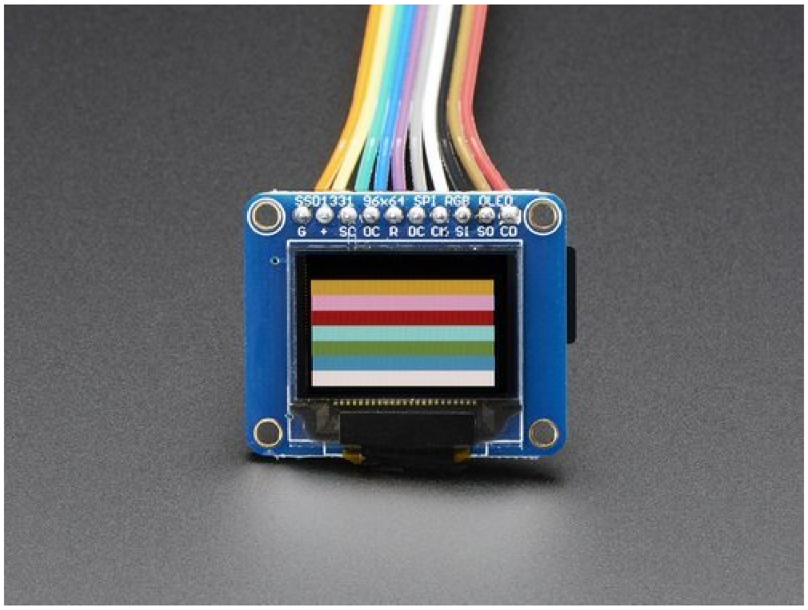
adafruit learning system

0.96" mini Color OLED

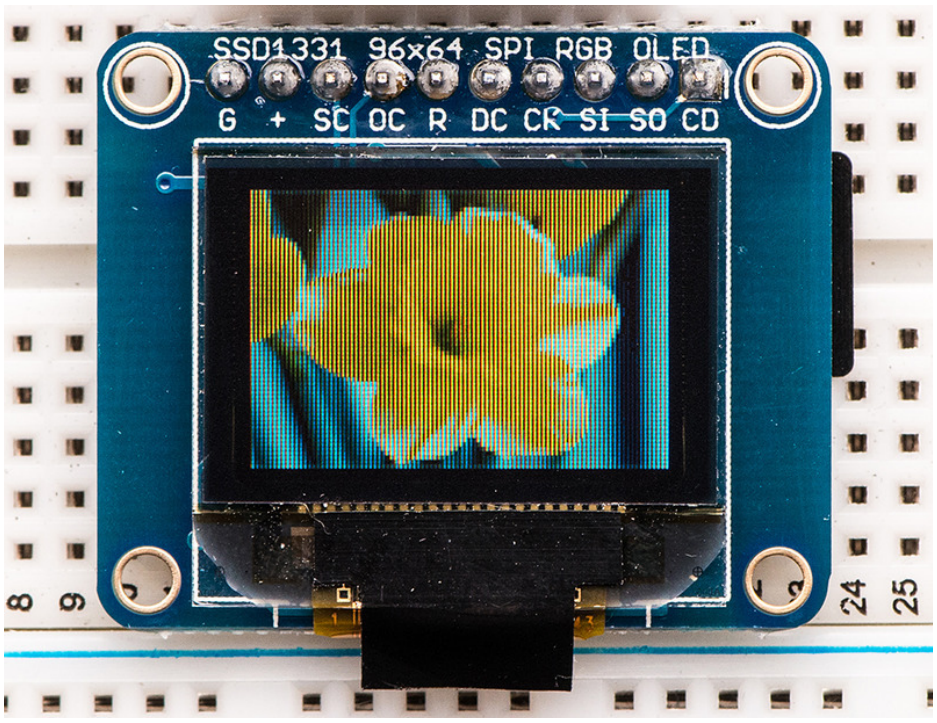
Created by lady ada



Last updated on 2019-08-15 04:18:39 PM UTC

Overview

We love our black and white monochrome displays but we also like to dabble with some color now and then. Our new 0.96" color OLED displays are perfect when you need an ultra-small display with vivid, high-contrast 16-bit color. The visible portion of the OLED measures 0.96" diagonal and contains 96x64 RGB pixels, each one made of red, green and blue OLEDs. Each pixel can be set with 16-bits of resolution for a large range of colors. Because the display uses OLEDs, there is no backlight, and the contrast is very high (black is really black). We picked this display for its excellent color, this is the nicest mini OLED we could find!



This OLED uses the SSD1331 driver chip, which manages the display. You can talk to the driver chip using either 3 or 4-wire write-only SPI (clock, data, chip select, data/command and an optional reset pin) or standard 8-bit parallel 8080/6800 which also permits reading pixel data from the display. Our example code shows how to use SPI since for such a display, its plenty fast. Inlcuded on the fully assembled breakout is the OLED display and a small boost converter (required for providing 12V to the OLED) and a microSD card holder. Our example code shows how to read a bitmap from the uSD card and display it all via SPI.

The logic levels for the microSD ard and OLED are 3.3V max. In order to make this breakout usable for bidirectional 8-bit and SPI interfaces, we left out an on-board level shifter. However, we include a DIP chip 75LVC245 8-bit level converter chip and our tutorial shows how to wire it to an Arduino so that you can use the breakout with 5V logic such as that of an Arduino. If you have a 3.3V logic level microcontroller system, you can skip the level shifter.

Of course, we wouldn't just leave you with a datasheet and a "good luck!" - we've written a full open source graphics library that can draw pixels, lines, rectangles, circles, text and bitmaps as well as example code and a wiring tutorial. The code is written for Arduino but can be easily ported to your favorite microcontroller!

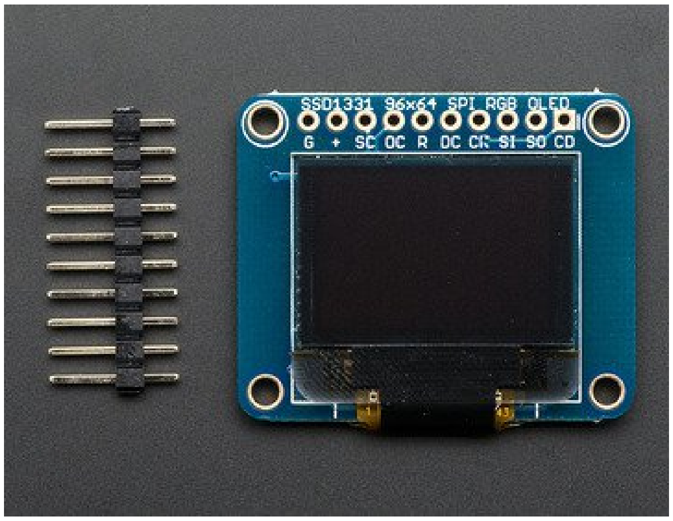
Pick one up today from the adafruit shop! (http://adafru.it/684)

Power

OLEDs tend to be fairly low power since they don't have a backlight, but they do require high voltage to drive the OLED segments. For this reason there is a boost converter on the back of the OLED. There's also a 3.3V regulator. However, we've found that it can be very sensitive to noisy 5V supplies such as that on an Arduino so try to run it off the 3.3V line which is filtered and cleaner. The power usage will vary with how many pixels are lit, the maximum is around 25mA.

