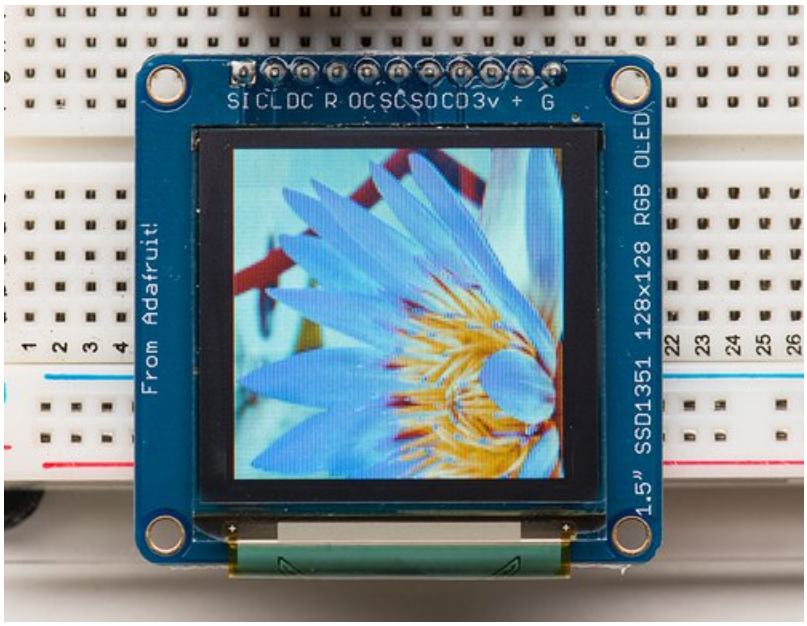
adafruit learning system

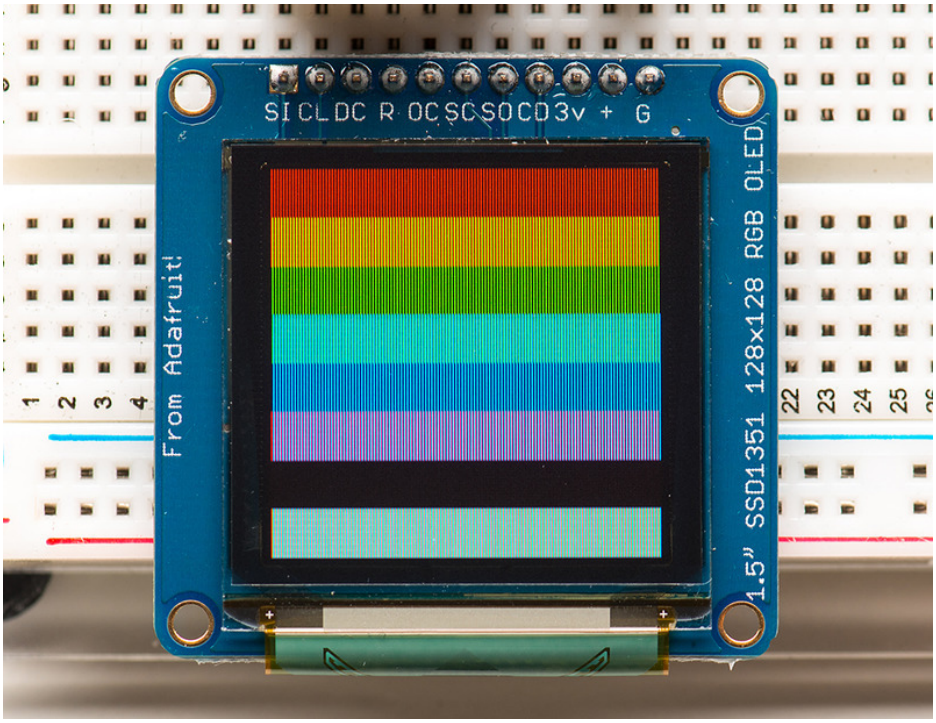
Adafruit 1.27" and 1.5" Color OLED Breakout Board

Created by Bill Earl



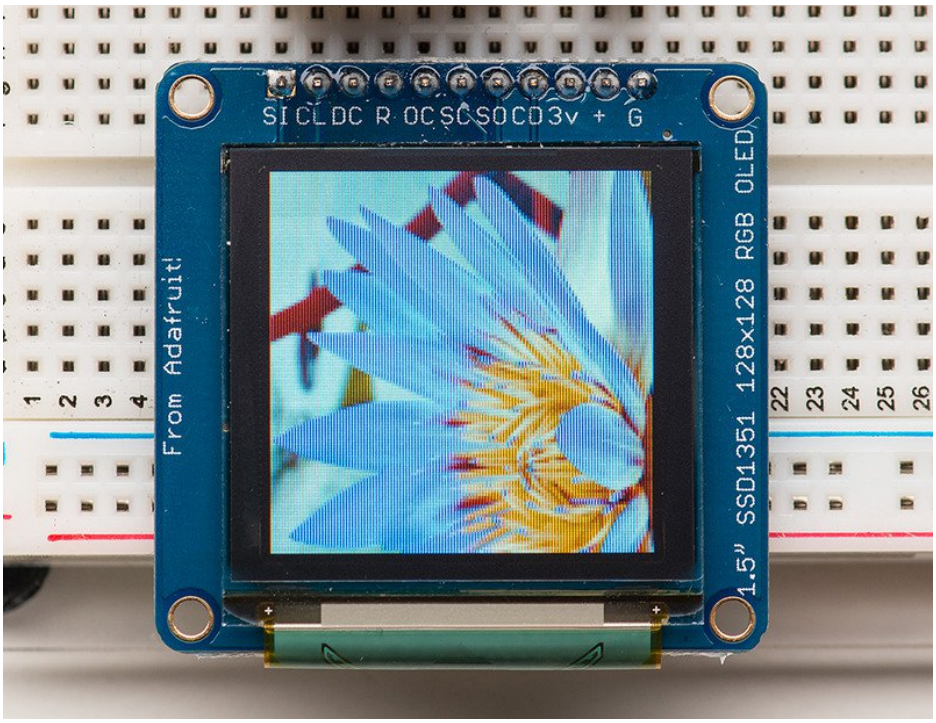
Last updated on 2019-08-15 04:09:37 PM UTC

