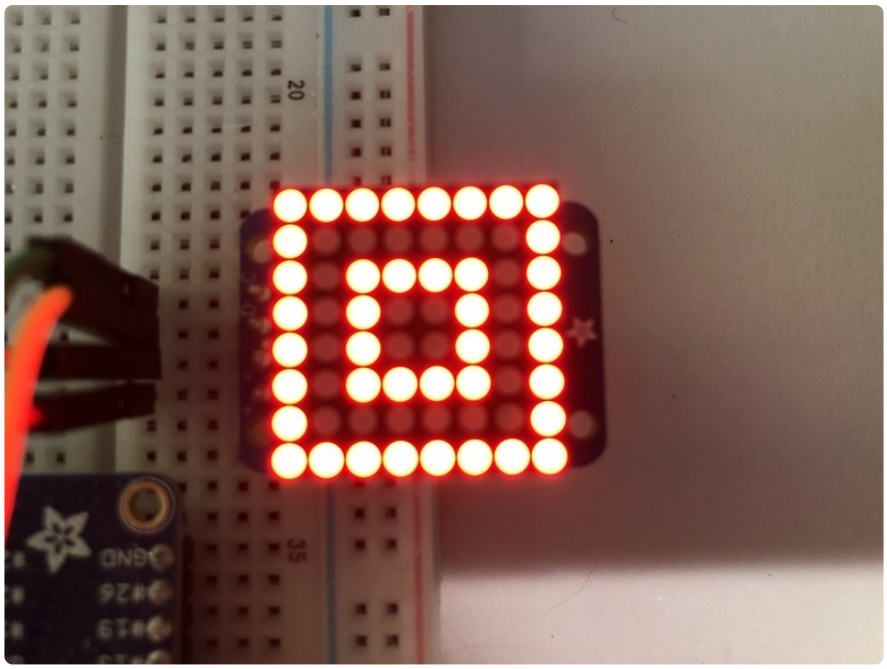
Additionally, when using with the Raspberry Pi, you can use the Pillow library to display an image to the Matrix. The image will need to be the same exact size as the Matrix. In this case, it should be **8x8** pixels. As an example, you can save the image below as **myimage.png**.

Download Image

https://adafru.it/ICR

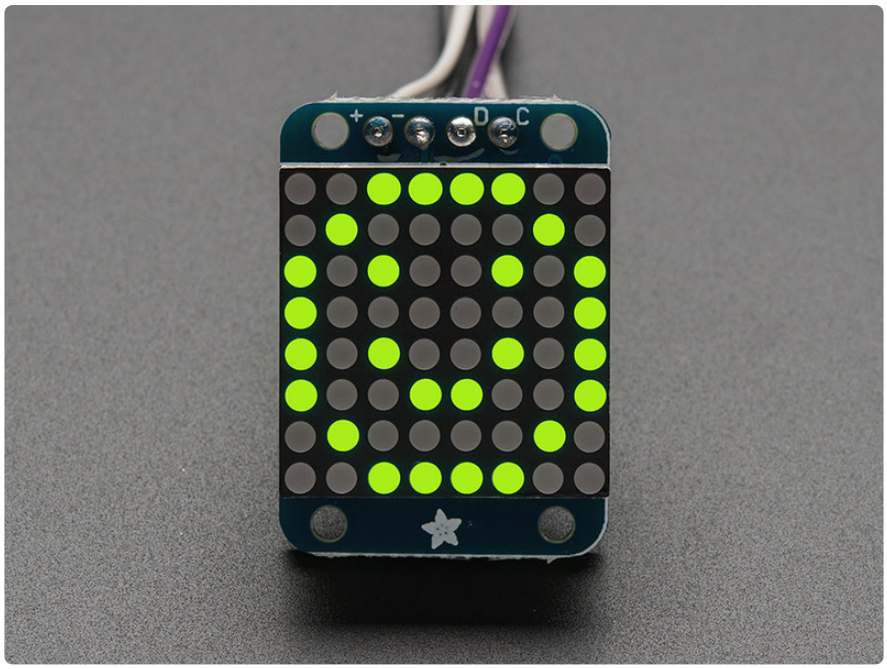
Then if you want to display the image called **myimage.png**, you would use something like this:

```
import board
from PIL import Image
from adafruit_ht16k33 import matrix

matrix = matrix.Matrix8x8(board.I2C())

image = Image.open("myimage.png")
matrix.image(image)
```



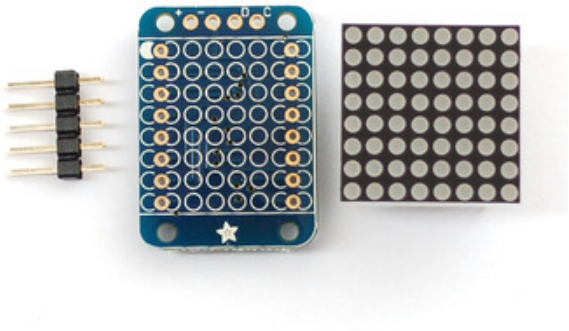
0.8" 8x8 Matrix

This version of the LED backpack is designed for these very cute miniature 8x8 matrices. They measure only 0.8"x0.8" so its a shame to use a massive array of chips to control it. This backpack solves the annoyance of using 16 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all 64 LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are two address select pins so you can select one of 4 addresses to control up to 4 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.

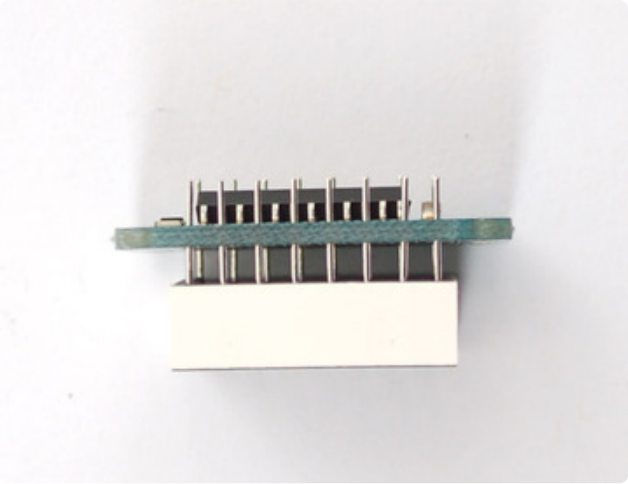


Assembly

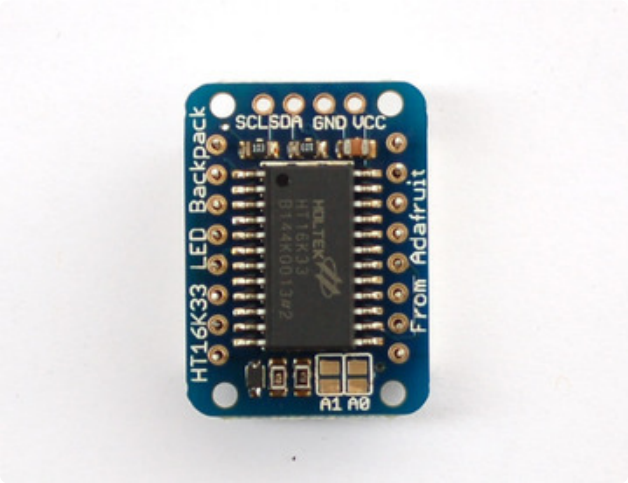
These instruction apply to the 0.8" Matrix only! If you have a Bi-Color or 1.2" square matrix, follow the links on the left side of the page.



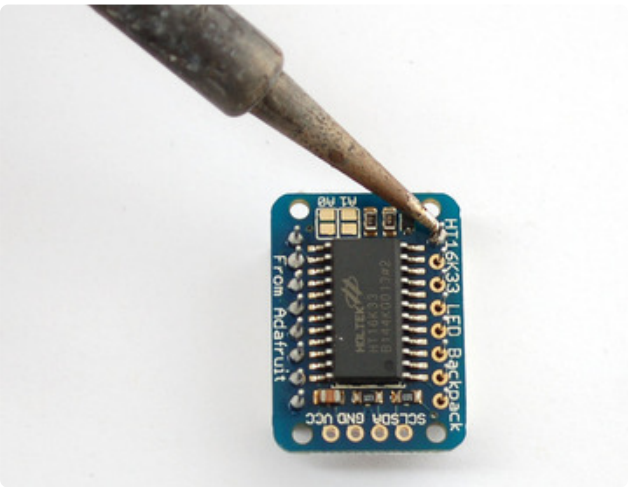
When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as a 8x8 matrix in one of the colors we provide (say, red, yellow or green). You'll need to solder the matrix onto the backpack but its an easy task.



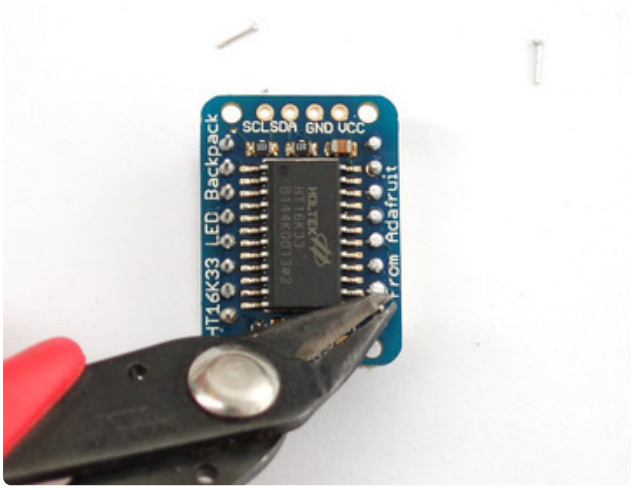
Remove the parts from packaging and place the LED matrix OVER the silkscreen side. It can go 'either way' - the matrix is symmetric so as long as it goes onto the front it will work in any orientation. Do not solder the matrix over the chip on the back of the backpack - it will not work then!



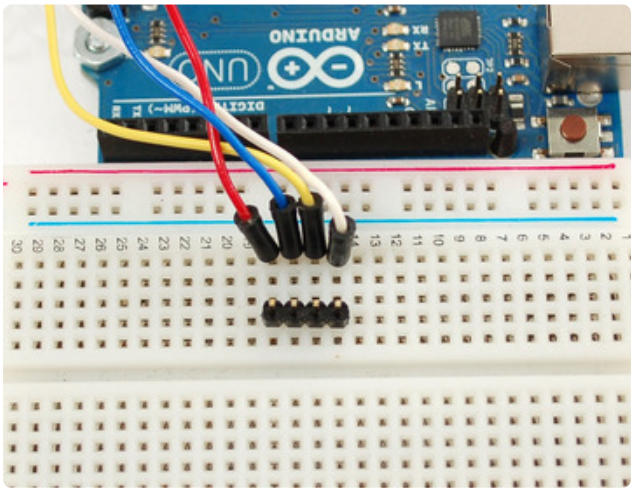
Turn the backpack over so its sitting flat on the matrix.



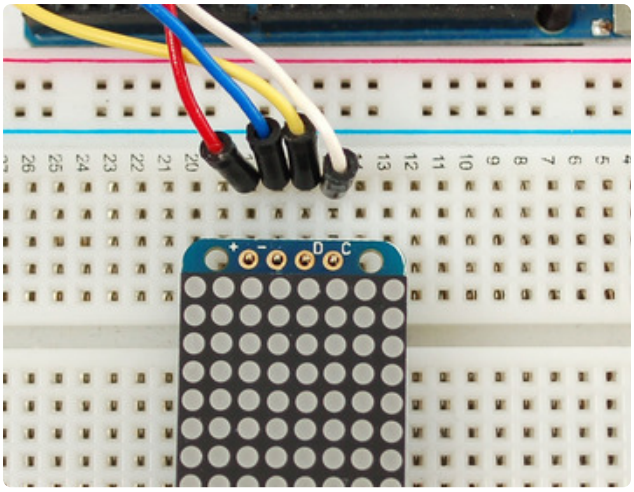
Solder all 16 pins.



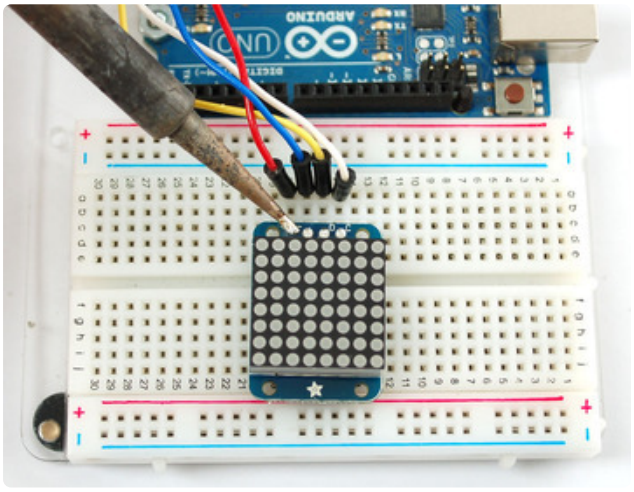
Clip the long pins.



Now you're ready to wire it up to a microcontroller. We'll assume you want to use a 4pin header. You can also of course solder wires directly. Place a 4-pin piece of header with the LONG pins down into the breadboard.



Place the soldered backpack on top of the header.



Solder 'em!

That's it! now you're ready to run the firmware!

Arduino Wiring and Setup

You can use these with a 3.3v or 5v microcontroller. Just connect the VCC+ pin is the same voltage as the logic on your microcontroller.

We wrote a basic library to help you work with the mini 8x8 matrix backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

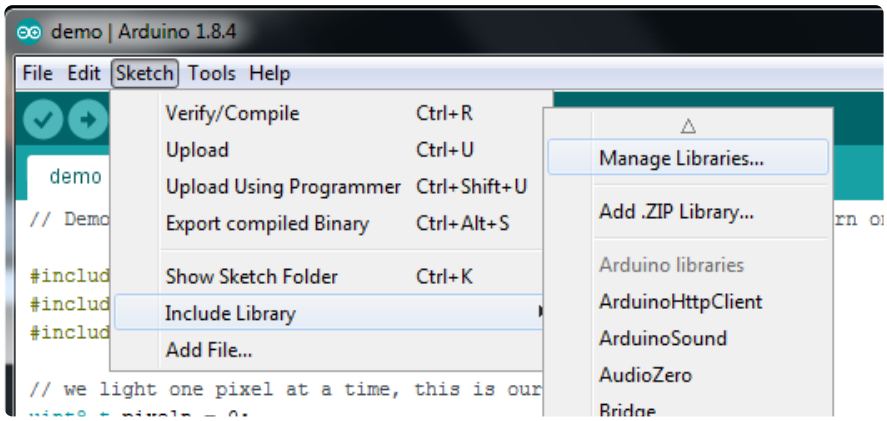
Wiring to the matrix is really easy

- Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo its Digital #3, on the Mega its digital #21

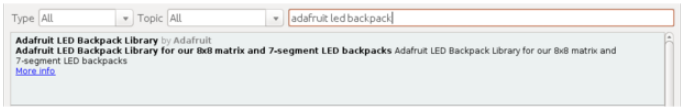
- Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo its Digital #2, on the Mega its digital #20
- Connect **GND** to common ground
- Connect **VCC**+ to power - 5V is best but 3V also seems to work for 3V microcontrollers.

Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

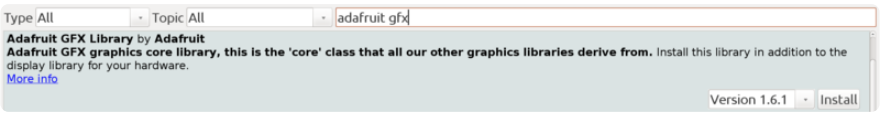
Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it



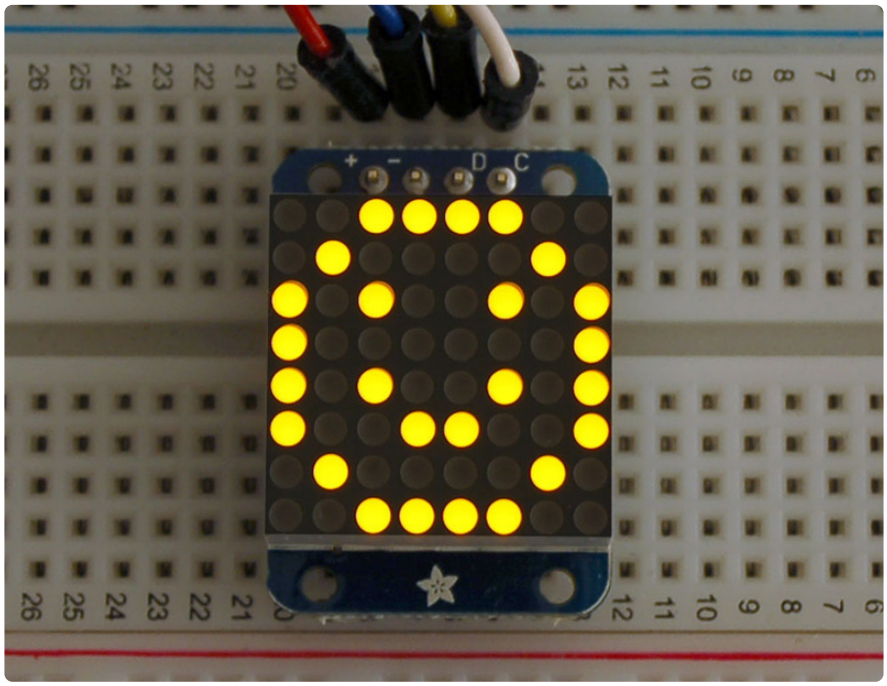
Search for the **Adafruit GFX** library and install it



If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

Once you've restarted you should be able to select the **File→Examples→Adafruit_LEDBackpack→matrix88** example sketch. Upload it to your Arduino as usual. You should see a basic test program that goes through a bunch of different drawing routines

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)



Once you're happy that the matrix works, you can write your own sketches. The 8x8 matrix supports everything the Adafruit GFX library - drawing pixels, lines, rectangles, circles, triangles, roundrects, and small bitmaps. For more details check out the GFX page which will detail all of the GFX routines (https://adafru.it/aPx).

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the matrix:

- **setBrightness(brighness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.

CircuitPython Wiring and Setup

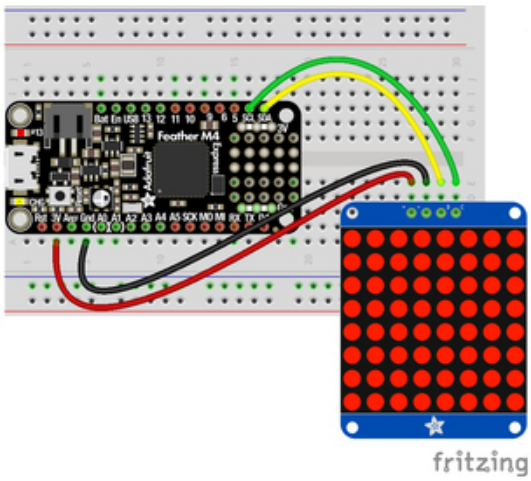
Wiring

It's easy to use LED Matrices with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the LED Matrix to your CircuitPython microcontroller board. First assemble your LED Matrix.

Connect the LED Matrix to your microcontroller board as shown below.



Microcontroller 3V to **LED Matrix VIN**
Microcontroller GND to **LED Matrix GND**
Microcontroller SCL to **LED Matrix SCL**
Microcontroller SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/lfy

Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

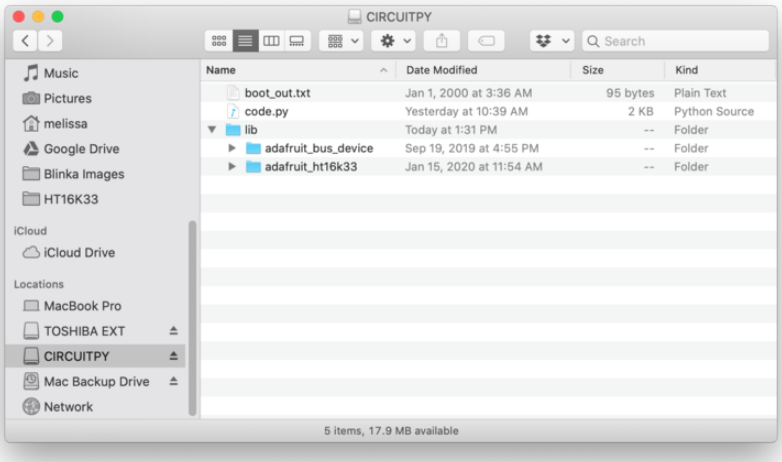
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

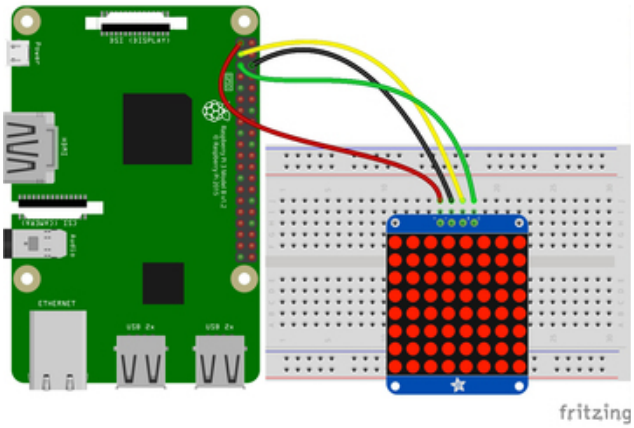
Wiring

It's easy to use LED Matrices with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the LED Matrix to your Raspberry Pi. First assemble your LED Matrix.

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).

Connect the LED Matrix as shown below to your Raspberry Pi.



Raspberry Pi 3.3V to **LED Matrix VIN**
Raspberry Pi GND to **LED Matrix GND**
Raspberry Pi SCL to **LED Matrix SCL**
Raspberry Pi SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/lfz

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `Matrix8x8`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the matrix, you just pass `i2c` in.

When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.matrix import Matrix8x8

i2c = board.I2C()
matrix = Matrix8x8(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
matrix = Matrix8x8(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
matrix.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual pixels. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
matrix.blink_rate = 3
```

Setting Individual Pixels

To set individual pixels to on, you simply treat the `matrix` object as a multidimensional list and set it to 1.

```
matrix[0, 0] = 1
matrix[4, 4] = 1
matrix[7, 7] = 1
```

Filling the Entire Matrix

To fill the entire matrix, just use the fill() function and pass in either 0 or 1 depending on whether you want all pixels off or on. For instance, if you wanted to set everything to on, you would use:

```
matrix.fill(1)
```

Shifting the Matrix

To shift the pixels on the matrix, there are 5 functions you can use. The main function, called shift(), is used to shift the pixels, up, down, left, right, or even diagonally. By passing a positive number, it will shift the pixels right/up and passing a negative number will shift them left/down. For instance:

```
matrix.shift(2, 0)    # shift pixels to the right by 2
matrix.shift(-1, 0)    # shift pixels to the left by 1
matrix.shift(0, -3)    # shift pixels down by 3
matrix.shift(-2, 2)    # shift pixels left by 2 and up by 2
```

You can pass `True` as a third parameter to loop all the pixels that get shifted off over to the other side.

```
matrix.shift(2, 0, True)    # loop pixels to the right by 2
matrix.shift(-1, 0, True)    # loop pixels to the left by 1
matrix.shift(0, -3, True)    # loop pixels down by 3
matrix.shift(-2, 2, True)    # loop pixels left by 2 and up by 2
```

Additionally, there are a few convenience functions that will shift the pixels by one. These can also be passed a value of `True` to loop the pixels.

```
matrix.shift_up()        # Shift pixels up
matrix.shift_left()        # Shift pixels left
matrix.shift_down()        # Shift pixels down
matrix.shift_right()        # Shift pixels right
matrix.shift_up(True)        # Loop pixels up
matrix.shift_left(True)    # Loop pixels left
matrix.shift_down(True)    # Loop pixels down
matrix.shift_right(True)    # Loop pixels right
```

Displaying an Image (Pillow Only)

Additionally, when using with the Raspberry Pi, you can use the Pillow library to display an image to the Matrix. The image will need to be the same exact size as the

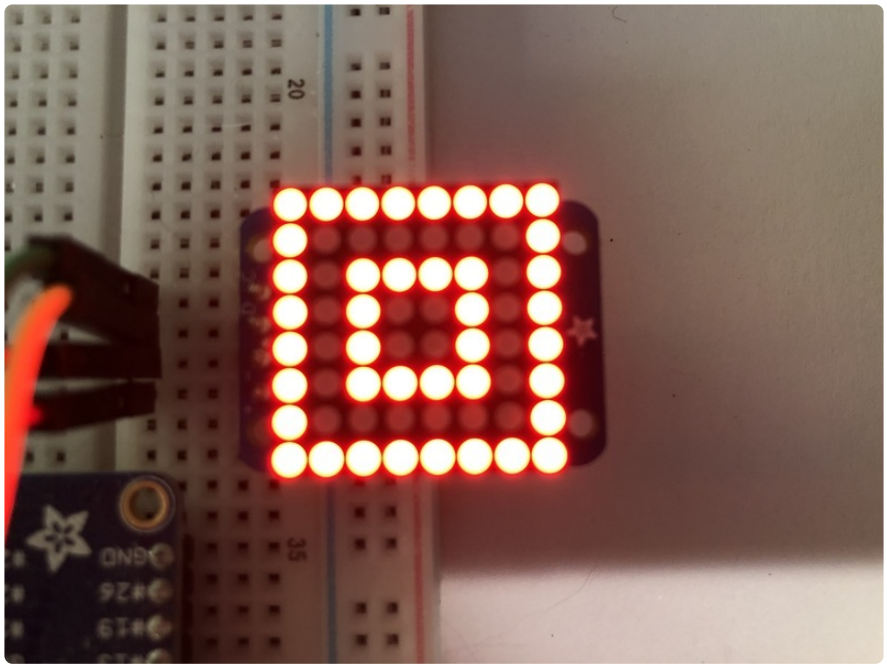
Matrix. In this case, it should be **8x8** pixels. As an example, you can save the image below as **myimage.png**.

Download Image

https://adafru.it/ICR

Then if you want to display the image called **myimage.png**, you would use something like this:

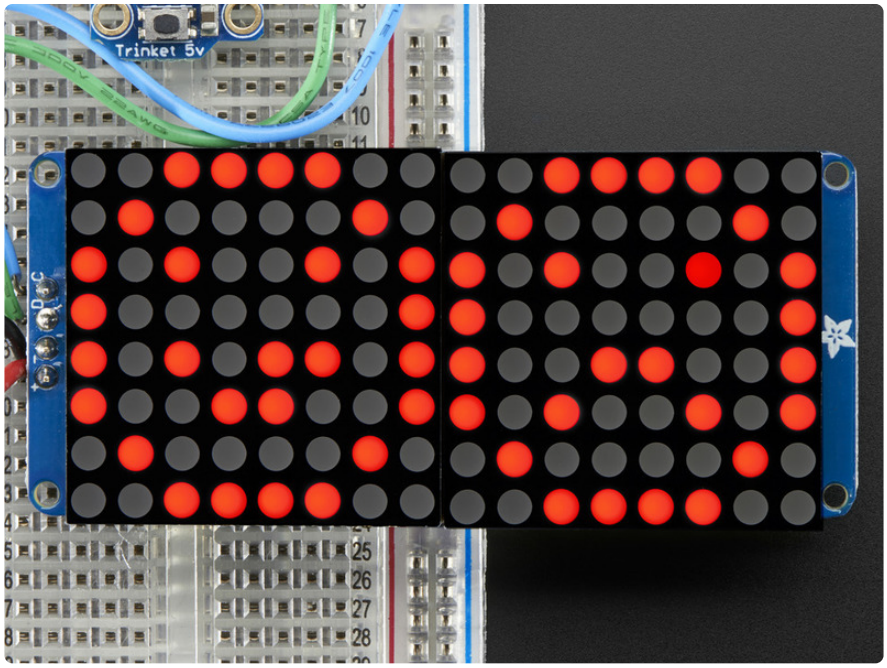
```
import board
from PIL import Image
from adafruit_ht16k33 import matrix

matrix = matrix.Matrix8x8(board.I2C())

image = Image.open("myimage.png")
matrix.image(image)
```



1.2" 16x8 Matrix

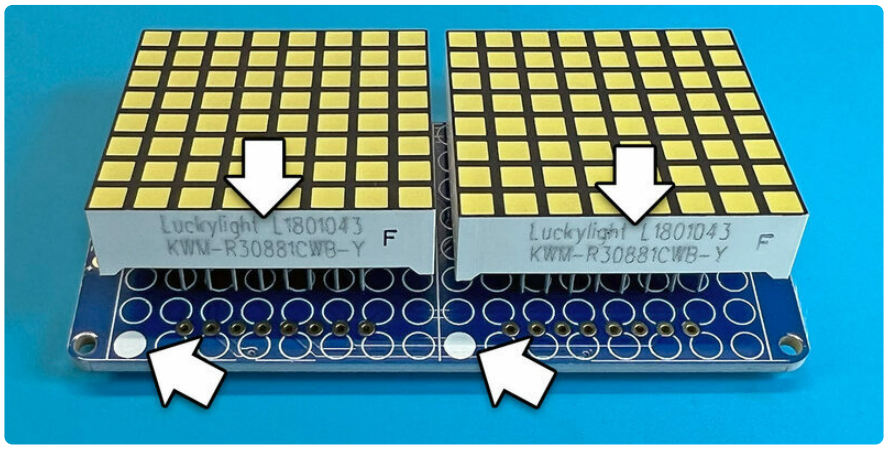
With the 16x8 LED Matrix we've doubled your project's matrix capacity by making it super easy to get two separate 8x8 matrices onto one handy board! This version of the LED backpack is designed for two of the 1.2" 8x8 matrices. They measure only 1.2"x1.2" so its a shame to use a massive array of chips to control it. This backpack solves the annoyance of using 32 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all 128 LEDs in the background. All you have to do is write

data to it using the 2-pin I2C interface. There are two address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.



Assembling the 1.2" 16x8 backpack is nearly the same as the 1.2" 8x8, so you can [follow that page](https://adafru.it/LAT) for directions, only difference is there's two matrices to install now.

The printed-on edge of the matrices face the white dots on the PCB, and there's a single set of address selection pads; the pair is addressed as one larger unit, not set independently.



Arduino Setup

You can use these with a 3.3v or 5v microcontroller. Just connect the VCC+ pin is the same voltage as the logic on your microcontroller.

16x8 Matrix Software

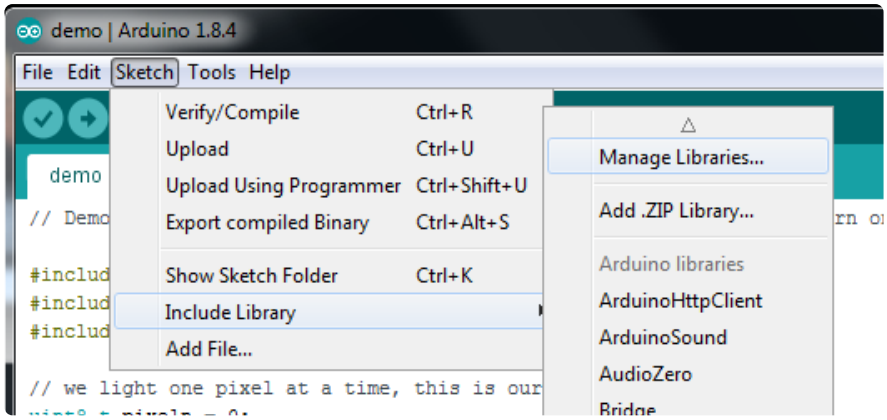
We wrote a basic library to help you work with the 16x8 matrix backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

Wiring to the matrix is really easy

- Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo its Digital #3, on the Mega its digital #21
- Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo its Digital #2, on the Mega its digital #20
- Connect **GND** to common ground
- Connect **VCC+** to power - 5V is best but 3V also seems to work for 3V microcontrollers.

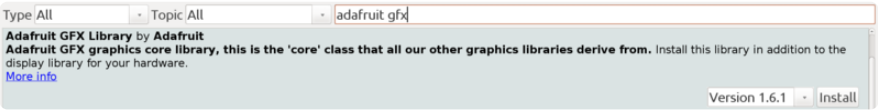
Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it

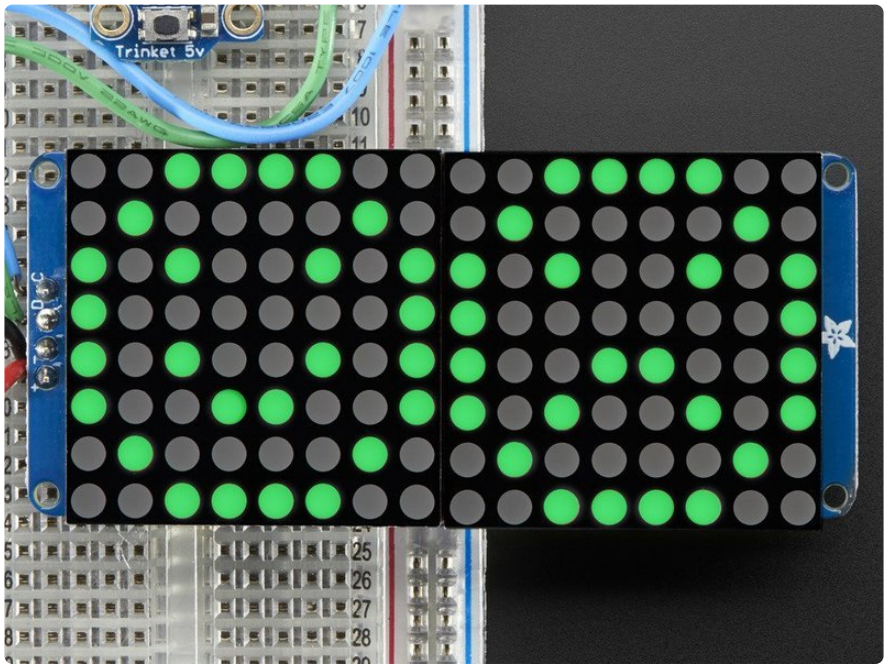
Search for the **Adafruit GFX** library and install it



If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

You should now be able to select the **File→Examples→Adafruit_LEDBackpack→matrix88** example sketch. Upload it to your Arduino as usual. You should see a basic test program that goes through a bunch of different drawing routine

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)

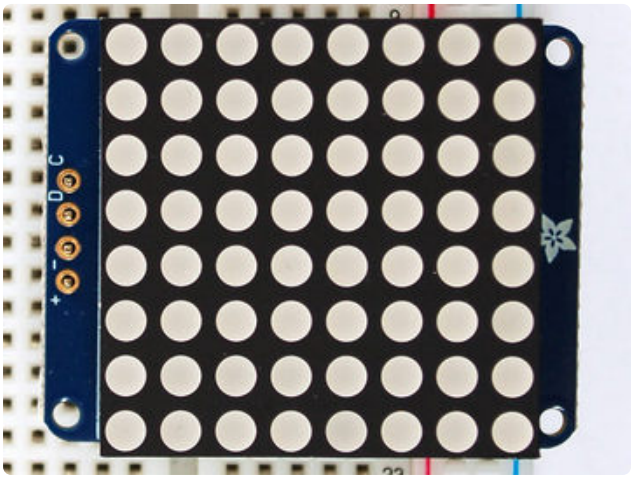


Once you're happy that the matrix works, you can write your own sketches. The 8x8 matrix supports everything the Adafruit GFX library - drawing pixels, lines, rectangles, circles, triangles, roundrects, and small bitmaps. For more details check out the GFX page which will detail all of the GFX routines (https://adafru.it/aPx).

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the matrix:

- **setBrightness(brighness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start. You can call this function at any time to change the brightness of the -entire- display
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.You can call this function at any time to change the blink rate of the -entire- display



The default orientation for graphics commands on this display places pixel (0,0) at the top-left when the header is at the left and Adafruit logo at the right. To use the matrix as shown above (header at top, logo at bottom), call matrix.setRotation(3) before issuing graphics commands.

CircuitPython Wiring and Setup

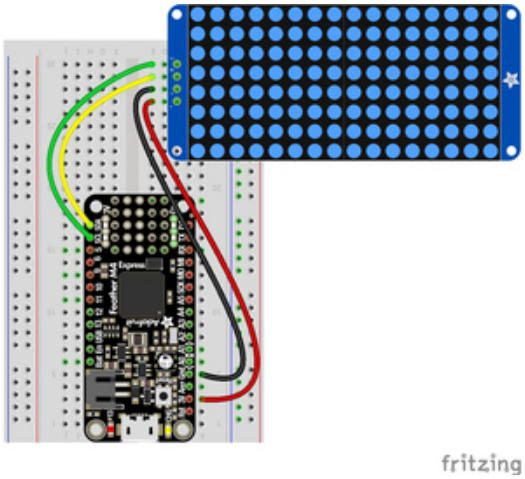
Wiring

It's easy to use LED Matrices with CircuitPython and the [Adafruit CircuitPython HT16K33 (https://adafru.it/u1E)](https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the LED Matrix to your CircuitPython microcontroller board. First assemble your LED Matrix.

Connect the LED Matrix to your microcontroller board as shown below.



Microcontroller 3V to **LED Matrix VIN**
Microcontroller GND to **LED Matrix GND**
Microcontroller SCL to **LED Matrix SCL**
Microcontroller SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/IB5

Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

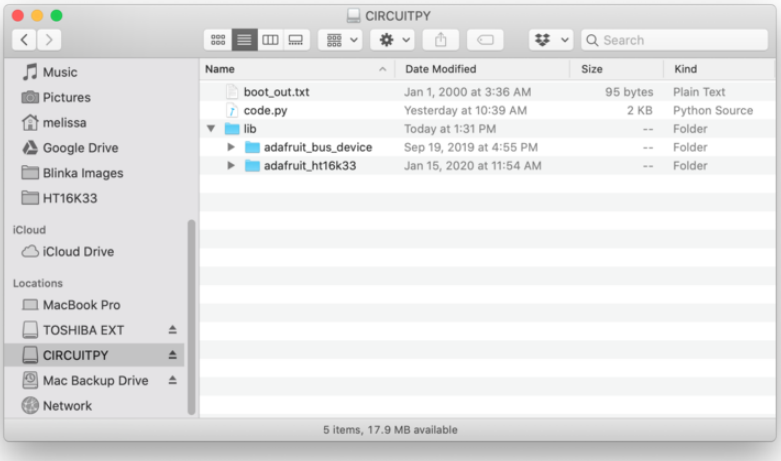
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

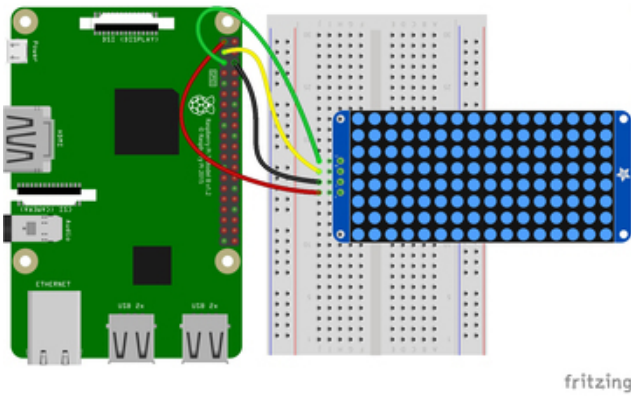
Wiring

It's easy to use LED Matrices with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the LED Matrix to your Raspberry Pi. First assemble your LED Matrix.

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).

Connect the LED Matrix as shown below to your Raspberry Pi.



Raspberry Pi 3.3V to **LED Matrix VIN**
Raspberry Pi GND to **LED Matrix GND**
Raspberry Pi SCL to **LED Matrix SCL**
Raspberry Pi SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/IB6

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `MatrixBackpack16x8`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the matrix, you just pass `i2c` in.

When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.matrix import MatrixBackpack16x8

i2c = board.I2C()
matrix = MatrixBackpack16x8(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
matrix = MatrixBackpack16x8(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual pixels. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Setting Individual Pixels

To set individual pixels to on, you simply treat the `matrix` object as a multidimensional list and set it to 1.

