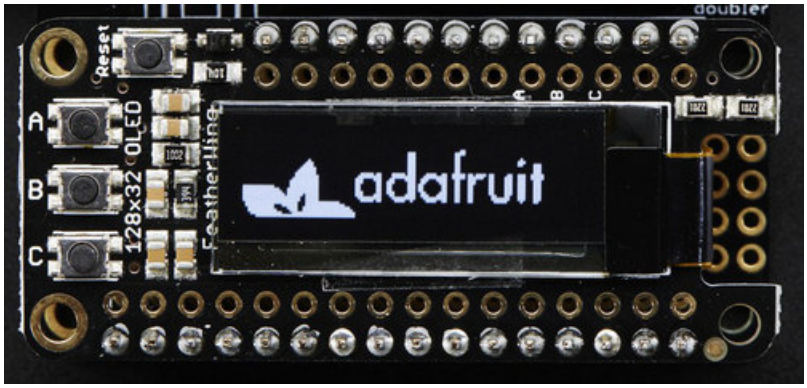
adafruit learning system

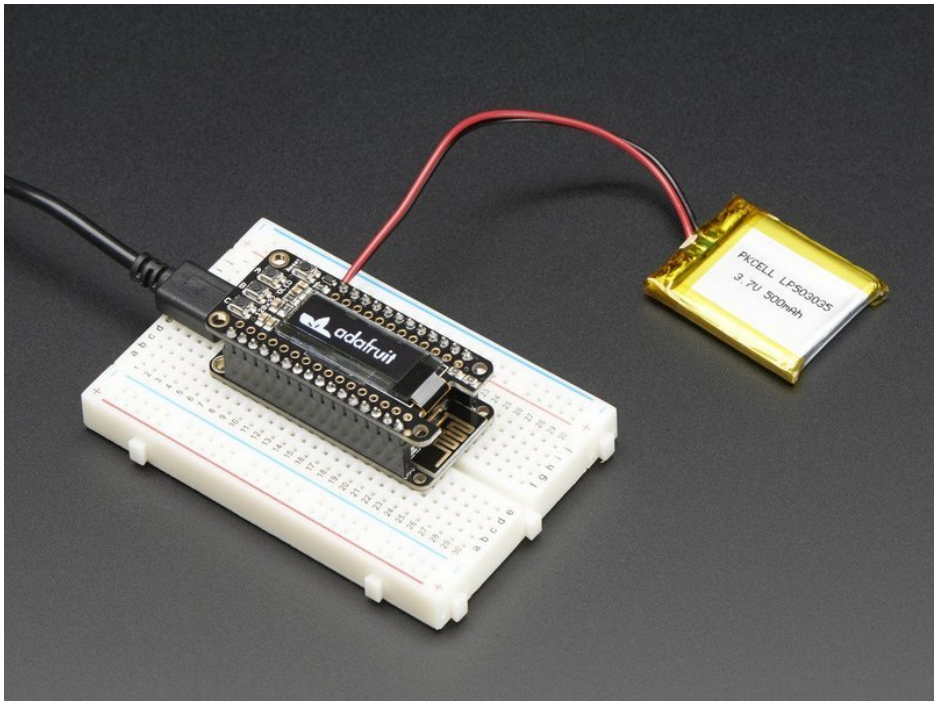
Adafruit OLED FeatherWing

Created by lady ada



Last updated on 2019-10-30 01:46:38 AM UTC

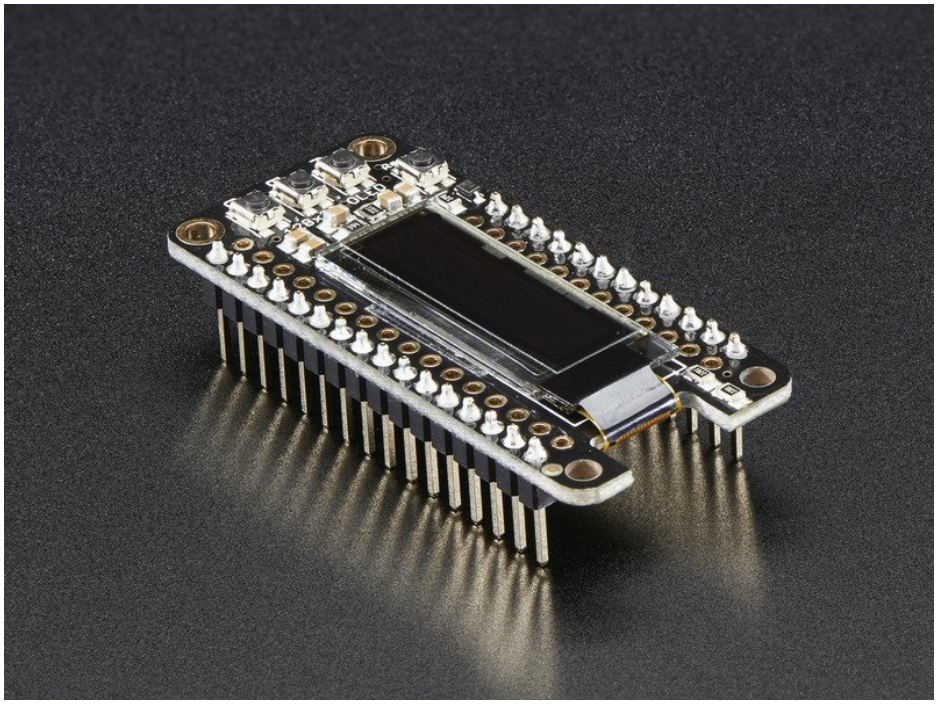
Overview



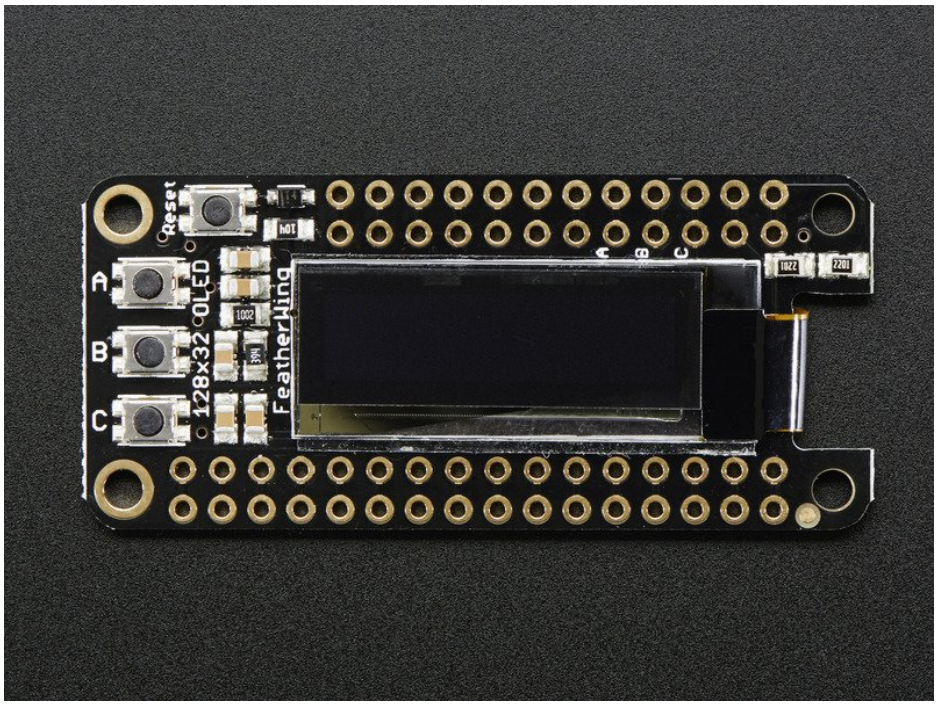
A Feather board without ambition is a Feather board without FeatherWings! This is the **FeatherWing OLED**: it adds a 128x32 monochrome OLED plus 3 user buttons to *any* Feather main board. Using our Feather Stacking Headers (http://adafru.it/2830) or Feather Female Headers (http://adafru.it/2886) you can connect a FeatherWing on top of your Feather board and let the board take flight!