Overview



We love our black and white monochrome displays but we also like to dabble with some color now and then. Our big 1.5" color OLED displays are perfect when you need a small display with vivid, high-contrast 16-bit color. The visible portion of the OLED measures 1.5" diagonal and contains 128x128 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!

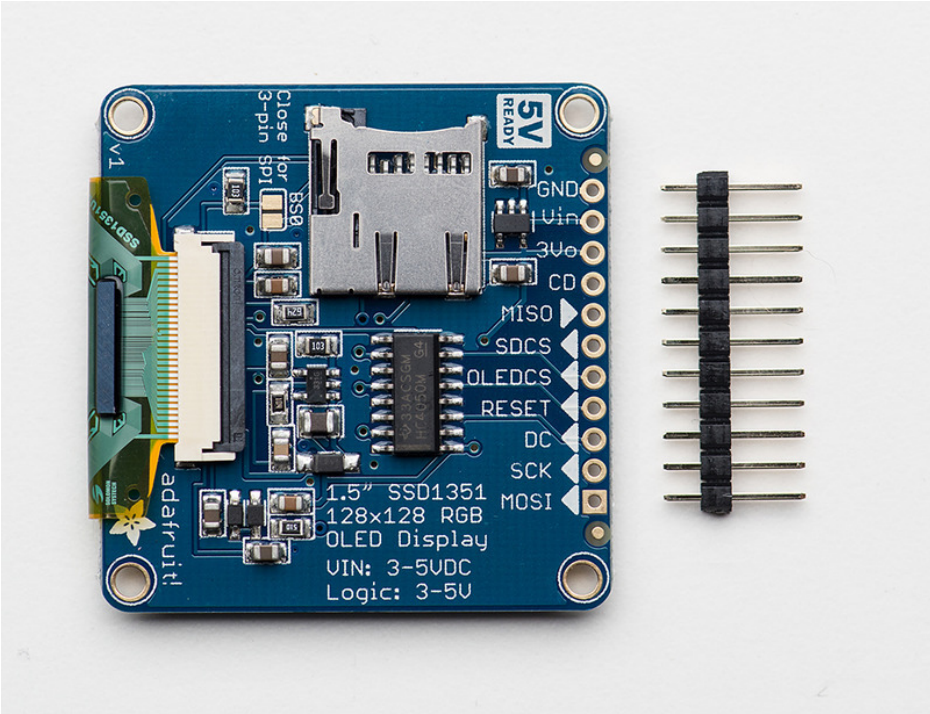
This OLED uses the SSD1351 driver chip, which manages the display. You can talk to the driver chip using 4-wire write-only SPI (clock, data, chip select, data/command and an optional reset pin). Included on the fully assembled breakout is the OLED display and a small boost converter (required for providing 12V to the OLED) and a microSD card holder. This design includes built-in logic level shifting so you can use it with 3-5VDC power and logic levels. Our example code shows how to read a bitmap from the uSD card and display it all via SPI.