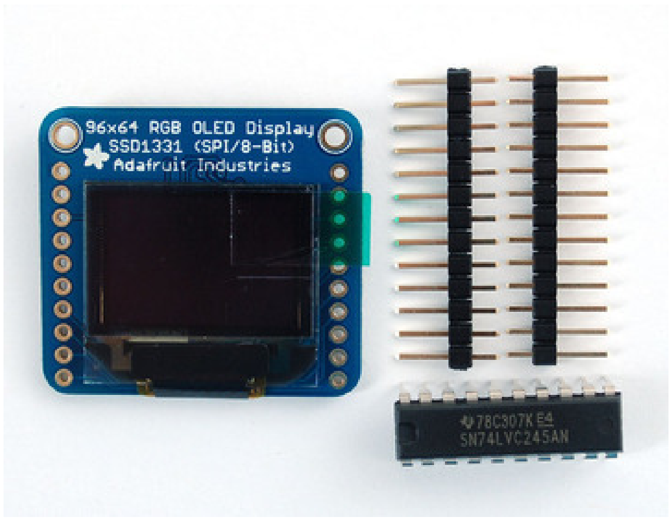
Of course, the microSD card is another matter - and may draw up to 150mA or more during writes (check the OEM document for the card to understand current usage!)

Wiring



New Model

If your display has a single row header across the top, it is the newer version. For wiring instructions, skip down to "Wiring Up the Newer Version"



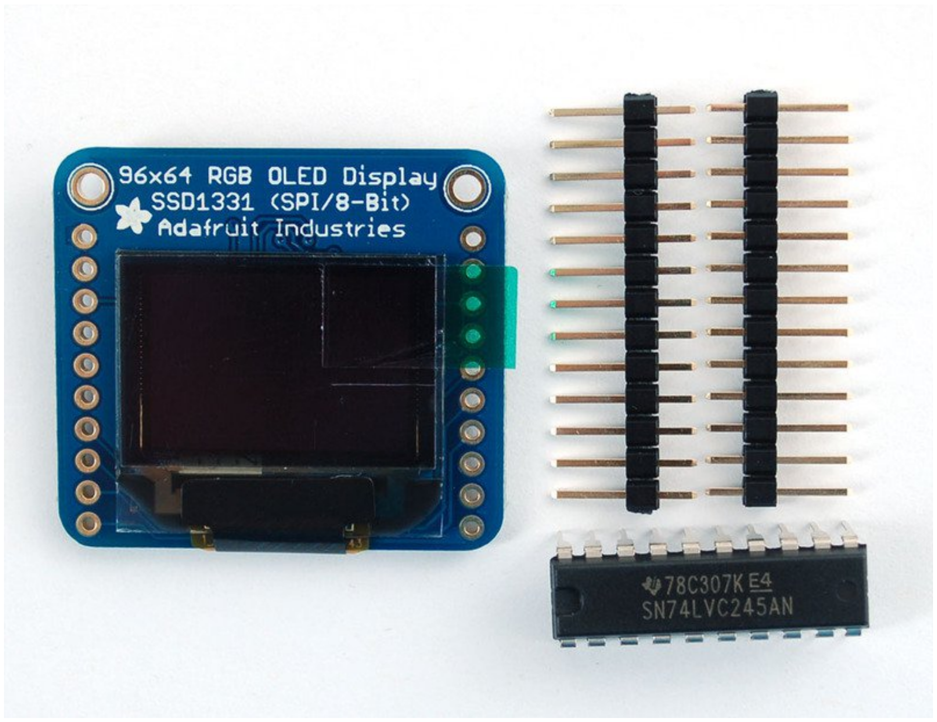
Older Model

If your display has a row of header pins down each side, it is the older model. See "Wiring the OLDER design" below.

Wiring the OLDER design (two rows of pins on either side)

The older breakout does not have a 5V level shifter on board, so its a little more complex to wire up!

The OLED module supports 3 methods of communication: 4 wire SPI, 8-bit parallel in 8080 and 6800 format. Since the display is small and we like to save pins, we'll be using the SPI protocol. Our tutorial, wiring and example code is all for SPI so if you need 8-bit, check the datasheets for details on how to wire up for 8-bit parallel.



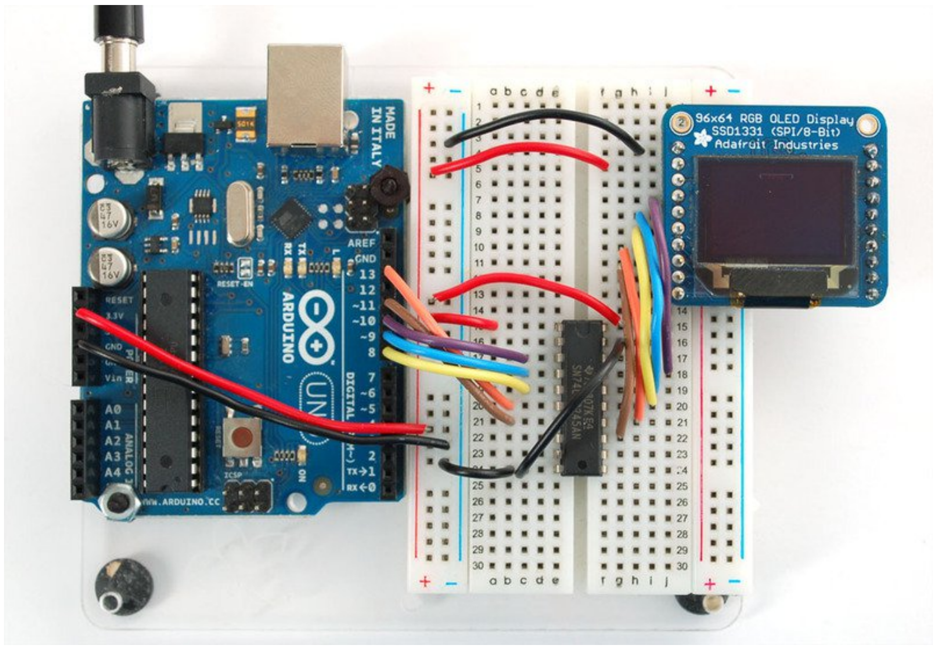
Since the OLED is 3.3V and also uses 3.3V logic, we need to use a logic shifter. We include a DIP logic shifter, the 74LVX245 with the OLED. If you're using a 3.3V logic chip, you can skip the logic shifter. Arduinos are all 5.0V so we'll be demonstrating that.

Don't forget to solder a piece of 0.1" header onto the left side of the OLED so you can plug it into a breadboard. You cannot skip this step, the header MUST be soldered in before plugging it in and wiring it up or it won't work!