These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. This screen is made of 128x32 individual white OLED pixels and because the display makes its own light, no backlight is

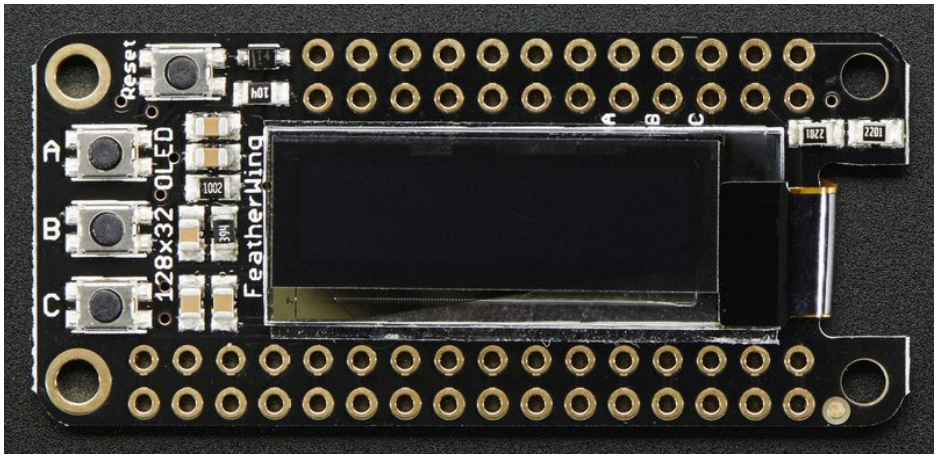
required. This reduces the power required to run the OLED and is why the display has such high contrast; we really like this miniature display for its crispness! We also toss on a reset button and three mini tactile buttons called A B and C so you can add a mini user interface to your feather.



Tested works with all of our Feather boards. The OLED uses only the two I2C pins on the Feather, and you can pretty much stack it with any other FeatherWing, even ones that use I2C since that is a shared bus.

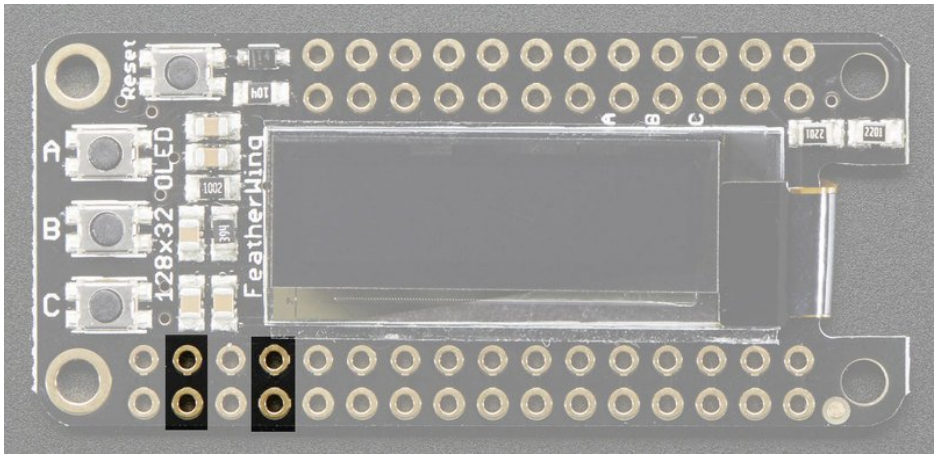


Pinouts



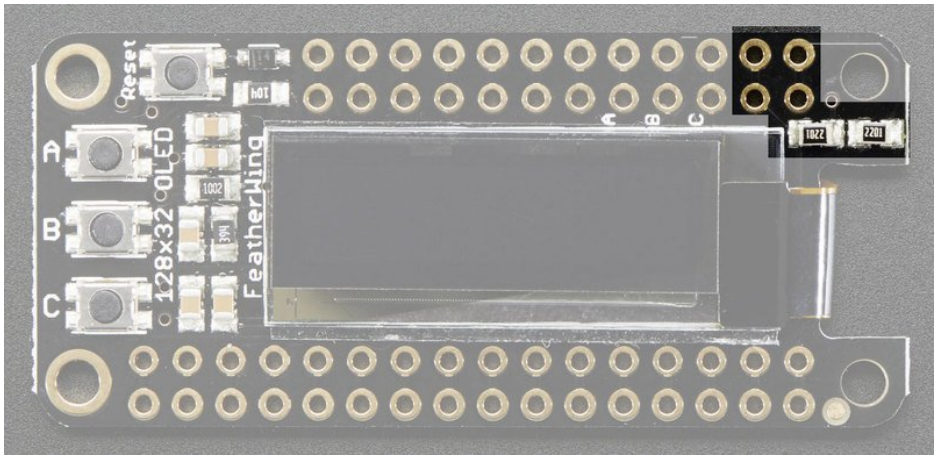
The OLED FeatherWing plugs into any Feather and adds a cute little display. To make it as cross-platform compatible as possible, we use only I2C to control the display. This is not as fast as SPI but it uses only two pins, can share the I2C bus and is fine for the small 128x32 pixel OLED.

Power Pins



OLED displays do not have a backlight, and are fairly low power, this display will draw about 10mA when in use. The display uses 3V power and logic so we just connect to the 3V and GND pins from the feather, as indicated above.

I2C Data Pins

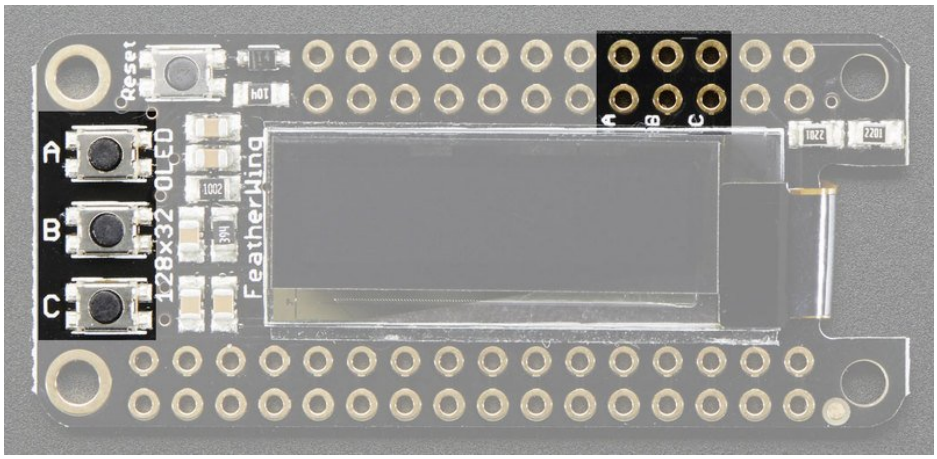


The cute little OLED does all of the data transfer over the I2C pins, highlighed above **SDA** and **SCL**. No other pins are required. There are two 2.2K pullups to 3V on each.

These pins can be shared with other I2C devices.