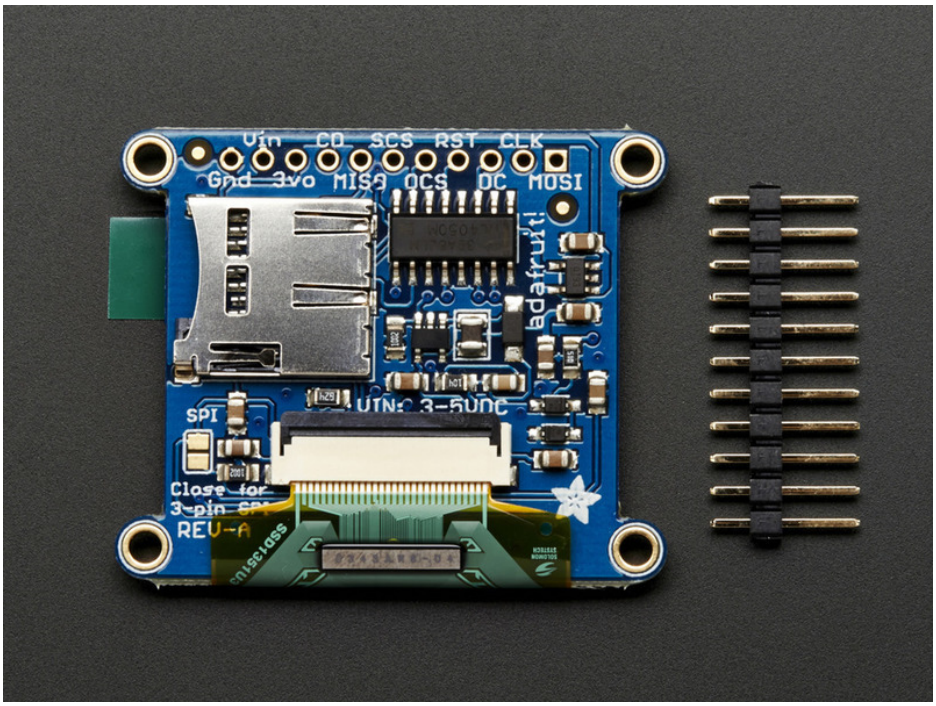


Board Technical Details

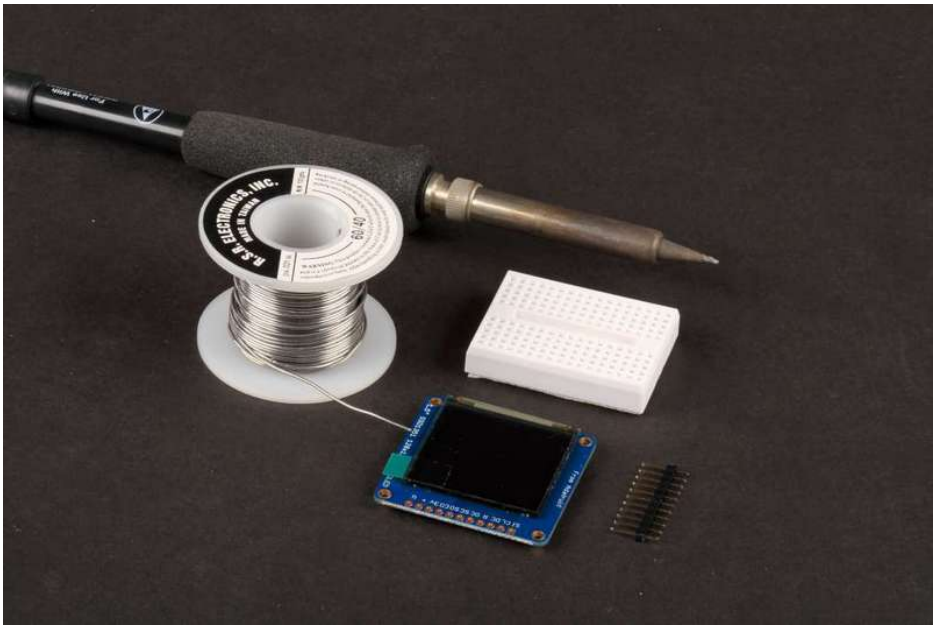
- 1.5" diagonal OLED, 16-bit color
- SPI interface
- 3.3-5V logic and power
- Micro-SD card holder
- Dimensions: 43.17mm / 1.7" x 42mm / 1.65" x 5.42mm / 0.2"

Assembly



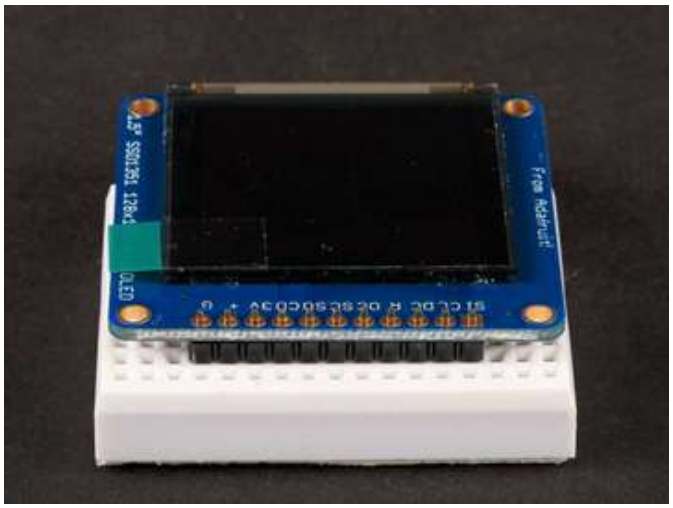


The breakout board comes fully assembled and tested. We include an optional strip of header pins to make it easier to use this display in a breadboard. The header can be installed in just a few minutes with your soldering iron:



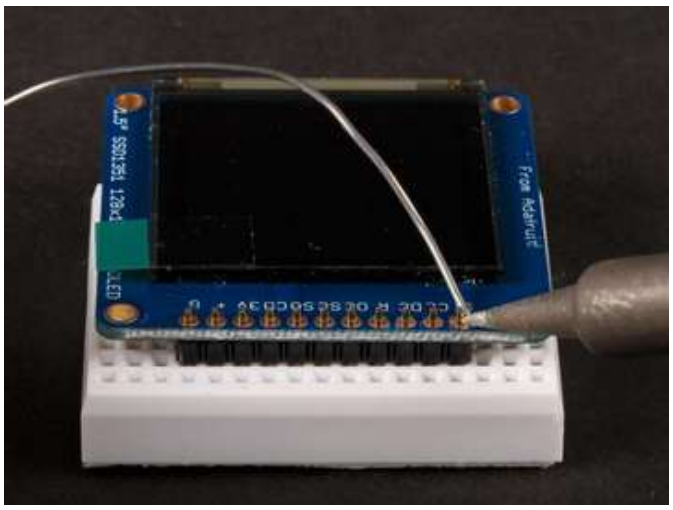
Prepare the header strip

Cut the header to size and insert (long pins down) into a breadboard to stabilize for soldering.