Plug in the OLED and the '245 chip. The Chip has the notch closest to the OLED. Click on the image to see a large photo if you need help orienting

Starting from the top pin of the OLED (closest to the Adafruit flower) Connect the following OLED pins:

- Common ground - black wire
- 3.3V (red wires from the Arduino)
- **SD CS Pin**- don't connect (microSD card, we'll get to this later)
- **OLED CS Pin** - purple wire - 74LVC245 pin #17
- **OLED Reset Pin** - blue wire - 74LVC245 pin #16
- **OLED D/C Pin** - yellow wire - 74LVC245 pin #15
- **OLED SCLK Pin** - orange wire - 74LVC245 pin #14
- **OLED DATA Pin** - brown - 74LVC245 pin #13
- **SD Detect Pin**- not used, don't connect. Later on, if you wish, you can use this pin to detect if a card is inserted, it will be shorted to ground when a card is in the holder



Next we'll connect the remaining 74LVC245 pins to the Arduino

- Pin #1 goes to 3.3V (red wire)
- Skip
- Purple wire - goes to Digital #10
- Blue wire - goes to Digital #9
- Yellow wire - goes to Digital #8
- Orange wire - goes to Digital #13
- Brown wires - goes to Digital #11
- Skip
- Skip
- Connect to common ground

Then connect pin #20 of the 74LVC245 to 3.3V and pin #19 to Ground.

Digital #12 isn't used yet (we'll connect this to the SD card later

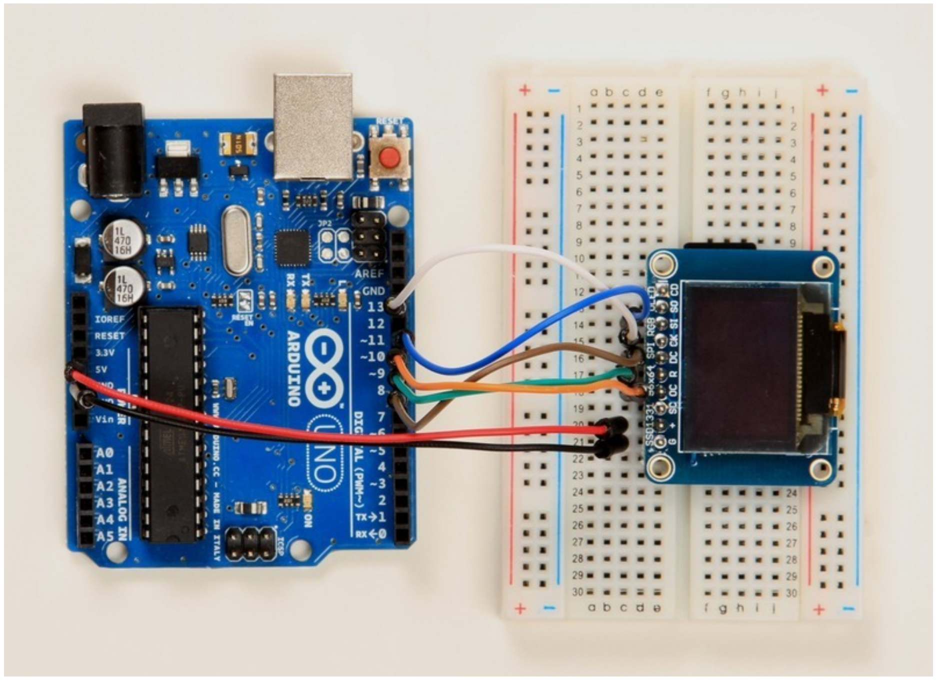
Wiring up the newer version (With one row of pins on top)

The updated 5v ready version of this display includes on-board level-shifting. So the 74LVC245 chip is not required and the wiring is much simpler! For the level shifter we use the CD74HC4050 (https://adafru.it/Boj) which has a typical propagation delay of ~10ns

The full pin names are marked on the back of the board, but there are abbreviations on the front to help identify pins when it is plugged into the breadboard. The chart below lists the full pin name, the abbreviated name (in parentheses) and the Arduino pin name to connect it to. Wire colors are as shown in the photo.

- **GND** (G) - Gnd (Black Wire)
- **VCC** (+) - 5v (Red Wire)
- **SDCS** (SC) - skip
- **OCS** (OC) - Digital #10 (Orange Wire)

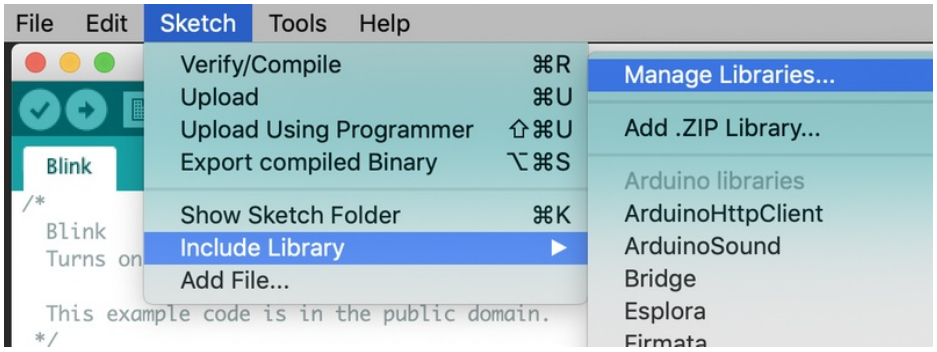
- **RST** (R) - Digital #9 (Green Wire)
- **D/C** (DC) - Digital #8 (Brown Wire)
- **SCK** (CK) - Digital #13 (White Wire)
- **MOSI** (SI) - Digital #11 (Blue Wire)
- **MISO** (SO) - skip
- **CD** (CD) - skip



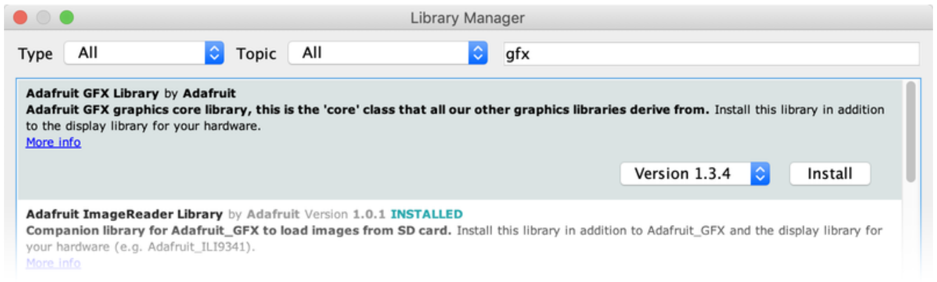
Installing and running Arduino software

Now we can run the test software on the Arduino. We'll need to download the library first and install it

Three libraries need to be installed using the **Arduino Library Manager**…this is the preferred and modern way. From the Arduino “Sketch” menu, select “Include Library” then “Manage Libraries…”



Type “gfx” in the search field to quickly find the first library — **Adafruit_GFX**:

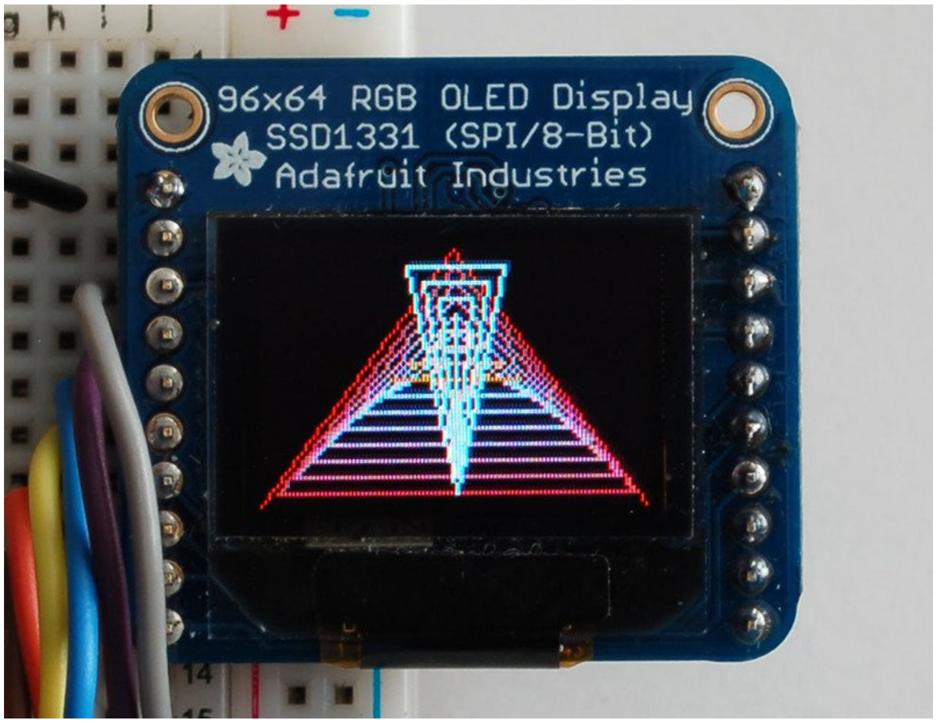


Adafruit GFX Library by **Adafruit**
Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from. Install this library in addition to the display library for your hardware.
More info

Adafruit ImageReader Library by Adafruit Version **1.0.1** **INSTALLED**
Companion library for Adafruit_GFX to load images from SD card. Install this library in addition to Adafruit_GFX and the display library for your hardware (e.g. Adafruit_ILI9341).
More info

Repeat the search and install steps, looking for the **Adafruit_ZeroDMA** and **Adafruit_SSD1331** libraries.

After you restart, you should be able to select **File→Examples→Adafruit_SSD1331→test** - this is the example sketch that just tests the display by drawing text and shapes. Upload the sketch and you should see the following:



If alls working, then you can start looking through the test sketch for demonstrations on how to print text, circles, lines, etc.

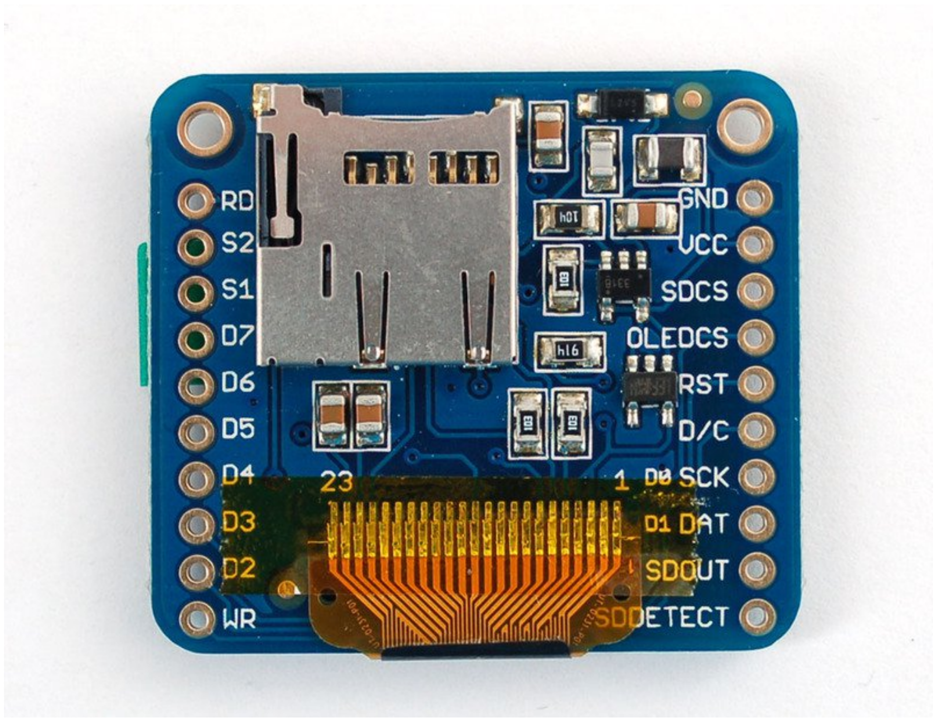
For a detailed tutorial on the Adafruit GFX library, including all the functions available please visit the GFX tutorial page (https://adafru.it/aPx)

Drawing Bitmaps

We have an example sketch in the library showing how to display full color bitmap images stored on an SD card. You'll need a microSD card such as this one . This example will only work for Arduino v1.0 and later.

You'll also need an image. We suggest starting with this bitmap of a flower If you want to later use your own image, use an image editing tool and crop your image to no larger than 64 pixels high and 96 pixels wide. Save it as a 24-bit color **BMP** file - it must be 24-bit color format to work, even if it was originally a 16-bit color image - becaue of the way BMPs are stored and displayed!

Copy the **violet.bmp** to the microSD card and insert it into the back of the breakout board.

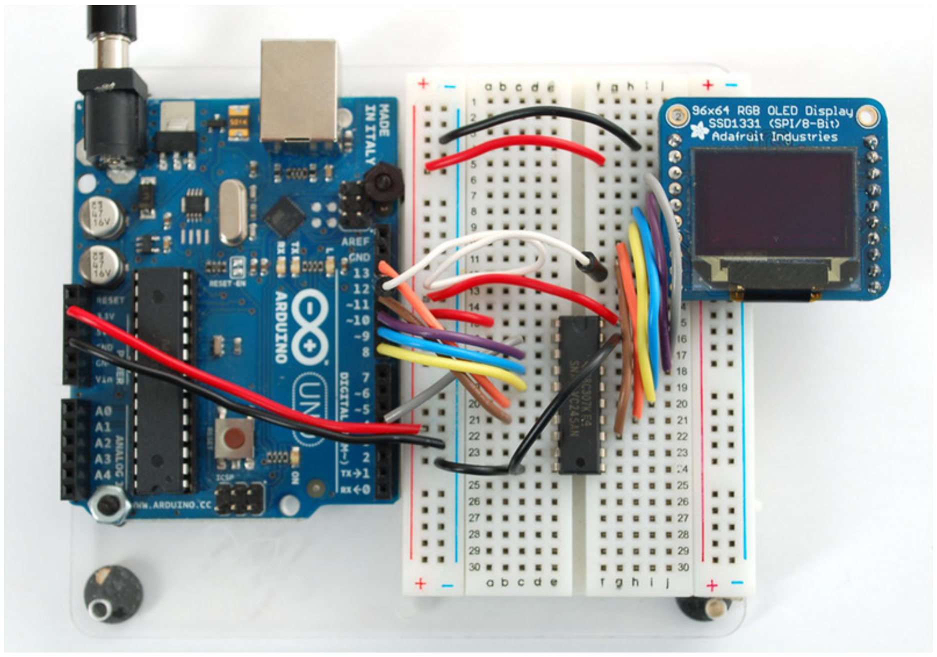


We'll have to add an extra 2 wires so we can 'select' and 'receive data' from the SD card

Old Style Board: (two rows of pins)

Connect the third pin **SD ChipSelect** of the OLED (gray wire) to pin #18 of the 74LVC245. Then connect pin #2 of the 74LVC245 to Arduino **Digital #4**. This is the pin to select that we want to talk to the microSD card.