The I2C address is **0x3C** and cannot be changed

Optional Buttons



We had a little bit of space so we added three mini tactile buttons that you can use for user interface. We label them **A B** and **C** because each Feather has slightly different pin numbering schemes and we wanted to make it 'universal'

If you're using ATmega328P, Atmega32u4, ATSAMD51 M4 or ATSAMD21 M0 Feather

- Button A is **#9 (note this is also used for the battery voltage divider** so if you want to use both make sure you disable the pullup when you analog read, then turn on the pullup for button reads)
- Button B is **#6**
- Button C is **#5**

If you're using ESP8266:

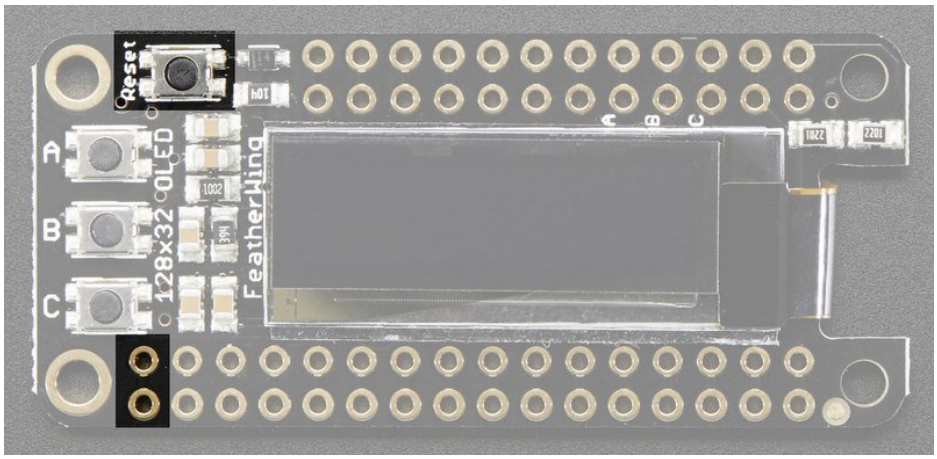
- Button A is **#0**
- Button B is **#16**
- Button C is **#2**

If you're using WICED/STM32 Feather

- Button A is **#PA15**
- Button B is **#PC7**
- Button C is **#PC5**

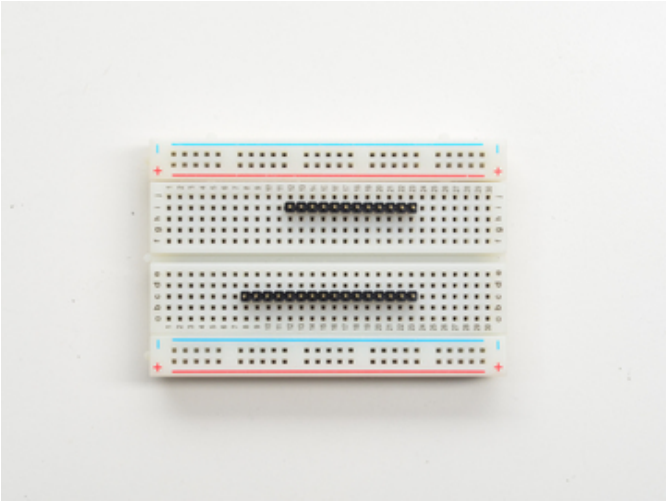
Button B has a 100K pullup on it so it will work with the ESP8266 (which does not have an internal pullup available on that pin). You will need to set up a pullup on all other pins for the buttons to work.

Reset Button



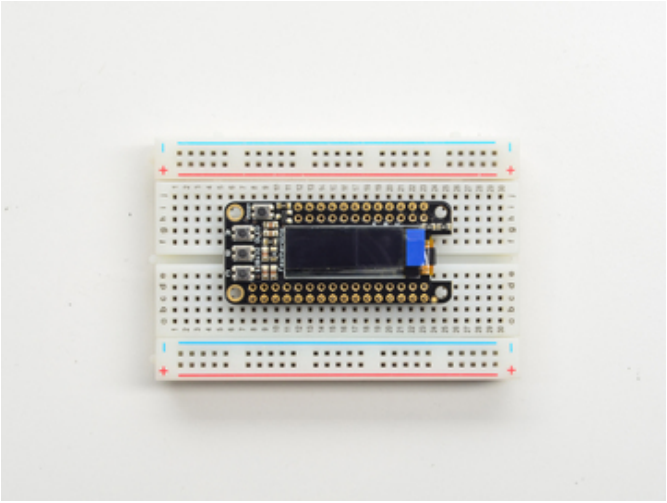
Sometimes its nice to be able to restart your program, so we also have a reset button. It is tied to the **RST** pin marked above.

Assembly



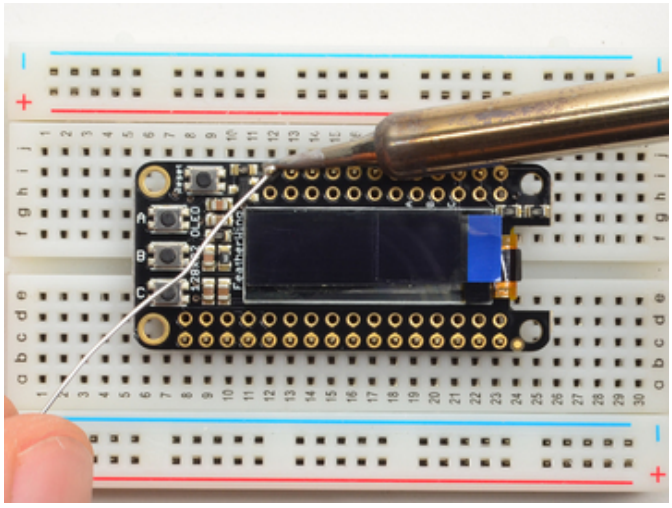
Prepare the header strip:

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



Add the FeatherWing:

Place the featherwing over the pins so that the short pins poke through the two rows of breakout pads

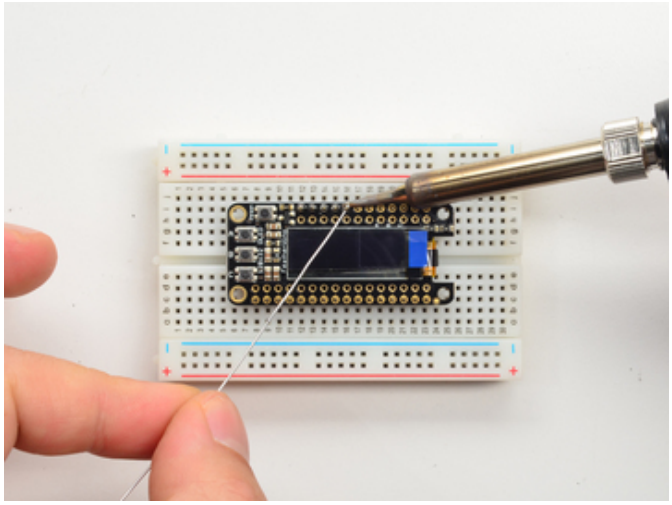


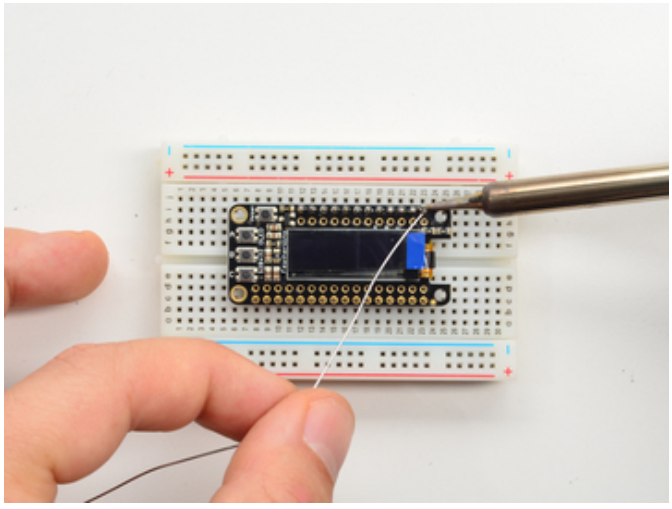
And Solder!

Be sure to solder all pins for reliable electrical contact.