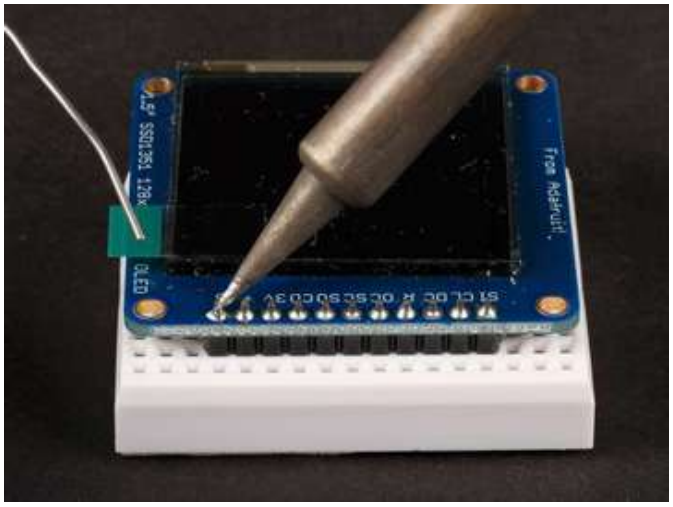
Position the display

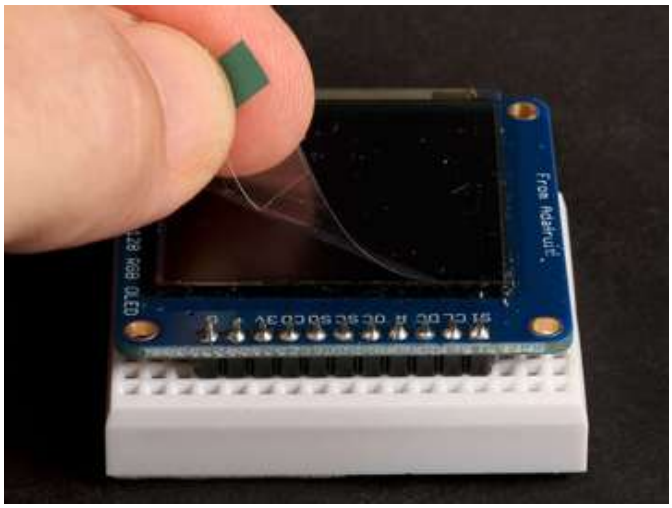
Place the display breakout on the header so that the short pins protrude through the holes.



And Solder!

Solder all pins to assure a good electrical connection.



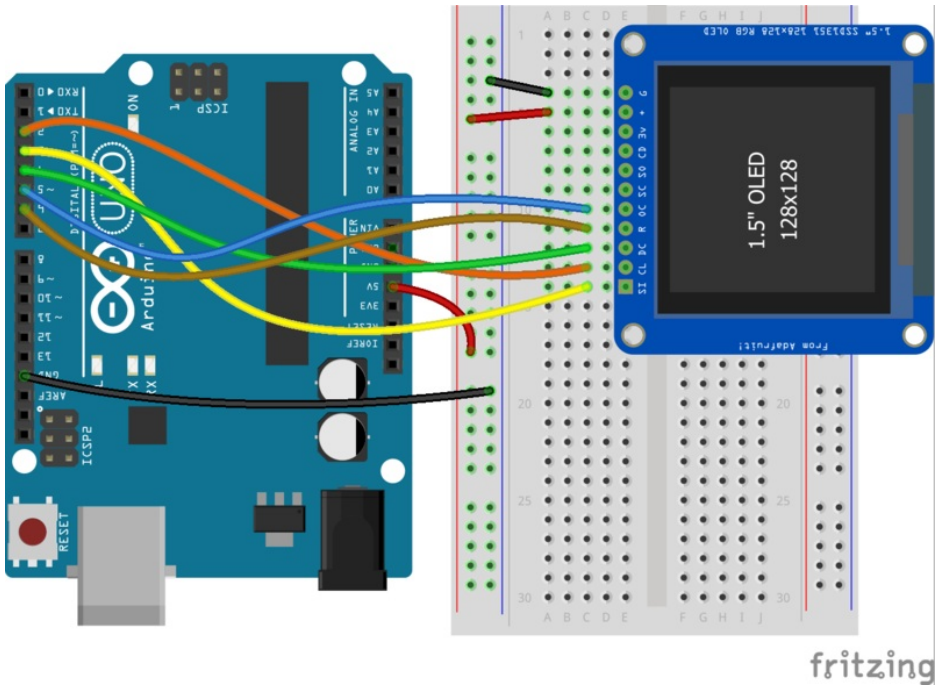


Remove the protective film

Gently pull up on the tab to remove the film.