```
matrix[0, 0] = 1
matrix[4, 4] = 1
matrix[7, 7] = 1
```

Filling the Entire Matrix

To fill the entire matrix, just use the fill() function and pass in either 0 or 1 depending on whether you want all pixels off or on. For instance, if you wanted to set everything to on, you would use:

```
matrix.fill(1)
```

Shifting the Matrix

To shift the pixels on the matrix, there are 5 functions you can use. The main function, called shift(), is used to shift the pixels, up, down, left, right, or even diagonally. By passing a positive number, it will shift the pixels right/up and passing a negative number will shift them left/down. For instance:

```
matrix.shift(2, 0)    # shift pixels to the right by 2
matrix.shift(-1, 0)    # shift pixels to the left by 1
matrix.shift(0, -3)    # shift pixels down by 3
matrix.shift(-2, 2)    # shift pixels left by 2 and up by 2
```

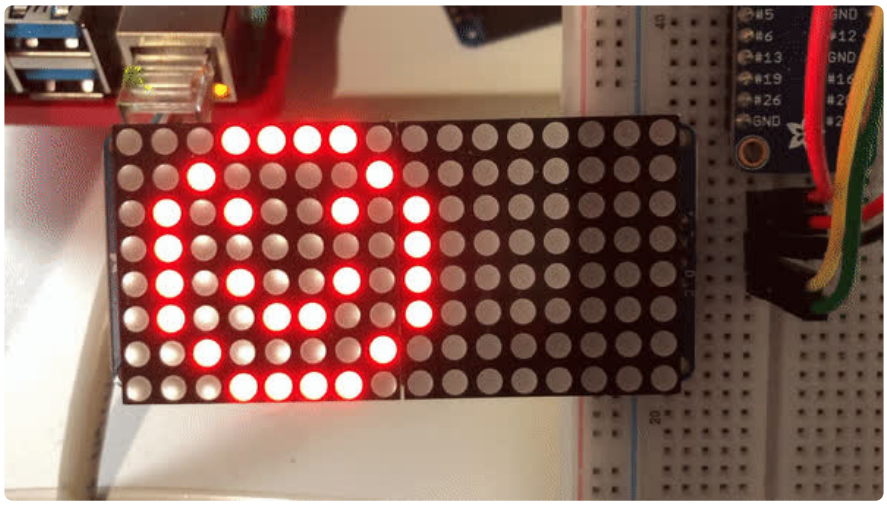
You can pass `True` as a third parameter to loop all the pixels that get shifted off over to the other side.

```
matrix.shift(2, 0, True)    # loop pixels to the right by 2
matrix.shift(-1, 0, True)    # loop pixels to the left by 1
matrix.shift(0, -3, True)    # loop pixels down by 3
matrix.shift(-2, 2, True)    # loop pixels left by 2 and up by 2
```

Additionally, there are a few convenience functions that will shift the pixels by one. These can also be passed a value of `True` to loop the pixels.

```
matrix.shift_up()         # Shift pixels up
matrix.shift_left()         # Shift pixels left
matrix.shift_down()         # Shift pixels down
matrix.shift_right()         # Shift pixels right
matrix.shift_up(True)         # Loop pixels up
matrix.shift_left(True)     # Loop pixels left
matrix.shift_down(True)     # Loop pixels down
matrix.shift_right(True)     # Loop pixels right
```

Here's what shifting a smiley face to the right looks like:



Displaying an Image (Pillow Only)

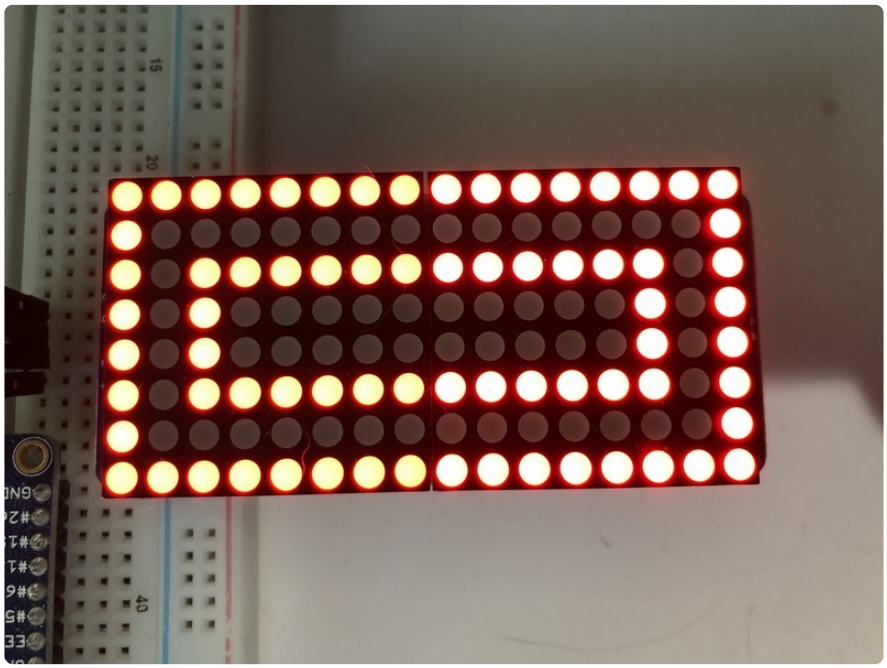
Additionally, when using with the Raspberry Pi, you can use the Pillow library to display an image to the Matrix. The image will need to be the same exact size as the Matrix. In this case, it should be **16x8** pixels. As an example, you can save the image below as **myimage.png**.

Download Image

https://adafru.it/ICS

Then if you want to display the image called **myimage.png**, you would use something like this:

```
import board
from PIL import Image
from adafruit_ht16k33 import matrix

matrix = matrix.MatrixBackpack16x8(board.I2C())

image = Image.open("myimage.png")
matrix.image(image)
```



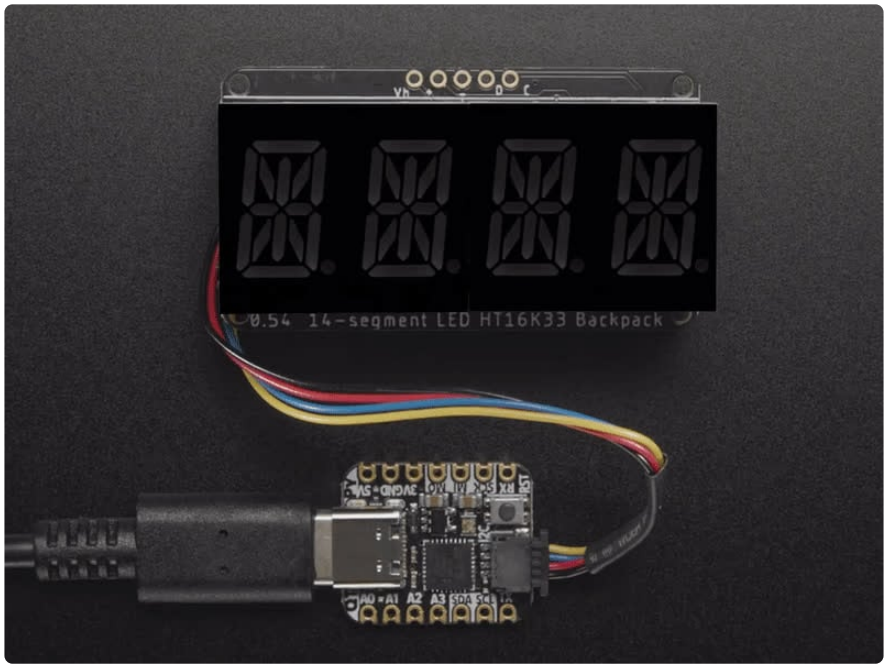
0.54" Alphanumeric Backpack

This version of the LED backpack is designed for two dual 14-segment "Alphanumeric" displays. These 14-segment displays normally require 18 pins (4 'characters' and 14 total segments each) This backpack solves the annoyance of using 18 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all the LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface.

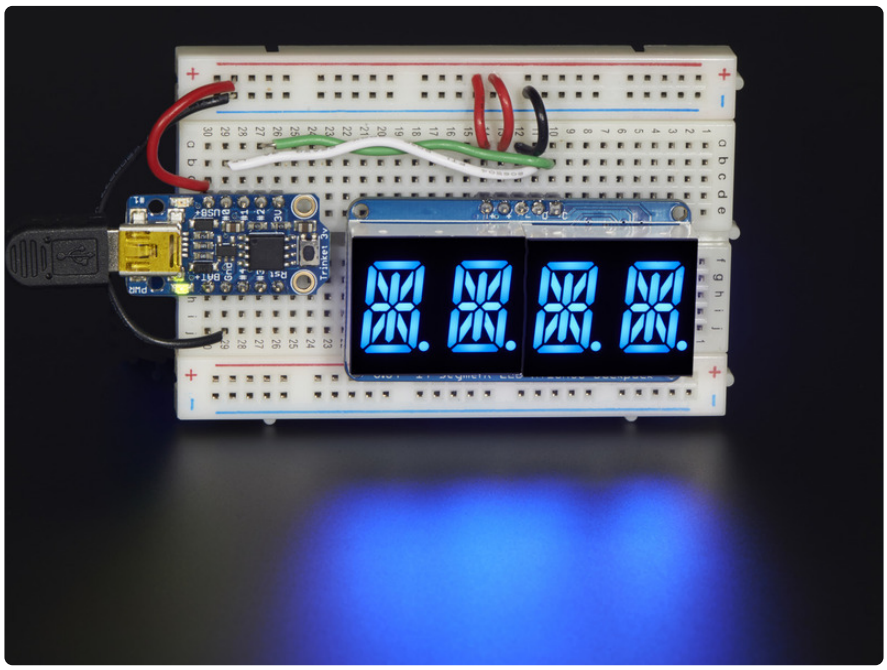
There are three address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.

To get you going fast, we have revised this popular board to be the same size and pinout as before but now with two STEMMA QT connectors (https://adafru.it/JqB) on

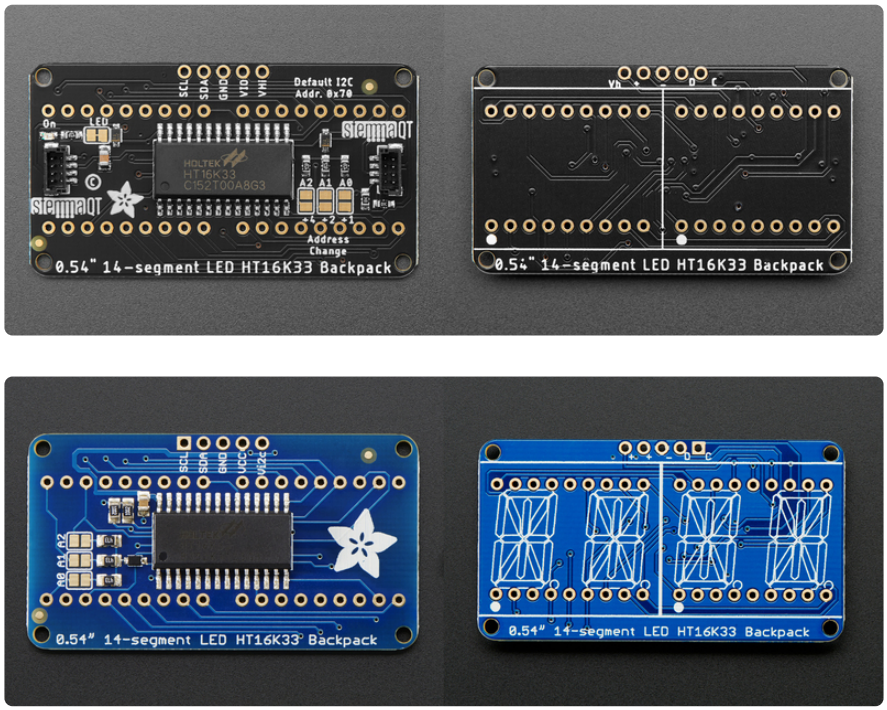
either side that are compatible with the SparkFun Qwiic (https://adafru.it/Fpw) I2C connectors. This allows you to make solderless connections between your development board and the HT16K33 or to chain it with a wide range of other sensors and accessories using a **compatible cable** (https://adafru.it/JnB).



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



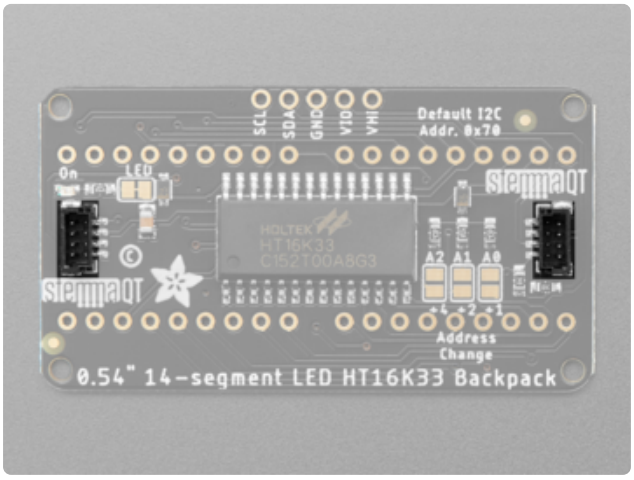
There are a number of features on the 0.54" Alphanumeric Backpack.

STEMMA QT Revision-Only Features

These features are only available on the STEMMA QT revision.

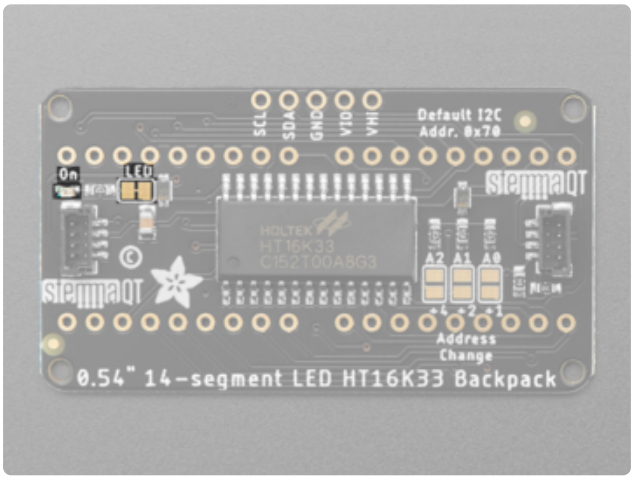
STEMMA QT Connectors

The default I2C address is **0x70**.



The **STEMMA QT connectors** (https://adafru.it/Ft4) provide a solder-free way to connect this backpack to development boards with STEMMA QT connectors, or to other things, with various associated accessories (https://adafru.it/Qgf).

On LED and LED Jumper



On LED - On the left side of the back of the board is a little green LED labeled **On**. This LED lights up when the board is successfully powered.
LED jumper - To the right of the On LED is a jumper labeled **LED**. If you wish to disable the On LED, you can cut the trace between the two pads. To enable it again, use solder to reconnect the two pads.

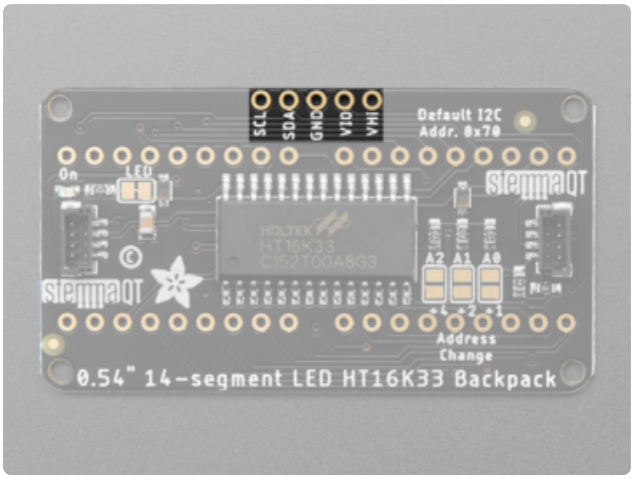
Original and STEMMA QT Version Features

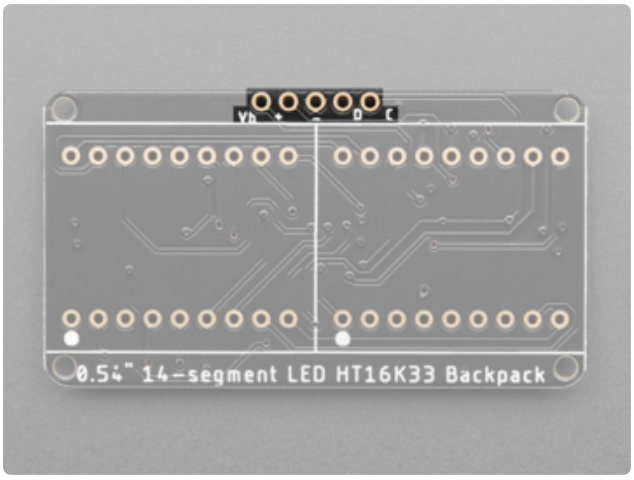
These features are available on both versions. There is one header pin difference between the two, which is explained in the next section. Everything else is the same.

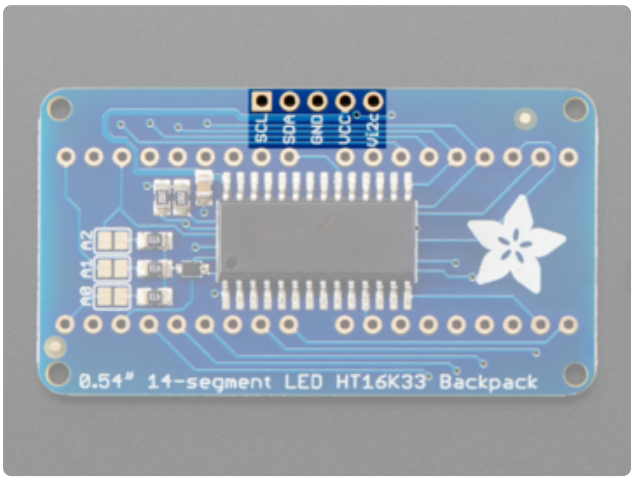
Header Pin Through-Hole Pads

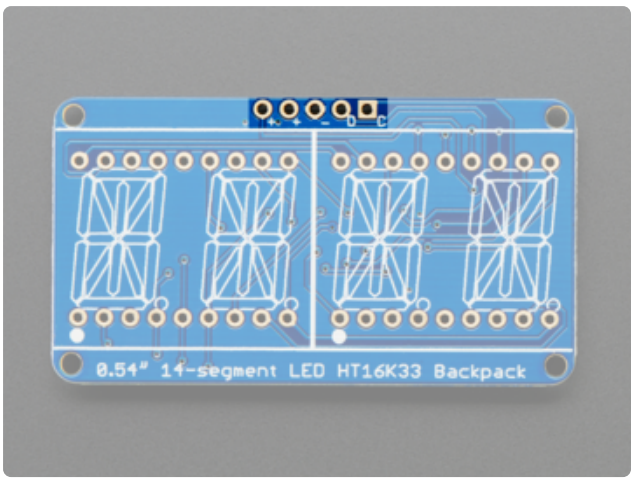
If you prefer to use a breadboard, there are through-hole header pin pads along the top of the board in the middle.

The default I2C address is **0x70**.









On both versions:

VIO/VCC - This is power for the backpack. It can be 3V-5V. To power the backpack, give it the same power as the logic level of your microcontroller - e.g. for a 5V microcontroller like Arduino, use 5V.
GND - This is common ground for power and logic.
SCL - This is the I2C clock pin. Connect it to your microcontroller I2C clock line. This pin is level shifted so you can use 3-5V logic, and there's a **10K pullup** on this pin.
SDA - This is the I2C data pin. Connect it to your microcontroller I2C data line. This pin is level shifted so you can use 3-5V logic, and there's a **10K pullup** on this pin.

Wiring VHi to 3v on the Stemma QT version will result in the display not activating.

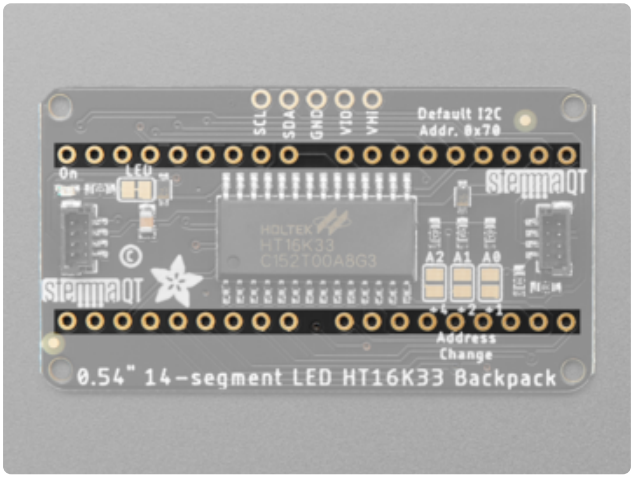
On the STEMMA QT revision ONLY:

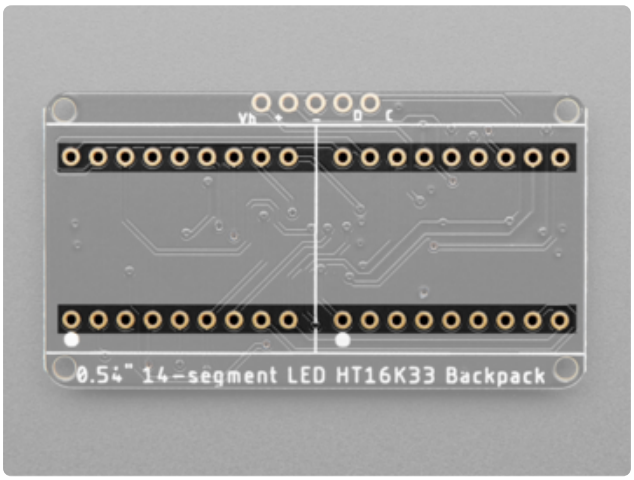
- **VHi** - This pin allows you to provide 5V to only the 14-segment displays when using a 3V device to control the backpack. If you're using a 3V device and you want your displays to be brighter, you can maintain the 3V I2C power level, and connect 5V to the VHi pin to make the 14-segment displays have a brighter look.

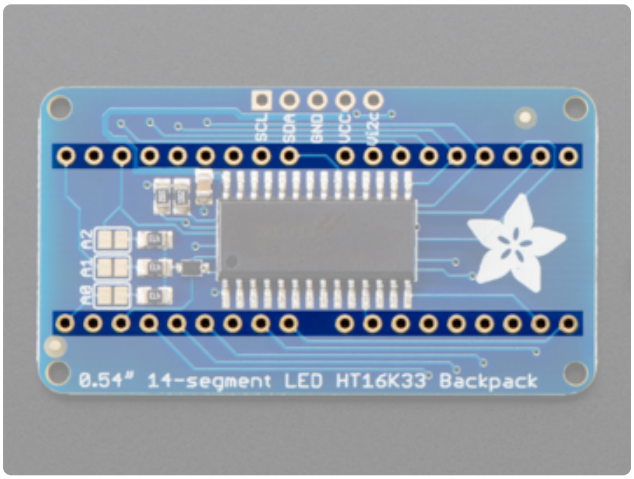
On the original version ONLY:

- **Vi2c** - This is the I2C voltage, which sets the logic level to I2C. Connect this pin to the voltage pin on your device that matches the device's logic level. For example, if you're using a 3.3V microcontroller, connect it to 3.3V.

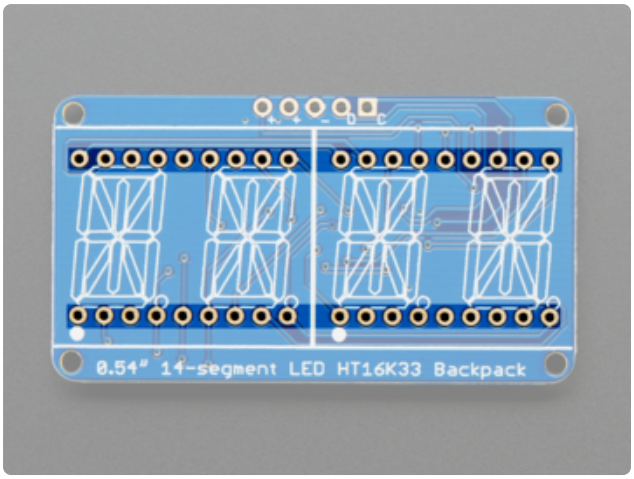
Display Pin Through-Hole Pads



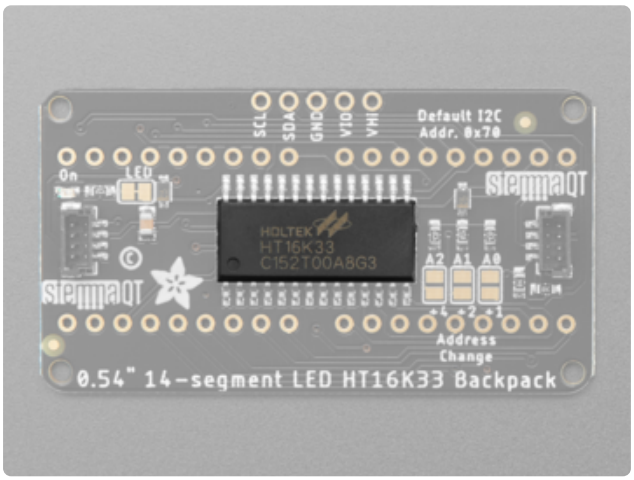


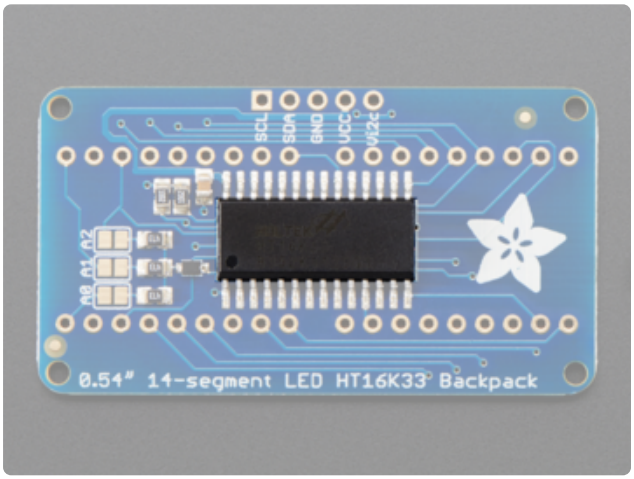


These two rows of through-hole pads are for soldering the alphanumeric LED displays onto the backpack. See the Assembly page (https://adafru.it/11dU) for details on attaching the displays properly.



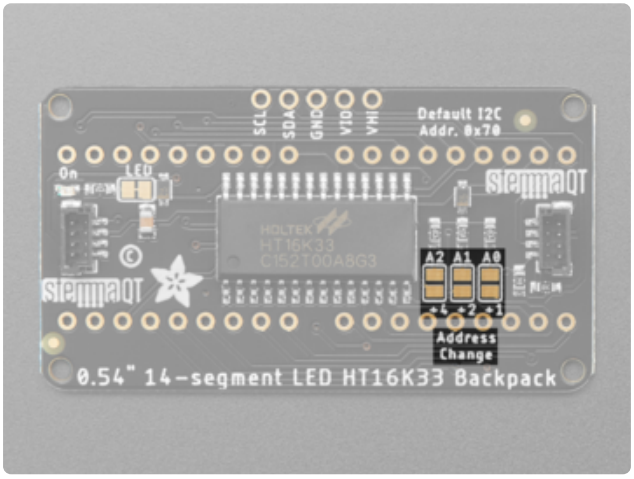
HT16K33 Matrix Driver

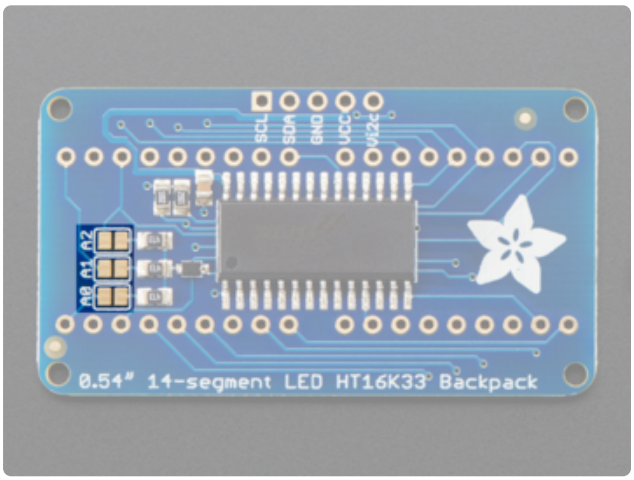




The chip located in the center of the back of the backpack is the HT16K33 matrix driver, which drives the 14-segment LED displays.

Address Jumper Pins





On the back of the board are **three address jumpers**, labeled **A0**, **A1**, and **A2**. These jumpers allow you to chain up to 8 of these boards on the same pair of I2C clock and data pins. To do so, you solder the jumpers "closed" in various combinations by connecting the two pads.

The default I2C address is **0x70**. The other address options can be calculated by “adding” the **A0/A1/A2** to the base of **0x70**.

A0 sets the lowest bit with a value of **1**, **A1** sets the next bit with a value of **2** and **A2** sets the next bit with a value of **4.** The final address is **0x70** + **A2** + **A1** + **A0** which would be **0x77**.

So for example if **A2** is soldered closed and **A0** is soldered closed, the address is **0x70** + **4** + **1** = **0x75**.

If only **A0** is soldered closed, the address is **0x70** + **1** = **0x71**

If only **A1** is soldered closed, the address is **0x70** + **2** = **0x72**

If only **A2** is soldered closed, the address is **0x70** + **4** = **0x74**

The table below shows all possible addresses, and whether the pin(s) should be high (closed) or low (open).

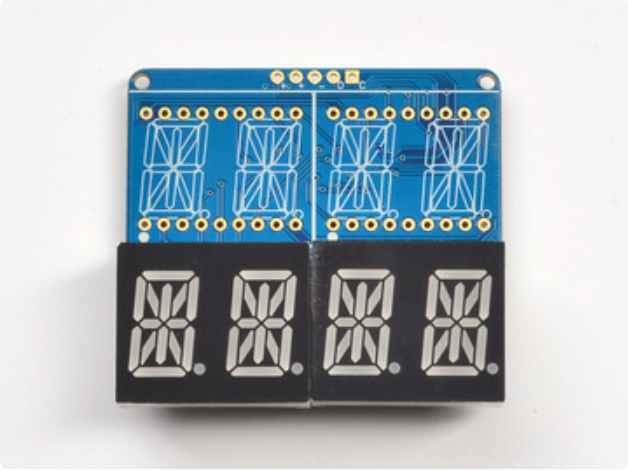
ADDR	A0	A1	A2	ADDR	A0	A1	A2
0x70	L	L	L	0x74	L	L	H
0x71	H	L	L	0x75	H	L	H
0x72	L	H	L	0x76	L	H	H
0x73	H	H	L	0x77	H	H	H

Assembly

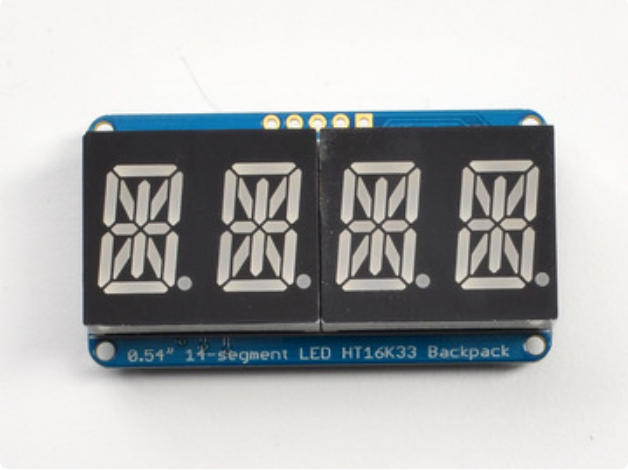
Attaching the Backpack

The assembly photos below are the original version. Assembly is the same for both the original version and the STEMMA QT version.

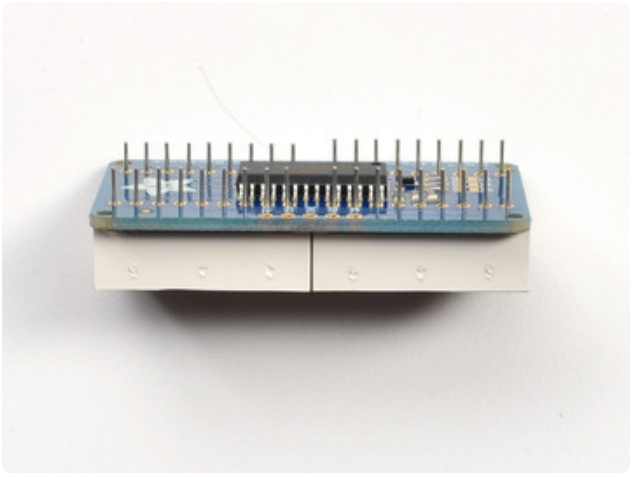
> This assembly is the same on the STEMMA QT version of the backpack!

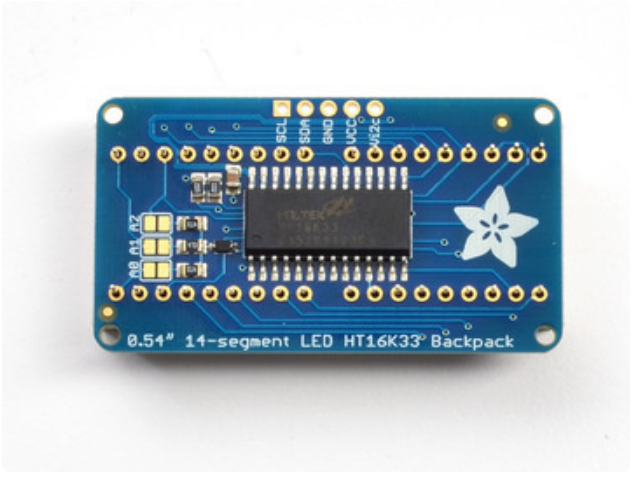


When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as two dual 14-segment display in one of the colors we provide (say, red, yellow, blue or green). You'll need to solder the matrix onto the backpack but it's an easy task.

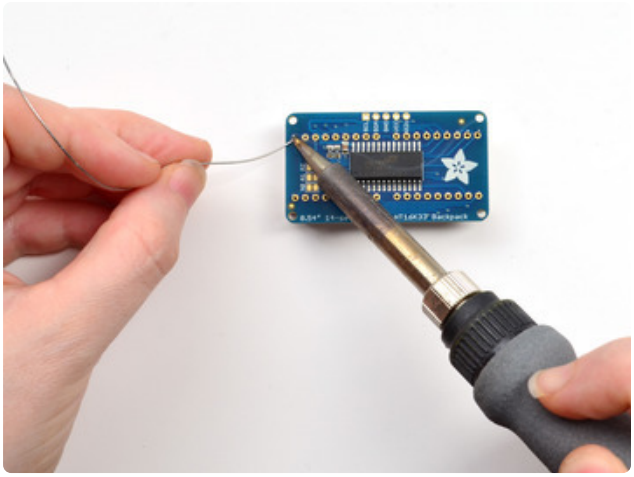


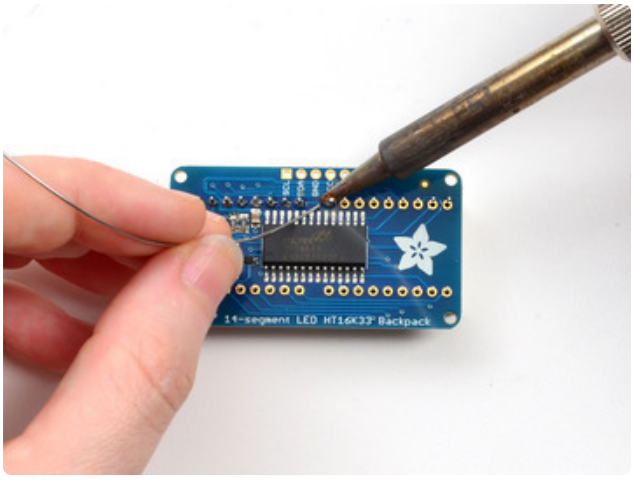
Remove the parts from packaging and place the LED matrices OVER the silkscreen side. **DO NOT PUT THE DISPLAY ON UPSIDE DOWN OR IT WONT WORK!! Check the image below to make sure the 'decimal point' dots are on the bottom, matching the silkscreen.**



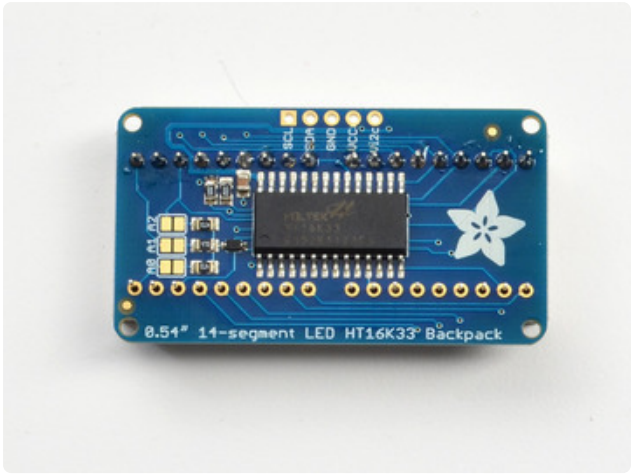


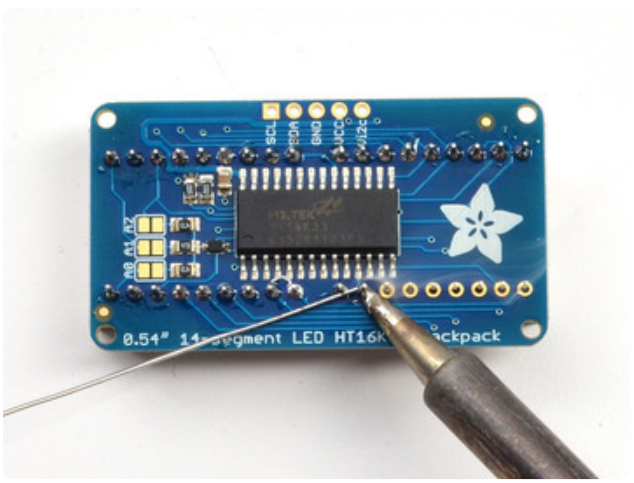
Turn the backpack over so it is sitting flat on the matrix.

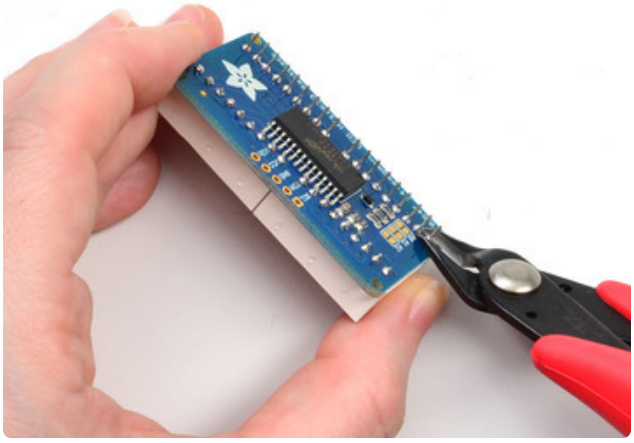




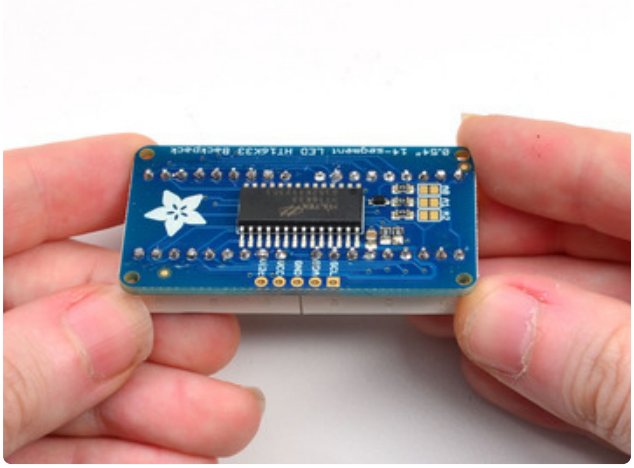
Solder all of the pins!





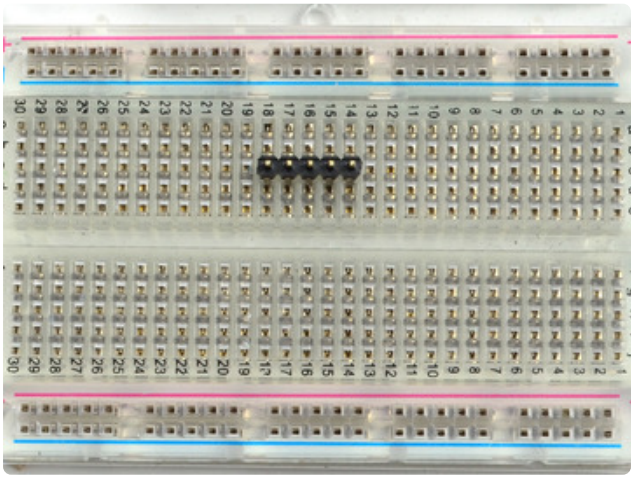


Clip the long pins.



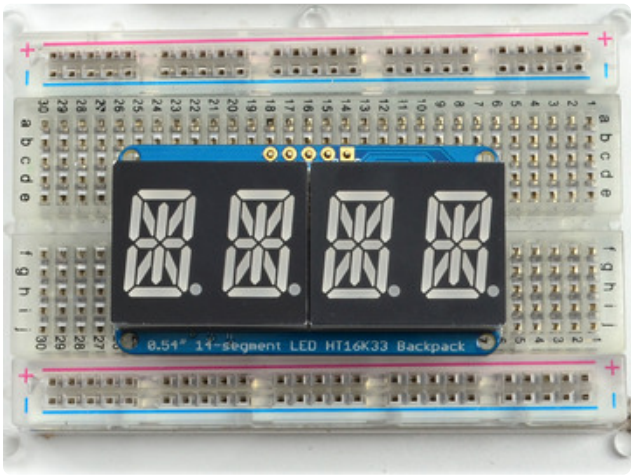
Check your work, making sure each pin is nicely soldered, and there's no cold solder joints or shorted pins

Attaching Header



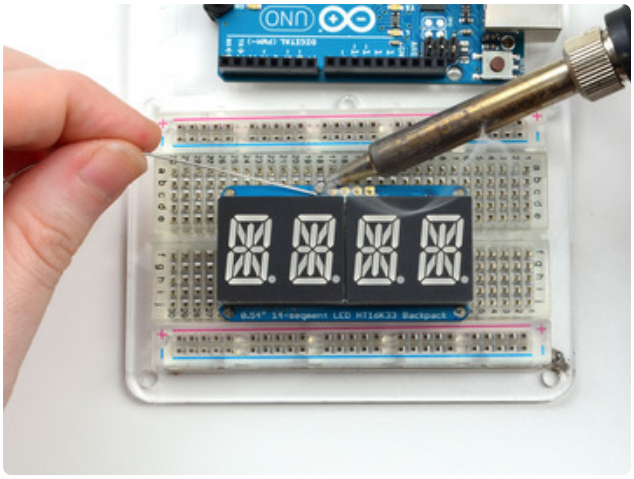
Prepare the header strip:

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 Backpack:

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



Solder all 5 pins!

That's it! now you're ready to run the firmware on your Arduino!

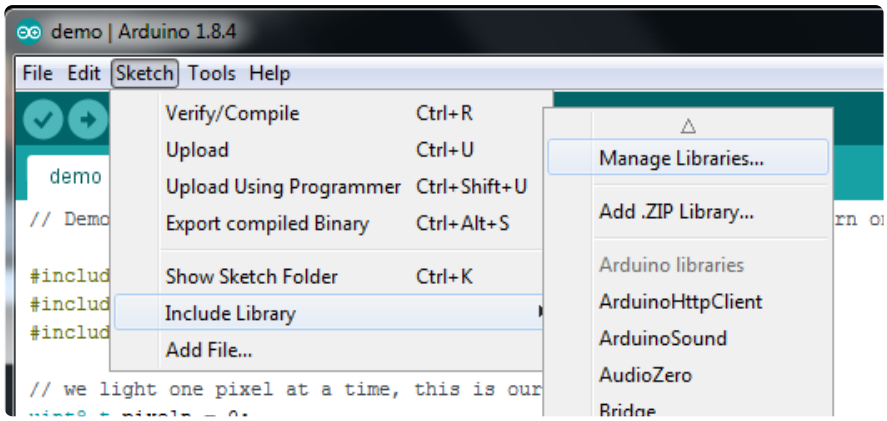
Arduino Wiring and Setup

Downloading the Arduino Library

We wrote a basic library to help you work with the alphanumeric backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

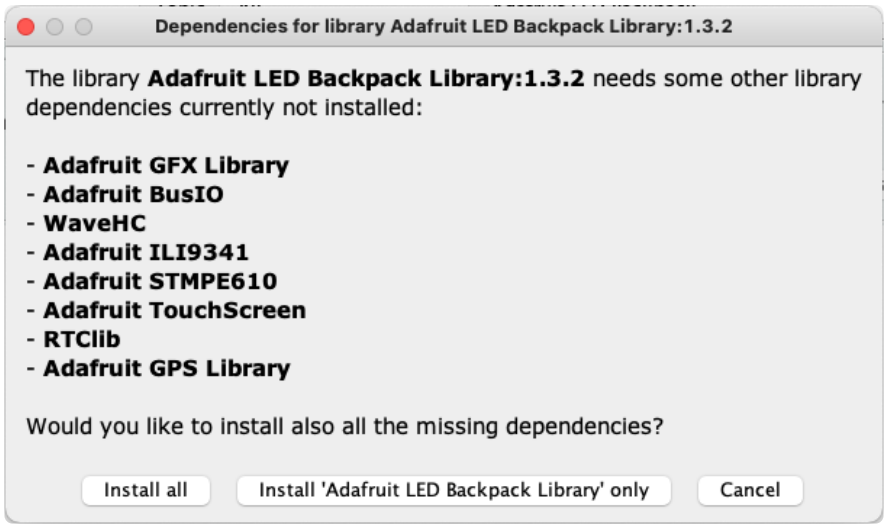
Begin by downloading our Adafruit LED Backpack library (https://adafru.it/aLI)and the Adafruit GFX library (https://adafru.it/aJa) from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it

When asked to install dependencies, click **Install all.**

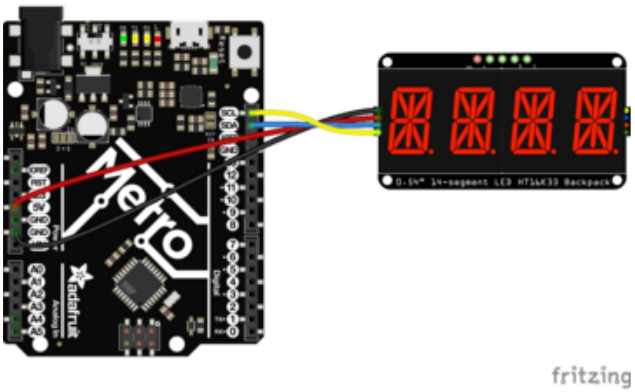


We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

You can use these with a 3.3v or 5v microcontroller. Just make sure the Vi2c pin is the same voltage as the logic on your microcontroller.

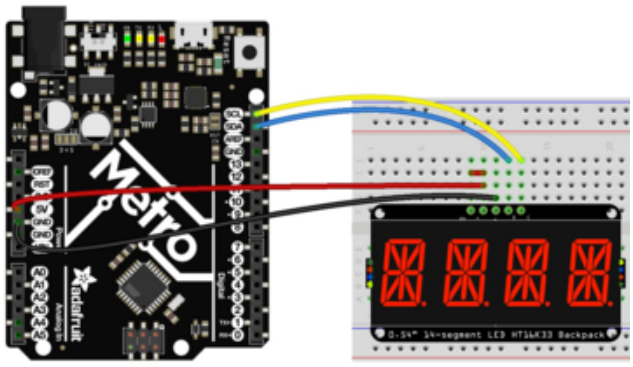
Wiring STEMMA QT Version

Here is an example of the STEMMA QT version wired to a Metro using the STEMMA QT connector on the backpack.



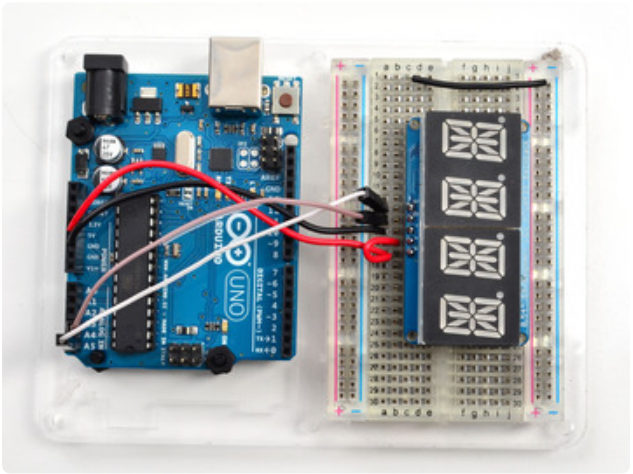
Board 5V to **backpack Vio (red wire)**
Board GND to **backpack GND (black wire)**
Board SCL to **backpack SCL (yellow wire)**
Board SDA to **backpack SDA (blue wire)**

Here is an example of the STEMMA QT version wired to a Metro using a solderless breadboard. This example also includes how to wire up the VHi pin, which makes the LEDs appear brighter.



Board 5V to **backpack Vio (long red wire)**
Board GND to **backpack GND (black wire)**
Board SCL to **backpack SCL (yellow wire)**
Board SDA to **backpack SDA (blue wire)**
Backpack VIO to **backpack VHi (short red wire)**

Wiring Original Version

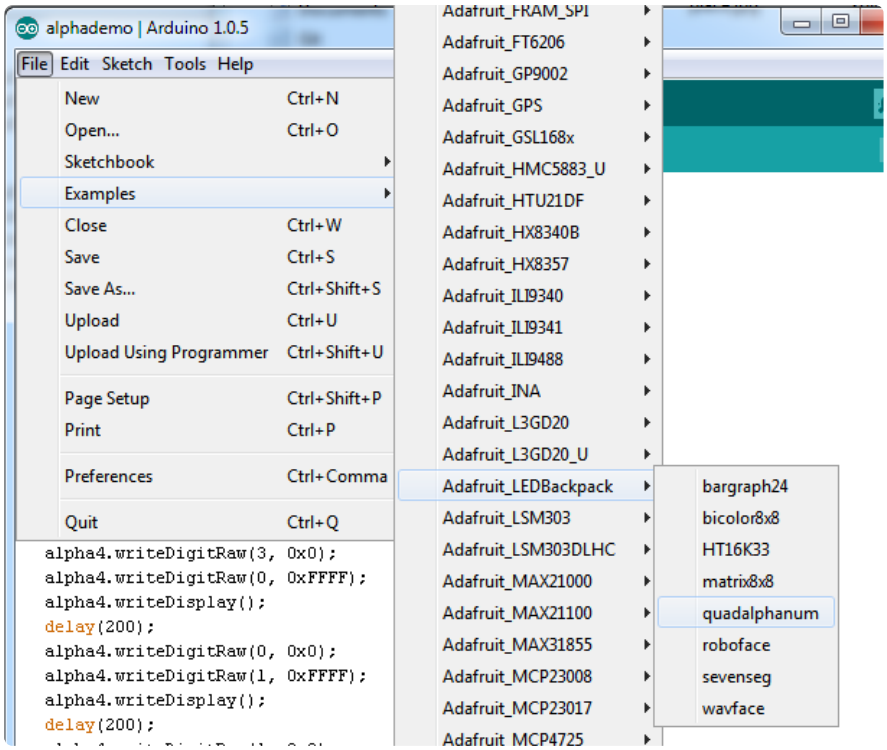


Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo it's Digital #3, on the Mega it's digital #21
Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo it's Digital #2, on the Mega it's digital #20
Connect **GND** to common ground
Connect **VCC+** to power - 5V is best but 3V will work if that's all you've got (it will be dimmer)
Connect **Vi2c** to your microcontroller's logic level (3-5V) - If you're using an Arduino, this is almost certainly 5V. If its a 3V Arduino such as a Due, connect it to 3V

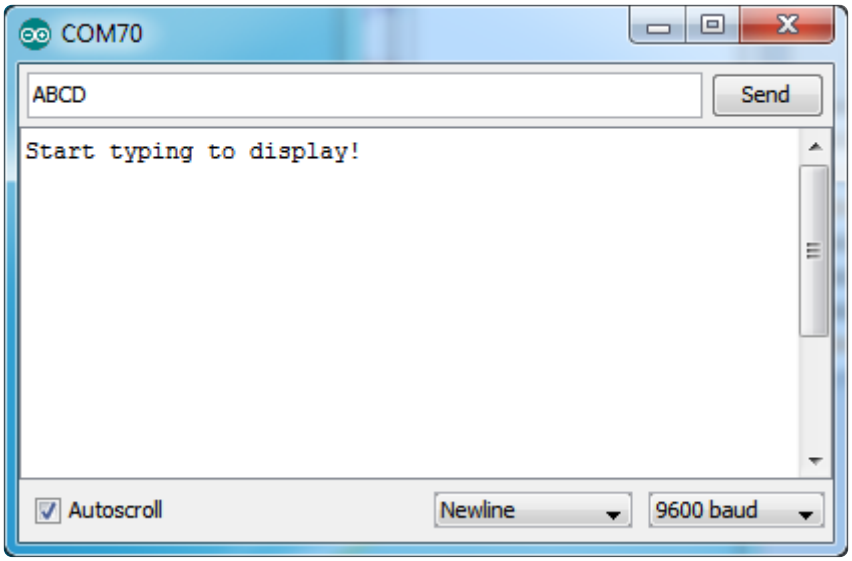
Both **Vi2c** and **Vcc** MUST be connected to 3 to 5VDC! Vcc is for the LED driver power, Vi2c is what sets the logic level for communication to the chip.

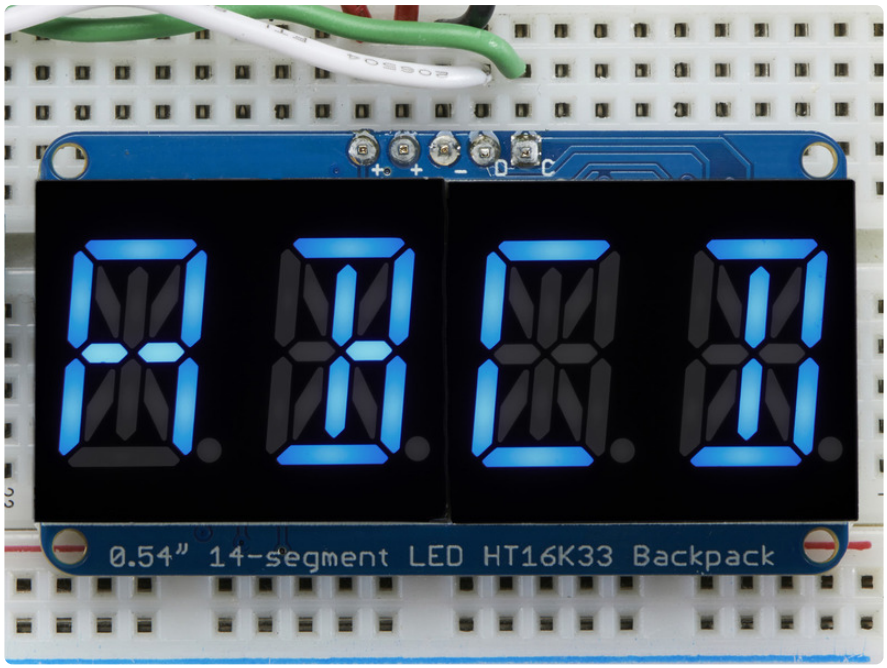
Load Demo

Restart the Arduino IDE and load up the **File→Adafruit_LEDBackpack→quadalphanum** demo



Upload to your Arduino, and open up the Serial console at 9600 baud speed. You'll see each digit light up all the segments, then the display will scroll through the 'font table' showing every character that it knows how to display. Finally, you'll get a notice to start typing into the serial console. Type a message and hit return, you'll see it scroll onto the display!





Library Reference

For the quad displays, we have a special object that can handle ascii data for easy printing.

You can create the object with