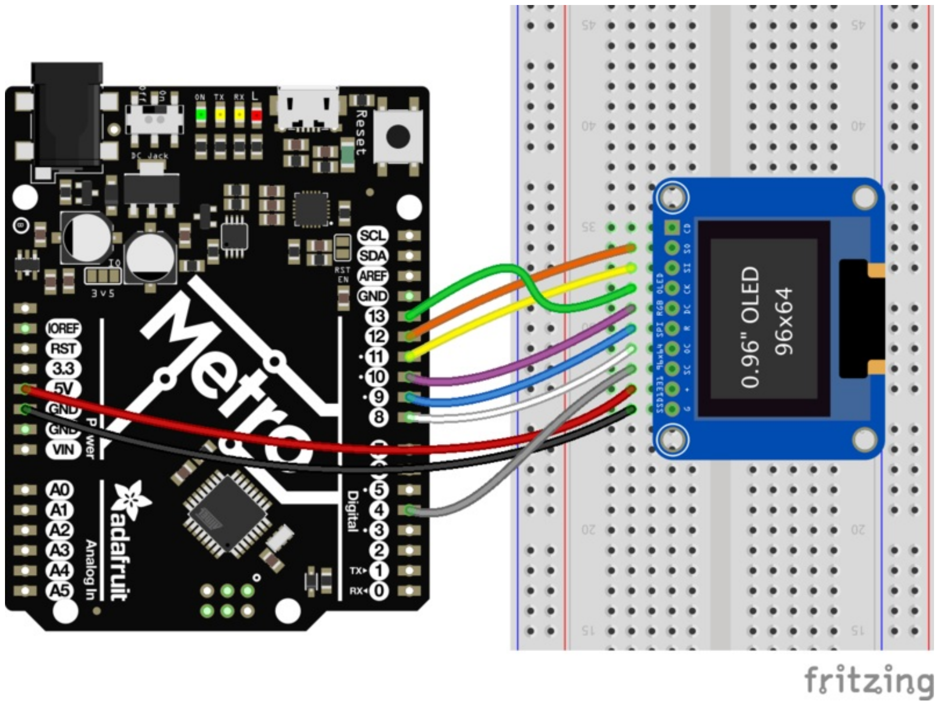
Then connect second-from-the bottom pin of the OLED - **SDOUT** - with a wire directly to **Arduino Digital #12** this is the longer white wire shown - this wire does not need to be level shifted.



New Style Board: (Single row of pins)

Add the following two connections between the breakout board and the Arduino:

- **SDCS** (SC) - Digital #4.(Gray Wire)
- **MISO** (SO) - Digital #12 (Orange Wire)

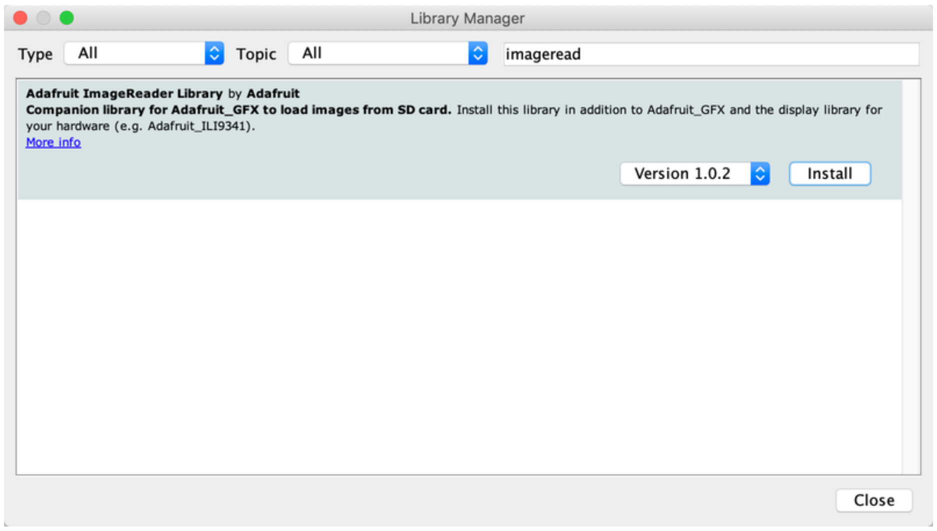


Bitmap Example Sketch

To display bitmaps from the on-board micro SD slot, you will need a micro SD card (http://adafru.it/102) formatted **FAT16 or FAT32** (they almost always are by default).

There is a built in microSD card slot on the rear of the breakout and we can use that to load bitmap images!

It's really easy to draw bitmaps. We have a library for it, Adafruit_ImageReader, which can be installed through the Arduino Library Manager (Sketch→Include Library→Manage Libraries…). Enter “imageread” in the search field and the library is easy to spot:

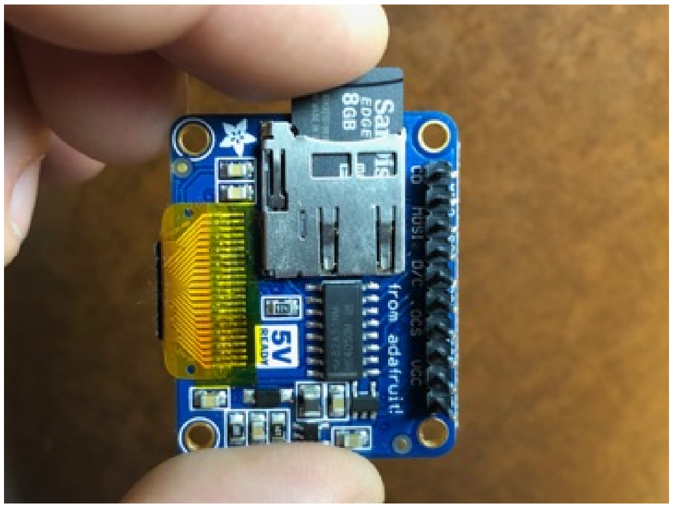


Adafruit ImageReader Library by **Adafruit**
Companion library for Adafruit_GFX to load images from SD card. Install this library in addition to Adafruit_GFX and the display library for your hardware (e.g. Adafruit_ILI9341).
More info

Next you can either download the image here or copy it from the images folder from inside the library files.