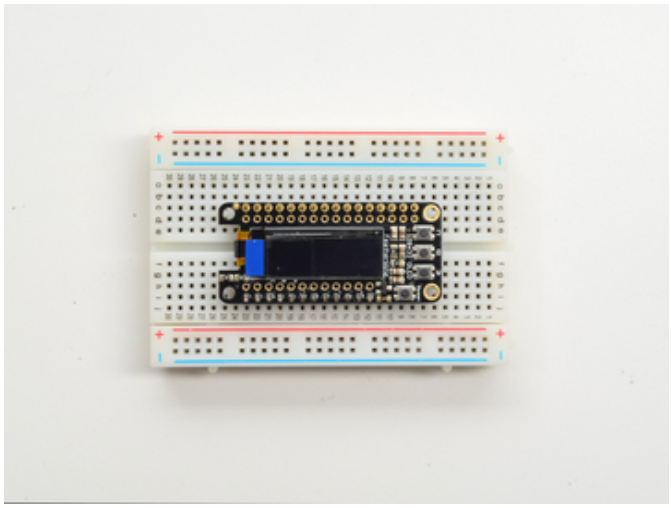
(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk).

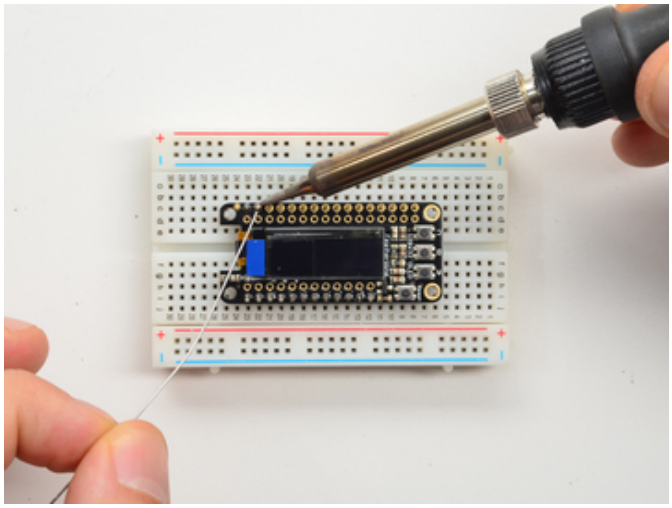
Start by soldering the first row of header

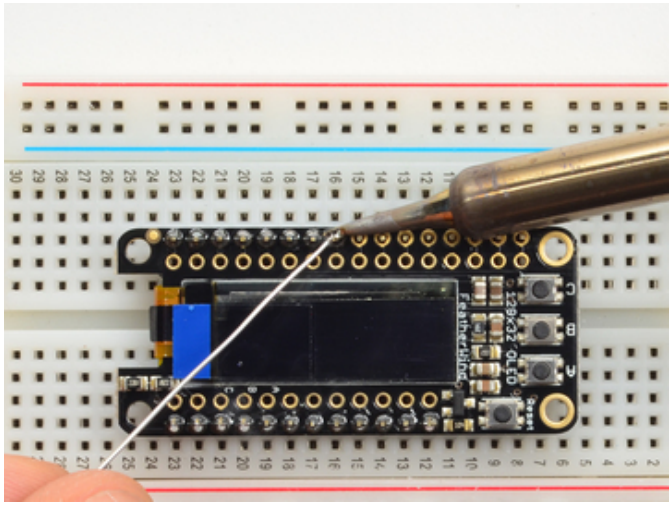


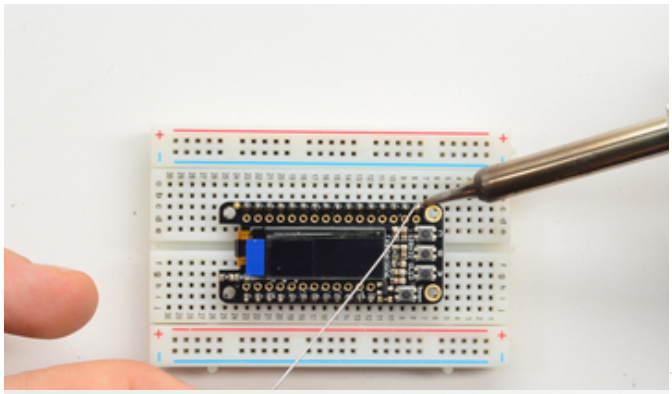


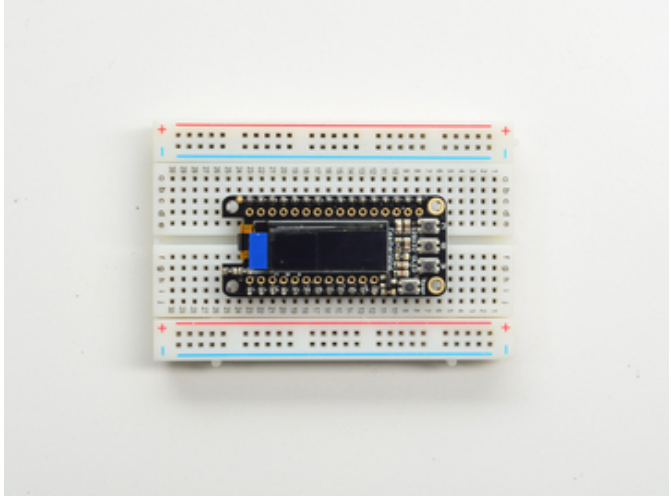
Now flip around and solder the other row completely





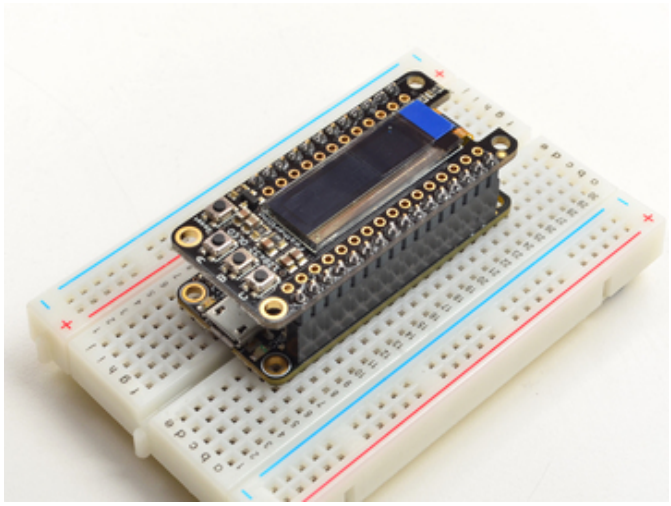

Featherwing
128x32 OLED





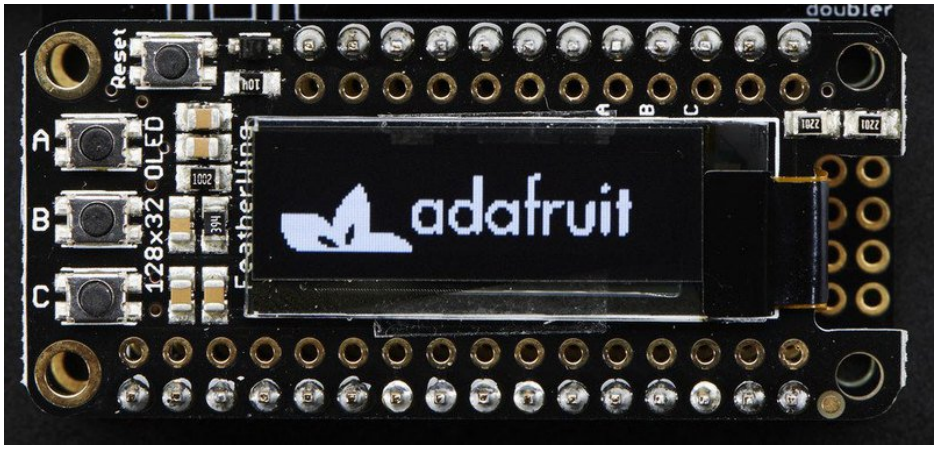
You're done with the two header strips.