Wiring and Graphics Test

The pinout ordering is the same for both the 1.27" and 1.5" version of the OLED!



https://adafru.it/sVa

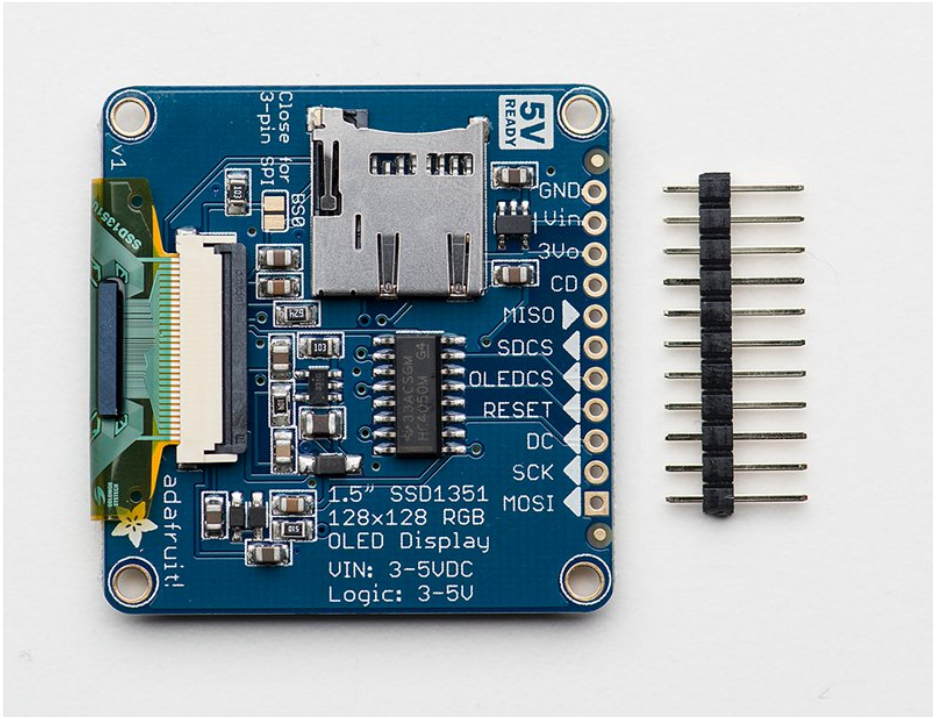
https://adafru.it/sVa

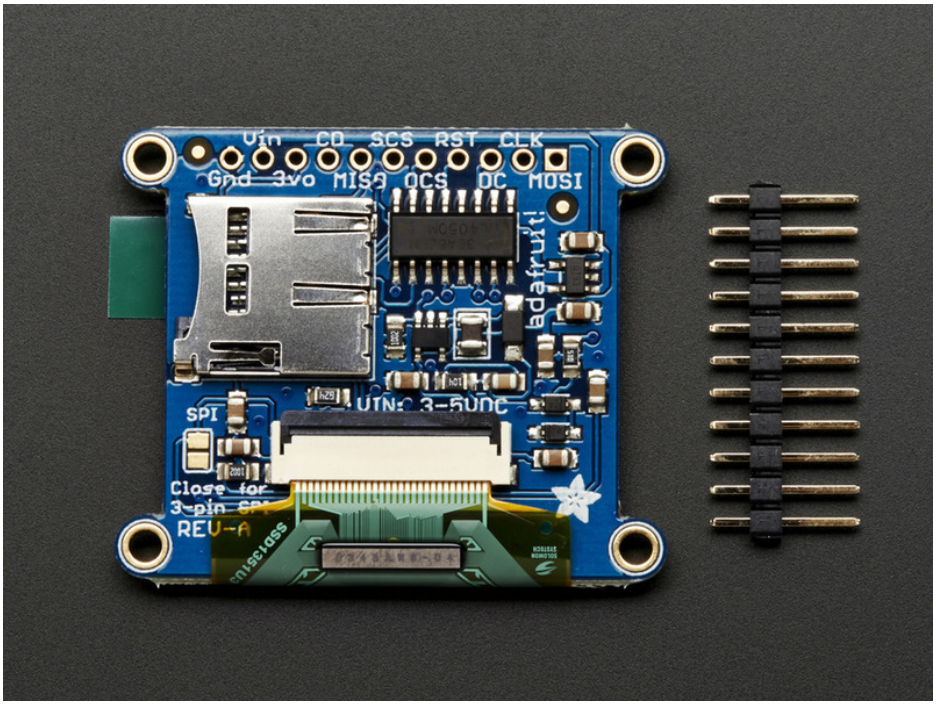
The library supports flexible wiring to minimize pin conflicts with other shields and breakouts. For the initial test, we'll use the same wiring as the "test" example from the library:

- **GND -> GND (G)**
- **5v -> VIN (+)**
- **#2 -> SCLK (CL)**
- **#3 -> MOSI (SI)**
- **#4 -> DC**
- **#5 -> OLEDCS (OC)**
- **#6 -> RST (R)**

Hint:

If you are confused by the abbreviations on the front of the board, the full signal names are printed on the back!