```
Adafruit_AlphaNum4 alpha4 = Adafruit_AlphaNum4();
```

There's no arguments or pins because the backpacks use the fixed I2C pins.
By default, the address is 0x70, but you can pass in the I2C address used when you initialize the display with **begin**

```
alpha4.begin(0x70);  // pass in the address
```

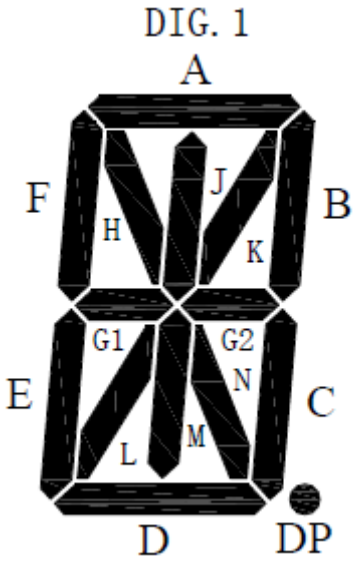
Next up, the segments can be turned on/off for each digit by writing the 'raw' bitmap you want, for example, all the LEDs off on digit #3 is

```
alpha4.writeDigitRaw(3, 0x0);
```

All the segments on for digit #0 is

```
alpha4.writeDigitRaw(0, 0x3FFF);
```

This is the segment map:



the 16 bit digit you pass in for raw image has this mapping:

0 **DP N M L K J H G2 G1 F E D C B A**

The first bit isn't used, you can make it 0 or 1

To turn on just the **A** segment, use 0x0001
To turn on just the **G1** segment, use 0x0040

ASCII data

If you're just looking to print 'text' you can use our font table, just pass in an ASCII character!

For example, to set digit #0 to **A** call:

```
alpha4.writeDigitAscii(0, 'A')
```

Writing Data

Don't forget to 'write' the data to the display with

```
alpha4.writeDisplay();
```

That's what actually 'sets' the data onto the LEDs!

CircuitPython Wiring and Setup

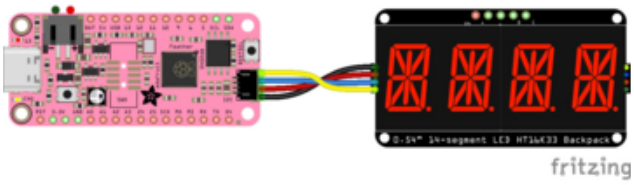
It's easy to use LED AlphaNumeric Displays with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this backpack with any CircuitPython microcontroller board.

First assemble your AlphaNumeric Display (https://adafru.it/11dU).

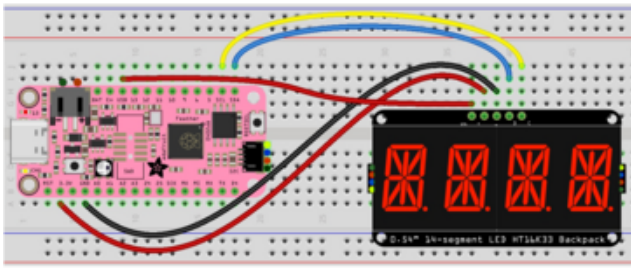
Wiring STEMMA QT Version

Here is an example of the STEMMA QT version wired to a Feather RP2040 using the STEMMA QT connector on the backpack.



Board 3.3V to **backpack Vio (red wire)**
Board GND to **backpack GND (black wire)**
Board SCL to **backpack SCL (yellow wire)**
Board SDA to **backpack SDA (blue wire)**

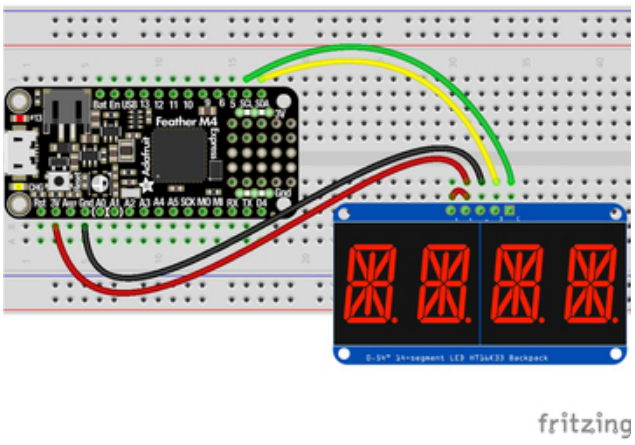
Here is an example of the STEMMA QT version wired to a Feather RP2040 using a solderless breadboard. This example also includes how to wire up the VHi pin, which makes the LEDs appear brighter.



Board 5V to **backpack Vio (red wire connected along the bottom of the Feather)**
Board GND to **backpack GND (black wire)**
Board SCL to **backpack SCL (yellow wire)**
Board SDA to **backpack SDA (blue wire)**
Board USB to **backpack VHi (red wire connected along the top of the Feather)**

Wiring Original Version

Connect the AlphaNumeric Display to your microcontroller board as shown below.



Microcontroller 3V to **AlphaNumeric Display I2C VIN**
Microcontroller 3V to **AlphaNumeric Display VIN**
Microcontroller GND to **AlphaNumeric Display GND**
Microcontroller SCL to **AlphaNumeric Display SCL**
Microcontroller SDA to **AlphaNumeric Display SDA**

HT16K33 Library Installation

To use with CircuitPython, you need to first install the HT16K33 library, and its dependencies, into the **lib** folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

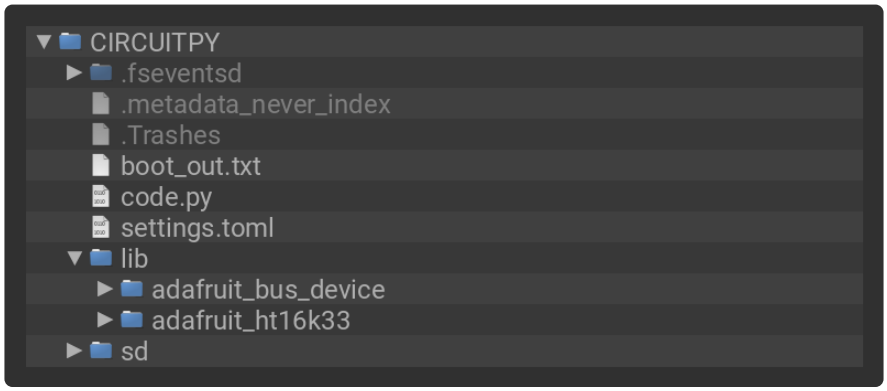
Thankfully, you can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder and the code.py file** to your **CIRCUITPY** drive.

Your **CIRCUITPY/lib** folder should contain the following folders:

- **adafruit_bus_device/**
- **adafruit_ht16k33/**

```
# SPDX-FileCopyrightText: 2022 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT

import time

import board

from adafruit_ht16k33 import segments

# Create the display object.
# Display connected to STEMMA QT connector.
display = segments.Seg14x4(board.STEMMA_I2C())
# Display connected to I2C pins.
# display = segments.Seg14x4(board.I2C())  # uses board.SCL and board.SDA

# This section displays four 0's across the display. The code shows four
```

```
# different ways to use the set_digit_raw function. Each is labeled below.
# 16-bit Hexadecimal number
display.set_digit_raw(0, 0x2D3F)
time.sleep(0.2)
# 16-bit Binary number
display.set_digit_raw(1, 0b0010110100111111)
time.sleep(0.2)
# 8-bit Binary Tuple
display.set_digit_raw(2, (0b00101101, 0b00111111))
time.sleep(0.2)
# 8-bit Hexadecimal List
display.set_digit_raw(3, [0x2D, 0x3F])
time.sleep(0.2)

# Delay between.
time.sleep(2)

# Scroll "Hello, world!" across the display. Setting the loop parameter to false
allows you to
# tell the marquee function to run only once. By default, marquee loops
indefinitely.
display.marquee("Hello, world!", loop=False)

# Delay between.
time.sleep(2)

# Scroll special characters, uppercase and lowercase letters, and numbers across
# the display in a loop. This section will continue to run indefinitely.
display.marquee("".join(chr(character) for character in range(ord("!"), ord("z") +
1)))
```



Python Wiring and Setup

Wiring

It's easy to use AlphaNumeric Displays with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

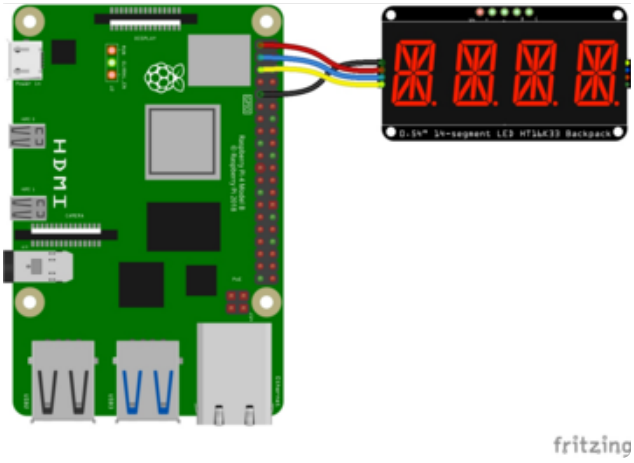
This section will cover how to wire the AlphaNumeric Display to your Raspberry Pi. First assemble your AlphaNumeric Display.

Since there's dozens of Linux computers/boards you can use, this guide will just 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).

Connect the AlphaNumeric Display as shown below to your Raspberry Pi.

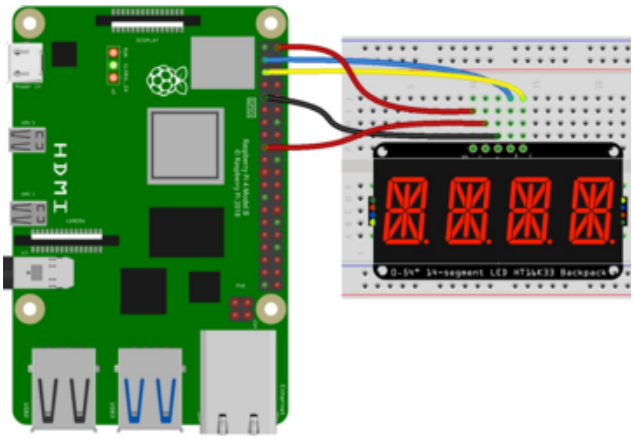
Wiring STEMMA QT Version

Here is an example of wiring the STEMMA QT version of the backpack to a Raspberry Pi using the STEMMA QT connector.



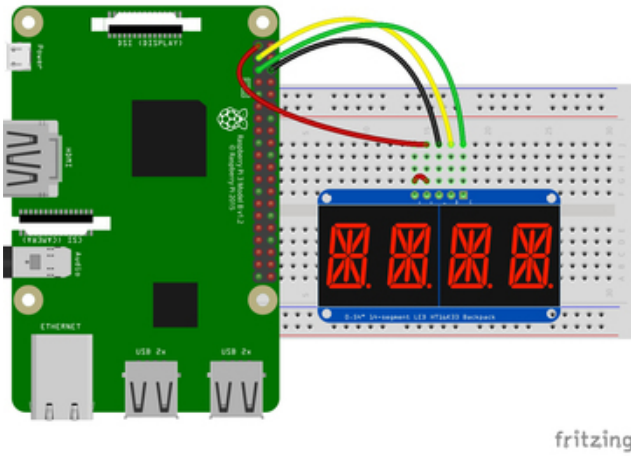
Pi GND to **backpack GND (black wire)**
Pi 3.3V to **backpack VIO (red wire)**
Pi SDA to **backpack SDA (blue wire)**
Pi SCL to **backpack SCL (yellow wire)**

Here is an example of wiring the STEMMA QT version of the backpack using a solderless breadboard. This example also includes how to wire up the VHi pin, which makes the LEDs appear brighter.



Pi GND to **backpack GND (black wire)**
Pi 3.3V to **backpack VIO (lower red wire)**
Pi SDA to **backpack SDA (blue wire)**
Pi SCL to **backpack SCL (yellow wire)**
Pi 5V to **backpack VHi (upper red wire)**

Wiring Original Version



Raspberry Pi 3.3V to **AlphaNumeric Display I2C VIN**
Raspberry Pi 3.3V to **AlphaNumeric Display VIN**
Raspberry Pi GND to **AlphaNumeric Display GND**
Raspberry Pi SCL to **AlphaNumeric Display SCL**
Raspberry Pi SDA to **AlphaNumeric Display SDA**

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

You also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, you'll use `board` and `Seg14x4`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the display, you just pass `i2c` in.

> When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.segments import Seg14x4

i2c = board.I2C()
display = Seg14x4(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
display = Seg14x4(i2c, address=0x71)
```

If you intend to chain multiple displays together, you will need to alter the address of subsequent boards by bridging the address pins in various combinations (https://

adafru.it/11eg). If you have an unsoldered board, and a board with the A0 pad solder-bridged, you would initialize the two displays as follows.

```
display = Seg14x4(i2c, address=(0x70, 0x71))
```

To add further displays, ensure the address is different on all of them (https://adafru.it/11eg), and initialize it the same way as above, but add more comma-separated addresses to the `address=()` tuple.