Check your solder joints visually and continue onto the next steps



OK You're done! You can now plug your FeatherWing into your Feather and get your OLED on!

Arduino Code

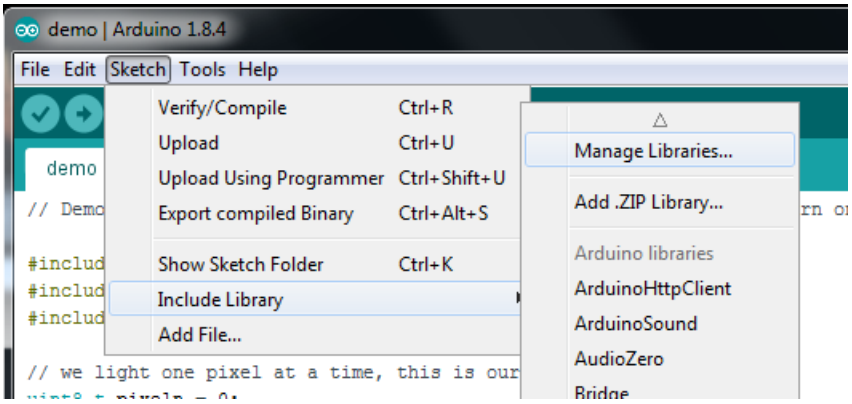


The OLED display we use is well supported and works for all Feathers, all you need is a little library support and you will be drawing in no time!

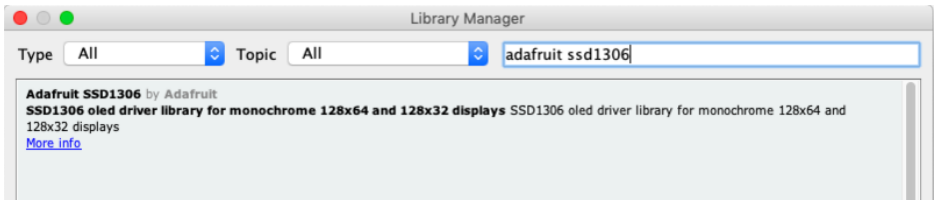
Install Arduino Libraries

Using the OLED FeatherWing with Arduino sketches requires that two libraries be installed: **Adafruit_SSD1306**, which handles the low-level communication with the hardware, and **Adafruit_GFX**, which builds atop this to add graphics functions like lines, circles and text.

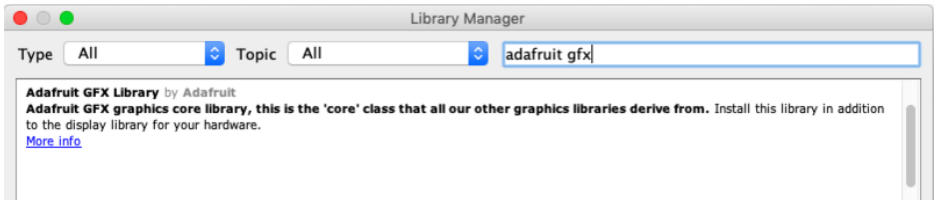
Open up the library manager:



Search for the **Adafruit SSD1306** library and install it



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



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

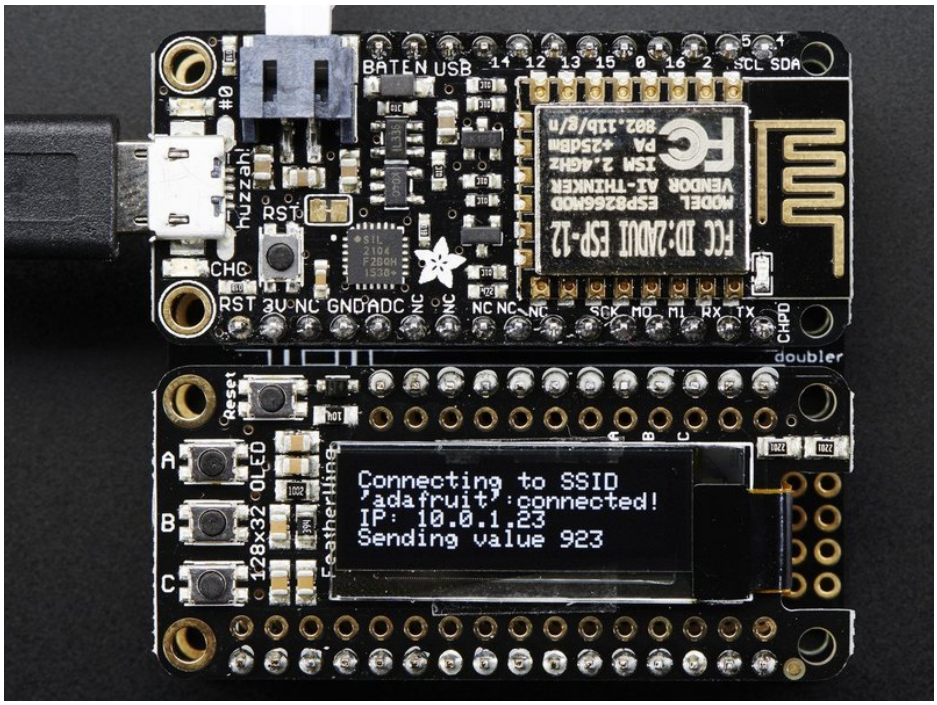
Run Example Code

We have a basic demo that works with all Feathers, so compile/upload this sketch:

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

// OLED FeatherWing buttons map to different pins depending on board:
#if defined(ESP8266)
  #define BUTTON_A  0
  #define BUTTON_B 16
  #define BUTTON_C  2
#elif defined(ESP32)
  #define BUTTON_A 15
  #define BUTTON_B 32
  #define BUTTON_C 14
#elif defined(ARDUINO_STM32_FEATHER)
  #define BUTTON_A PA15
  #define BUTTON_B PC7
  #define BUTTON_C PC5
#elif defined(TEENSYDUINO)
  #define BUTTON_A  4
  #define BUTTON_B  3
  #define BUTTON_C  8
#elif defined(ARDUINO_FEATHER52832)
  #define BUTTON_A 31
  #define BUTTON_B 30
  #define BUTTON_C 27
#else // 32u4, M0, M4, nrf52840 and 328p
  #define BUTTON_A  9
  #define BUTTON_B  6
  #define BUTTON_C  5
#endif

void setup() {
  Serial.begin(9600);

  Serial.println("OLED FeatherWing test");
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // Address 0x3C for 128x32

  Serial.println("OLED begun");

  // Show image buffer on the display hardware
```

```
  // Show image buffer on the display hardware.
  // Since the buffer is intialized with an Adafruit splashscreen
  // internally, this will display the splashscreen.
  display.display();
  delay(1000);

  // Clear the buffer.
  display.clearDisplay();
  display.display();

  Serial.println("IO test");

  pinMode(BUTTON_A, INPUT_PULLUP);
  pinMode(BUTTON_B, INPUT_PULLUP);
  pinMode(BUTTON_C, INPUT_PULLUP);