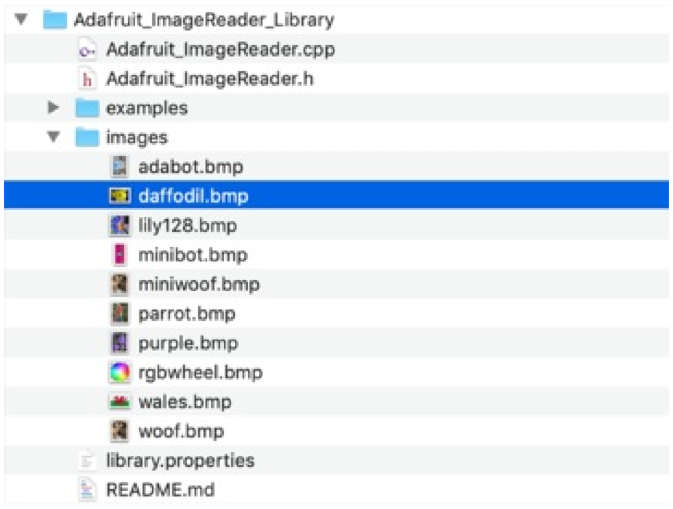
https://adafru.it/EUL

https://adafru.it/EUL



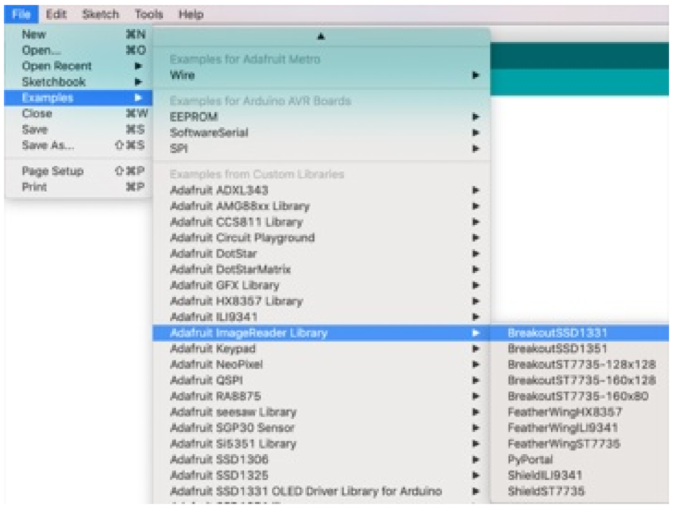
Insert the card

Insert the micro SD card into the slot on the back of the SSD1331 breakout board.



Copy the bitmap file

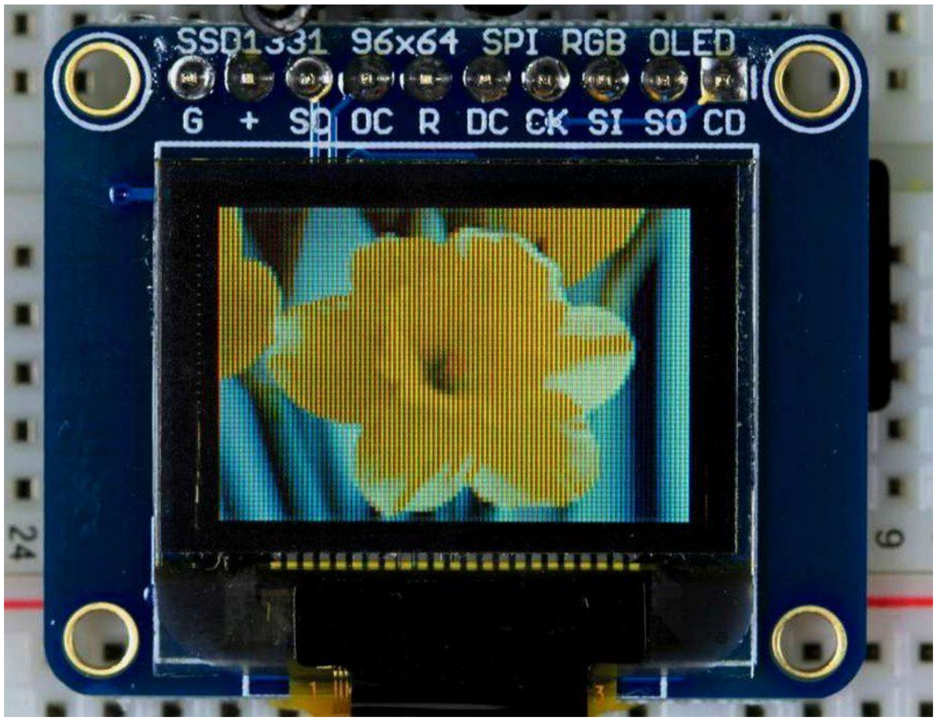
Copy the file **"daffodil.bmp"** from the Adafruit_ImageReader_Library\images folder (or wherever you saved it if you downloaded the file) over to the root directory of your micro-SD card.



Load the bitmap example sketch

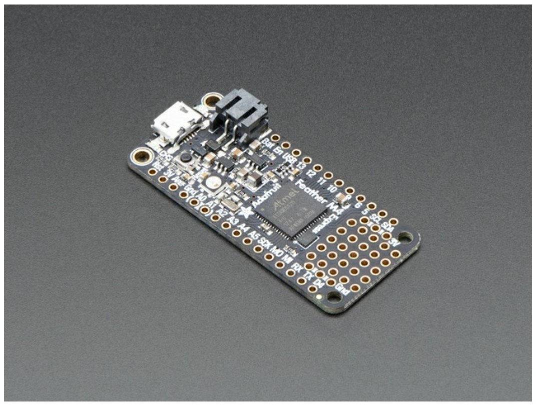
Select **"Examples->Adafruit_ImageReader_Library->BreakoutSSD1331"** and upload it to your Arduino.

You should see the daffodil! If you don't see it, check the Serial Monitor for hints on what might have gone wrong (maybe the microSD card had an issue).



CircuitPython Displayio Quickstart

You will need a board capable of running CircuitPython such as the Metro M0 Express or the Metro M4 Express. You can also use boards such as the Feather M0 Express or the Feather M4 Express. We recommend either the Metro M4 or the Feather M4 Express because it's much faster and works better for driving a display. For this guide, we will be using a Feather M4 Express. The steps should be about the same for the Feather M0 Express or either of the Metros. If you haven't already, be sure to check out our Feather M4 Express (https://adafru.it/EEm) guide.



Adafruit Feather M4 Express - Featuring ATSAMD51

$22.95
IN STOCK

ADD TO CART

Preparing the Breakout

Before using the TFT Breakout, you will need to solder the headers or some wires to it. Be sure to check out the Adafruit Guide To Excellent Soldering (https://adafru.it/drl). After that the breakout should be ready to go.

Required CircuitPython Libraries

To use this display with `displayio`, there is only one required library.

https://adafru.it/Fxd

https://adafru.it/Fxd

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

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

Remember for non-express boards, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_ssd1331**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_ssd1331** file copied over.

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/FiA

https://adafru.it/FiA

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.