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Printing Text

To print text to the display, you just use the print function. So if you want to print ABCD, you would use the following:

```
display.print("ABCD")
```

Printing Numbers

Printing numbers is done similar to printing text, except without the quotes, though you can still print numbers in a string as well.

```
display.print(1234)
```

Printing Hexidecimal Values

To print hexidecimal values, you use the `print_hex` function:

```
display.print_hex(0x1A2B)
```

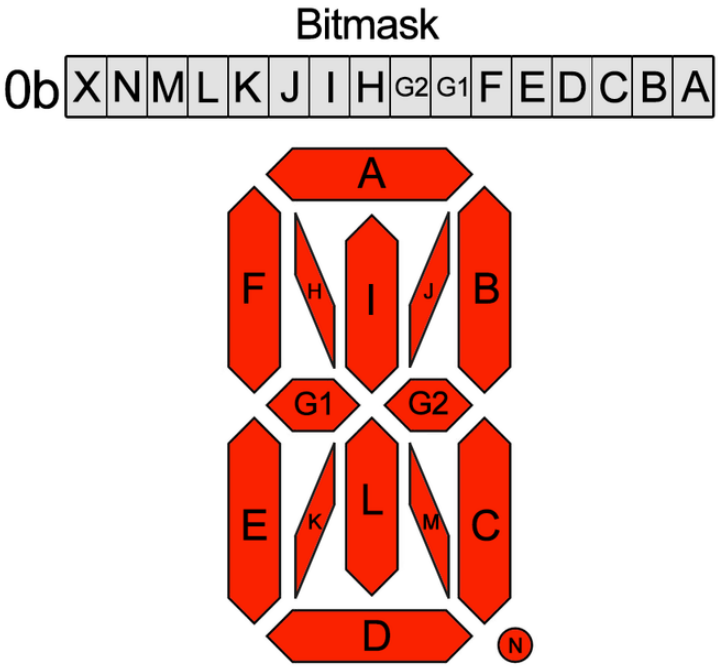
Setting Individual Characters

To set individual characters, you simply treat the `display` object as a list and set it to the value that you would like.

```
display[0] = '1'
display[1] = '2'
display[2] = 'A'
display[3] = 'B'
```

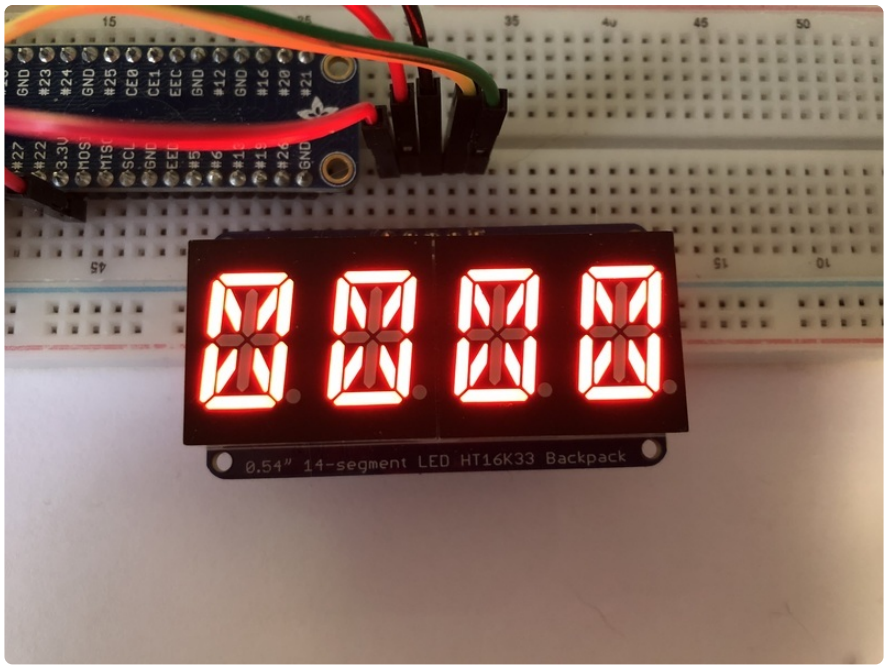
Setting Individual Segments

To set individual segments to turn on or off, you would use the set_digit_raw function to pass the digit that you want to change and the bitmask. This can be really useful for creating your own characters. The bitmask corresponds to the following diagram. The highest bit is not used, so an X represents that spot to indicate that.



The bitmask is a 16-bit number that can be passed in as a single Hexidecimal, Decimal, or binary number. It can also be passed in as a list or tuple containing 2 separate 8-bit numbers. Here are some of the ways to set the digits. All of these different methods create a box with an X in the center:

```
display.set_digit_raw(0, 0x2D3F)
display.set_digit_raw(1, 0b0010110100111111)
display.set_digit_raw(2, (0b00101101, 0b00111111))
display.set_digit_raw(3, [0x2D, 0x3F])
```



Filling all Segments

To fill the entire display, just use the fill() function and pass in either 0 or 1 depending on whether you want all segments off or on. For instance, if you wanted to set everything to on, you would use:

```
display.fill(1)
```

Scrolling Display Manually

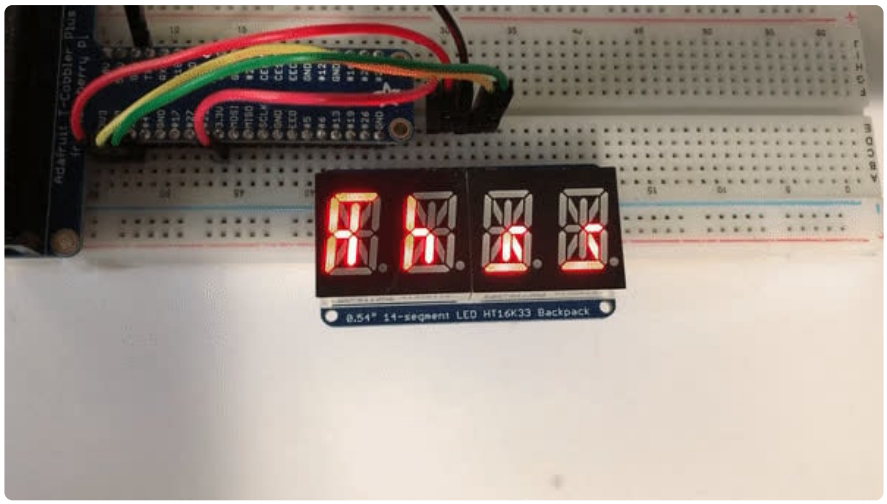
If you want to scroll the displayed data to the left, you can use the `scroll()` function. You can pass in the number of places that you want to scroll. The right-most digit will remain unchanged and you will need to set that manually. After scrolling, you will need to call the show function. For example if you wanted to print an A and then scroll it over to spaces, you would do the following.

```
display.print("A")
display.scroll(2)
display[3] = " "
display.show()
```

Displaying an Automatic Scrolling Marquee

To make displaying long text easier, you can use the marquee function. You just pass it the full string. Optionally, you can pass it the amount of delay between each character:

```
display.marquee("This is a really long string ")
```



By default it is 0.25 seconds, but you can change this by providing a second parameter. You can optionally pass `False` for a third parameter if you would not like to have it loop. So if you wanted each character to display for half a second and didn't want it to loop, you would use the following:

```
display.marquee('This is a really long string ', 0.5, False)
```

Full Example

Click **Download Project Bundle** below to download a full example **code.py** and necessary libraries to run it.

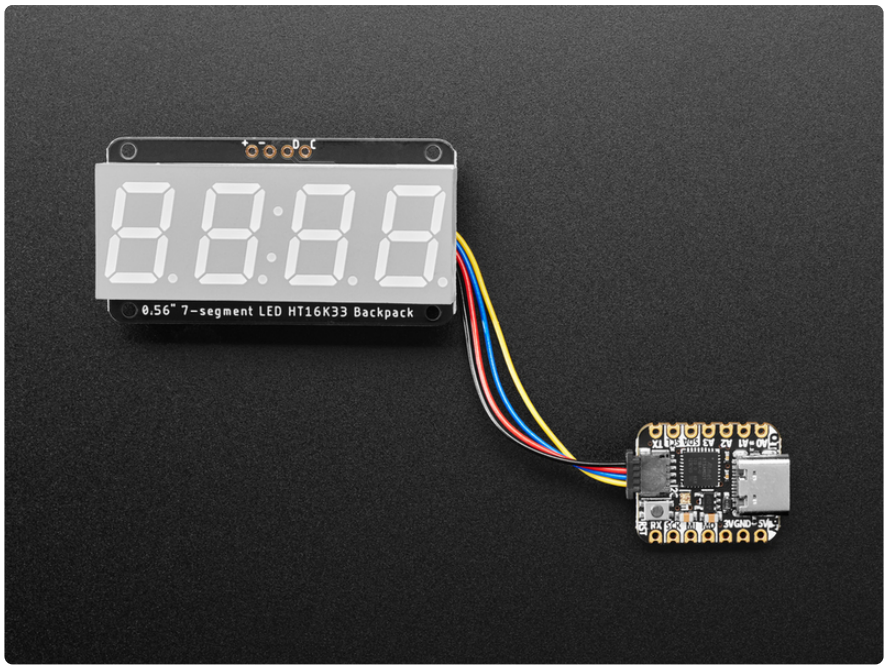
```
# SPDX-FileCopyrightText: 2022 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT

import time

import board

from adafruit_ht16k33 import segments

# Create the display object.
# Display connected to STEMMA QT connector.
display = segments.Seg14x4(board.STEMMA_I2C())
# Display connected to I2C pins.
# display = segments.Seg14x4(board.I2C())  # uses board.SCL and board.SDA

# This section displays four 0's across the display. The code shows four
# different ways to use the set_digit_raw function. Each is labeled below.
# 16-bit Hexadecimal number
display.set_digit_raw(0, 0x2D3F)
time.sleep(0.2)
# 16-bit Binary number
display.set_digit_raw(1, 0b0010110100111111)
time.sleep(0.2)
# 8-bit Binary Tuple
display.set_digit_raw(2, (0b00101101, 0b00111111))
time.sleep(0.2)
# 8-bit Hexadecimal List
display.set_digit_raw(3, [0x2D, 0x3F])
time.sleep(0.2)
```

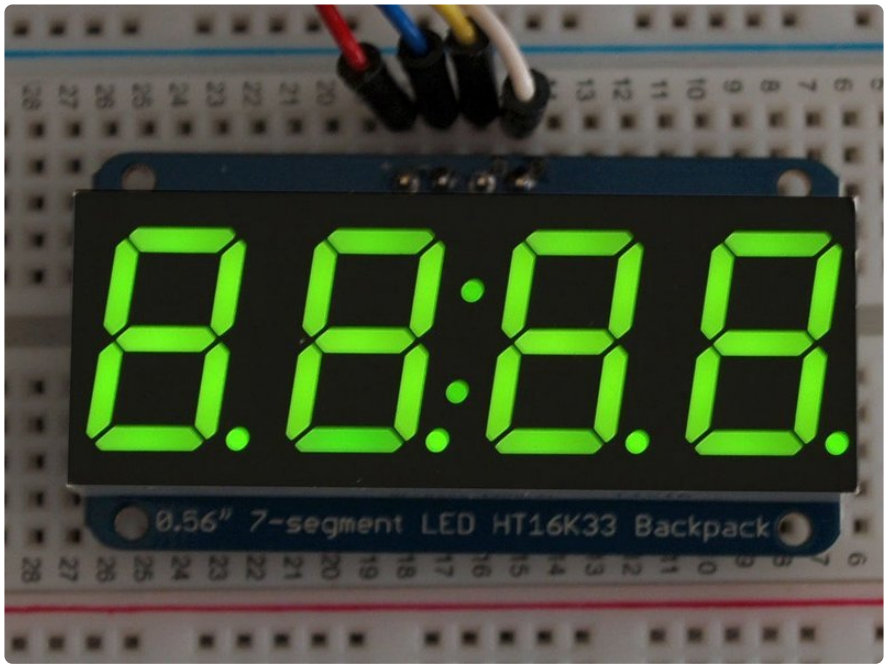
```
# Delay between.
time.sleep(2)

# Scroll "Hello, world!" across the display. Setting the loop parameter to false
allows you to
# tell the marquee function to run only once. By default, marquee loops
indefinitely.
display.marquee("Hello, world!", loop=False)

# Delay between.
time.sleep(2)

# Scroll special characters, uppercase and lowercase letters, and numbers across
# the display in a loop. This section will continue to run indefinitely.
display.marquee("".join(chr(character) for character in range(ord("!"), ord("z") +
1)))
```

0.56" 7-Segment Backpack

This version of the LED backpack is designed for these big bright 7-segment displays. These 7-segment displays normally require 13 pins (5 'characters' and 8 total segments each) This backpack solves the annoyance of using 13 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all the LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are three address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.

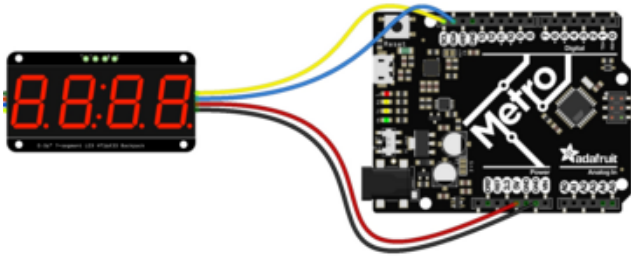
To get you going fast, we have revised this popular board to be the same size and pinout as before but now with two STEMMA QT connectors (https://adafru.it/JqB) on either side that are compatible with the SparkFun Qwiic (https://adafru.it/Fpw) I2C connectors. This allows you to make solderless connections between your development board and the HT16K33 or to chain it with a wide range of other sensors and accessories using a **compatible cable** (https://adafru.it/JnB).

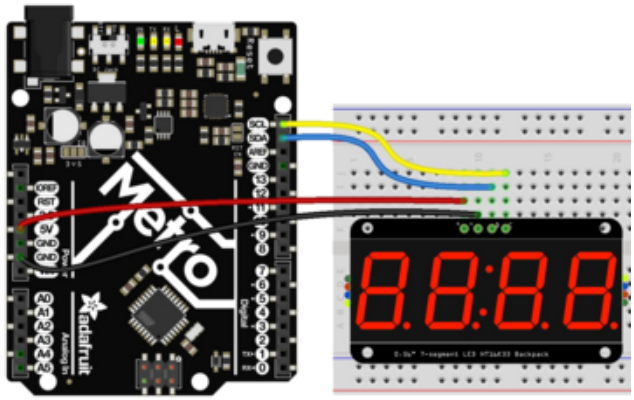




Assembly and Arduino Wiring

For the STEMMA QT version, you can solder the headers on, or you can simply use a STEMMA QT cable!

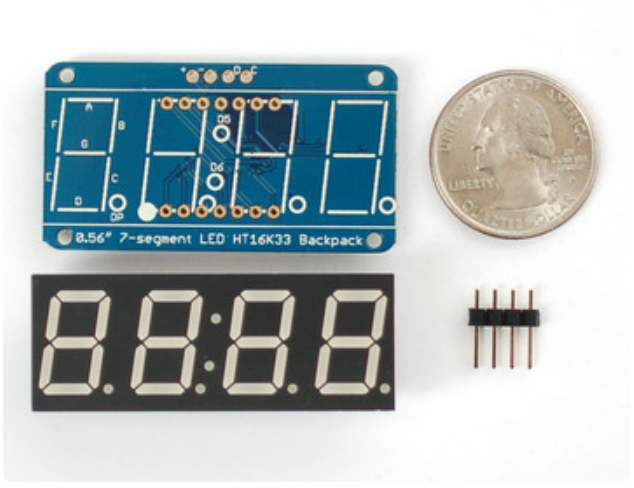




Backpack + (VCC) to **microcontroller 5V** (if using a board with 5V logic, otherwise use 3V for boards with 3V logic) **(red wire)**
Backpack - (GND) to **microcontroller GND (black wire)**
Backpack D (SDA) to **microcontroller SDA (blue wire)**
Backpack C (SCL) to **microcontroller SCL (yellow wire)**

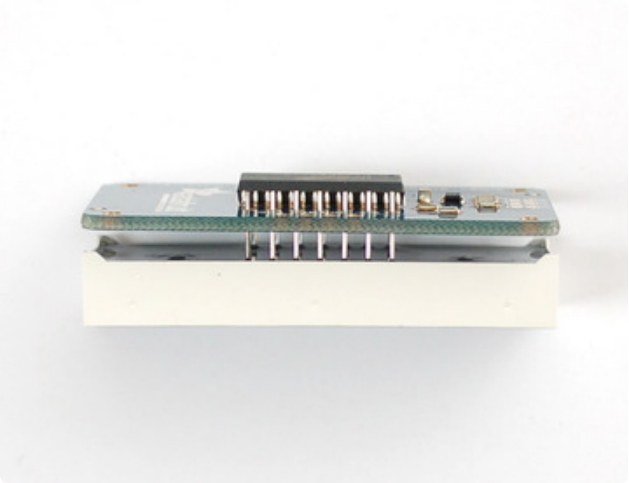
To assemble the LED backpack, follow along below.

> Soldering the 7-segment display is the same for the STEMMA QT version and the original version.

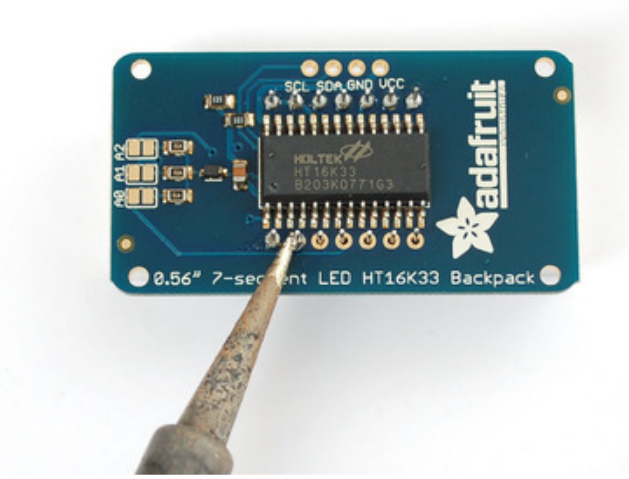


When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as a 7-segment display in one of the colors we provide (say, red, yellow, blue or green). You'll need to solder the matrix onto the backpack but it's an easy task.

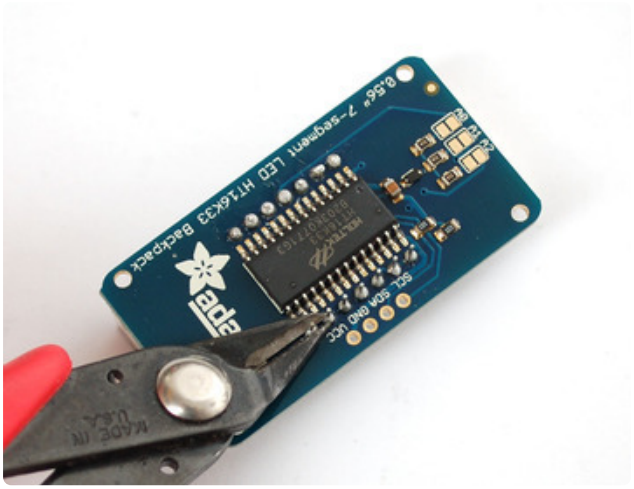
Remove the parts from packaging and place the LED matrix OVER the silkscreen side. **DO NOT PUT THE DISPLAY ON UPSIDE DOWN OR IT WONT WORK!! Check the image below to make sure the 'decimal point' dots are on the bottom, matching the silkscreen.**



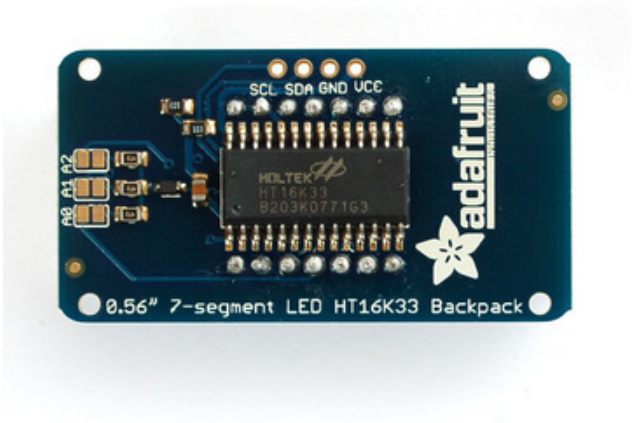
Turn the backpack over so it is sitting flat on the matrix.

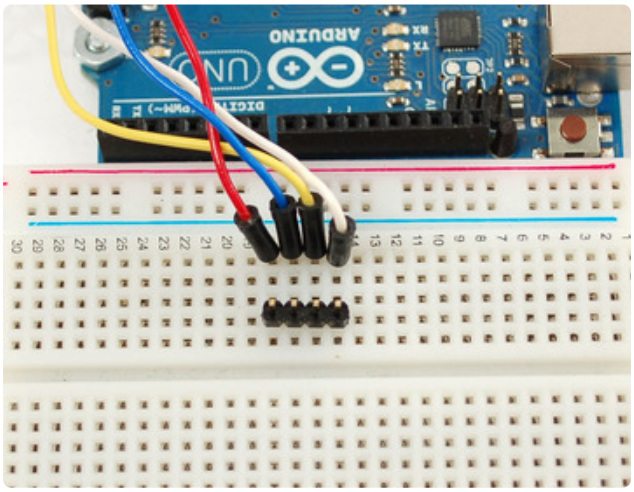


Solder all 14 pins.

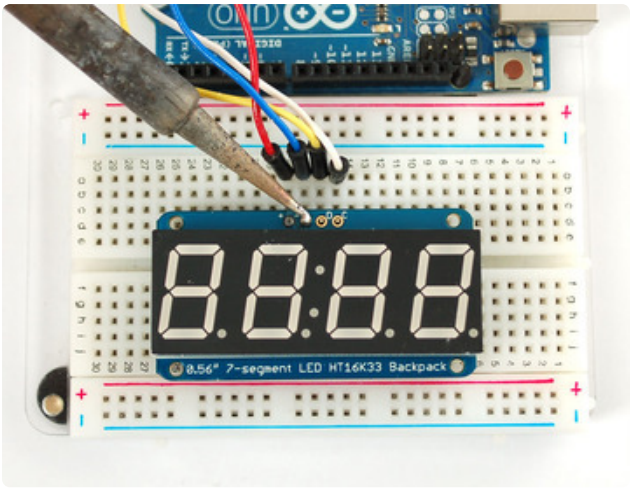


Clip the long pins.





Now you're ready to wire it up to a microcontroller. We'll assume you want to use a 4pin header. You can also of course solder wires directly. Place a 4-pin piece of header with the LONG pins down into the breadboard.



Place the soldered backpack on top of the header and Solder 'em!

That's it! now you're ready to run the firmware!

Arduino Setup

You can use these with a 3.3v or 5v microcontroller. Just connect the VCC+ pin is the same voltage as the logic on your microcontroller.

Seven-Segment Backpack Firmware

We wrote a basic library to help you work with the 7-segment backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

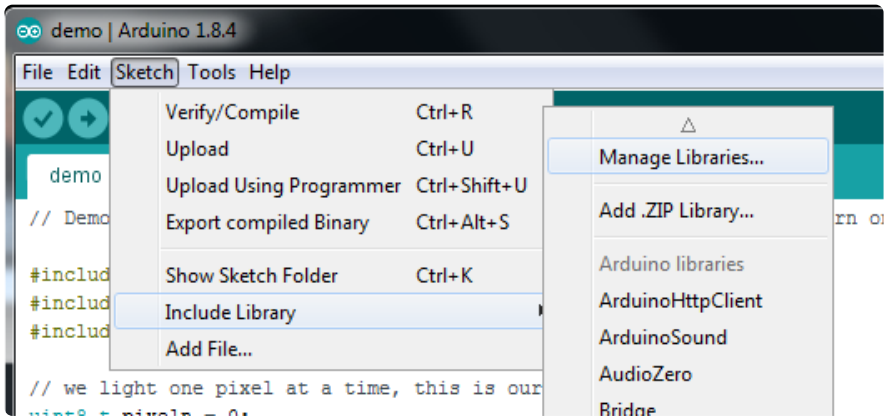
Wiring to the matrix is really easy

- Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo it's Digital #3, on the Mega it's digital #21
- Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo it's Digital #2, on the Mega it's digital #20
- Connect **GND** to common ground
- Connect **VCC**+ to power - 5V is best but 3V also seems to work for 3V microcontrollers.

If using the STEMMA QT version of the backpack, simply plug in the STEMMA QT connectors between boards.

Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

Open up the Arduino library manager:



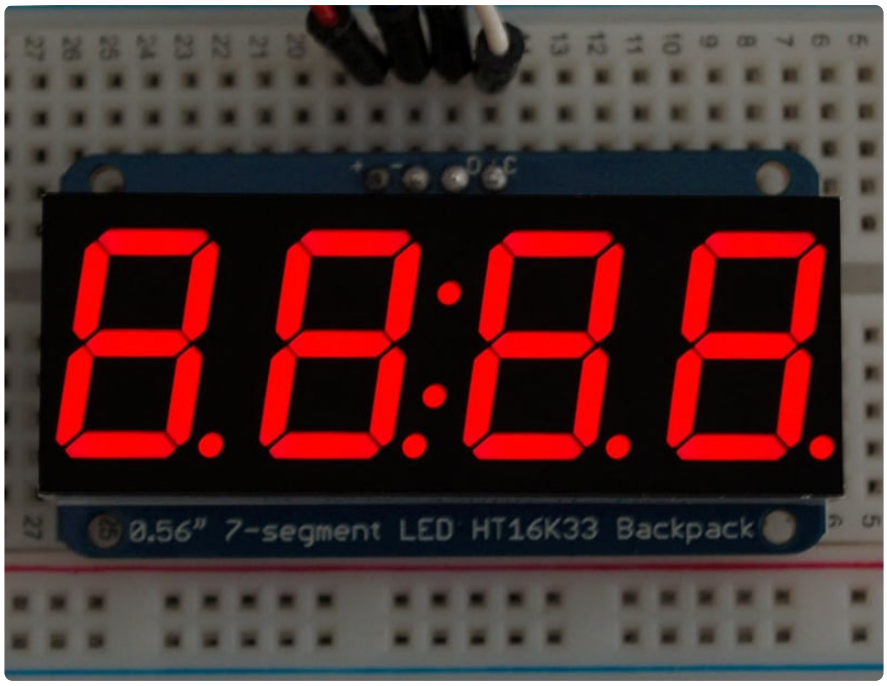
Search for the **Adafruit LED Backpack** library and install it

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

If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

Once you've restarted you should be able to select the **File→Examples→Adafruit_LEDBackpack→sevenseg** example sketch. Upload it to your Arduino as usual. You should see a basic test program that goes through a bunch of different routines.

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)



If using the STEMMA QT version of the board, you'll need to use Wire1 like this: matrix.begin(0x70, &Wire1) ;

Once you're happy that the matrix works, you can write your own sketches.

There's a few ways you can draw to the display. The easiest is to just call **print -** just like you do with **Serial**

- **print(variable,HEX) -** this will print a hexidecimal number, from 0000 up to FFFF
- **print(variable,DEC)** or **print(variable) -** this will print a decimal integer, from 0000 up to 9999

If you need more control, you can call **writeDigitNum(location, number)** - this will write the number (0-9) to a single location. Location #0 is all the way to the left, location #2 is the colon dots so you probably want to skip it, location #4 is all the way to the right. If you want a decimal point, call **writeDigitNum(location, number, true)** which will paint the decimal point. To draw the colon, use**drawColon(true** or **false)**

If you want even more control, you can call **writeDigitRaw(location,bitmask)** to draw a raw 8-bit mask (as stored in a uint8_t) to that location.

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the backpack:

- **setBrightness(brightness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.

CircuitPython Wiring and Setup

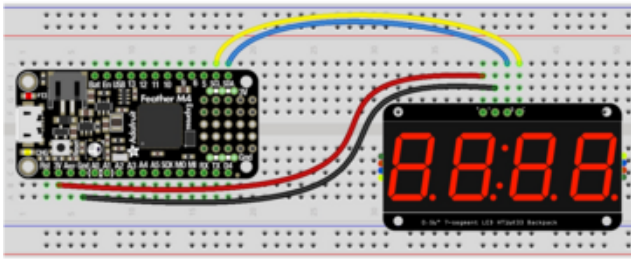
Wiring

It's easy to use LED 7-Segment Displays with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

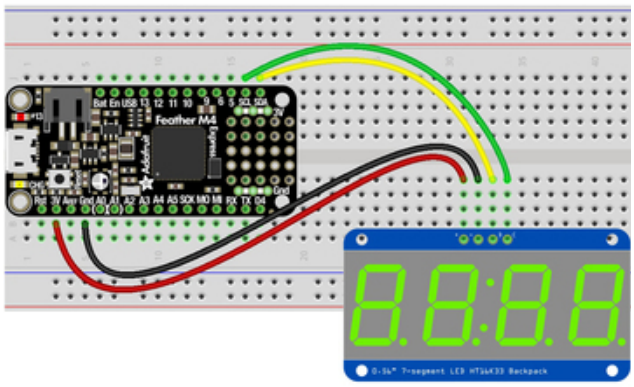
You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the 7-Segment Display to your CircuitPython microcontroller board. First assemble your 7-Segment Display.

Connect the 7-Segment Display to your microcontroller board as shown below.



Microcontroller 3V to **7-Segment Display VIN (red wire)**
Microcontroller GND to **7-Segment Display GND (black wire)**
Microcontroller SCL to **7-Segment Display SCL (yellow wire)**
Microcontroller SDA to **7-Segment Display SDA (blue wire)**



Download Fritzing Object

https://adafru.it/ICk

Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

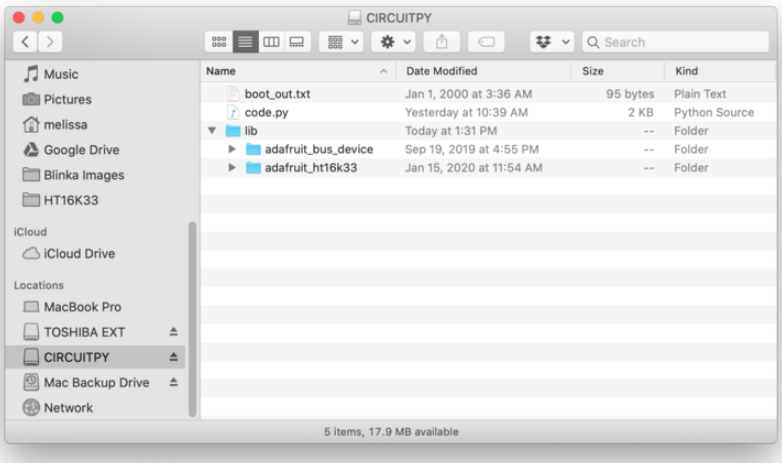
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

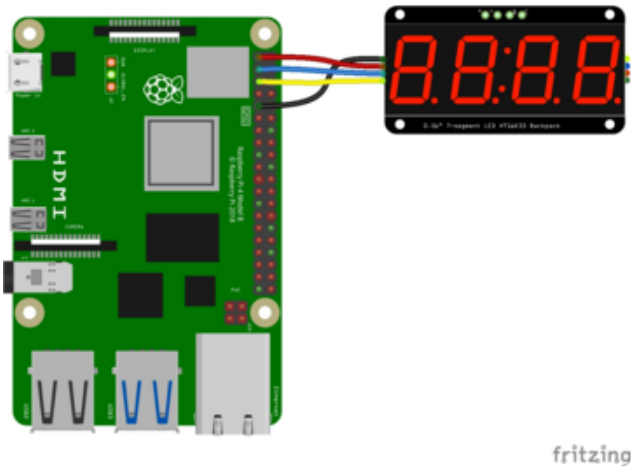
Wiring

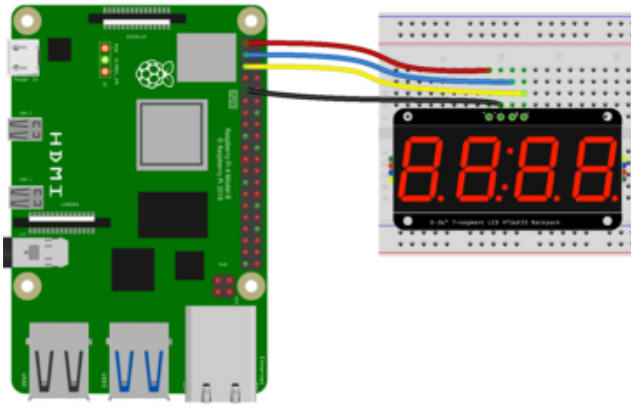
It's easy to use 7-Segment Displays with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the 7-Segment Display to your Raspberry Pi. First assemble your 7-Segment Display.

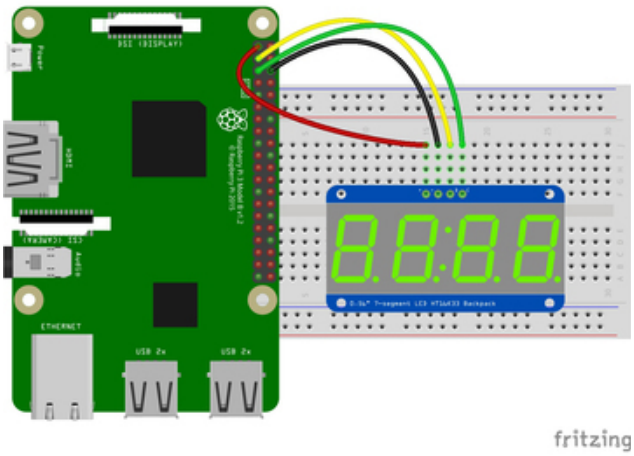
Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Connect the 7-Segment Display as shown below to your Raspberry Pi.





Raspberry Pi 3.3V to **7-Segment Display VIN (red wire)**
Raspberry Pi GND to **7-Segment Display GND (black wire)**
Raspberry Pi SCL to **7-Segment Display SCL (yellow wire)**
Raspberry Pi SDA to **7-Segment Display SDA (blue wire)**



Download Fritzing Object

https://adafru.it/ICI

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `Seg7x4`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the display, you just pass `i2c` in.

> When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.segments import Seg7x4

i2c = board.I2C()
display = Seg7x4(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
display = Seg7x4(i2c, address=0x71)
```

If you intend to chain multiple displays together, you will need to alter the address of subsequent boards by bridging the address pins in various combinations (https://adafru.it/11eg). If you have an unsoldered board, and a board with the A0 pad solder-bridged, you would initialisz the two displays as follows.

```
display = Seg14x4(i2c, address=(0x70, 0x71))
```

To add further displays, ensure the address is different on all of them (https://adafru.it/11eg), and initialize it the same way as above, but add more comma-separated addresses to the `address=()` tuple.

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Printing Text

To print text to the display, you just use the print function. For the 7-segment display, valid characters are 0-9, letters A-F, a period, and a hyphen. So if we want to print ABCD, we would use the following:

```
display.print("ABCD")
```

Printing Numbers

Printing numbers is done similar to printing text, except without the quotes, though you can still print numbers in a string as well.