  // text display tests
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0,0);
  display.print("Connecting to SSID\n'adafruit':");
  display.print("connected!");
  display.println("IP: 10.0.1.23");
  display.println("Sending val #0");
  display.setCursor(0,0);
  display.display(); // actually display all of the above
}

void loop() {
  if(!digitalRead(BUTTON_A)) display.print("A");
  if(!digitalRead(BUTTON_B)) display.print("B");
  if(!digitalRead(BUTTON_C)) display.print("C");
  delay(10);
  yield();
  display.display();
}
```

You should see the OLED display a splash screen then spit out some text (it's a make-believe WiFi connection status screen...this doesn't actually do anything, just showing how typical project might look). If you press the **A B** or **C** buttons it will also print those out.



Do more!

You can use any of the Adafruit GFX library commands to draw onto your OLED, that means that you get all sorts of shapes, fonts, lines, etc available. Check out GFX for all the underlying graphics support functions and how they work (https://adafru.it/doL)

Remember you need to call display() after drawing to refresh the screen!

CircuitPython Wiring

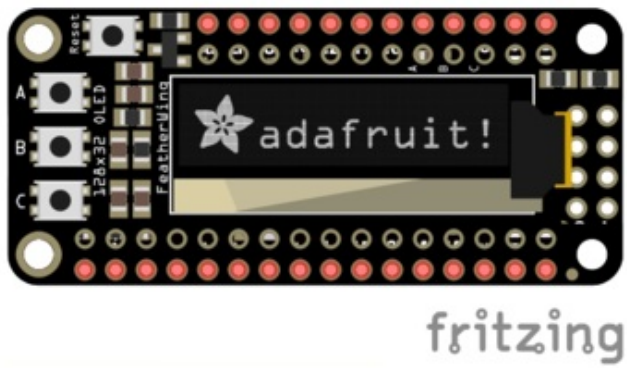
It's easy to use OLEDs with CircuitPython and the Adafruit CircuitPython SSD1306 (https://adafru.it/u1f) module. This module allows you to easily write Python code to control the display.

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

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

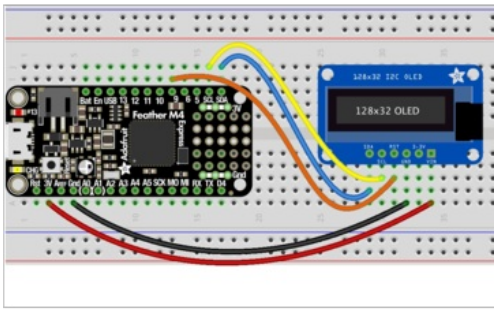
Connect the OLED to your microcontroller board as shown below.

Adafruit OLED FeatherWing



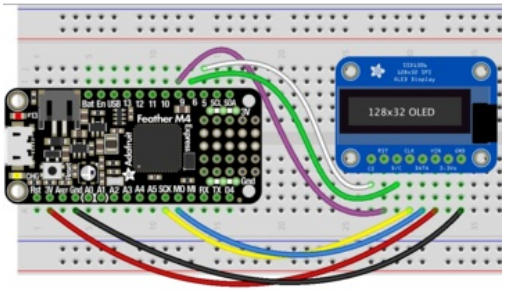
- **Solder the Feather with female headers on top or stacking headers.**
- **Attach the OLED FeatherWing using the stacking method.**

Adafruit 128x32 I2C OLED Display



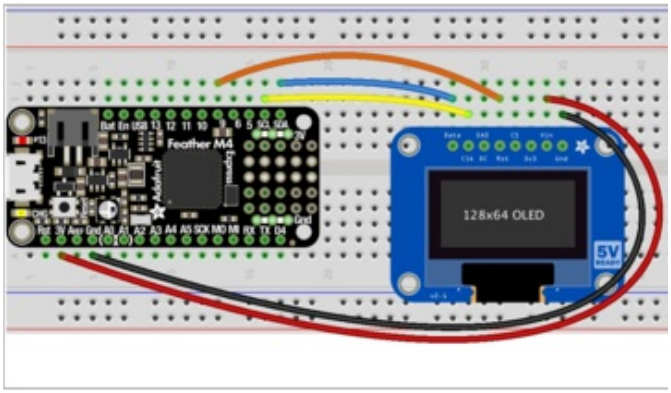
- **Microcontroller 3V** to **OLED VIN**
- **Microcontroller GND** to **OLED GND**
- **Microcontroller SCL** to **OLED SCL**
- **Microcontroller SDA** to **OLED SDA**
- **Microcontroller D9** to **OLED RST**

Adafruit 128x32 SPI OLED Display



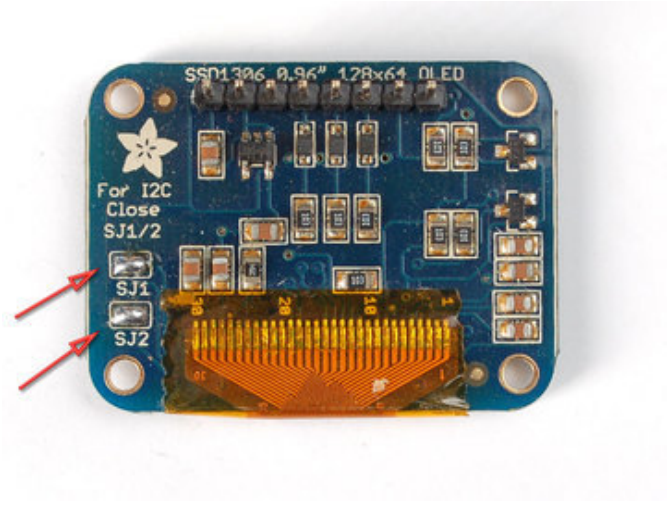
- **Microcontroller 3V** to **OLED VIN**
- **Microcontroller GND** to **OLED GND**
- **Microcontroller SCK** to **OLED CLK**
- **Microcontroller MOSI** to **OLED Data**
- **Microcontroller D5** to **OLED CS**
- **Microcontroller D6** to **OLED D/C**
- **Microcontroller D9** to **OLED RST**

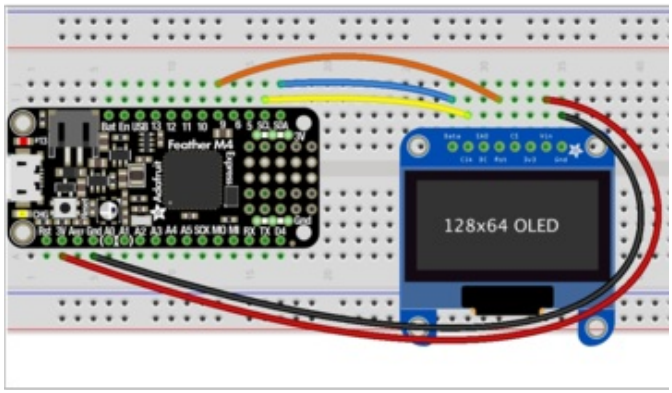
Adafruit 0.96" or 1.3" 128x64 OLED Display - I2C Wiring

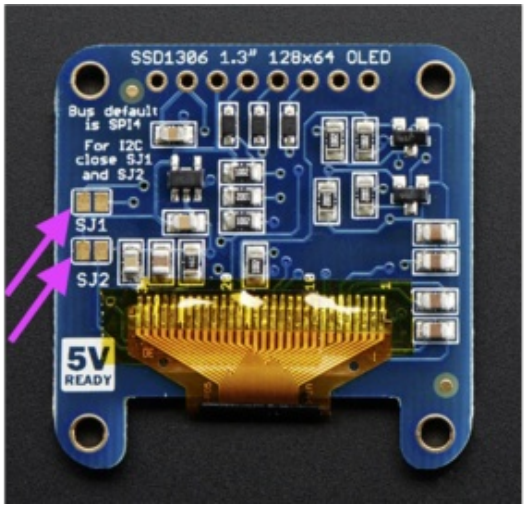


You must solder two jumpers closed on the back of the display to use with I2C!

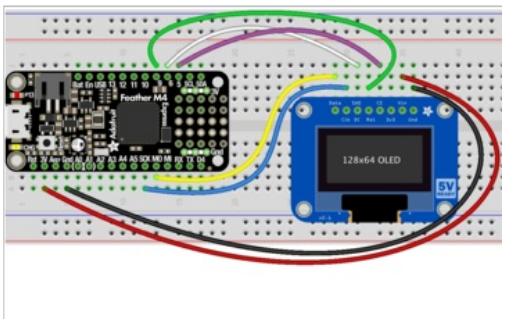
- **Microcontroller 3V** to **OLED Vin**
- **Microcontroller GND** to **OLED Gnd**
- **Microcontroller SCL** to **OLED Data**
- **Microcontroller SDA** to **OLED Clk**
- **Microcontroller D9** to **OLED Rst**



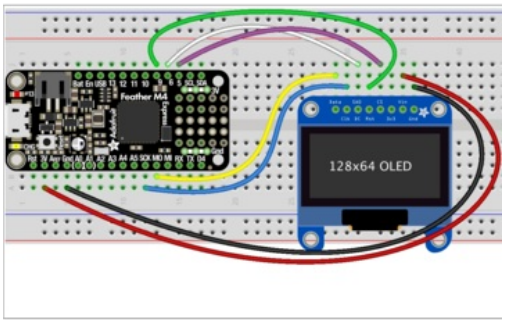




Adafruit 0.96" or 1.3" 128x64 OLED Display - SPI Wiring



- **Microcontroller 3V** to **OLED Vin**
- **Microcontroller GND** to **OLED Gnd**
- **Microcontroller SCK** to **OLED Clk**
- **Microcontroller MOSI** to **OLED Data**
- **Microcontroller D5** to **OLED CS**
- **Microcontroller D6** to **OLED DC**
- **Microcontroller D9** to **OLED Rst**



CircuitPython Installation of DisplayIO SSD1306 Library

To use the SSD1306 OLED with your Adafruit CircuitPython board you'll need to install the Adafruit CircuitPython DisplayIO SSD1306 (https://adafru.it/FRA) module on your board.

First make sure you are running the latest version 5.0 or later of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

If you choose, you can manually install the libraries individually on your board:

- **adafruit_displayio_ssd1306**
- **adafruit_bus_device**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_displayio_ssd1306.mpy and adafruit_bus_device** files and folders copied over.

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

Code Example Additional Libraries

For the Code Example, you will need an additional library. We decided to make use of a library so the code didn't get overly complicated.

https://adafru.it/FRB

https://adafru.it/FRB

Go ahead and install this in the same manner as the driver library by copying the **adafruit_display_text** folder over to the **lib** folder on your CircuitPython device.

Displayio is only available on express board due to the smaller memory size on non-express boards.

It's easy to use OLEDs with Python and the Adafruit CircuitPython DisplayIO SSD1306 (https://adafru.it/FRA) module. This module allows you to easily write Python code to control the display.

To demonstrate the usage, we'll initialize the library and use Python code to control the OLED from the board's Python REPL.

I2C Initialization

If your display is connected to the board using I2C (like if using a Feather and the FeatherWing OLED) you'll first need to initialize the I2C bus. First import the necessary modules:

```
import board
```

Now for either board run this command 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):

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

After initializing the I2C interface for your firmware as described above, you can create an instance of the I2CDisplay bus:

```
import displayio
import adafruit_displayio_ssd1306
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c)
```

Finally, you can pass the display_bus in and create an instance of the SSD1306 I2C driver by running:

```
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=32)
```

Now you should be seeing an image of the REPL. Note that the last two parameters to the `SSD1306` class initializer are the **width** and **height** of the display in pixels. Be sure to use the right values for the display you're using!

```
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c, reset=board.D9)
```

At this point the I2C bus and display are initialized.

Example Code

```
"""
This test will initialize the display using displayio and draw a solid white
background, a smaller black rectangle, and some white text.
"""

import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_displayio_ssd1306

displayio.release_displays()

i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c)
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=32)

# Make the display context
splash = displayio.Group(max_size=10)
display.show(splash)

color_bitmap = displayio.Bitmap(128, 32, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF # White

bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)
splash.append(bg_sprite)

# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(118, 24, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000 # Black
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=5, y=4)
splash.append(inner_sprite)

# Draw a label
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=28, y=15)
splash.append(text_area)

while True:
    pass
```

Let's take a look at the sections of code one by one. We start by importing the board so that we can initialize `SPI`, `displayio`, `terminalio` for the font, a `label`, and the `adafruit_displayio_ssd1306` driver.

```
import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_displayio_ssd1306
```

Next we release any previously used displays. This is important because if the microprocessor is reset, the display pins are not automatically released and this makes them available for use again.

```
displayio.release_displays()
```

The FeatherWing uses I2C, so we set the I2C object to the board's I2C with the easy shortcut function `board.I2C()`. By using this function, it finds the SPI module and initializes using the default SPI parameters. We also set the display bus to I2CDisplay which makes use of the I2C bus.

```
# Use for I2C
i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c)
```

Finally, we initialize the driver with a width of the **128** variable and a height of the **32** variable. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update.

```
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=32)
```

Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

```
splash = displayio.Group(max_size=10)
display.show(splash)
```

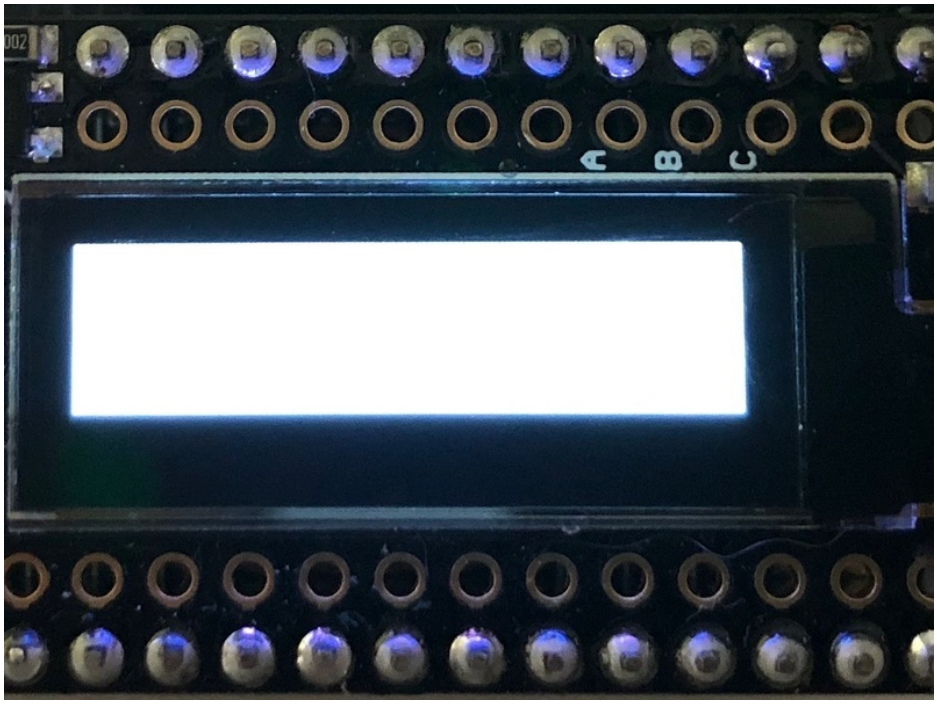
Next we create a Bitmap that is the full width and height of the display. The Bitmap is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. Although the Bitmaps can handle up to 256 different colors, the display is monochrome so we only need one. We create a Palette

with one color and set that color to `0xFFFFFF` which happens to be white. If were to place a different color here, `displayio` handles color conversion automatically, so it may end up black or white depending on the calculation.

```
color_bitmap = displayio.Bitmap(128, 32, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF # White
```

With all those pieces in place, we create a TileGrid by passing the bitmap and palette and draw it at `(0, 0)` which represents the display's upper left.