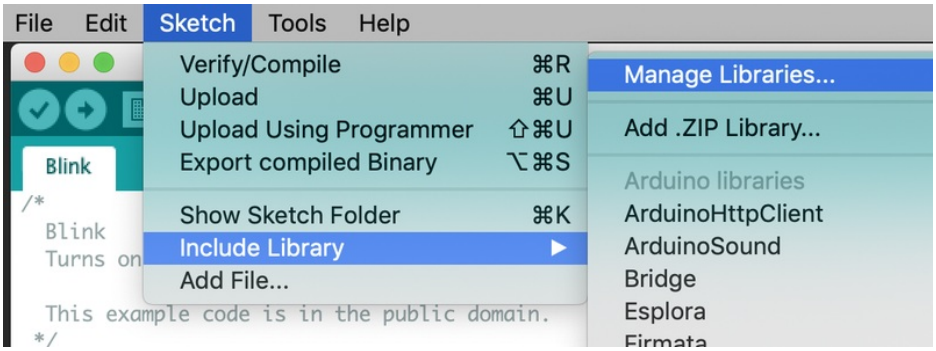




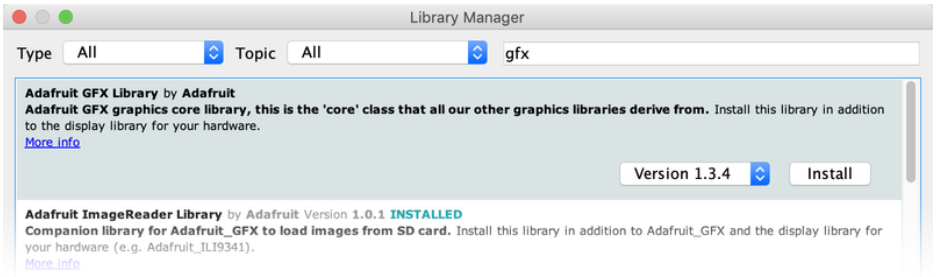
Installing the Arduino software

Now we can run the test software on the Arduino.

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**:

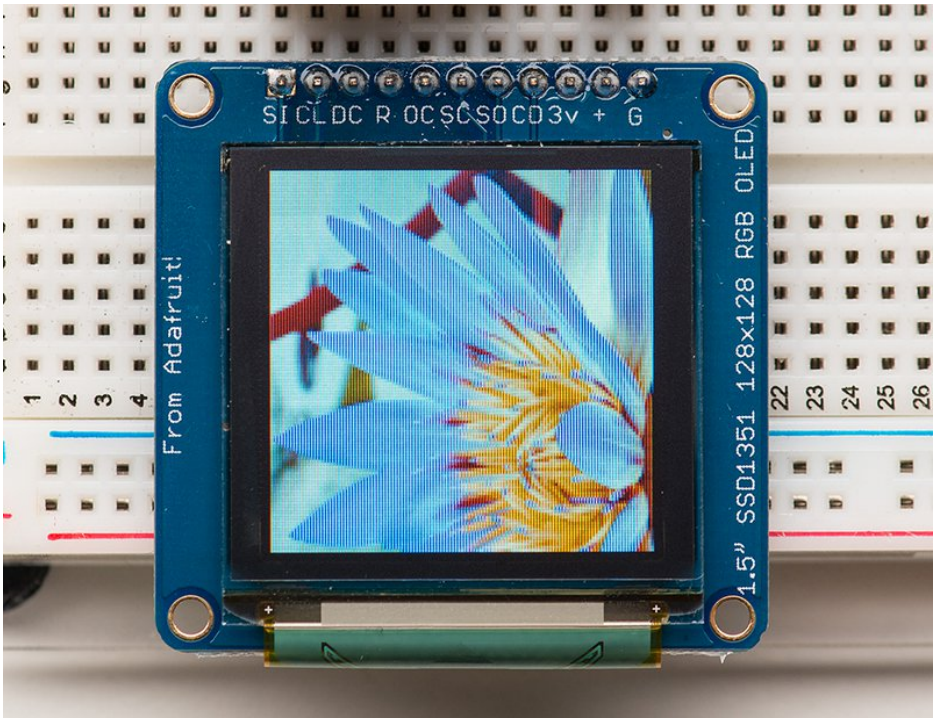


Repeat the search and install steps, looking for the **Adafruit_ZeroDMA** and **Adafruit_SSD1351** libraries.

After you restart, you should be able to select **File→Examples→Adafruit_SSD1351→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:

The test sketch demonstrates all the basic drawing functions of the Adafruit GFX Library. Read through the code to see how to draw 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)

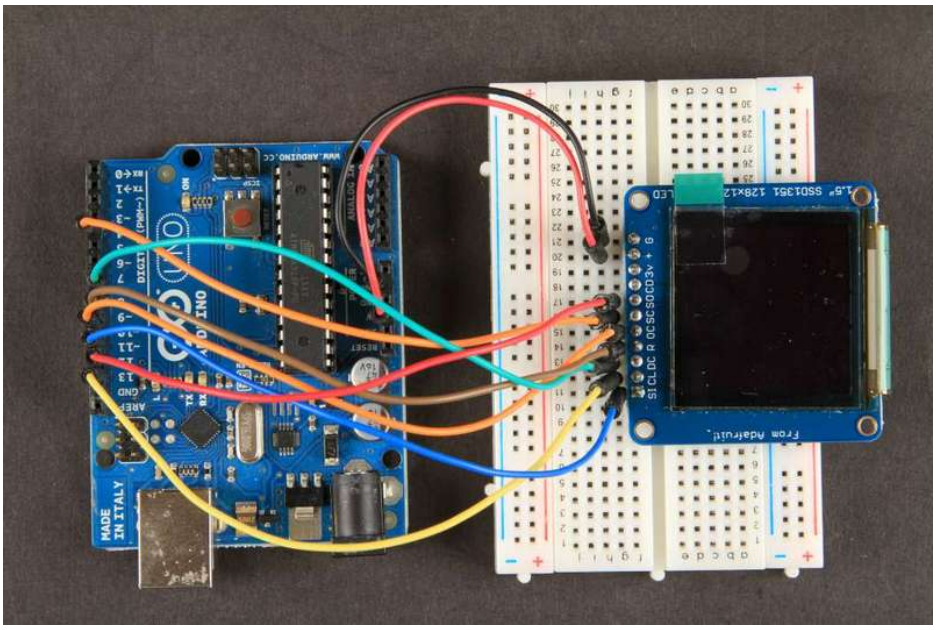


Wiring for the Bitmap Example

Drawing bitmaps from the on-board micro SD card requires a few more connections to communicate with the SD card. The library allows you to use any pins. The Arduino connections listed below match the code in the "bmp" example from the library:

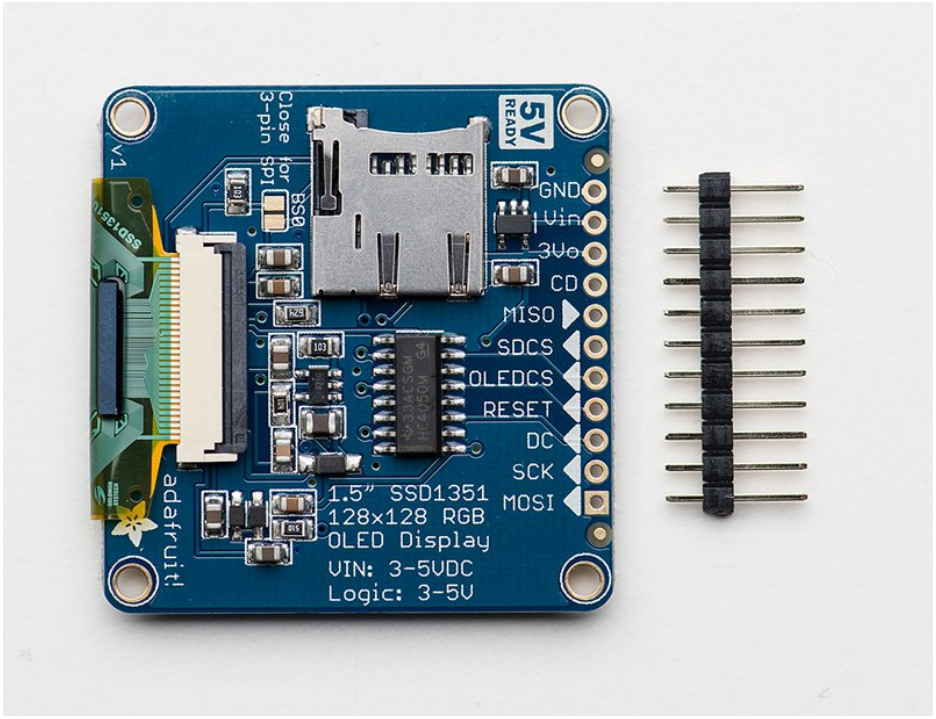
- **GND -> GND (G)**
- **5v -> VIN (+)**
- **#7 -> SDCS (SC)**
- **#4 -> DC**
- **#6 -> RST (R)**
- **#5 -> OLEDCS (OC)**
- **#11 -> MOSI (SI)**
- **#12 -> MISO (SO)**
- **#13 -> SCLK (CL)**

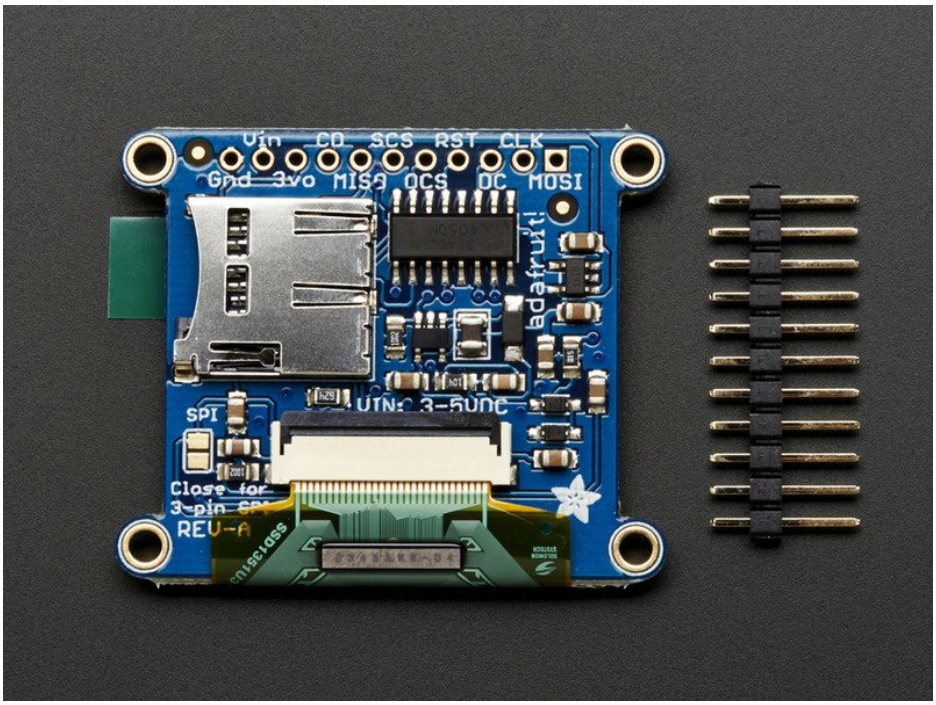
Note that the Bitmap example code uses hardware SPI wiring for maximum speed. You can still use software SPI, but make sure that the pin definitions match your wiring and that you modify the example to select the Software SPI option (#1) in the code. The SPI pins shown are for Atmega-328 processors. To use this wiring on other processors, software SPI must be used.



Hint:

If you are confused by the abbreviations on the front of the board, the full signal names are printed on the back!