```
display.print(1234)
```

Printing Hexadecimal Values

To print hexadecimal values, you use the `print_hex` function:

```
display.print_hex(0x1A2B)
```

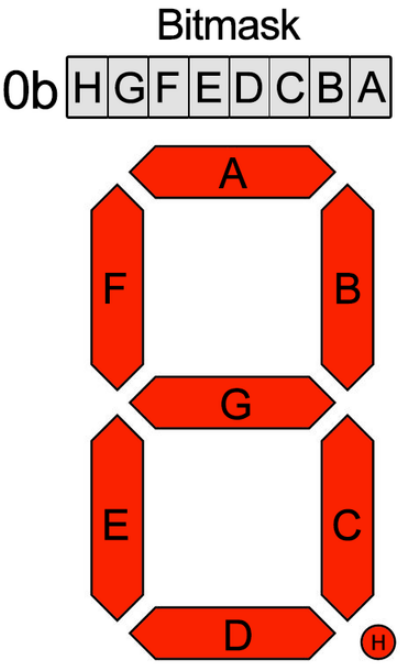
Setting Individual Characters

To set individual characters, you simply treat the `display` object as a list and set it to the value that you would like.

```
display[0] = '1'
display[1] = '2'
display[2] = 'A'
display[3] = 'B'
```

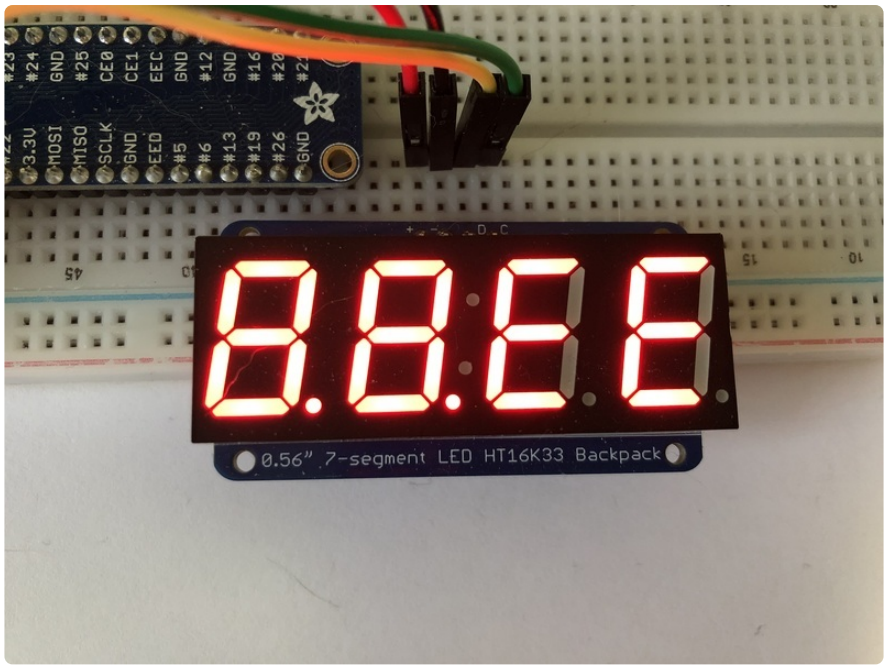
Setting Individual Segments

To set individual segments to turn on or off, you would use the set_digit_raw function to pass the digit that you want to change and the bitmask. This can be really useful for creating your own characters. The bitmask corresponds to the following diagram:



The bitmask is a single 8-bit number that can be passed in as a single Hexadecimal, Decimal, or binary number. This will use a couple different methods to display `8.8.EE`:

```
display.set_digit_raw(0, 0xFF)
display.set_digit_raw(1, 0b11111111)
display.set_digit_raw(2, 0x79)
display.set_digit_raw(3, 0b01111001)
```



Filling all Segments

To fill the entire display, just use the fill() function and pass in either 0 or 1 depending on whether you want all segments off or on. For instance, if you wanted to set everything to on, you would use:

```
display.fill(1)
```

Scrolling Display Manually

If you want to scroll the displayed data to the left, you can use the `scroll()` function. You can pass in the number of places that you want to scroll. The right-most digit will remain unchanged and you will need to set that manually. After scrolling, you will need to call the show function. For example if you wanted to print an A and then scroll it over to spaces, you would do the following.

```
display.print("A")
display.scroll(2)
display[3] = " "
display.show()
```

Displaying the Colon

There are a couple of different ways to display a colon on the 7-segment display. The first and easiest way is to use the print function:

```
display.print("12:30")
```

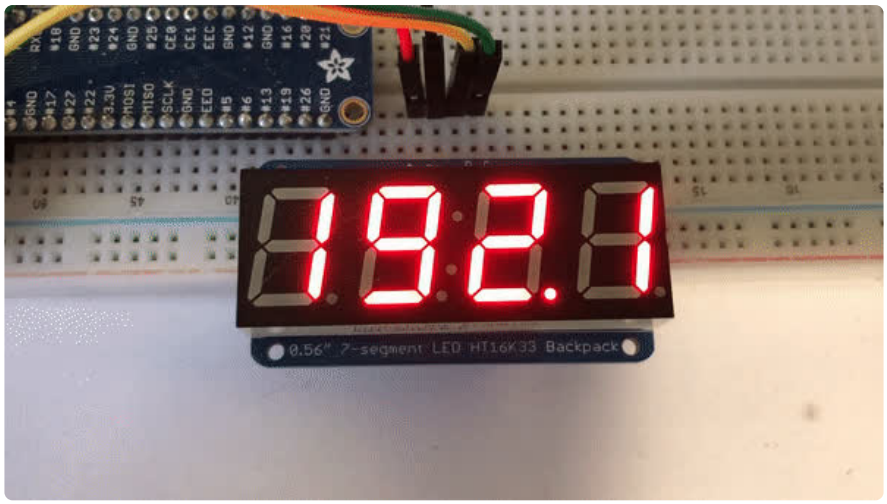
The other way to control it is to access the colon with the colon property and set it to `True` or `False`:

```
display.colon = False
```

Displaying an Automatic Scrolling Marquee

To make displaying long text easier, we've added a marquee function. You just pass it the full string. Optionally, you can pass it the amount of delay between each character. This may be useful for displaying an IP address, a phone number, or other numeric data:

```
display.marquee('192.168.100.102... ')
```



By default it is 0.25 seconds, but you can change this by providing a second parameter. You can optionally pass `False` for a third parameter if you would not like to have it loop. So if you wanted each character to display for half a second and didn't want it to loop, you would use the following:

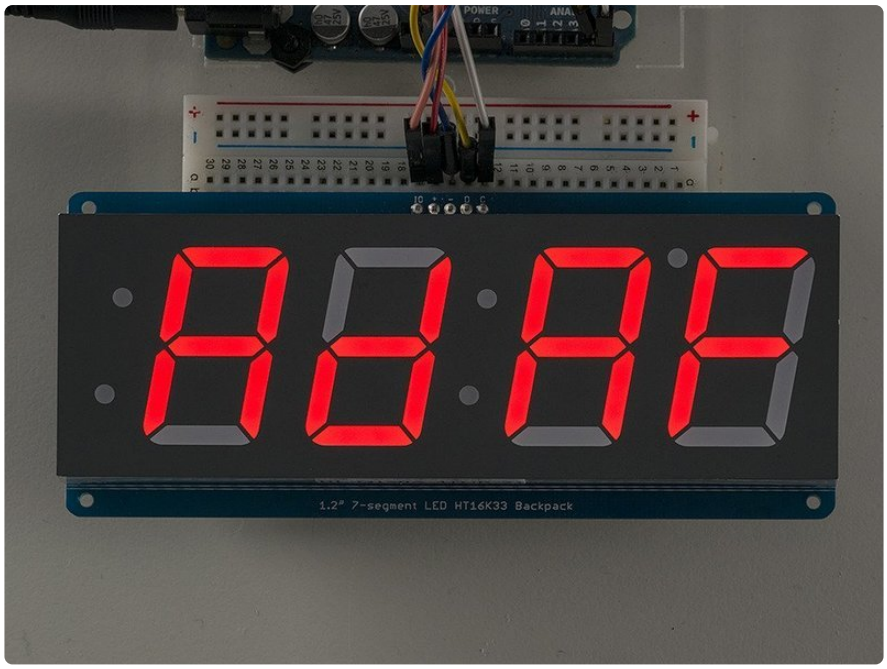
```
display.marquee('192.168.100.102... ', 0.5, False)
```

1.2" 7-segment Backpack

These backpacks drive the massive 1.2" 7-segment modules. With 2 leds per segment these make a gorgeous and impressive display. The 7-segment displays normally require 16 pins to drive. This backpack uses an I2C constant-current matrix controller on the back of the PCB, so you only need 2 pins to drive it!

The controller chip takes care of multiplexing all the LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are three address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.

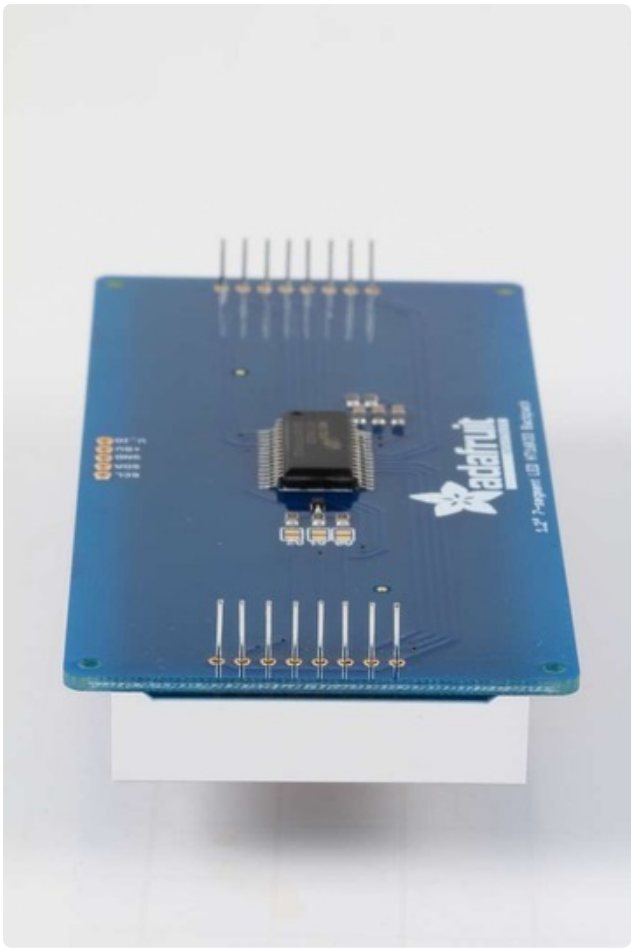
New! As of June 13, 2023 we have updated the backpack board to have a 5V boost converter which means you can use it with 3.3V power and get nice bright LED segments without having to wire up a separate 5V power supply. It also has Stemma QT ports for plug-n-play connectivity



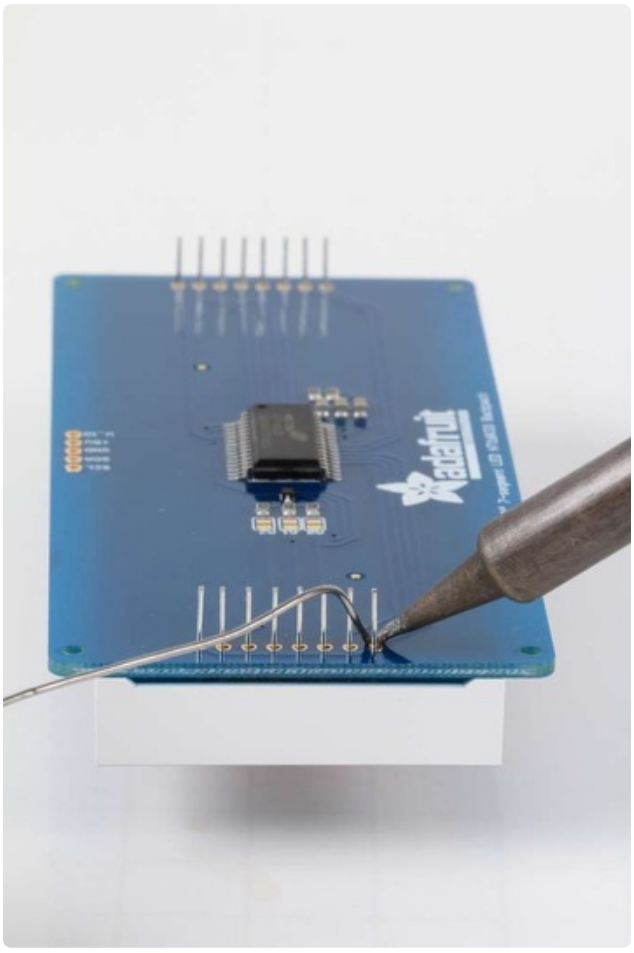
Assembly

When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as a 7-segment display in one of the colors we provide (say, red, yellow, blue or green). You'll need to solder the matrix onto the backpack but its an easy task.

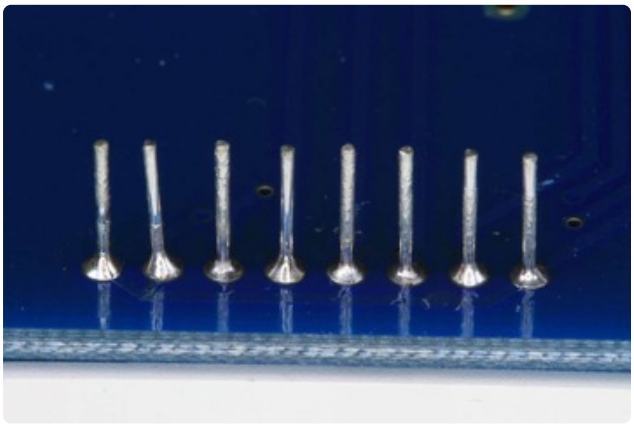
Remove the parts from packaging and place the LED matrix OVER the silkscreen side. **DO NOT PUT THE DISPLAY ON UPSIDE DOWN OR IT WONT WORK!! Check the image below to make sure the 'decimal point' dots are in the same location as the ones on the silkscreen.**



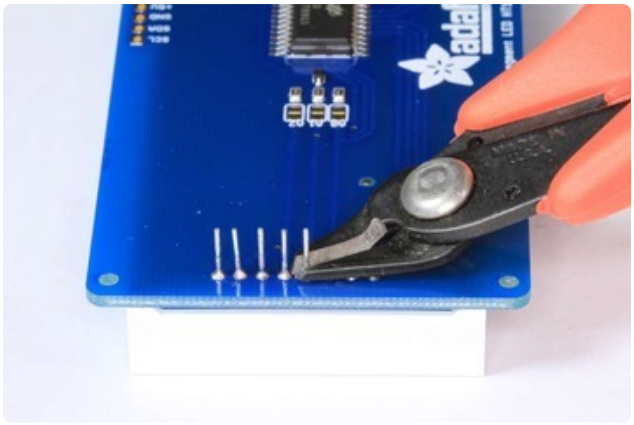
Turn the backpack over so its sitting flat on the matrix and ready to solder.



Then solder each pin. There are 8 on each end for a total of 16.

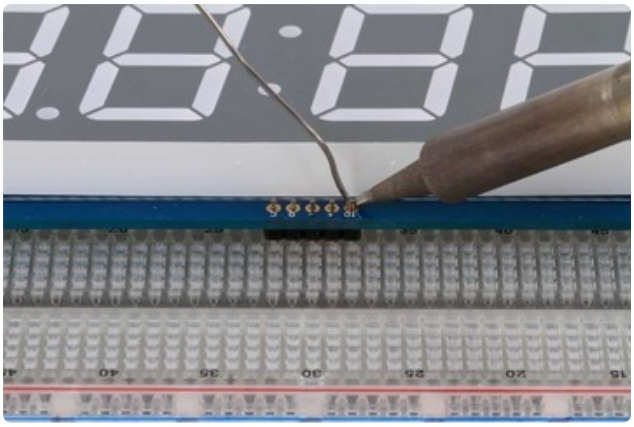


That completes the basic assembly. For use on a breadboard, you will want to also install a 5-pin header on the edge of the board.



Clip the long pins close to the board.

Cut the header strip to length if necessary and insert LONG pins down into the breadboard.



Then solder all 5 pins.

Arduino Wiring and Setup

You can use these with a 3.3v or 5v microcontroller. Just make sure the IO pin is the same voltage as the logic on your microcontroller.

Now you are ready to wire it to your microcontroller. The required connections are:

- **"D"** - I2C Data Pin (SDA)
- **"C"** - I2C Clock Pin (SCL)
- **"+"** - 5v. (Will not run on 3.3v!)
- **"-"** - GND
- **"IO"** - I2C bus voltage.

Due to the size of this display, there are 2 LEDs in series for each segment. Because of this, the display requires 5v to run. It will not run on 3.3v.

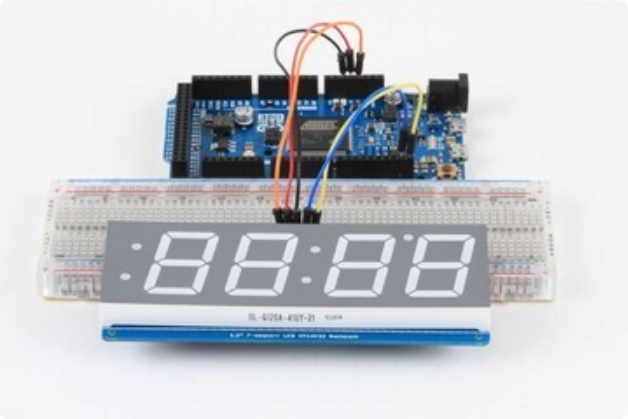
For use with 3.3v processors, connect the IO pin to 3.3v. This will keep the I2C bus signals at a safe level for your processor.

With 5v processors like the Arduino UNO, this pin can be connected to either 5v or 3.3v. (use 3.3v if there will be other 3.3v devices on the bus)

Arduino Wiring - R3 and later

Connect:

D → SDA
C → SCL
+ → 5v
- → GND
IO → jumper to + for 5v.



Arduino Due and Other 3.3v Processors

Connect:

D → SDA
C → SCL
+ → 5v
- → GND
IO → 3.3v

Arduino "Classic" Wiring

Connect:

D → Analog-4 or Digital 20 for the Mega
C → Analog-5 or Digital 21 for the Mega
+ → 5v
- → GND
IO → jumper to + for 5v.

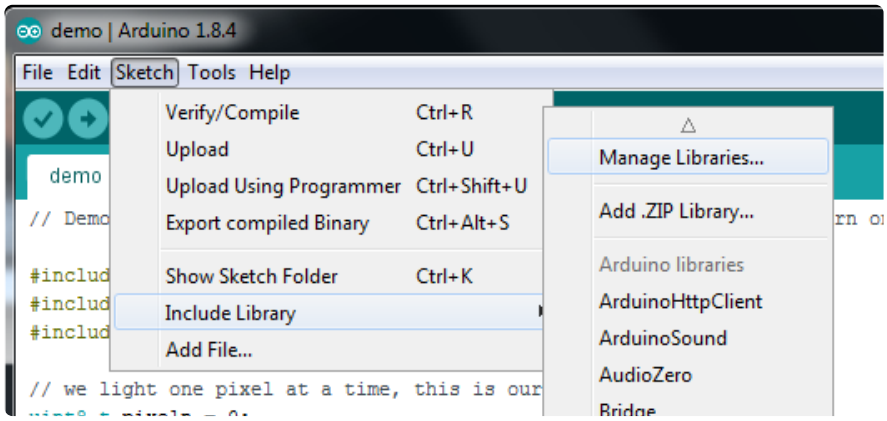
OK, now on to the firmware!

Seven-Segment Backpack Firmware

Our 7-segment backpack library makes it easy to program these displays. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

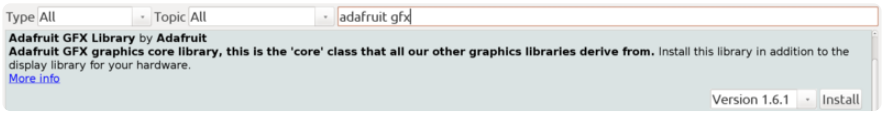
Download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it

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



If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

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)

You should now be able to select the **File→Examples→Adafruit_LEDBackpack→sevenseg** example sketch. Upload it to your Arduino as usual. You should see a "sevenseg" example sketch that will demonstrate various capabilities of the library and the display.

Once you're happy that the matrix works, you can write your own sketches.

There's a few ways you can draw to the display. The easiest is to just call **print -** just like you do with **Serial**

- **print(variable,HEX) -** this will print a hexadecimal number, from 0000 up to FFFF
- **print(variable,DEC)** or **print(variable) -** this will print a decimal integer, from 0000 up to 9999

If you need more control, you can call **writeDigitNum(location, number)** - this will write the number (0-9) to a single location. Location #0 is all the way to the left, location #2 is the colon dots so you probably want to skip it, location #4 is all the way to the right.

To control the colon and decimal points, use the writeDigitRaw(location, bitmap) function. (Note that both dots of the center colon are wired together internal to the display, so it is not possible to address them separately.) Specify 2 for the location and the bits are mapped as follows:

- 0x02 - center colon (both dots)
- 0x04 - left colon - lower dot
- 0x08 - left colon - upper dot
- 0x10 - decimal point (upper right)

If you want a decimal point, call **writeDigitNum(location, number, true)** which will paint the decimal point. To draw the colon, use **drawColon(true** or **false)**

If you want full control of the segments in all digits, you can call **writeDigitRaw(location, bitmask)** to draw a raw 8-bit mask (as stored in a uint8_t) to any location.

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the backpack:

- **setBrightness(brightness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.

CircuitPython Wiring and Setup

Wiring

It's easy to use LED 7-Segment Displays with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

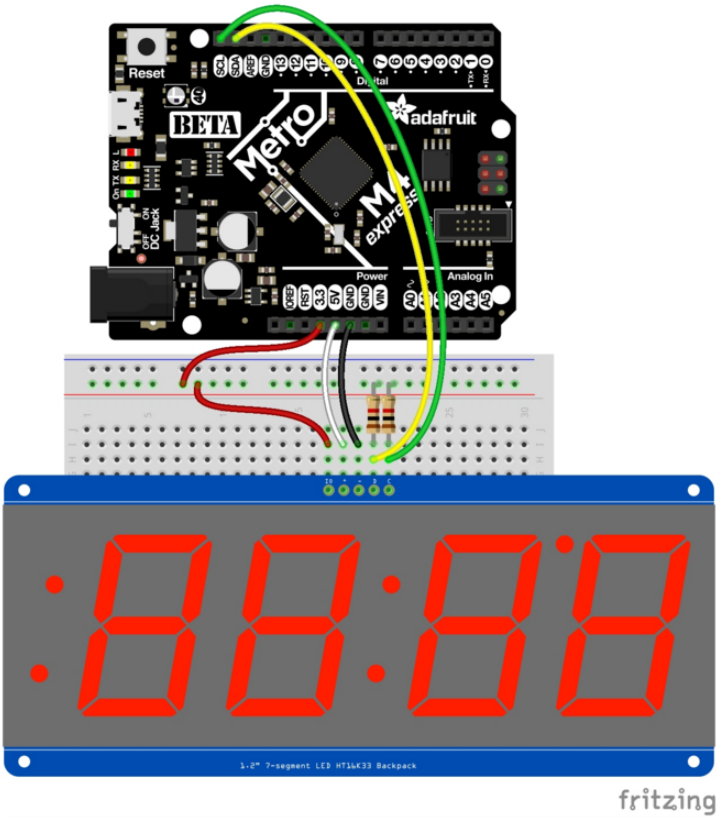
You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the 7-Segment Display to your CircuitPython microcontroller board. First assemble your 7-Segment Display.

Connect the 7-Segment Display to your microcontroller board as shown below.

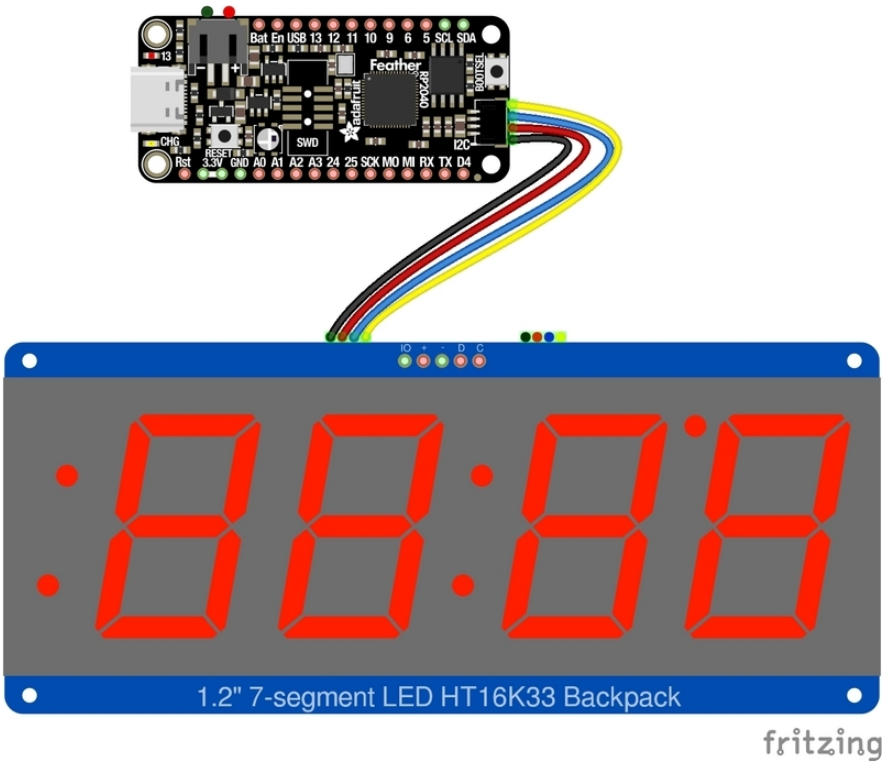
Make sure your device has enough RAM. Devices such as those based on the SAMD21 will likely not be able to run the examples. Devices such as those with the RP2040, ESP32-S2, ESP32-S3, or an M4 chip should work well.

In the latest CircuitPython, you will need to connect 1K resistor Pull-ups to the SCL and SDA inputs.



- **Microcontroller 3.3V** to **3.3v Bus**
- **3.3v Bus** to **7-Segment Display IO**
- **Microcontroller 5V** to **7-Segment Display VIN**
- **Microcontroller GND** to **7-Segment Display GND**
- **Microcontroller SCL** to **7-Segment Display SCL**
- **Microcontroller SDA** to **7-Segment Display SDA**
- **1K Resistor** between **3.3v Bus** and **7-Segment Display SCL**
- **1K Resistor** between **3.3v Bus** and **7-Segment Display SDA**

You can also use a STEMMA QT cable to connect any board with a STEMMA QT port to the updated display.



Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

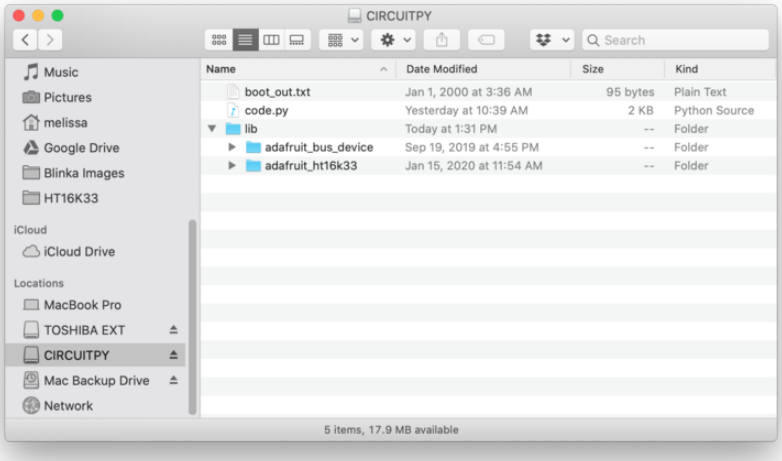
Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**

- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

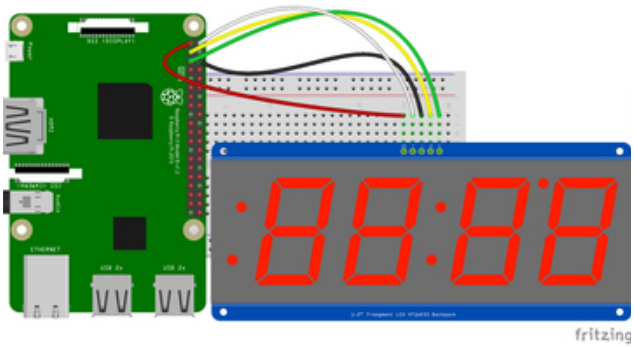
Wiring

It's easy to use 7-Segment Displays with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the 7-Segment Display to your Raspberry Pi. First assemble your 7-Segment Display.

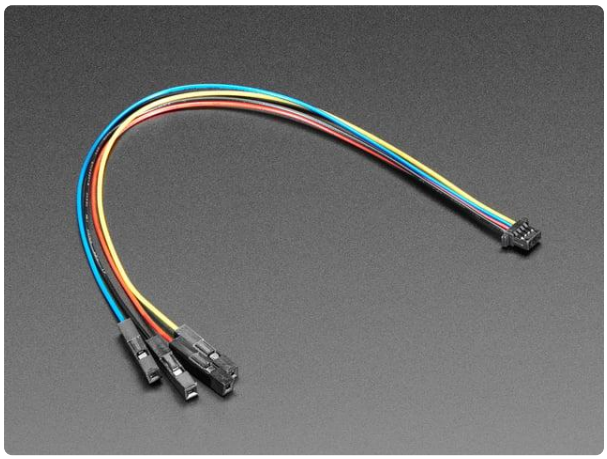
Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Using wires, connect the 7-Segment Display as shown below to your Raspberry Pi.



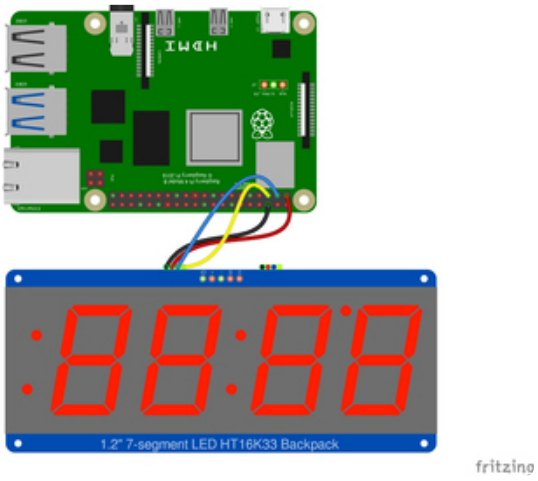
Raspberry Pi 3.3V to **7-Segment Display IO**
Raspberry Pi 5V to **7-Segment Display VIN**
Raspberry Pi GND to **7-Segment Display GND**
Raspberry Pi SCL to **7-Segment Display SCL**
Raspberry Pi SDA to **7-Segment Display SDA**

To wire up the display using the STEMMA QT port, if you have a board with STEMMA QT connected to the 40-pin header, you can just use a STEMMA QT cable. If you want to connect the display directly to the header pins, you can just use a cable such as:



STEMMA QT / Qwiic JST SH 4-pin Cable with Premium Female Sockets
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium female headers on the other. Compared with the chunkier...
https://www.adafruit.com/product/4397

Then connect the 7-Segment Display as shown below to your Raspberry Pi:



Raspberry Pi 3.3V to STEMMA QT **red wire**
Raspberry Pi GND to STEMMA QT **black wire**
Raspberry Pi SCL to STEMMA QT **yellow wire**
Raspberry Pi SDA to STEMMA QT **blue wire**

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

If you are using one of the matrix displays, you also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `BigSeg7x4`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the display, you just pass `i2c` in.

> When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.segments import BigSeg7x4

i2c = board.I2C()
display = BigSeg7x4(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
display = BigSeg7x4(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/4 increments

between 0 and 3 with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Printing Text

To print text to the display, you just use the print function. For the 7-segment display, valid characters are 0-9, letters A-F, and a hyphen. So if we want to print ABCD, we would use the following:

```
display.print("ABCD")
```

Printing Numbers

Printing numbers is done similar to printing text, except without the quotes, though you can still print numbers in a string as well.

```
display.print(1234)
```

Printing Hexidecimal Values

To print hexidecimal values, you use the `print_hex` function:

```
display.print_hex(0x1A2B)
```

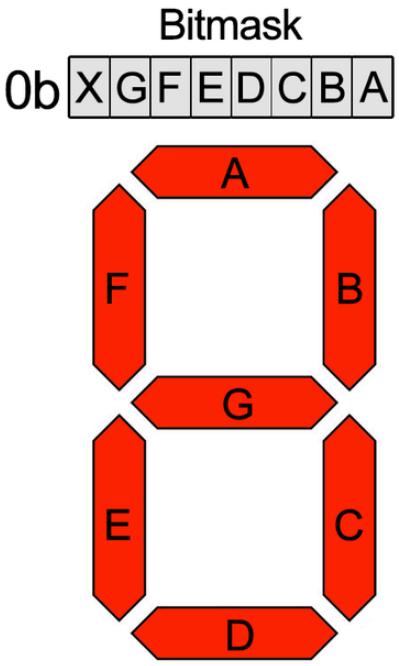
Setting Individual Characters

To set individual characters, you simply treat the `display` object as a list and set it to the value that you would like.

```
display[0] = '1'
display[1] = '2'
display[2] = 'A'
display[3] = 'B'
```

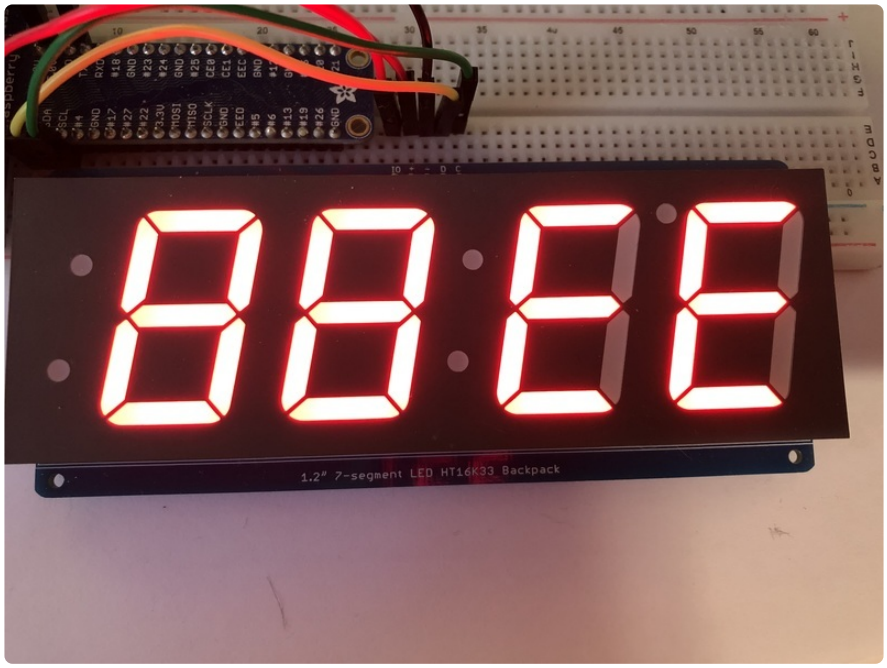
Setting Individual Segments

To set individual segments to turn on or off, you would use the set_digit_raw function to pass the digit that you want to change and the bitmask. This can be really useful for creating your own characters. The bitmask corresponds to the following diagram. The highest bit is not used, so an X represents that spot to indicate that.



The bitmask is a single 8-bit number that can be passed in as a single Hexidecimal, Decimal, or binary number. This will use a couple different methods to display `8E8E`:

```
display.set_digit_raw(0, 0xFF)
display.set_digit_raw(1, 0b11111111)
display.set_digit_raw(2, 0x79)
display.set_digit_raw(3, 0b01111001)
```



Filling all Segments

To fill the entire display, just use the fill() function and pass in either 0 or 1 depending on whether you want all segments off or on. For instance, if you wanted to set everything to on, you would use:

```
display.fill(1)
```

Scrolling Display Manually

If you want to scroll the displayed data to the left, you can use the `scroll()` function. You can pass in the number of places that you want to scroll. The right-most digit will remain unchanged and you will need to set that manually. After scrolling, you will need to call the show function. For example if you wanted to print an A and then scroll it over to spaces, you would do the following.