```
bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)
splash.append(bg_sprite)
```

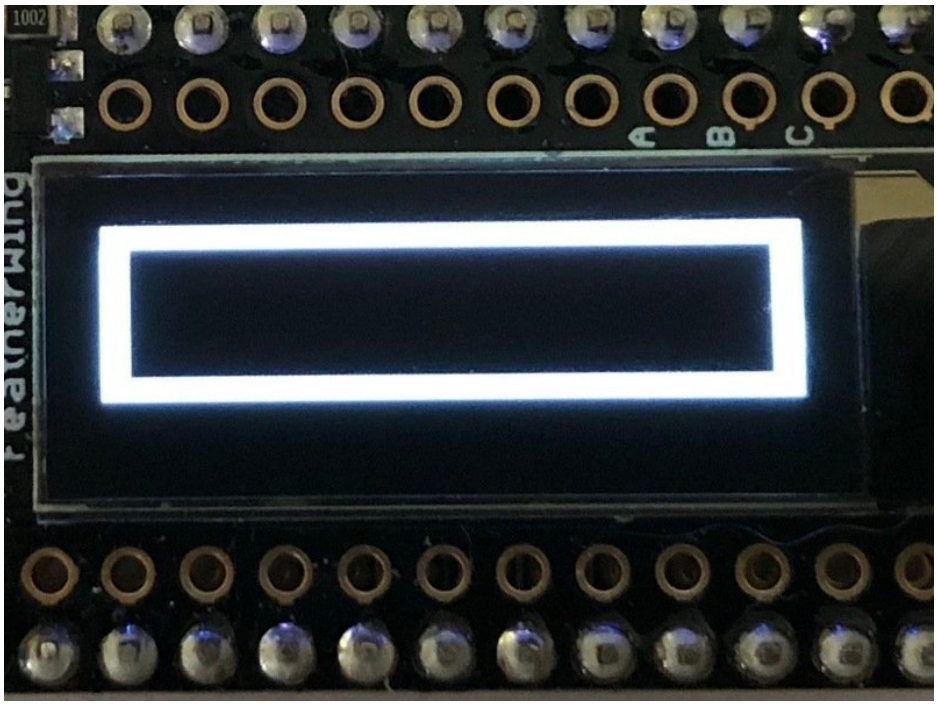


Next we will create a smaller black rectangle. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color of `0x000000`, which is **black**, and place it in a specific location. In this case, we will create a bitmap that is 5 pixels smaller on each side. The screen we're using here is **128x32** and we have the border set to **5** , so we'll want to subtract 10 from each of those numbers.

We'll also want to place it at the position `(5, 5)` so that it ends up centered.

```
inner_bitmap = displayio.Bitmap(118, 24, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000 # Black
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=5, y=4)
splash.append(inner_sprite)
```

Since we are adding this after the first square, it's automatically drawn on top. Here's what it looks like now.



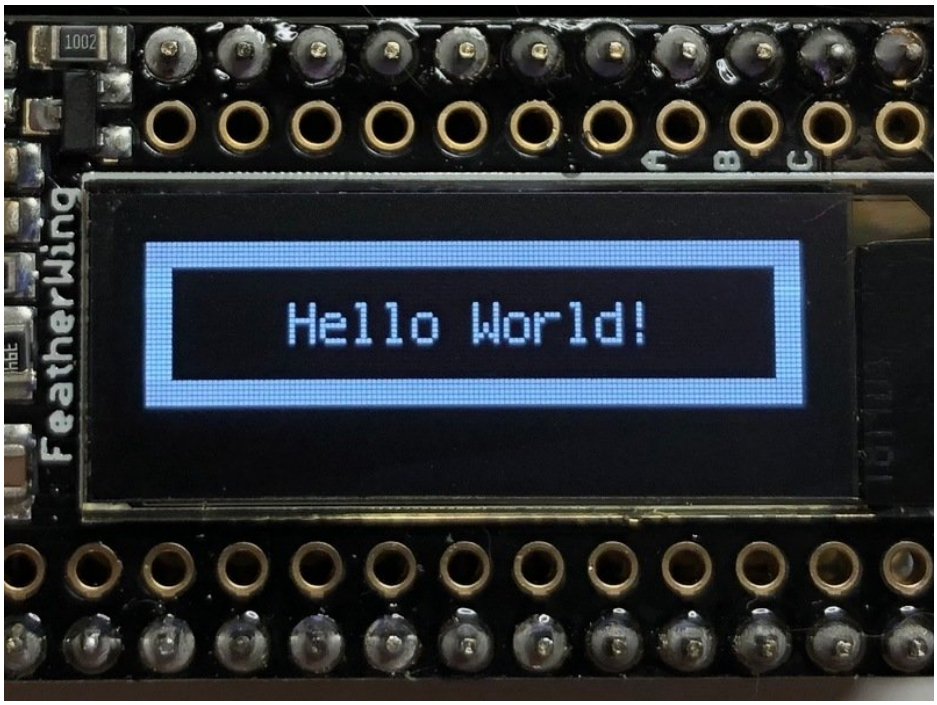
Next add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font. In this example, we won't be doing any scaling because of the small resolution, so we'll add the label directly the main group. If we were scaling, we would have used a subgroup.

Labels are centered vertically, so we'll place it at half the heigh for the Y coordinate and subtract one so it looks good. We'll set the width to around 28 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some white text, so we'll pass it a value of `0xFFFFFF`.

```
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=28, y=15)
splash.append(text_area)
```

Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

```
while True:
    pass
```



Where to go from here

Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafru.it/EGh)

Display does not work on initial power but does work after a reset.

The OLED driver circuit needs a small amount of time to be ready after initial power. If your code tries to write to the display too soon, it may not be ready. It will work on reset since that typically does not cycle power. If you are having this issue, try adding a small amount of delay before trying to write to the OLED.

In Arduino, use **delay()** to add a few milliseconds before calling **oled.begin()**. Adjust the amount of delay as needed to see how little you can get away with for your specific setup.

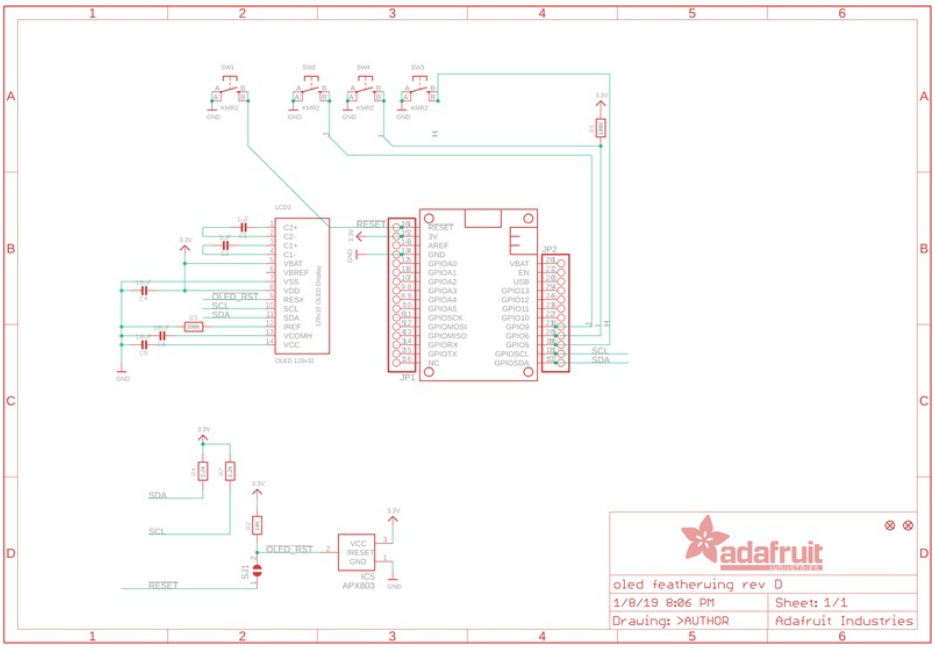
- Display is showing burn in on some pixels.

  The display can have image burn in for any pixels left on over a long period of time - many days. Try to avoid having the display on constantly for that length of time.

Download

- SSD1306 (https://adafru.it/aJK) Datasheet
- OLED UG-2832HSWEG02 Datasheet (https://adafru.it/qrf)
- PCB Files in EagleCAD format (https://adafru.it/nbc)
- Fritzing object available in the Adafruit Fritzing Library (https://adafru.it/aP3)

Schematics



Fabrication Print

Dimensions in inches