CircuitPython Code Example

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

import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_ssd1331 import SSD1331

spi = board.SPI()
tft_cs = board.D5
tft_dc = board.D6

displayio.release_displays()
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.D9)

display = SSD1331(display_bus, width=96, height=64)

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

color_bitmap = displayio.Bitmap(96, 64, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green

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(86, 54, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=5, y=5)
splash.append(inner_sprite)

# Draw a label
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=12, y=32)
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_ssd1331` driver.

```
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_ssd1331 import SSD1331
```

Next, we set the SPI object to the board's SPI with the easy shortcut function `board.SPI()`. By using this function, it finds the SPI module and initializes using the default SPI parameters.

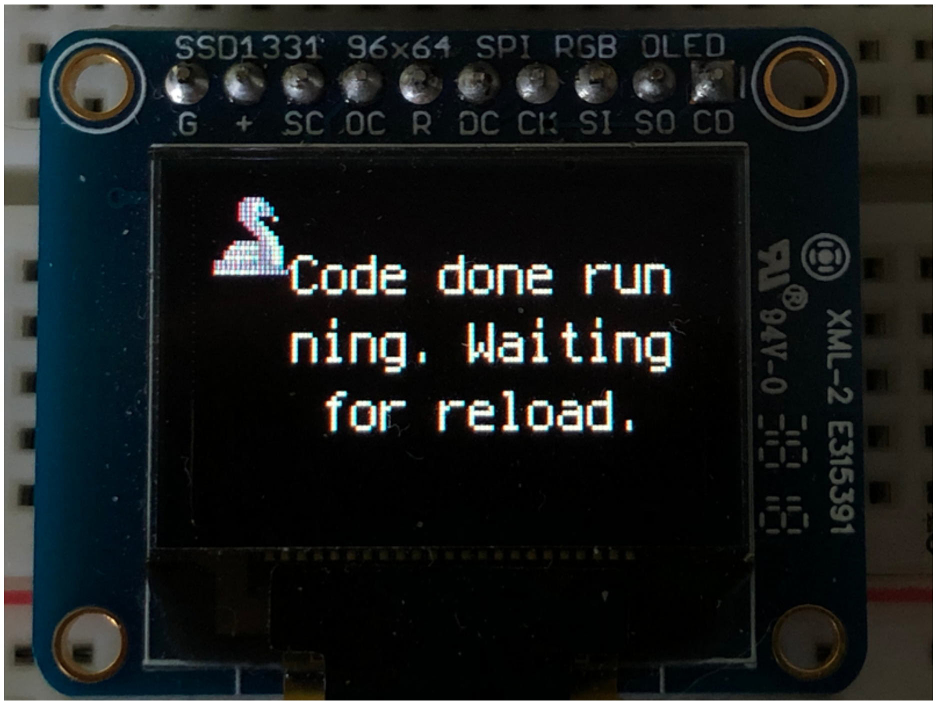
```
spi = board.SPI()
tft_cs = board.D5
tft_dc = board.D6
```

In the next two lines, we release the displays. This is important because if the Feather is reset, the display pins are not automatically released and this makes them available for use again. We set the display bus to FourWire which makes use of the SPI bus.

```
displayio.release_displays()
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.D9)
```

Finally, we initialize the driver with a width of 96 and a height of 64. 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 = SSD1331(display_bus, width=96, height=64)
```


SSD1331 96x64 SPI RGB OLED
G + SC OC R DC CK SI SO CD
Code done running. Waiting for reload.
XML-2 E315391
94V-0

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.

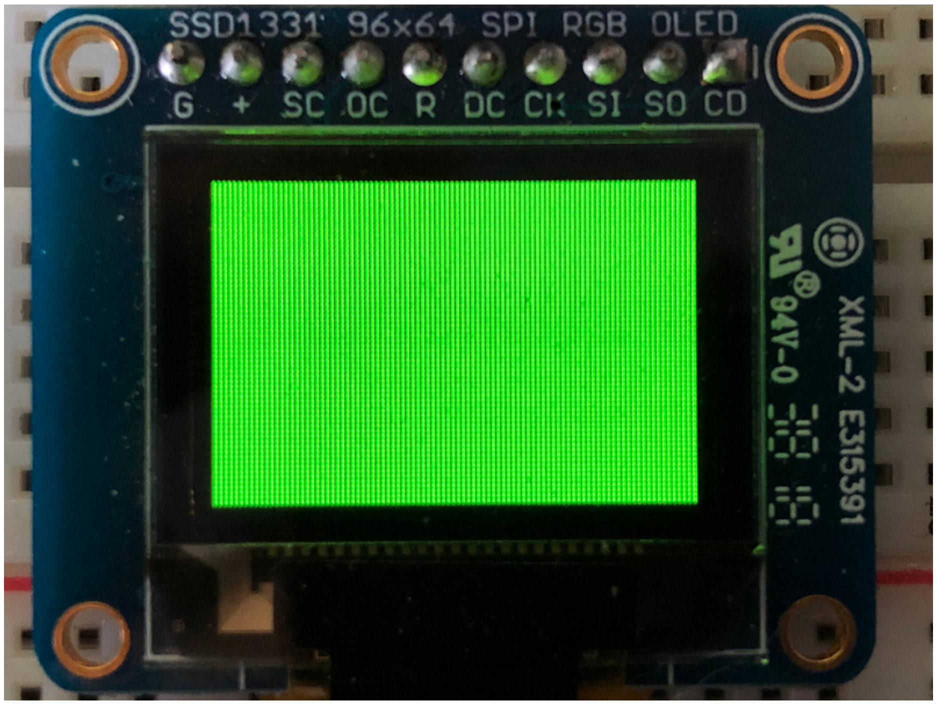
```
splash = displayio.Group(max_size=10)
display.show(splash)
```

Next we create a Bitmap which 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. The Bitmaps can currently handle up to 256 different colors. We create a Palette with one color and set that color to 0x00FF00 which happens to be green. Colors are Hexadecimal values in the format of RRGGBB. Even though the Bitmaps can only handle 256 colors at a time, you get to define what those 256 different colors are.

```
color_bitmap = displayio.Bitmap(96, 64, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