```
display.print("A")
display.scroll(2)
display[3] = " "
display.show()
```

Displaying the Colon

There are a couple of different ways to display a colon on the 7-segment display. The first and easiest way is to use the print function:

```
display.print("12:30")
```

The other way to control it is to access the colon with the colons property and set index 0 to `True` or `False`:

```
display.colons[0] = False
```

Setting the Left-Side Dots

There are a couple of different ways to set the left-side dots on the large 7-segment display. The first way is to use the colons property like above:

```
display.colons[1] = False
```

If you would like to set the dots individually, you can do that using the `top_left_dot` and `bottom_left_dot` properties and set them to `True` or `False`:

```
display.top_left_dot = True
display.bottom_left_dot = True
```

Setting the AM/PM Indicator

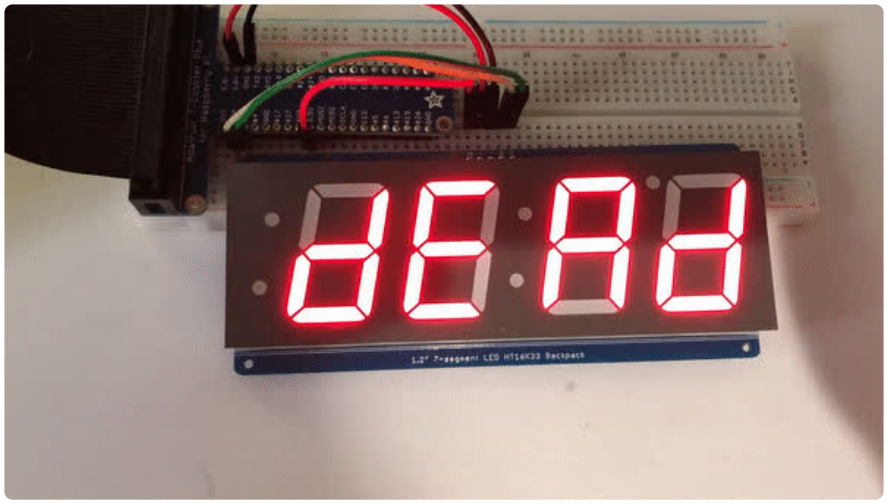
If you would like to set the upper-right dot, you can do this using the `ampm` property:

```
display.ampm = True
```

Displaying an Automatic Scrolling Marquee

To make displaying long text easier, we've added a marquee function. You just pass it the full string. Optionally, you can pass it the amount of delay between each character. This may be useful for displaying a phone number, words using only letters A-F, or other numeric data:

```
display.marquee('Deadbeef ')
```



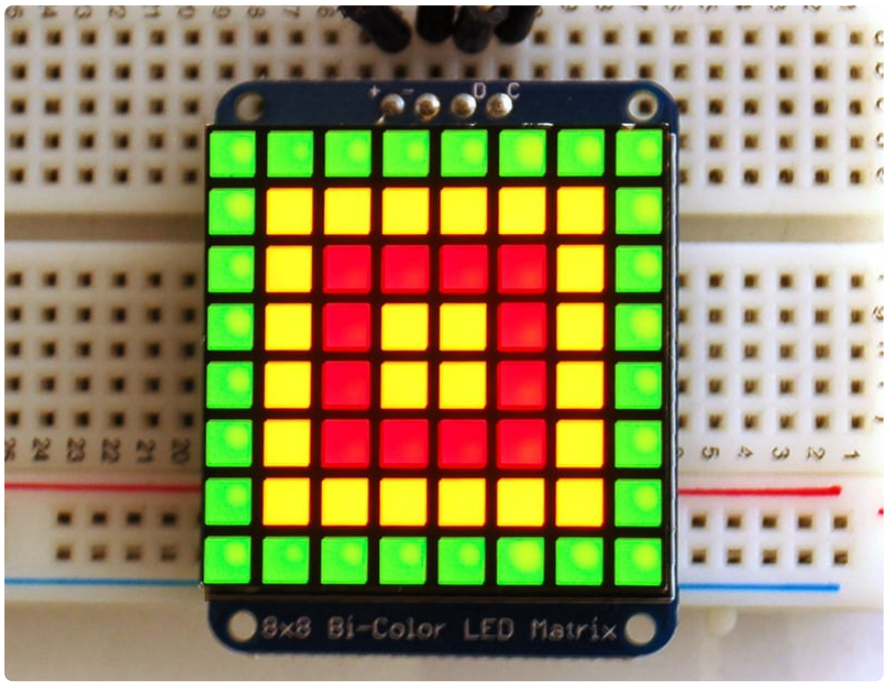
By default it is 0.25 seconds, but you can change this by providing a second parameter. You can optionally pass `False` for a third parameter if you would not like to have it loop. So if you wanted each character to display for half a second and didn't want it to loop, you would use the following:

```
display.marquee('Deadbeef', 0.5, False)
```

Bi-Color 8x8 Matrix

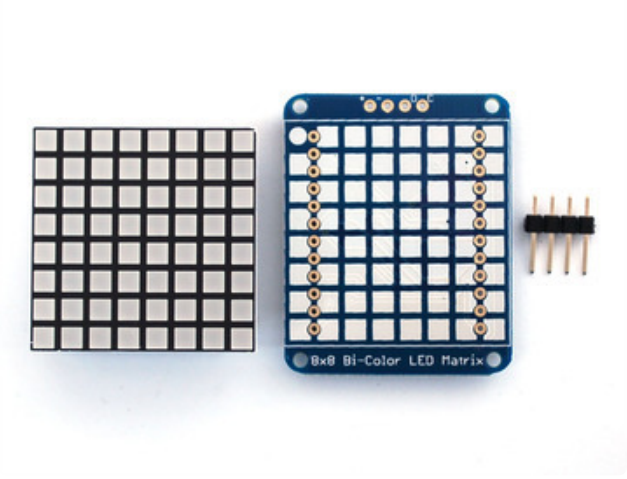
This version of the LED backpack is designed for these bright and colorful square=pixeled 8x8 matrices. They have 64 red and 64 green LEDs inside, for a total of 128 LEDs controlled as a 8x16 matrix. This backpack solves the annoyance of using 24 pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all 128 LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are three address select pins so you can select one of 8

addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.



Assembly

Pay close attention to the instructions for positioning the matrix. It must be oriented correctly to work and is almost impossible to remove it once it has been soldered to the backpack!



When you buy a pack from Adafruit, it comes with the fully tested and assembled backpack as well as a 8x8 matrix. You'll need to solder the matrix onto the backpack but its an easy task.

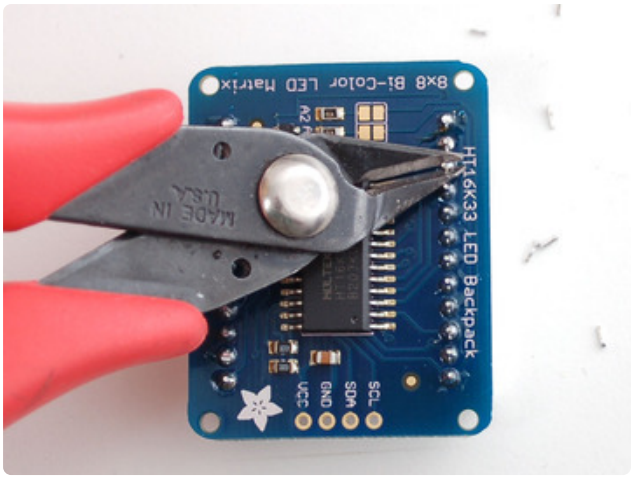
Remove the parts from packaging and place the LED matrix OVER the silkscreen side.

The matrix must be soldered on the correct orientation or it will not work! Check for the side of the matrix that has printing on it. Then look for the front of the PCB that has a circle instead of a square in the corner and line those up as shown on the left

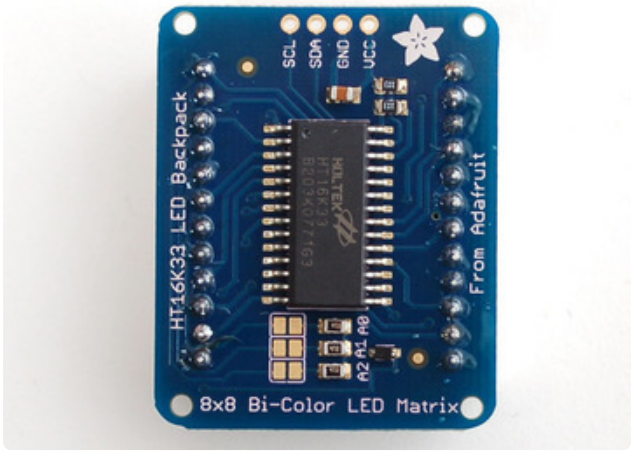
Do not solder the matrix over the chip on the back of the backpack - it will not work then!

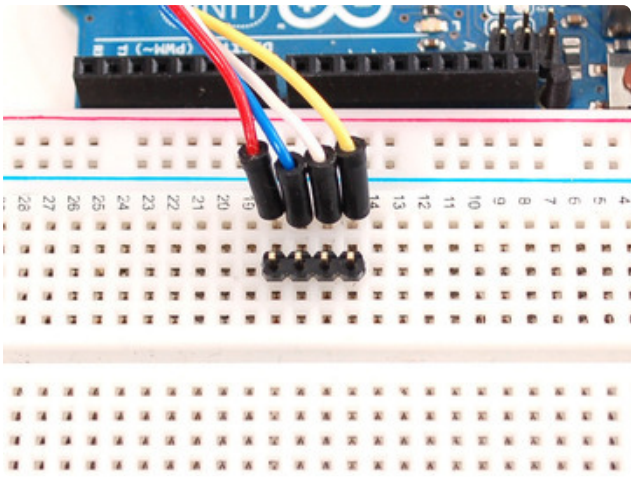
Turn the backpack over so its sitting flat on the matrix.

Solder all 24 pins.

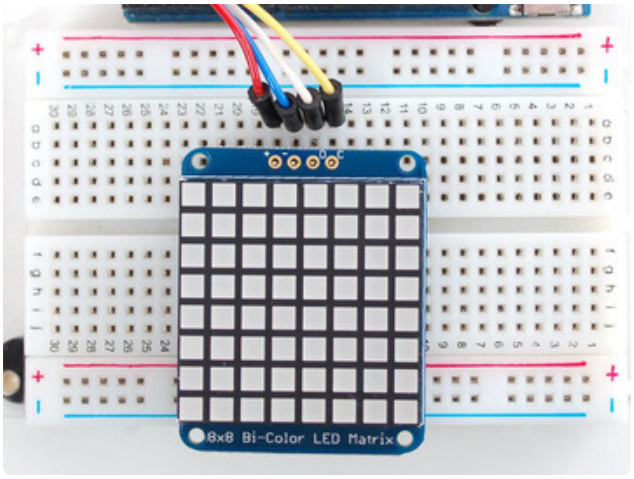


Clip the long pins

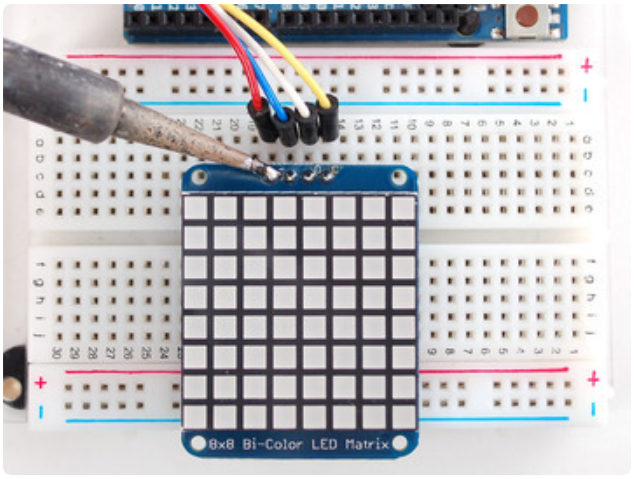




Now you're ready to wire it up to a microcontroller. We'll assume you want to use a 4pin header. You can also of course solder wires directly. Place a 4-pin piece of header with the LONG pins down into the breadboard.



Place the soldered backpack on top of the header.



Solder 'em!

Arduino Setup

You can use these with a 3.3v or 5v microcontroller. Just connect the VCC+ pin is the same voltage as the logic on your microcontroller.

We wrote a basic library to help you work with the bi-color 8x8 matrix backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

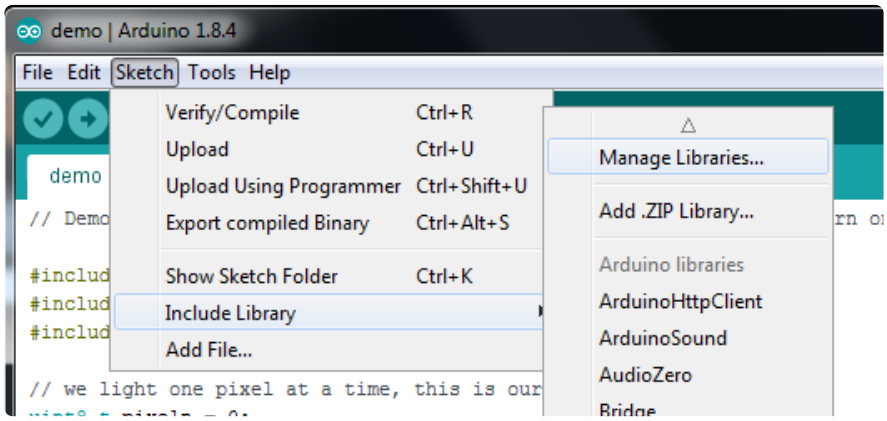
Wiring to the matrix is really easy

- Connect **CLK** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo its Digital #3, on the Mega its digital #21
- Connect **DAT** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo its Digital #2, on the Mega its digital #20
- Connect **GND** to common ground

- Connect **VCC**+ to power - 5V is best but 3V also seems to work for 3V microcontrollers.

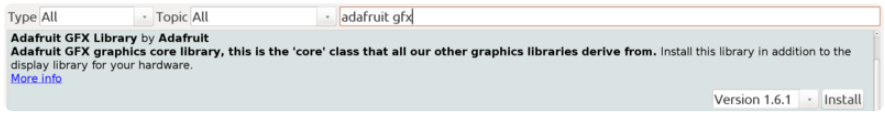
Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it

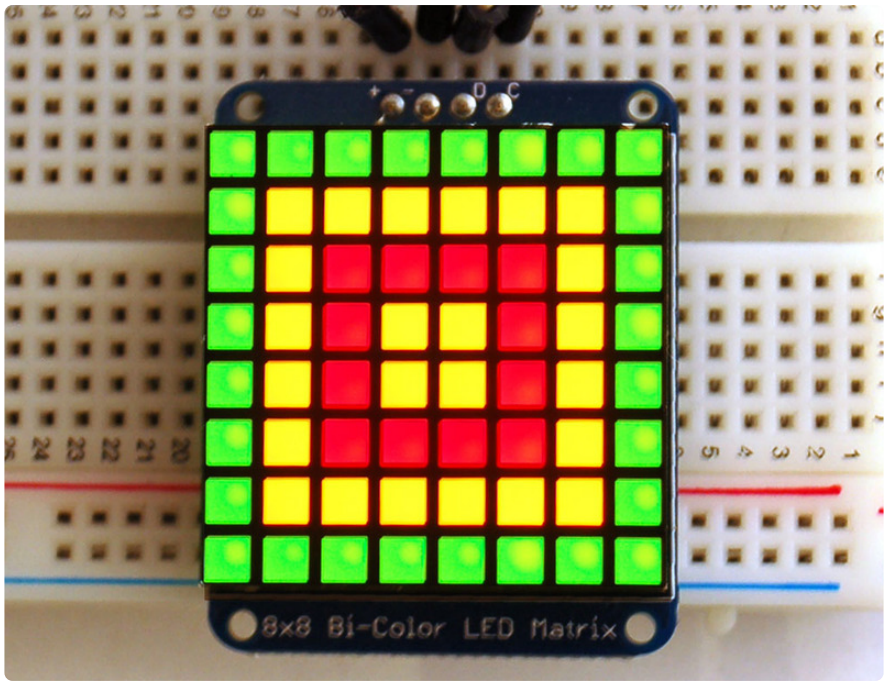
Search for the **Adafruit GFX** library and install it



If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

Once you've restarted you should be able to select the **File→Examples→Adafruit_LEDBackpack→bicolor88** example sketch. Upload it to your Arduino as usual. You should see a basic test program that goes through a bunch of different drawing routines.

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)

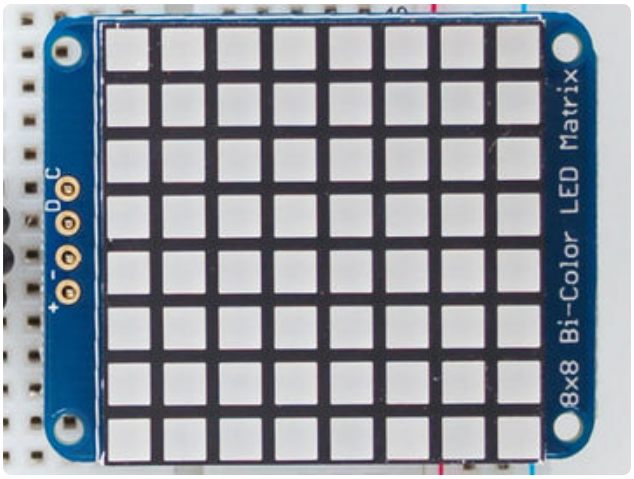


Once you're happy that the matrix works, you can write your own sketches. The 8x8 matrix supports everything the Adafruit GFX library - drawing pixels, lines, rectangles, circles, triangles, roundrects, and small bitmaps. For more details check out the GFX page which will detail all of the GFX routines (https://adafru.it/aPx).

All the drawing routines only change the display memory kept by the Arduino. Don't forget to call **writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the matrix:

- **setBrightness(brightness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.



The default orientation for graphics commands on this display places pixel (0,0) at the top-left when the header is at the left and Adafruit logo at the right. To use the matrix as shown above (header at top, logo at bottom), call matrix.setRotation(3) before issuing graphics commands.

CircuitPython Wiring and Setup

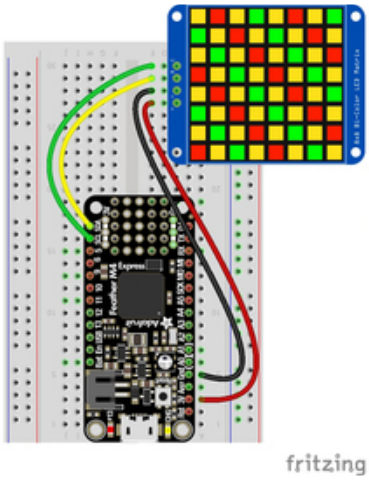
Wiring

It's easy to use LED Matrices with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the LED Matrix to your CircuitPython microcontroller board. First assemble your LED Matrix.

Connect the LED Matrix to your microcontroller board as shown below.



Microcontroller 3V to **LED Matrix VIN**
Microcontroller GND to **LED Matrix GND**
Microcontroller SCL to **LED Matrix SCL**
Microcontroller SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/ICo

Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

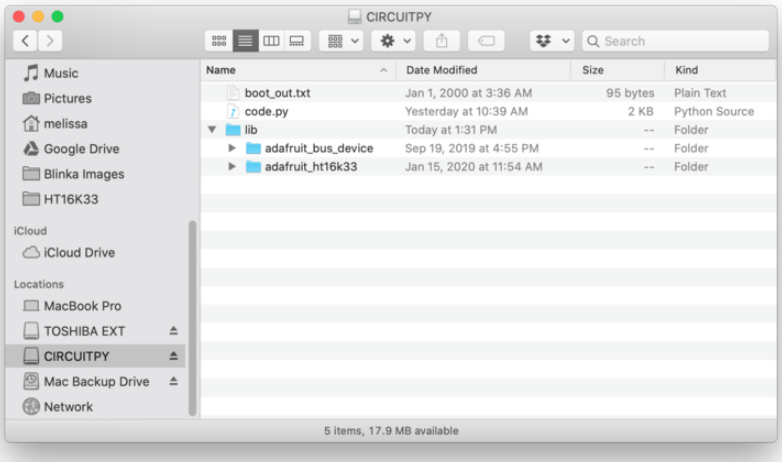
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

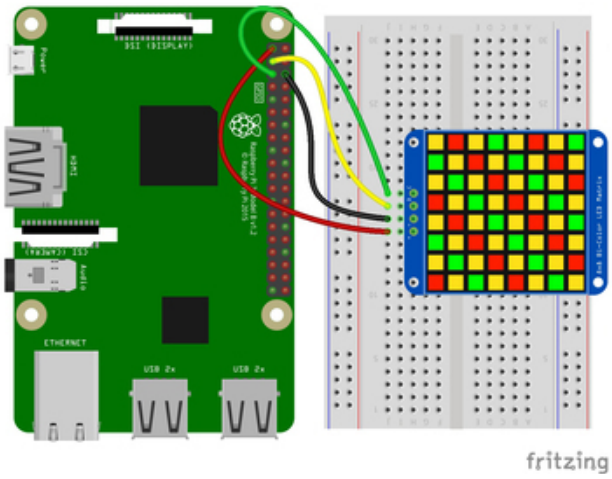
Wiring

It's easy to use LED Matrices with Python and the [Adafruit CircuitPython HT16K33 (https://adafru.it/u1E)](https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the LED Matrix to your Raspberry Pi. First assemble your LED Matrix.

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).

Connect the LED Matrix as shown below to your Raspberry Pi.



Raspberry Pi 3.3V to **LED Matrix VIN**
Raspberry Pi GND to **LED Matrix GND**
Raspberry Pi SCL to **LED Matrix SCL**
Raspberry Pi SDA to **LED Matrix SDA**

Download Fritzing Object

https://adafru.it/ICp

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

Pillow Library

We also need PIL, the Python Imaging Library, to allow using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- `sudo apt-get install python3-pil`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `Matrix8x8x2`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the matrix, you just pass `i2c` in.

> When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.matrix import Matrix8x8x2

i2c = board.I2C()
matrix = Matrix8x8x2(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **A0** pad, you would use `0x71` like this:

```
matrix = Matrix8x8x2(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual pixels. If can be adjusted in 1/4 increments **between**

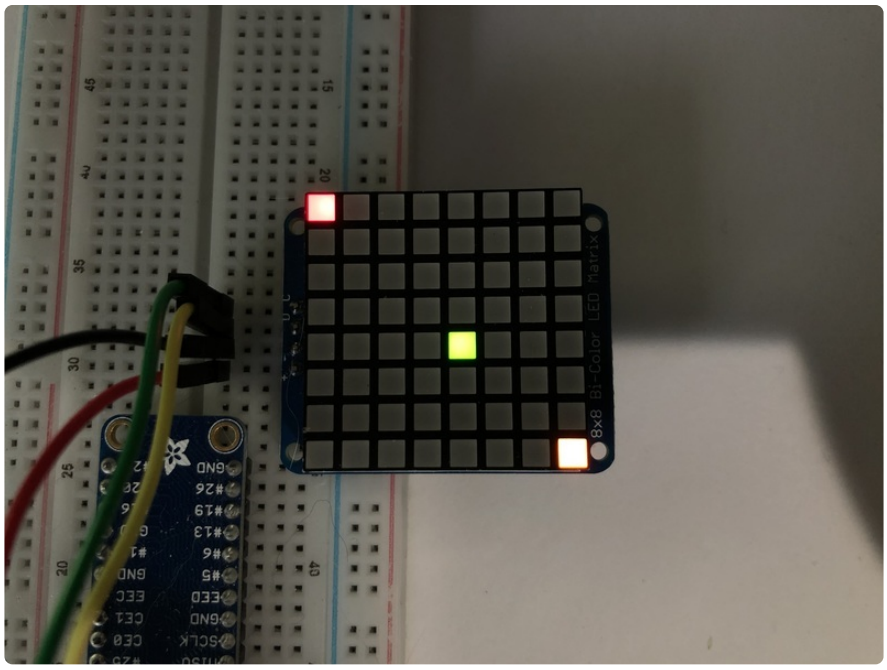
0 and 3 with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Setting Individual Pixels

To set individual pixels to specific colors, you simply treat the `matrix` object as a multidimensional list and set it to `matrix.LED_RED`, `matrix.LED_GREEN`, `matrix.LED_YELLOW` or `matrix.LED_OFF`.

```
matrix[0, 0] = matrix.LED_RED
matrix[4, 4] = matrix.LED_GREEN
matrix[7, 7] = matrix.LED_YELLOW
matrix[7, 0] = matrix.LED_OFF
```



Filling the Entire Matrix

To fill the entire matrix, just use the fill() function and pass in the color you want to set it to. For instance, if you wanted to set everything to green, you would use:

```
matrix.fill(matrix.LED_GREEN)
```

Shifting the Matrix

To shift the pixels on the matrix, there are 5 functions you can use. The main function, called shift(), is used to shift the pixels, up, down, left, right, or even diagonally. By passing a positive number, it will shift the pixels right/up and passing a negative number will shift them left/down. For instance:

```
matrix.shift(2, 0)    # shift pixels to the right by 2
matrix.shift(-1, 0)    # shift pixels to the left by 1
matrix.shift(0, -3)    # shift pixels down by 3
matrix.shift(-2, 2)    # shift pixels left by 2 and up by 2
```

You can pass `True` as a third parameter to loop all the pixels that get shifted off over to the other side.

```
matrix.shift(2, 0, True)    # loop pixels to the right by 2
matrix.shift(-1, 0, True)    # loop pixels to the left by 1
matrix.shift(0, -3, True)    # loop pixels down by 3
matrix.shift(-2, 2, True)    # loop pixels left by 2 and up by 2
```

Additionally, there are a few convenience functions that will shift the pixels by one. These can also be passed a value of `True` to loop the pixels.

```
matrix.shift_up()        # Shift pixels up
matrix.shift_left()        # Shift pixels left
matrix.shift_down()        # Shift pixels down
matrix.shift_right()        # Shift pixels right
matrix.shift_up(True)        # Loop pixels up
matrix.shift_left(True)    # Loop pixels left
matrix.shift_down(True)    # Loop pixels down
matrix.shift_right(True)    # Loop pixels right
```

Displaying an Image (Pillow Only)

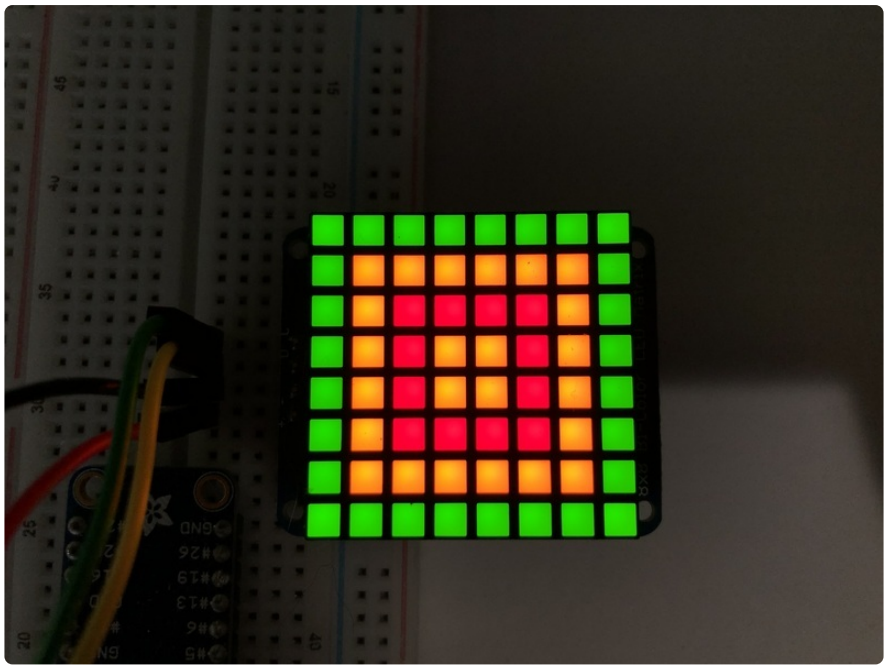
Additionally, when using with the Raspberry Pi, you can use the Pillow library to display an image to the Matrix. The image will need to be the same exact size as the Matrix and should include pure Green, Red, or Yellow. Anything else will be considered to be **off**. In this case, it should be **8x8** pixels. As an example, you can save the image below as **myimage.png**.

Download Image

https://adafru.it/ICQ

Then if you want to display the image called **myimage.png**, you would use something like this:

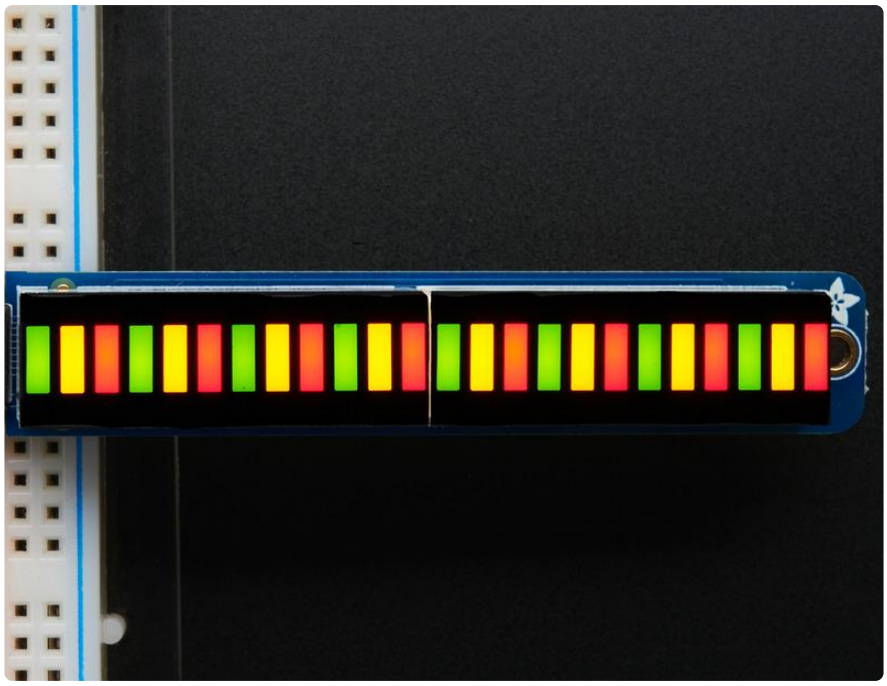
```
import board
from PIL import Image
from adafruit_ht16k33 import matrix

matrix = matrix.Matrix8x8x2(board.I2C())

image = Image.open("myimage.png")
matrix.image(image)
```



Bi-Color 24 Bargraph

This version of the LED backpack is designed for these bright and colorful bi-color bargraph modules. Each module has 12 red and 12 green LEDs inside, for a total of 24 LEDs controlled as a 1x12 matrix. We put two modules on each backpack for a 24-bar long bargraph (48 total LEDs).

This backpack solves the annoyance of using lots of pins or a bunch of chips by having an I2C constant-current matrix controller sit neatly on the back of the PCB. The controller chip takes care of everything, drawing all 48 LEDs in the background. All you have to do is write data to it using the 2-pin I2C interface. There are three address select pins so you can select one of 8 addresses to control up to 8 of these on a single 2-pin I2C bus (as well as whatever other I2C chips or sensors you like). The driver chip can 'dim' the entire display from 1/16 brightness up to full brightness in 1/16th steps. It cannot dim individual LEDs, only the entire display at once.



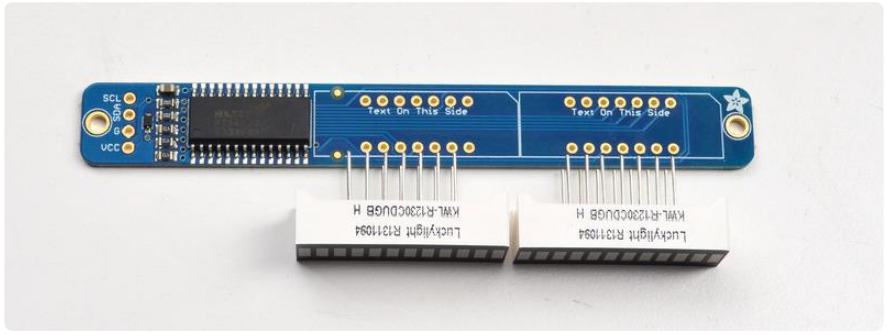
Assembly

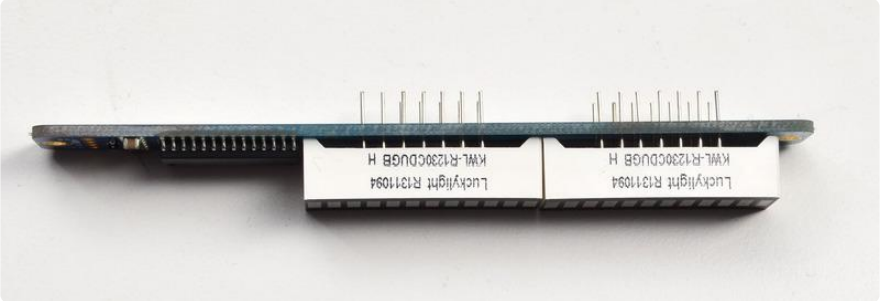
> Pay close attention to the instructions for positioning the bargraphs. They must be oriented correctly to work and is almost impossible to remove them once soldered to the backpack!

Remove the parts from packaging and place the LED bargraphs over the outlines on the top of the PCB.

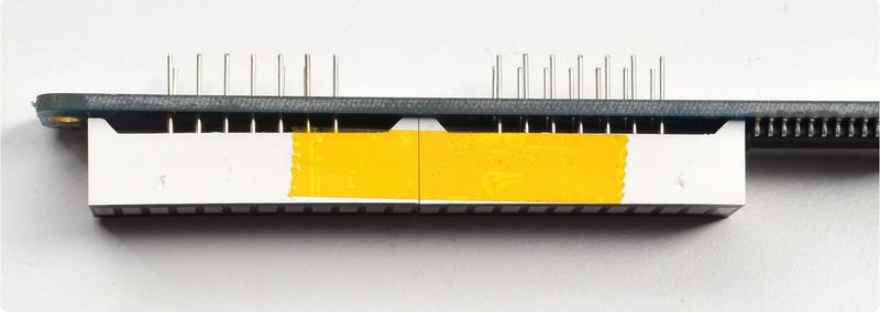
The bargraph must be soldered on the correct orientation or it will not work! Check for the side of the bargraph that has printing on it. Then look for the outline on the PCB that has "Text on this side" marked!

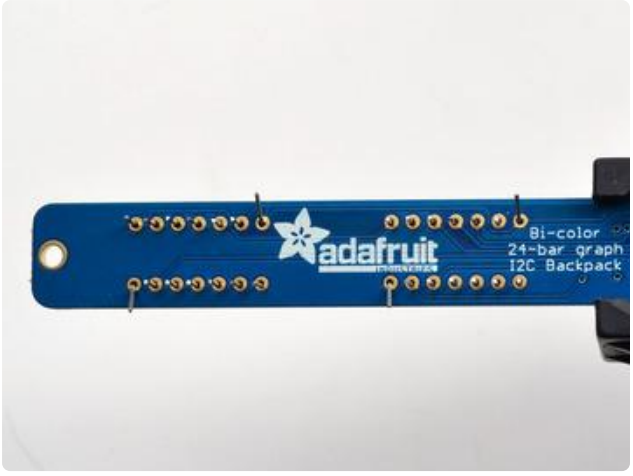
Do not solder the matrix onto the back of the PCB, it won't work either!



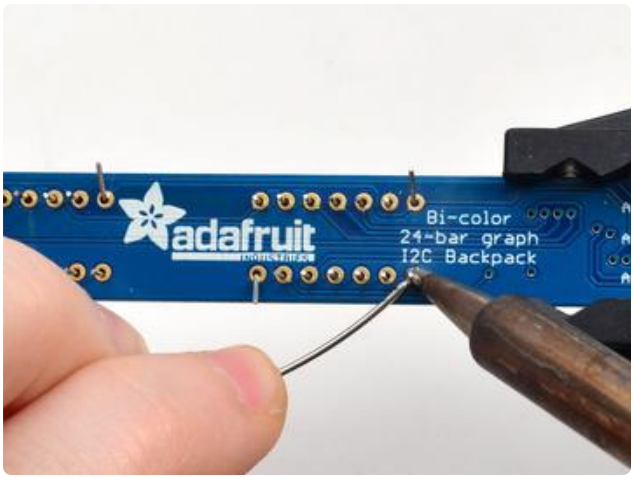


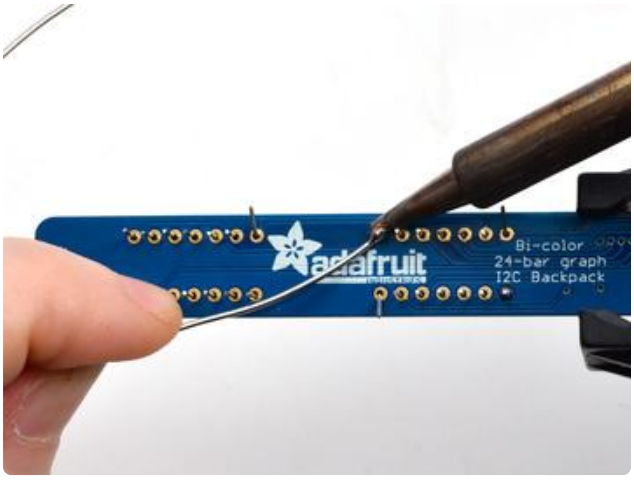
To keep the two bargraphs lined up nicely, you can use a little masking or scotch tape on the bargraph modules, tape them so they are in a straight line. There is a little play during soldering so if you don't do this the two modules may not be in a perfect line.



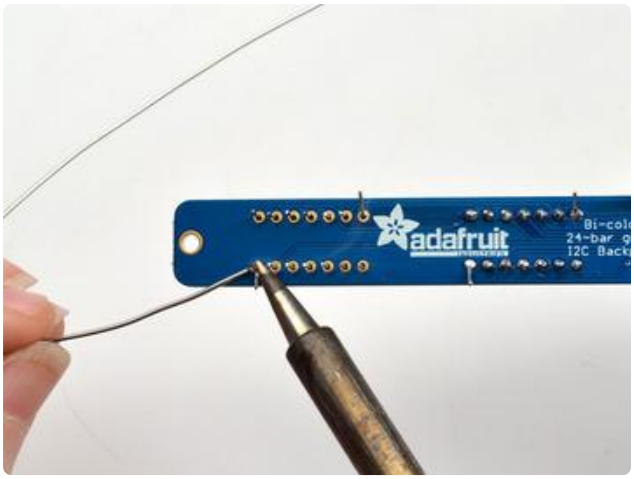


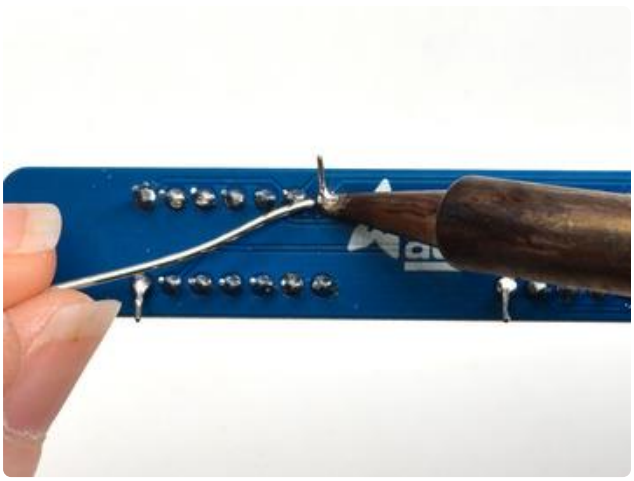
Turn over the PCB and bend opposite-corner pins of the modules out so that the modules are fixed in place against the PCB. Now is a good time to do a last check that you oriented the modules the right way!

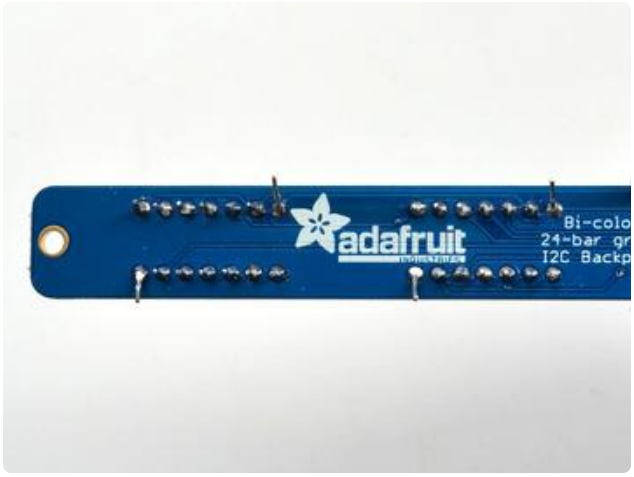




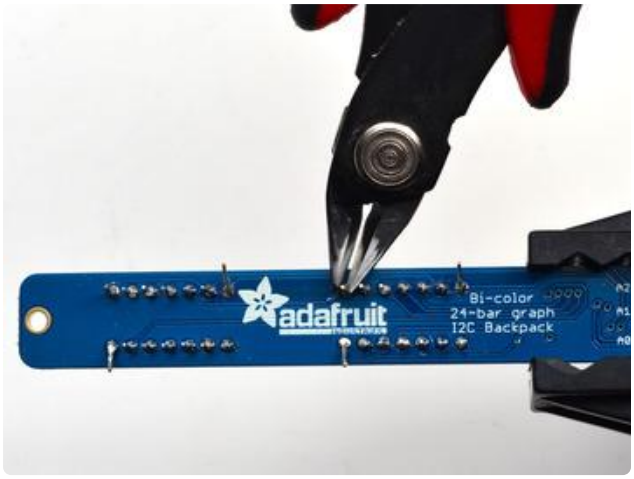
Solder all the module pins in!



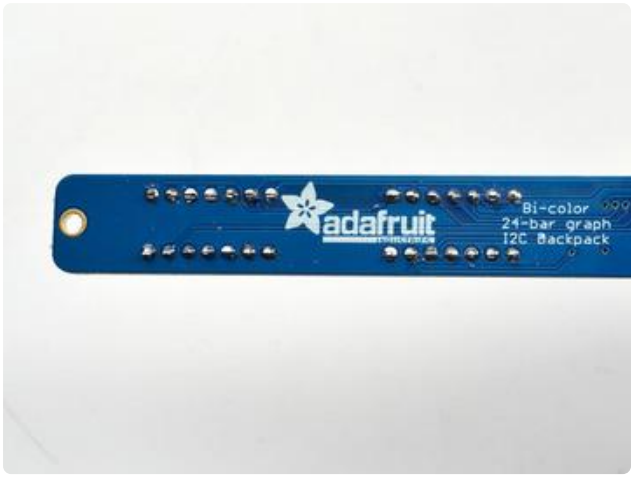




OK nice work!



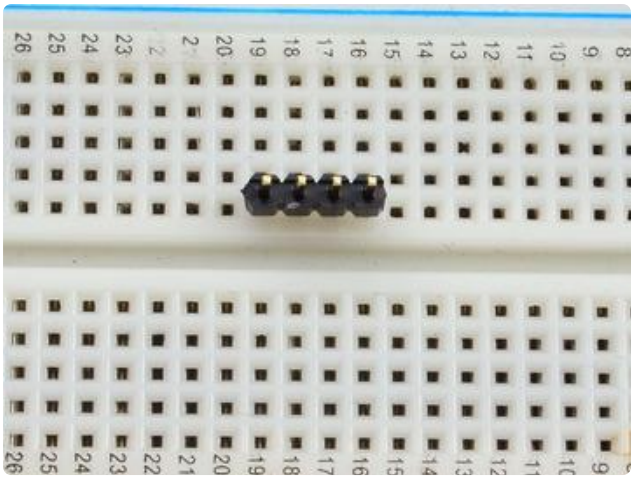
Once soldered, clip each pin. They're quite short and the pins are thicker than usual, so do this over/inside a trash bin so that the pins don't fly off and it you or your pets.



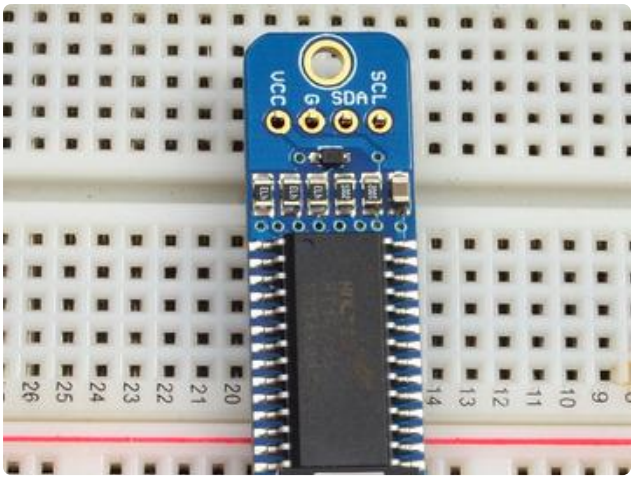
Everything should be neat and clipped, you're done!

Soldering on breadboard pins

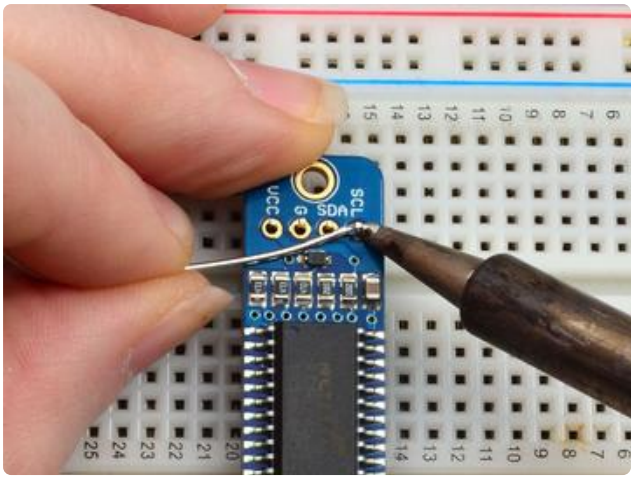
This is an optional step - you only need to do this step if you're planning on using the bargraph in a breadboard. Chances are you may want to solder wires directly to the pads instead, so you can mount the bargraph elsewhere. Anyhow, skip this step if its not for you!

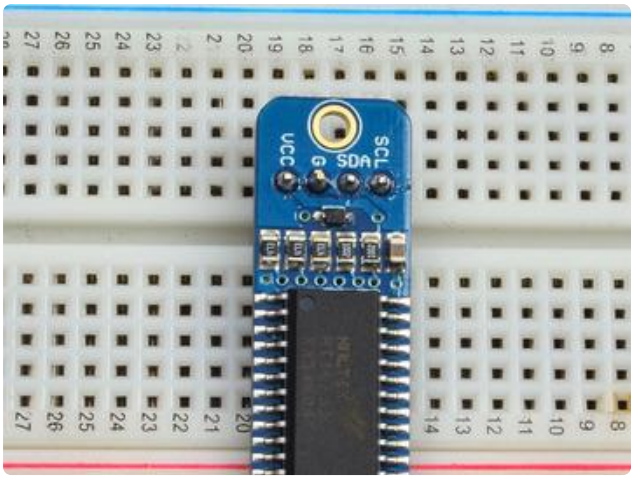


Break off a piece of male header, 4 pins long. Plug the **long ends** into a solderless breadboard.



Place the PCB on top. you may need to support it a little since its quite long.





Solder these 4 pins too, since you're good at it now this should be easy.

Arduino Wiring and Setup

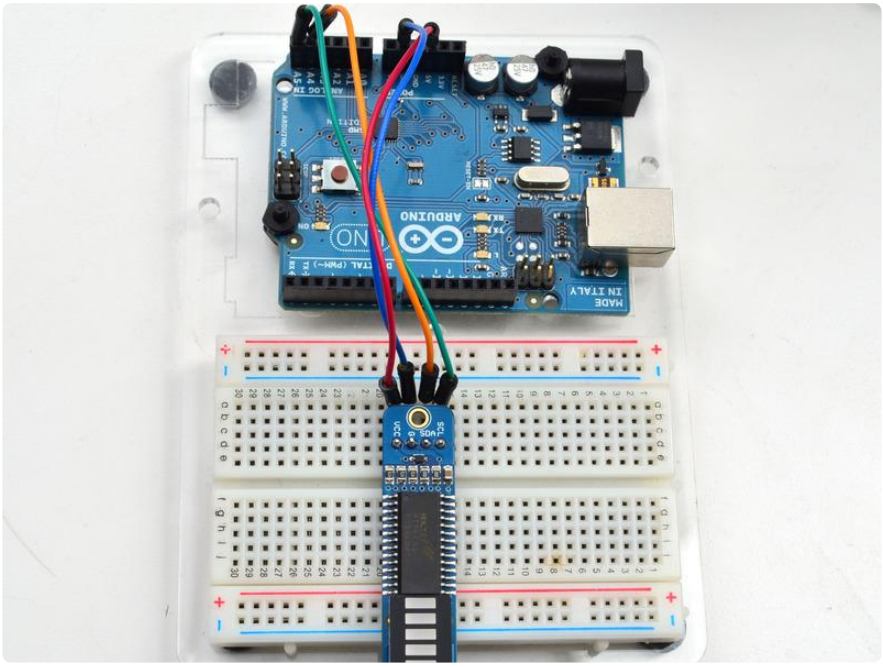
You can use these with a 3.3v or 5v microcontroller. Just connect the VCC+ pin is the same voltage as the logic on your microcontroller.

We wrote a basic library to help you work with the bi-color bargraph backpack. The library is written for the Arduino and will work with any Arduino as it just uses the I2C pins. The code is very portable and can be easily adapted to any I2C-capable micro.

Wiring to the bargraph is really easy

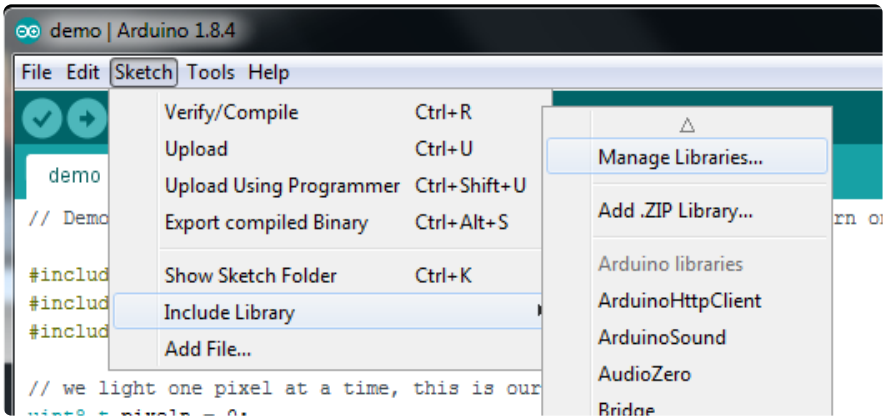
- Connect **SCL** to the I2C clock - on Arduino UNO thats Analog #5 (or SCL), on the Leonardo its Digital #3, on the Mega its digital #21
- Connect **SDA** to the I2C data - on Arduino UNO thats Analog #4 (or SDA), on the Leonardo its Digital #2, on the Mega its digital #20
- Connect **GND** to common ground

- Connect **VCC**+ to power - 5V is best but 3V also seems to work for 3V microcontrollers.



Next, download the **Adafruit LED Backpack** library and the **Adafruit GFX** library from the Arduino library manager.

Open up the Arduino library manager:



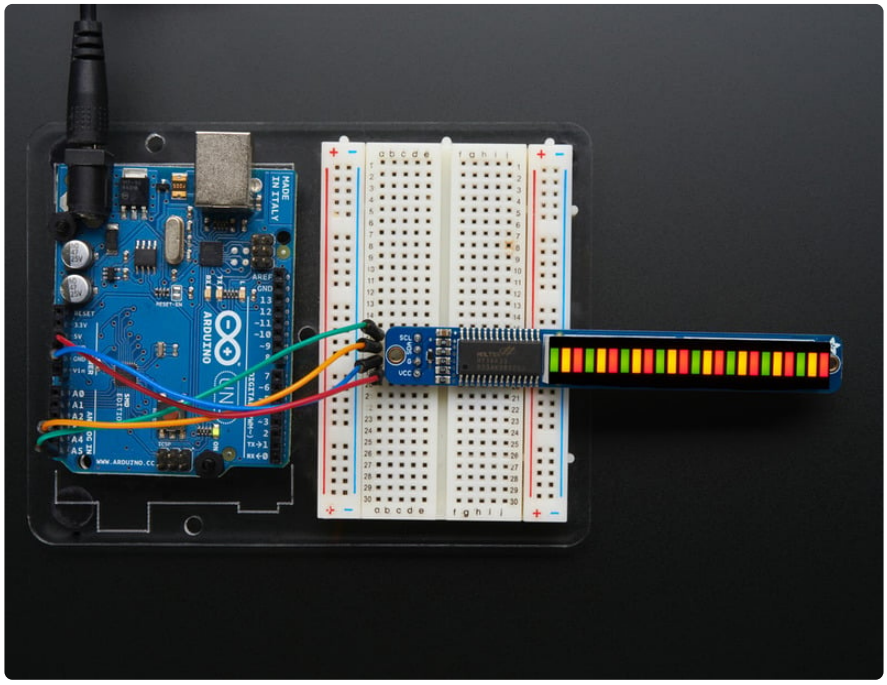
Search for the **Adafruit LED Backpack** library and install it

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

If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

Once you've restarted you should be able to select the **File→Examples→Adafruit_LEDBackpack→bargraph24** example sketch. Upload it to your Arduino as usual. You should see a basic test program that tests all the LEDs with different colors.

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)



Using the library interface is very easy. Start by creating the object with

```
Adafruit_24bargraph bar = Adafruit_24bargraph();
```

you can name it whatever you want, not just **bar**

Then initialize it with

```
bar.begin(0x70); // pass in the address
```

You can init with any address from 0x70 to 0x77, just make sure you solder in the matching solder jumpers!

Finally, write to the bargraph with

bar.setBar(lednumber, ledcolor);

Where lednumber is 0 thru 23. ledcolor can be **LED_RED, LED_YELLOW, LED_GREEN** or **LED_OFF**

The drawing routines only change the display memory kept by the Arduino. Don't forget to call **bar.writeDisplay()** after drawing to 'save' the memory out to the matrix via I2C.

There are also a few small routines that are special to the matrix:

- **setBrightness(brightness)**- will let you change the overall brightness of the entire display. 0 is least bright, 15 is brightest and is what is initialized by the display when you start
- **blinkRate(rate)** - You can blink the entire display. 0 is no blinking. 1, 2 or 3 is for display blinking.

CircuitPython Wiring and Setup

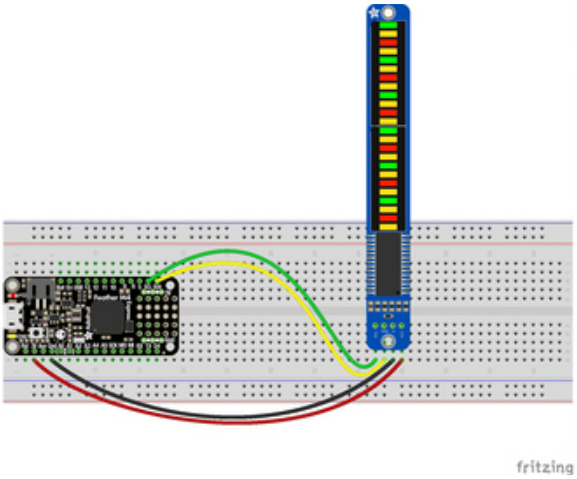
Wiring

It's easy to use LED Matrices with CircuitPython and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

We'll cover how to wire the Bargraph to your CircuitPython microcontroller board. First assemble your Bargraph.

Connect the Bargraph to your microcontroller board as shown below.



Microcontroller 3V to **Bargraph VIN**
Microcontroller GND to **Bargraph GND**
Microcontroller SCL to **Bargraph SCL**
Microcontroller SDA to **Bargraph SDA**

Download Fritzing Object

https://adafru.it/IfA

Library Setup

To use the LED backpack with your Adafruit CircuitPython (https://adafru.it/BIM) board you'll need to install the Adafruit_CircuitPython_HT16K33 (https://adafru.it/u1E) library on your board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board. Next you'll need to install the necessary libraries to use the hardware--read below and carefully follow the referenced steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx).

Bundle Install

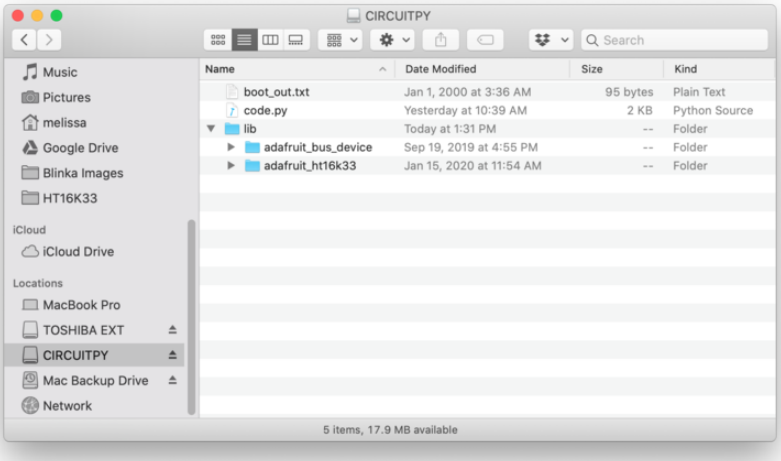
For express boards that have extra flash storage, like the Feather/Metro M0 express and Circuit Playground express, you can easily install the necessary libraries with Adafruit's CircuitPython bundle (https://adafru.it/zdx). This is an all-in-one package that includes the necessary libraries to use the LED backpack display with CircuitPython. For details on installing the bundle, read about CircuitPython Libraries (https://adafru.it/ABU).

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries (https://adafru.it/ABU) from the bundle:

- **adafruit_ht16k33**
- **adafruit_bus_device**

If your board supports USB mass storage, like the M0-based boards, then simply drag the files to the board's file system. **Note on boards without external SPI flash, like a Feather M0 or Trinket/Gemma M0, you might run into issues on Mac OSX with hidden files taking up too much space when drag and drop copying,** see this page for a workaround **(https://adafru.it/u1d).**

Before continuing make sure your board's **lib** folder or root filesystem has at least the **adafruit_ht16k33** and **adafruit_bus_device** folders/modules copied over.



Python Wiring and Setup

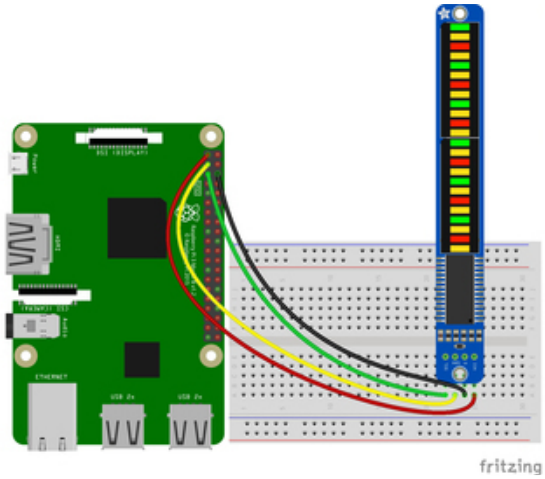
Wiring

It's easy to use LED Matrices with Python and the Adafruit CircuitPython HT16K33 (https://adafru.it/u1E) library. This library allows you to easily write Python code to control the display.

We'll cover how to wire the Bargraph to your Raspberry Pi. First assemble your Bargraph.

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).

Connect the Bargraph as shown below to your Raspberry Pi.



Raspberry Pi 3.3V to **Bargraph VIN**
Raspberry Pi GND to **Bargraph GND**
Raspberry Pi SCL to **Bargraph SCL**
Raspberry Pi SDA to **Bargraph SDA**

Download Fritzing Object

https://adafru.it/IfB

Setup

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)!

Python Installation of HT16K33 Library

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

- `pip3 install adafruit-circuitpython-ht16k33`

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!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

That's it. You should be ready to go.

CircuitPython and Python Usage

The following section will show how to control the LED backpack from the board's Python prompt / REPL. You'll walk through how to control the LED display and learn how to use the CircuitPython module built for the display.

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

Initialization

First you'll need to initialize the I2C bus for your board. It's really easy, first import the necessary modules. In this case, we'll use `board` and `BiColor24`.

Then just use `board.I2C()` to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board).

Then to initialize the bargraph, you just pass `i2c` in.

When using the STEMMA QT port, some board may have an alternate I2C such as board.STEMMA_I2C().

```
import board
from adafruit_ht16k33.bargraph import Bicolor24

i2c = board.I2C()
bc24 = Bicolor24(i2c)
```

If you bridged the address pads on the back of the display, you could pass in the address. The addresses for the HT16K33 can range between 0x70 and 0x77 depending on which pads you have bridged, with 0x70 being used if you haven't bridged any of them. For instance, if you bridge only the **a0** pad, you would use `0x71` like this:

```
bc24 = Bicolor24(i2c, address=0x71)
```

Setting the Brightness

You can set the brightness of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/16 increments **between 0 and 1.0** with 1.0 being the brightest. So to set the display to half brightness, you would use the following:

```
display.brightness = 0.5
```

Setting the Blink Rate

You can set the blink rate of the display, but changing it will set the brightness of the entire display and not individual segments. If can be adjusted in 1/4 increments **between 0 and 3** with 3 being the fastest blinking. So to set the display to blink at full speed, you would use the following:

```
display.blink_rate = 3
```

Setting Individual Bars

To set individual bars to specific colors, you simply treat the `bc24` object as a list and set it to `bc24.LED_RED`, `bc24.LED_GREEN`, `bc24.LED_YELLOW` or `bc24.LED_OFF`.

```
bc24[0] = bc24.LED_RED
bc24[1] = bc24.LED_GREEN
bc24[2] = bc24.LED_YELLOW
bc24[3] = bc24.LED_OFF
```

Filling the Entire Bargraph

To fill the entire bargraph, just use the fill() function and pass in the color you want to set it to. For instance, if you wanted to set everything to green, you would use:

```
bc24.fill(bc24.LED_GREEN)
```

Connecting Multiple Backpacks

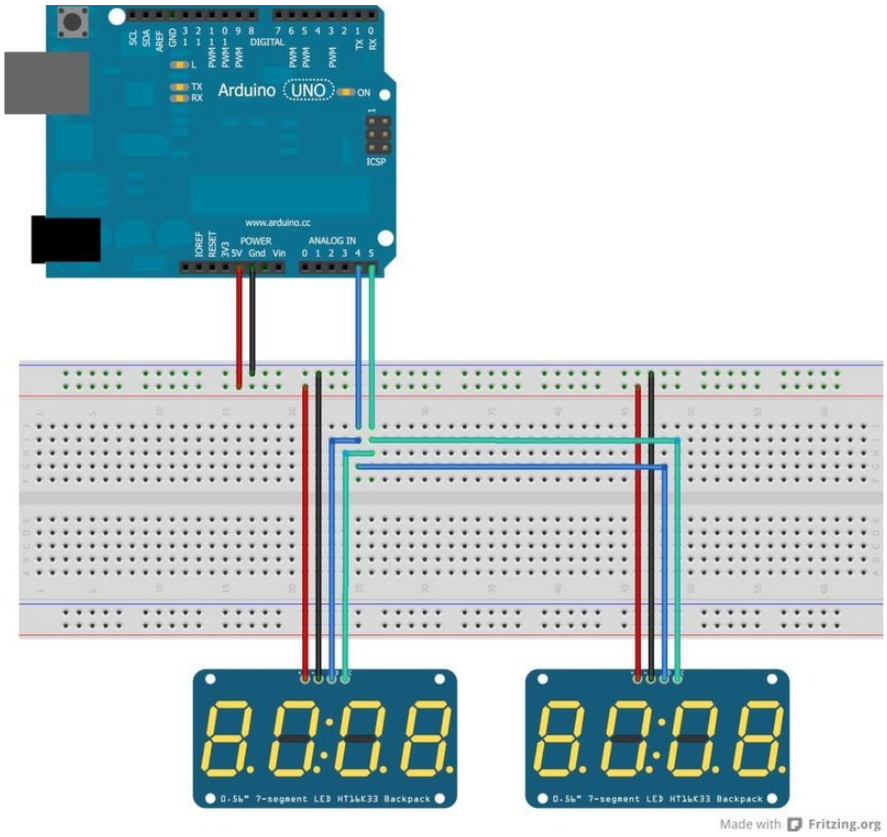
The coolest part about the I2C backpacks is that you can connect more than one using just the same 2 pins. This opens possibilities for [all kinds of multi-display projects](https://adafru.it/aQt).

[For a project that shows this is practice, check out this page](https://adafru.it/aQF) on animating multiple LED backpacks

Wire it Up

To connect another backpack to your project, just wire it in parallel with the first one as in the diagram below.

If your backpack has 5 connectors, just wire all 5 in parallel.



Configure the Address

For each backpack you add, you need to configure a different I2C address. You can keep adding backpacks in the same way until you run out of addresses. See the next page for how to configure the address on your backpack.

Changing I2C Address

The HT16K33 driver chip on these LED backpacks 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 2 or 3 address adjust pins, so that the address can be changed! The mini 0.8" 8x8 matrix backpack has 2 address adjust pins. The 1.2" 8x8, bi-color 8x8, bi-color bargraph and 4 x 7-segment backpacks have 3 address adjust pins.

That means that you can set the backpacks to these addresses:

- **Mini 0.8" 8x8:** 0x70, 0x71, 0x72, 0x73
- **Small 1.2" 8x8:** 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77
- **4 x 7-segment:** 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77
- **Bi-color 1.2" 8x8:** 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77

- **Bi-color 24-bargraph:** 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77

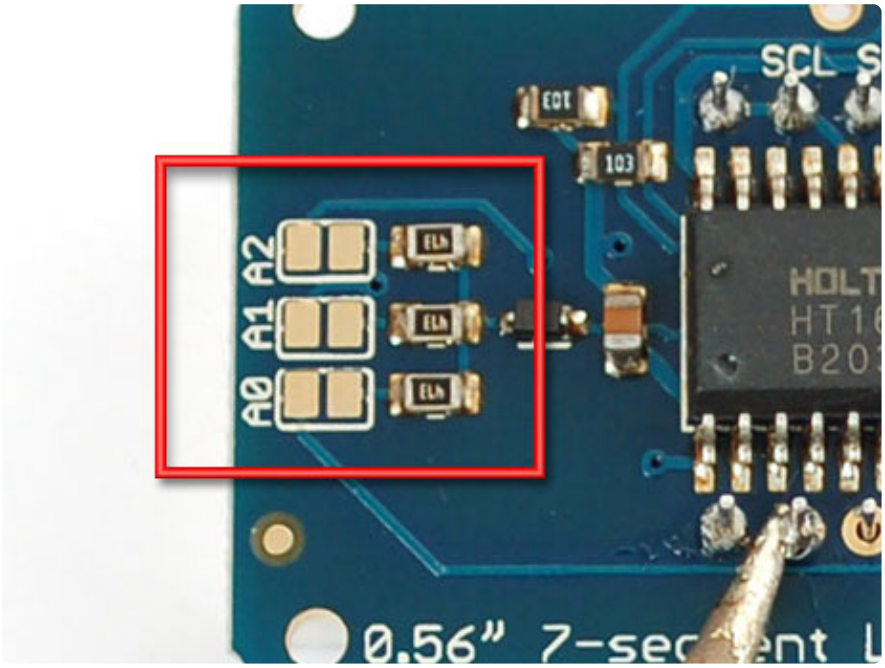
You can mix-and-match matrices, as long as each one has a unique address!

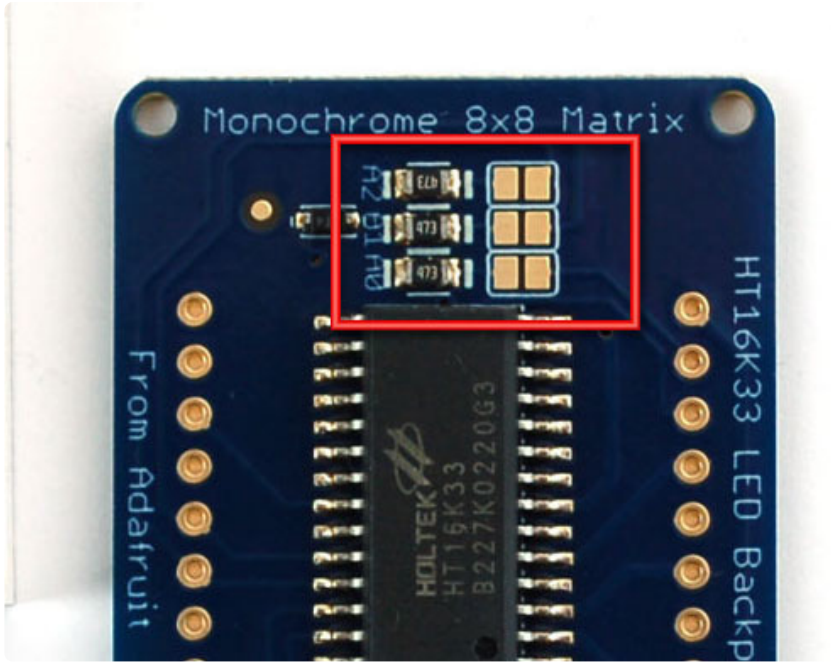
Changing Addresses

You can change the address of a backpack very easily. Look on the back to find the two or 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**

A2 does not appear on the mini 0.8" 8x8 matrix, so you cannot set the address higher than 0x73

On the 1.2" 8x8 backpacks, the labels for A1 and A2 are swapped! Sorry about that!





Changing the address in your code

Once you've adjusted the address on the backpack, you'll also want to adjust the address in the code!
For the Arduino library we wrote, its simple. For example, lets say you want to have two seven-segment matrices. One is set to address 0x70 and the other is set to 0x71. Find this code in the example

```
Adafruit_7segment matrix = Adafruit_7segment();

void setup() {
  Serial.begin(9600);
  Serial.println("7 Segment Backpack Test");

  matrix.begin(0x70);
}
```

And change it to this:

```
Adafruit_7segment matrix1 = Adafruit_7segment();
Adafruit_7segment matrix2 = Adafruit_7segment();

void setup() {
  Serial.begin(9600);
  Serial.println("Double 7 Segment Backpack Test");

  matrix1.begin(0x70);
  matrix2.begin(0x71);
}
```

That is, instantiate two matrix objects. Then one is called with **begin(0x70)** and the other is called with **begin(0x71)**. Each one can be used individually. If you need more

matrices, just instantiate more objects at the top and **begin()** each one with the unique i2c address.

F.A.Q.

I want to use these modules with other non-Arduino, how can I port the code?

The best way to get up and running is to read the HT16K33 driver datasheet available at http://learn.adafruit.com/adafruit-led-backpack/downloads (https://adafru.it/aMx) - the backpacks all use this chip to do all the LED driving. You can cross-reference this document with the Arduino library code to adapt it to your platform. Any microcontroller that has I2C host support should be able to drive the backpacks but we only provide Arduino example code at this time

I'd like to use these backpacks with Python / Linux (e.g. a Raspberry Pi)

You're in luck! We have a full tutorial here that covers using the 7-segment and 8x8 matrices on a Pi with Python code -> http://learn.adafruit.com/matrix-7-segment-led-backpack-with-the-raspberry-pi (https://adafru.it/aPj)

I am having strange problems when combining Adafruit Motor Shield/ Servo Shield (PCA9685 based) with the Adafruit LED Matrix/7Seg Backpacks

We are not sure why this occurs but there is an address collision even though the address are different! Set the backpacks to address 0x71 or anything other than the default 0x70 to make the issue go away

What is the current draw of the backpacks?