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)
```

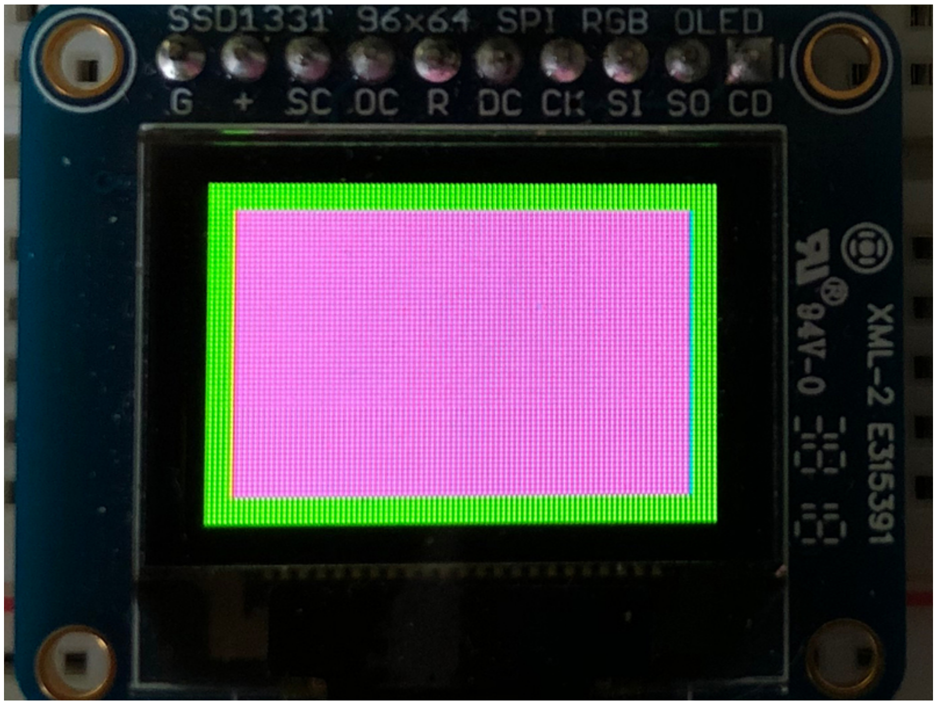


Next we will create a smaller purple square. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case, we will create a bitmap that is 5 pixels smaller on each side. The screen is **96x64**, 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(86, 54, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=5, y=5)
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 let's 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 32 for the Y coordinate, and around 12 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 yellow text, so we'll pass it a value of `0xFFFF00`.

```
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=12, y=32)
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
```

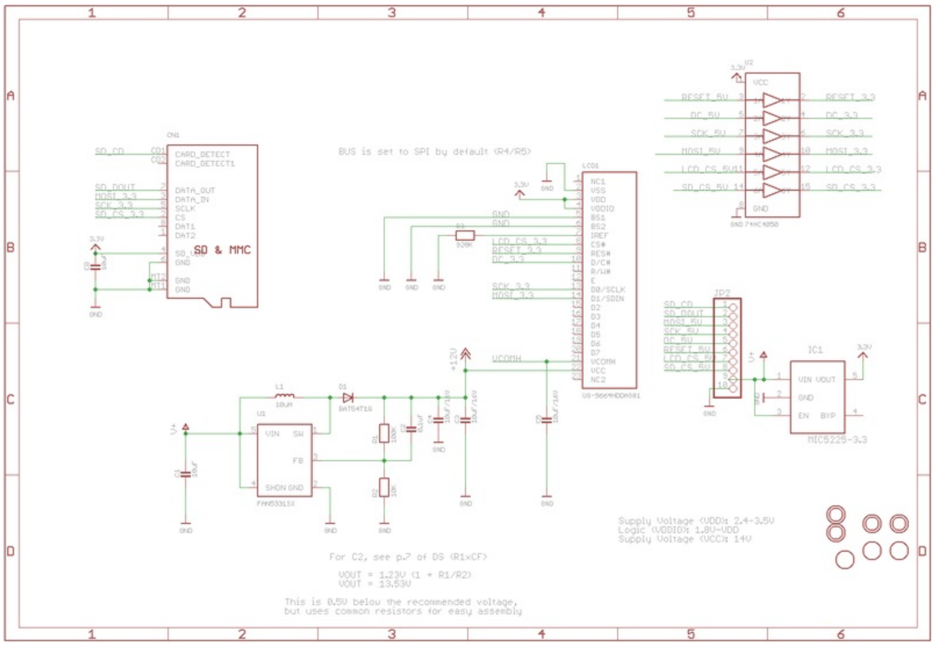
Where to go from here

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

Files

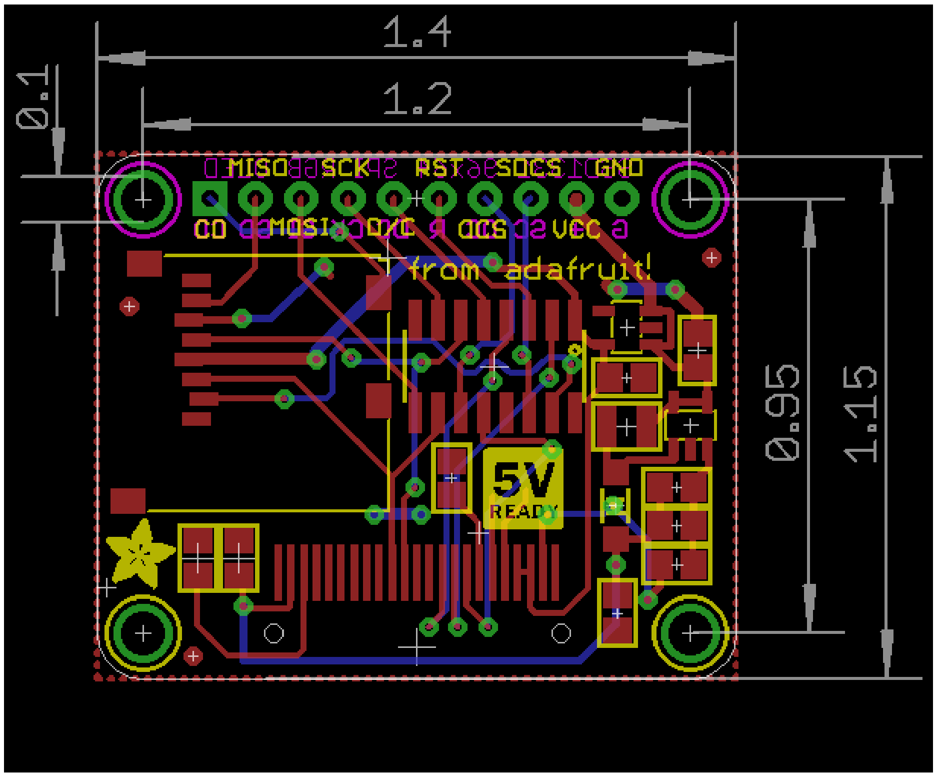
- Datasheet for the SSD1331 driver (https://adafru.it/aJb)
- Datasheet for the raw OLED module itself (https://adafru.it/aJc)
- Adafruit SSD1331 Arduino Library repository (https://adafru.it/aHp)
- You'll also have to install the Adafruit GFX graphics core library at this github repo (https://adafru.it/aJa) and install it after you've gotten the OLED driver library.
- Adafruit breakout board PCB files (https://adafru.it/aJd)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/aP3)

Schematic & Fabrication Print



BUS is set to SPI by default (R4/R5)
SD & MMC
Supply Voltage (VDD): 2.4-3.5V
Logic (VDDIO): 1.8V-VDD
Supply Voltage (VCC): 14V
For C2, see p.7 of DS (R1xCF)
VOUT = 1.23V (1 + R1/R2)
VOUT = 13.53V
This is 0.5V below the recommended voltage, but uses common resistors for easy assembly
MIC5225-3.3
FAN5331SX

For the level shifter we use the CD74HC4050 (https://adafru.it/Boj) which has a typical propagation delay of ~10ns


1.4
1.2
0.1
0.95
1.15
MISO
SCK
RST
SDCS
GND
CD
MOSI
D/C
OCS
VCC
from adafruit!
5V
READY