It depends on how many LEDs you have lit at once!

But a rough estimation is 20 milliamps per segment. Note that segments are multiplexed per row so that means

- 8x8 Mono Matrix (8 rows)= 8 x 20mA = 160mA max
- 7-segment backpacks (7 segments + 1 dot) = 8 x 20mA = 160 mA
- Alphanumeric (14 segments) = 14 x 20mA = 280mA
- Bi-color 8x8 and 8x16 matrix (16 rows) = 8 x 16 = 320mA

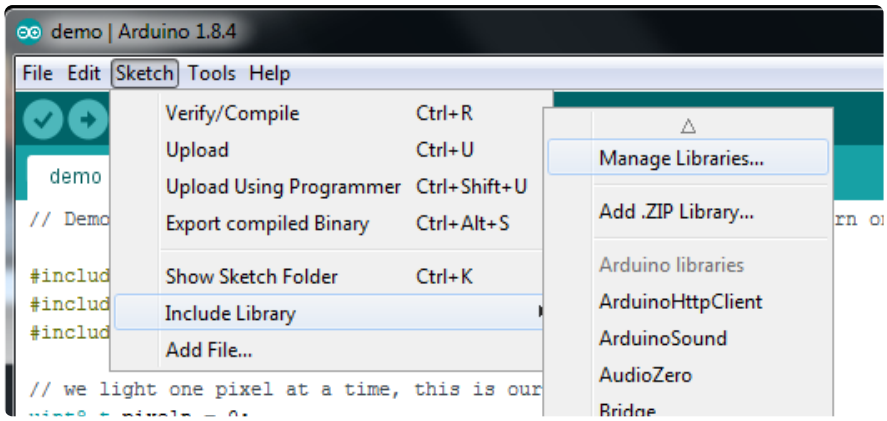
But again, this is maximum and assumes all digits and all segments are lit up! Your average use may be 1/10 to 1/2 of this amount

Downloads

Software

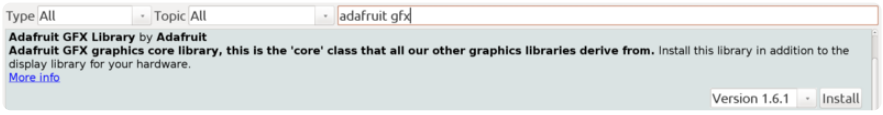
You'll need to install the following libraries to use any of the LED backpacks.

Open up the Arduino library manager:



Search for the **Adafruit LED Backpack** library and install it

Search for the **Adafruit GFX** 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)

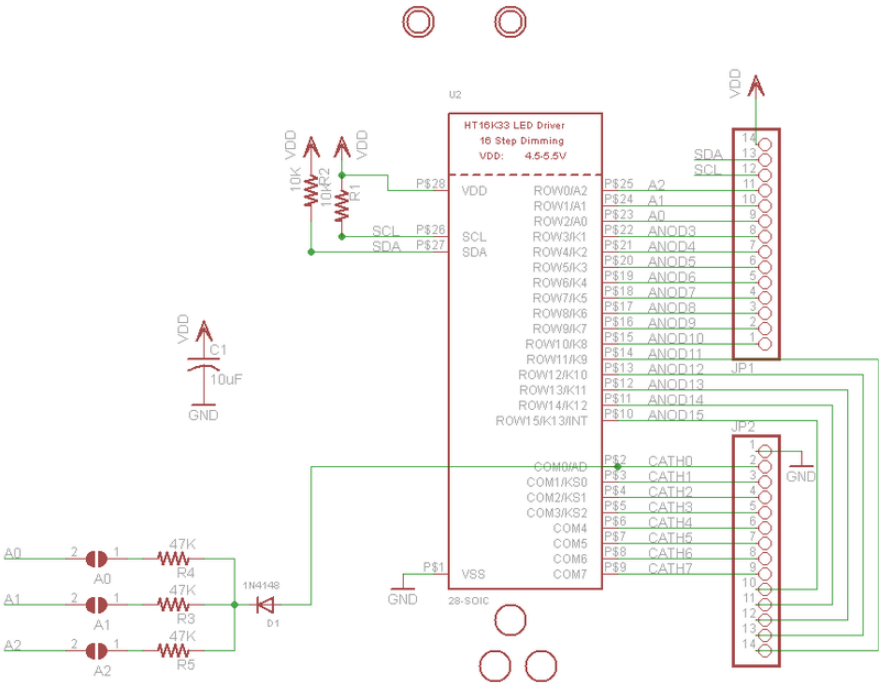
Files

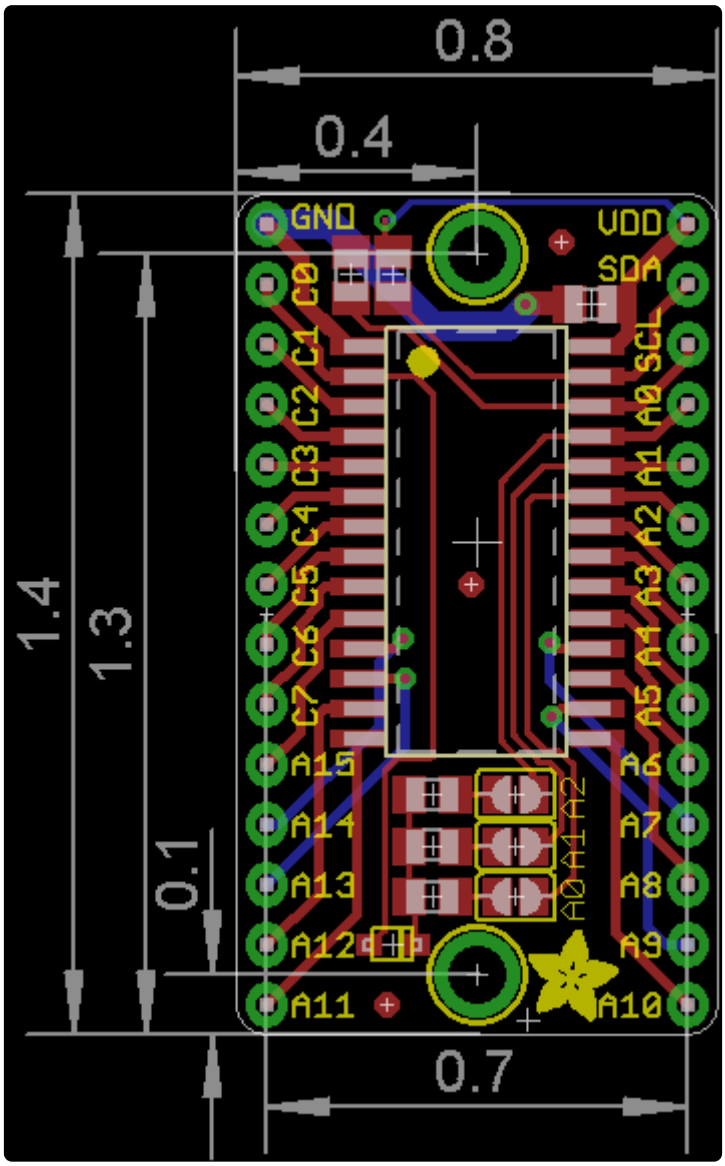
- Fritzing objects in Adafruit Fritzing library (https://adafru.it/aP3)
- EagleCAD PCB files for all backpacks in GitHub (https://adafru.it/aLJ)
- Alphanumeric STEMMA QT Display 3D models on GitHub (https://adafru.it/19Bc)

- The backpacks all use the HT16K33 chip solely for LED driving (https://adafru.it/aMy) - the mini 8x8's use the 24 pin version and the others use the 28 pin vesion

HT16K33 8x16 LED Backpack Breakout

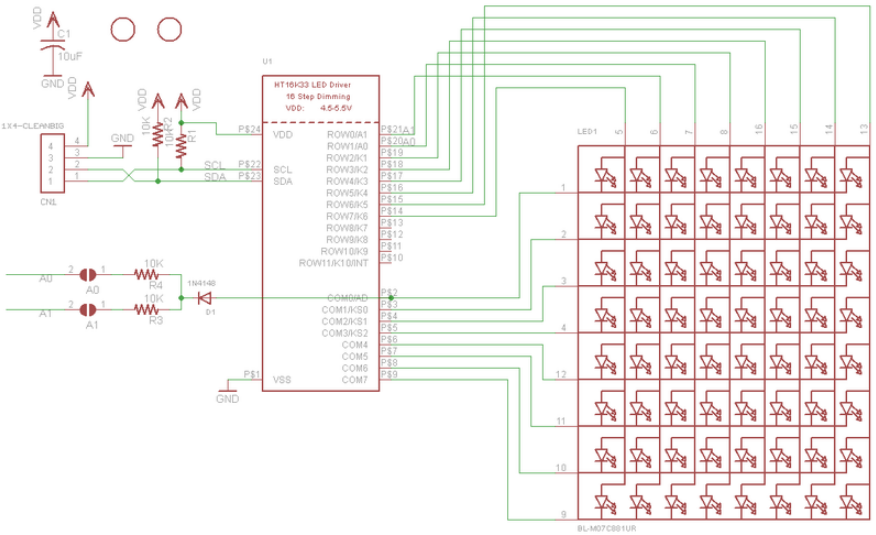
Schematic & fabrication print

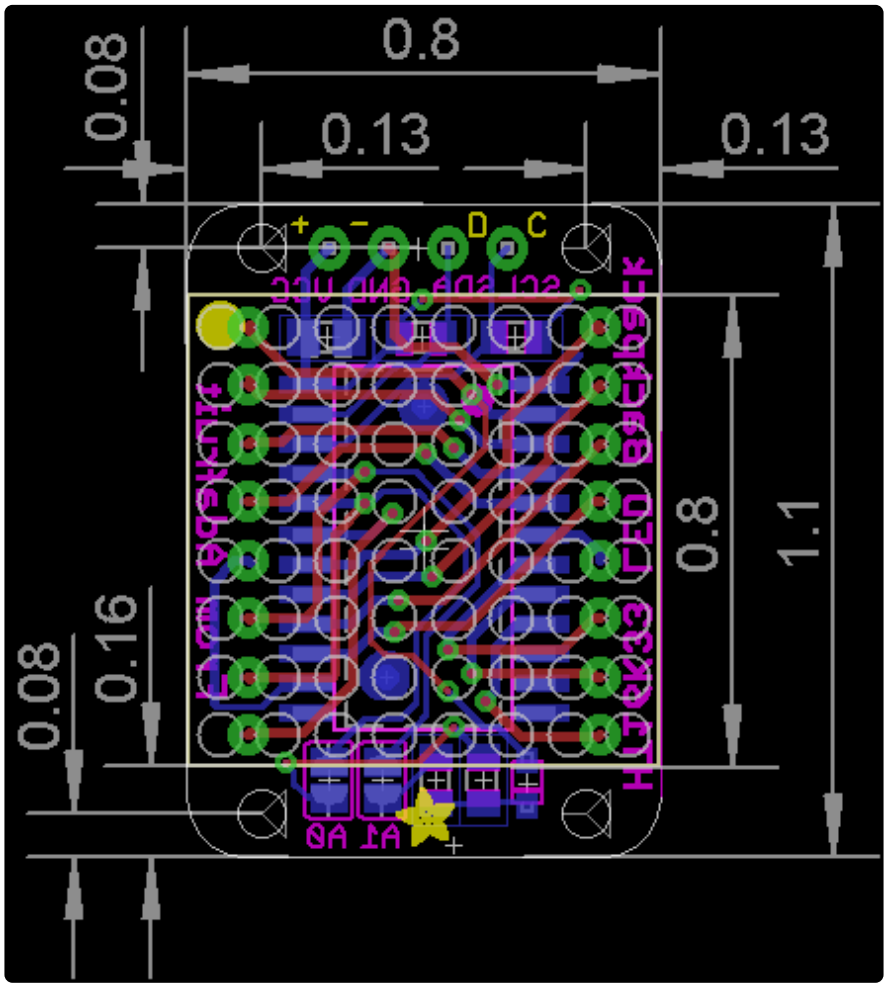

U2
HT16K33 LED Driver
16 Step Dimming
VDD: 4.5-5.5V
VDD
SCL
SDA
GND
JP1
JP2
C1
10uF
R1
R2
10K
R3
R4
R5
47K
D1
1N4148
A0
A1
A2
28-SOIC


0.8
0.4
1.4
1.3
0.1
0.7
GND
VDD
SDA
SCL
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
C0
C1
C2
C3
C4
C5
C6
C7

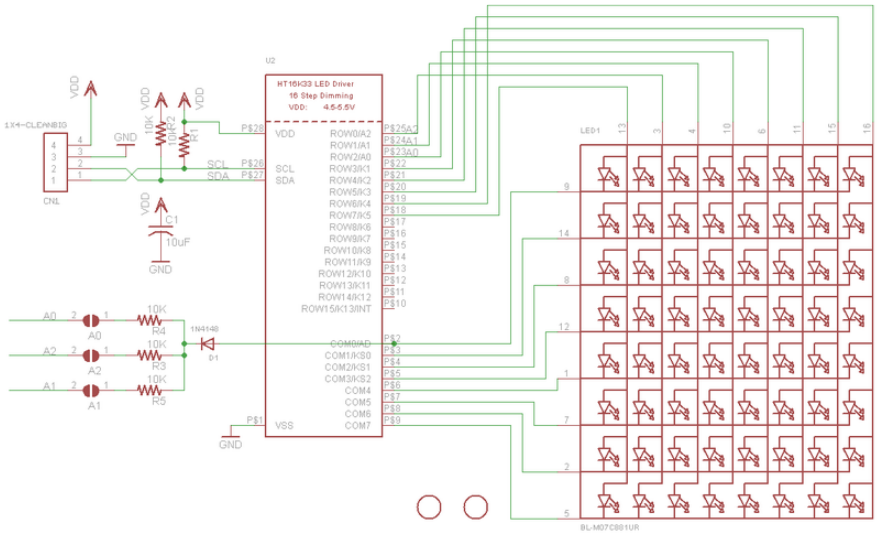
NOTE - The silkscreen labels for the A0/A2 header pins are swapped. The silkscreen labels for the solder pads are correct.

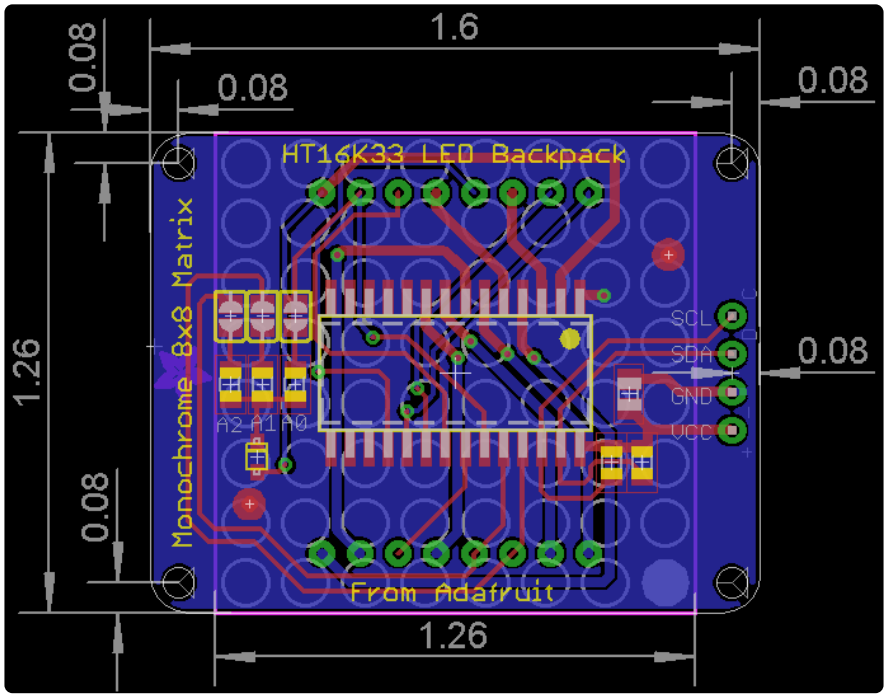
8x8 0.8" LED Backpack



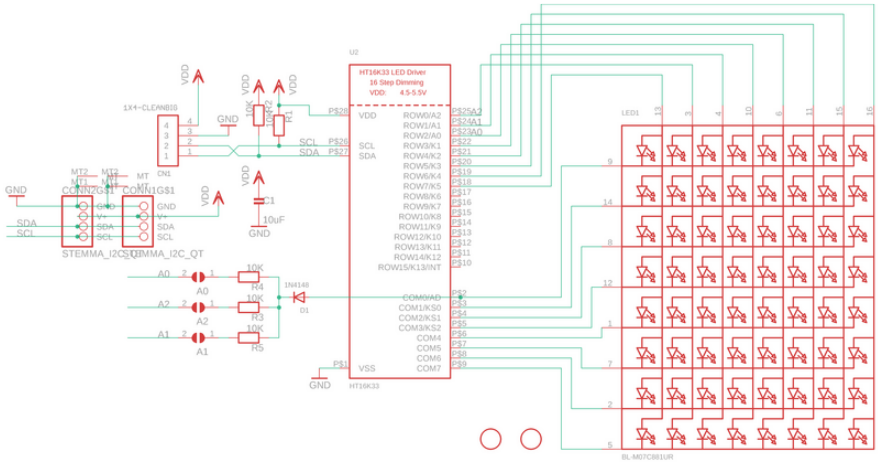


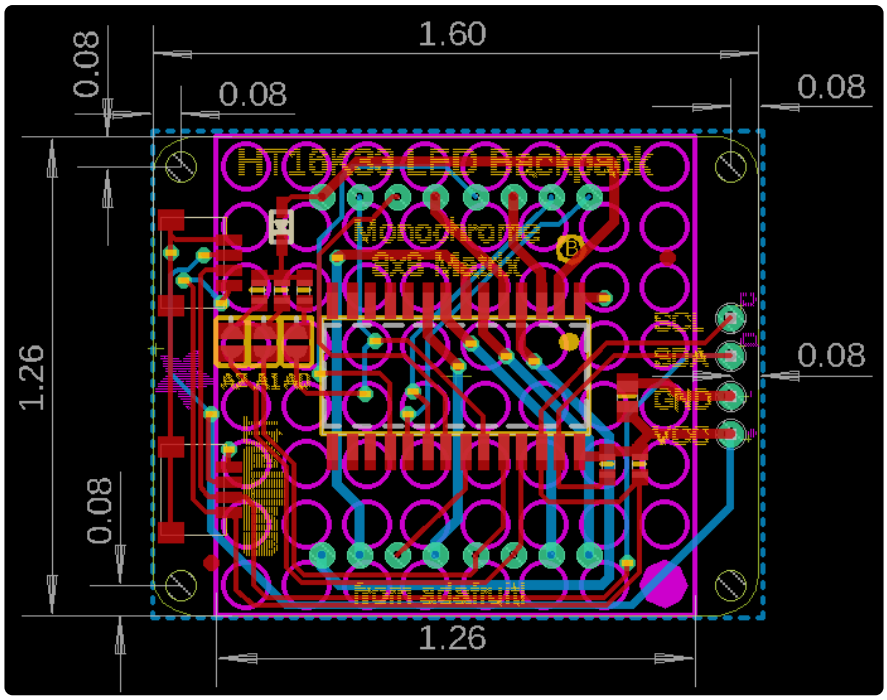
8x8 1.2" LED Backpack



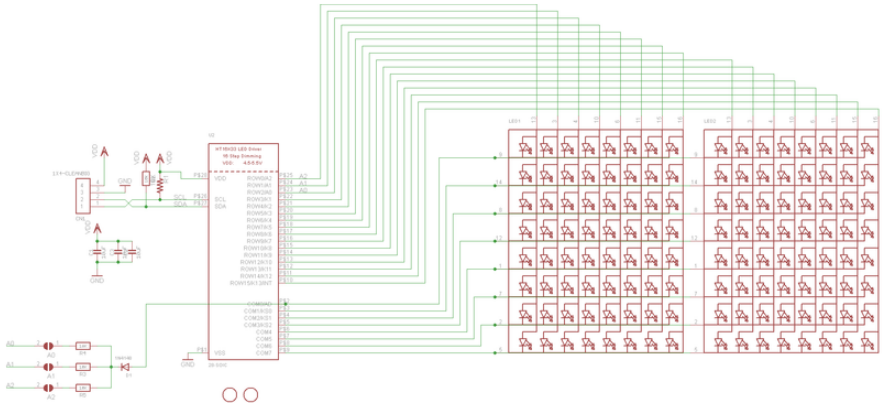


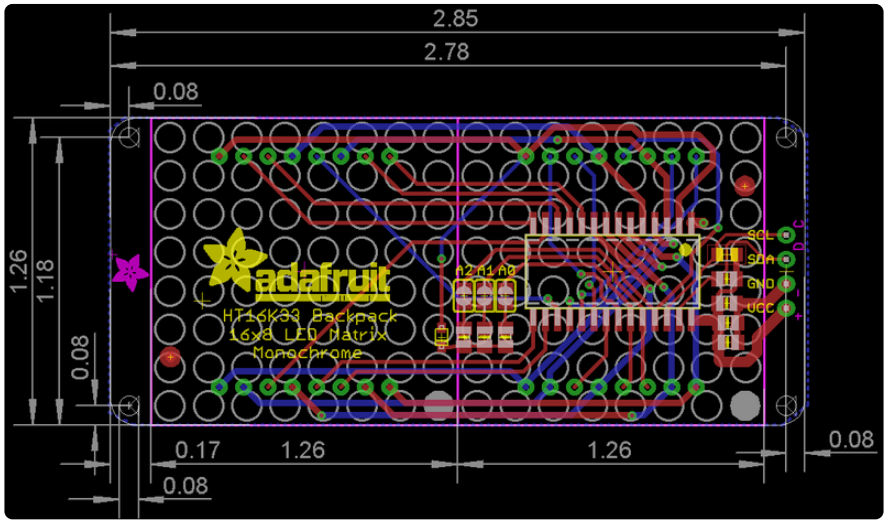
8x8 1.2" Bi-Color LED Backpack



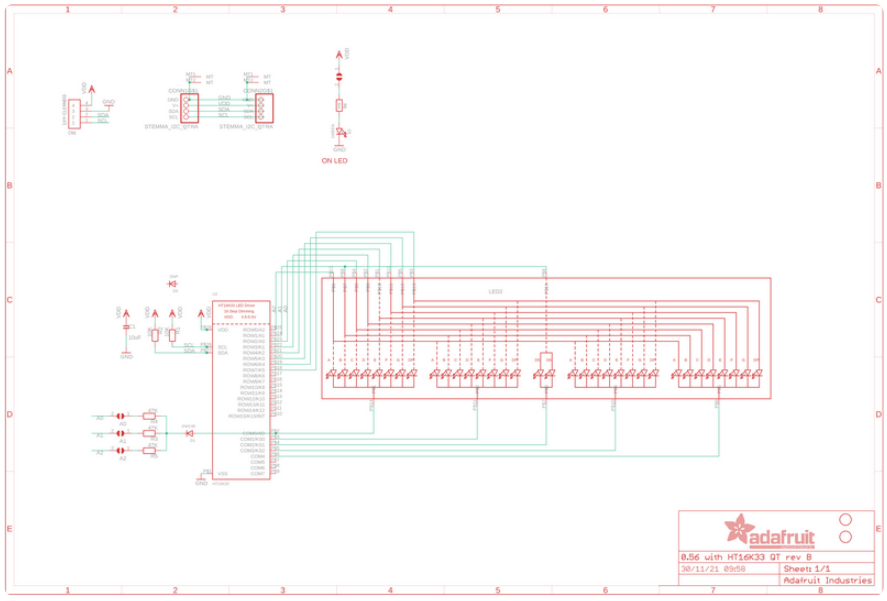


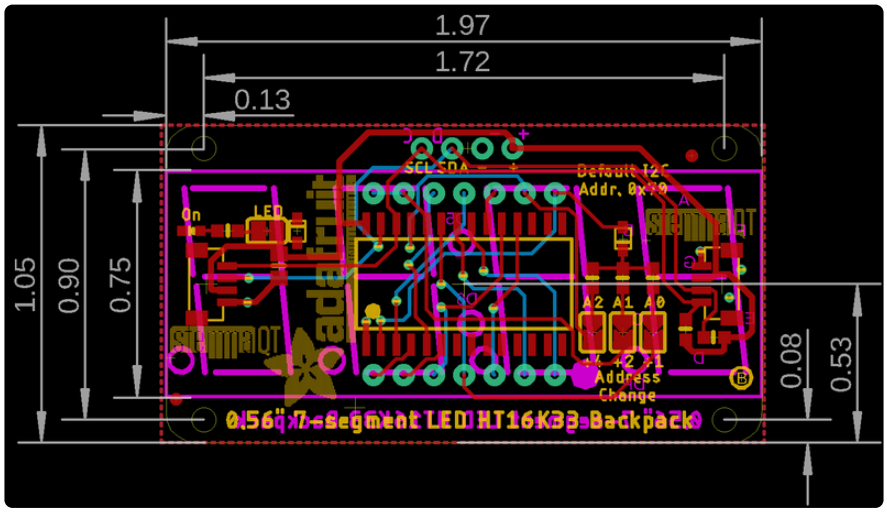
16x8 1.2" LED Backpacks



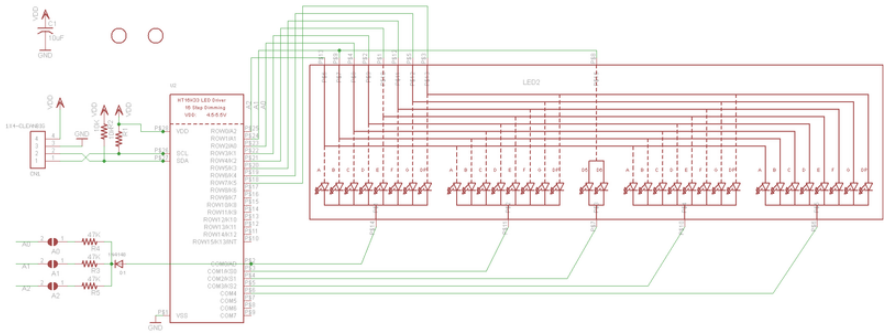


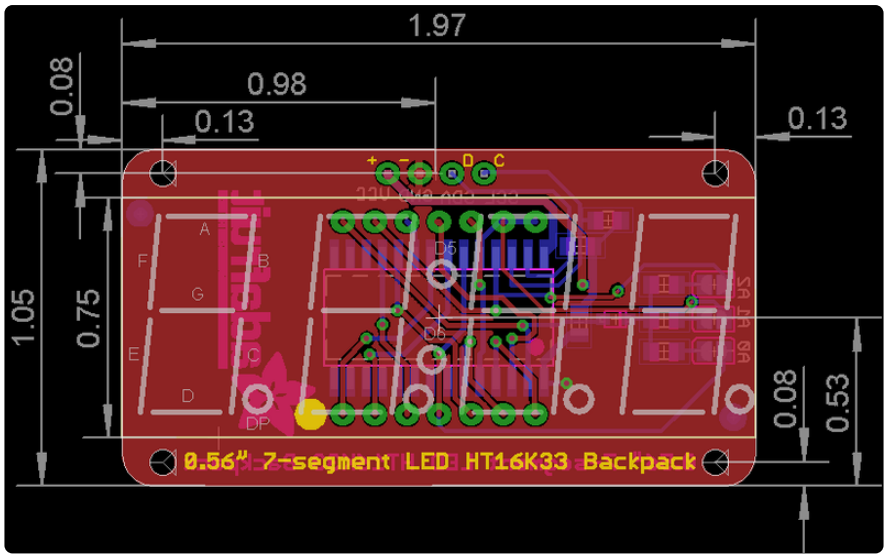
0.56" 7-Segment LED Backpack STEMMA QT



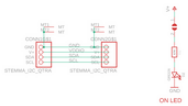


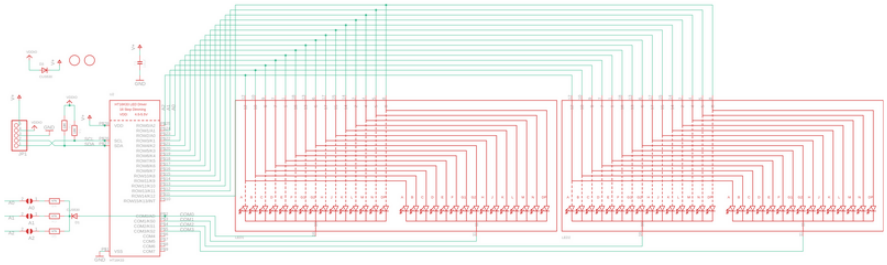
Quad 0.56" 7-Segment Original

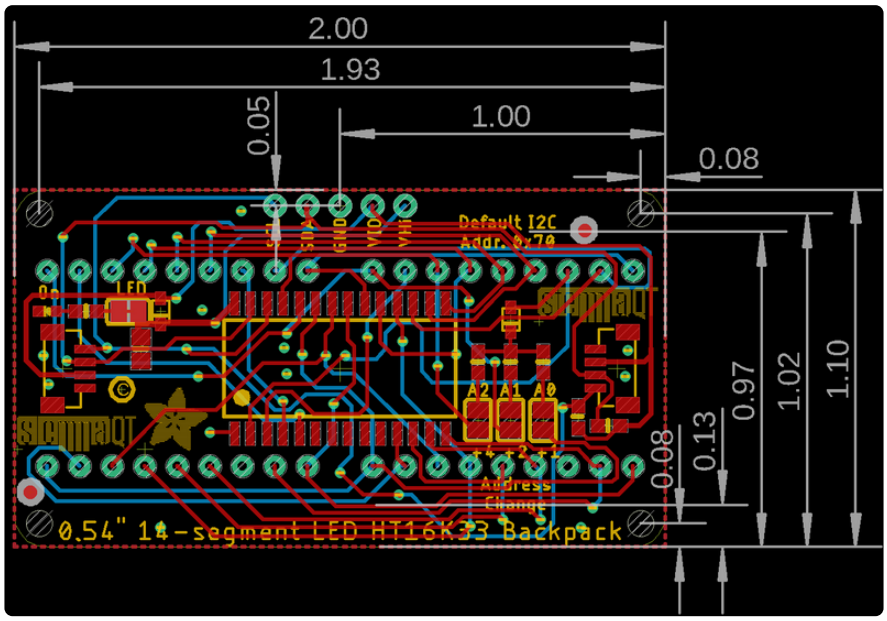




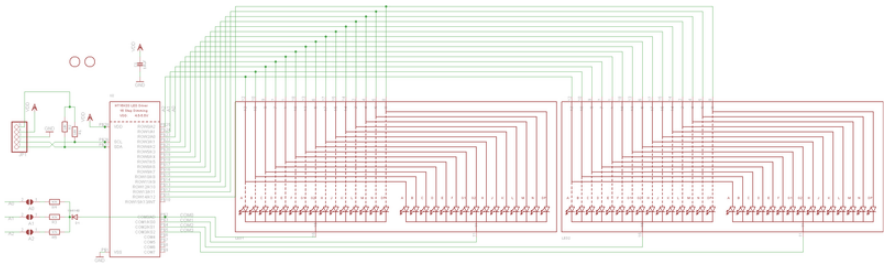
Quad 0.54" 14-segment Alphanumeric STEMMA QT Version

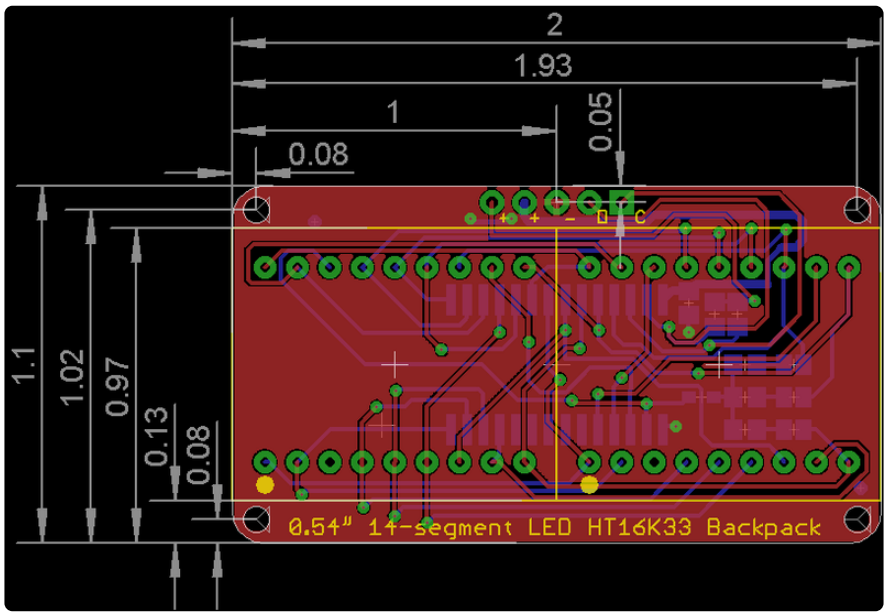




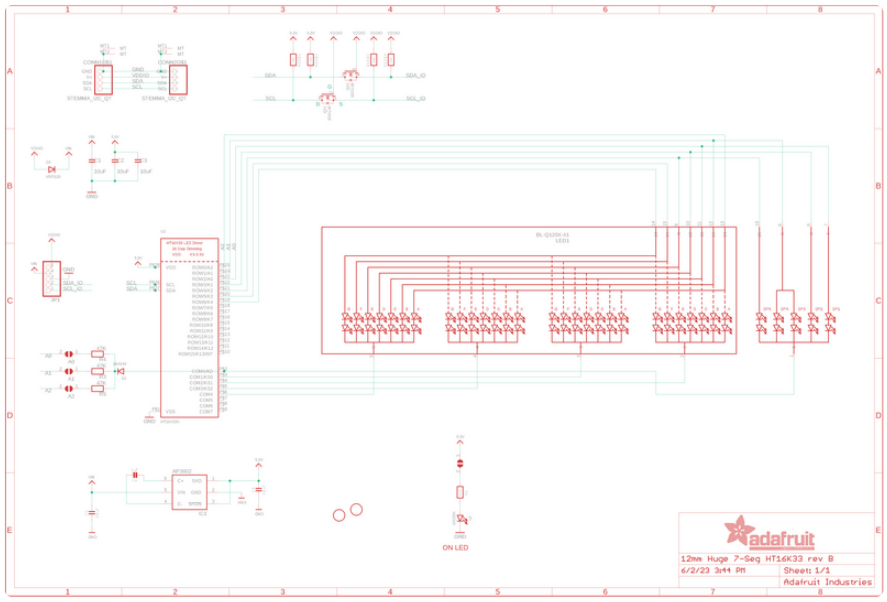


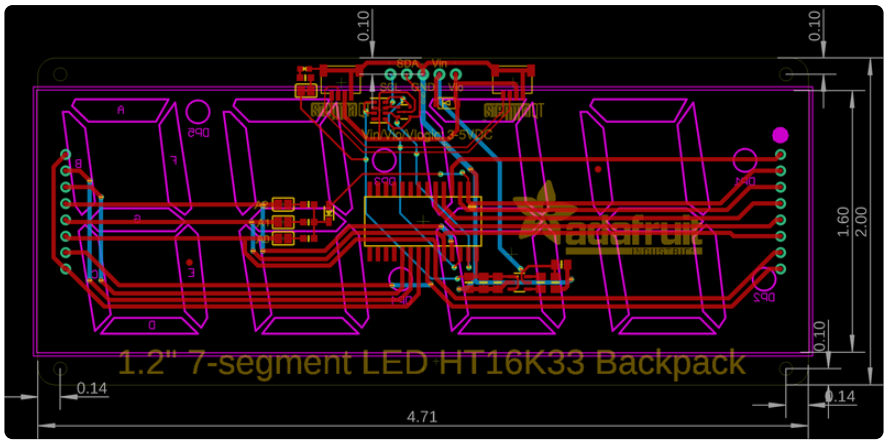
Quad 0.54" 14-segment Alphanumeric Original Version





Quad 1.2" 7-Segment





Bicolor 24-Bargraph