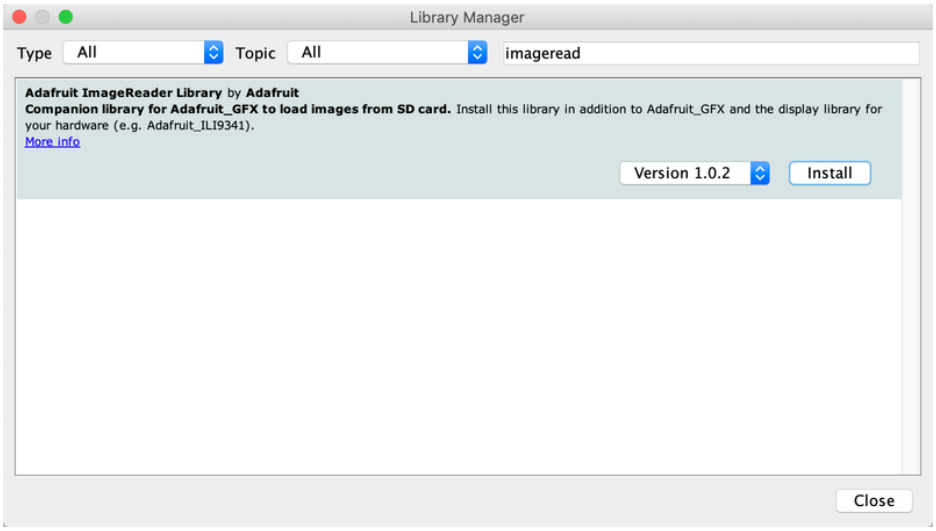


Bitmap Example Sketch

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!

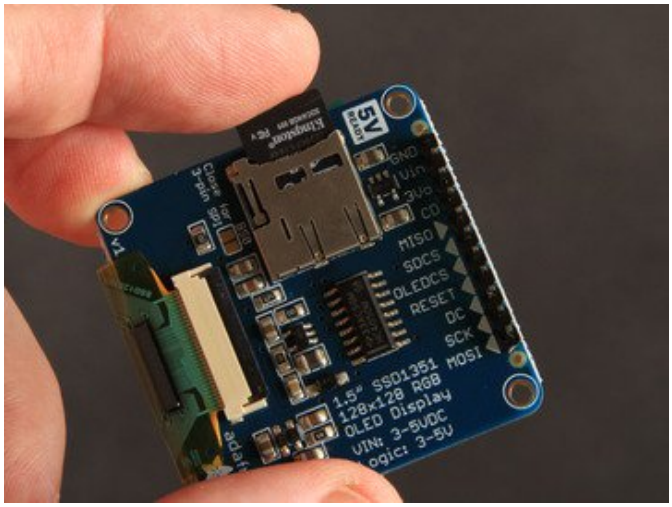
It's really easy to draw bitmaps. We have a library for it, Adafruit_ImageReader, which can be installed through the Arduino Library Manager (Sketch→Include Library→Manage Libraries...). Enter “imageread” in the search field and the library is easy to spot:



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

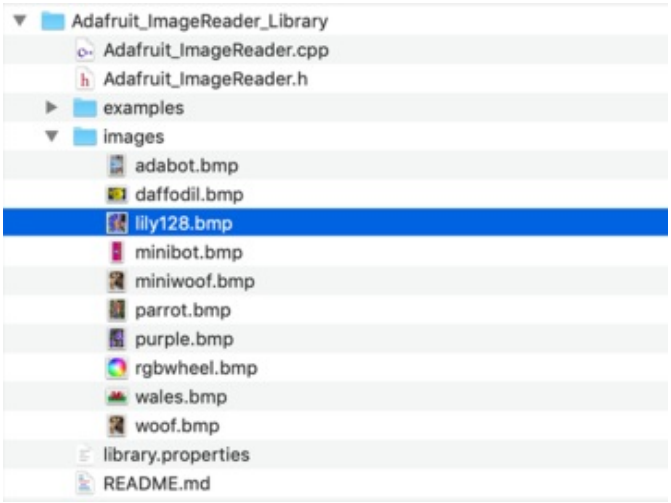
https://adafru.it/Ey0

https://adafru.it/Ey0



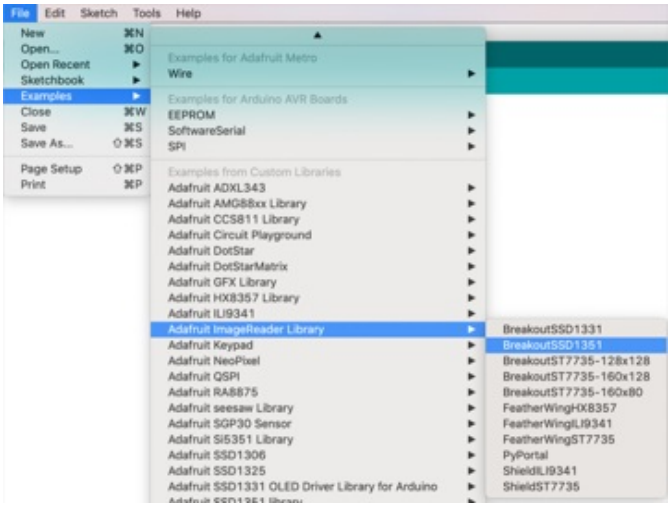
Insert the card

Insert the micro SD card into the slot on the back of the SSD1351 breakout board.



Copy the bitmap file

Copy the file **"lily128.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

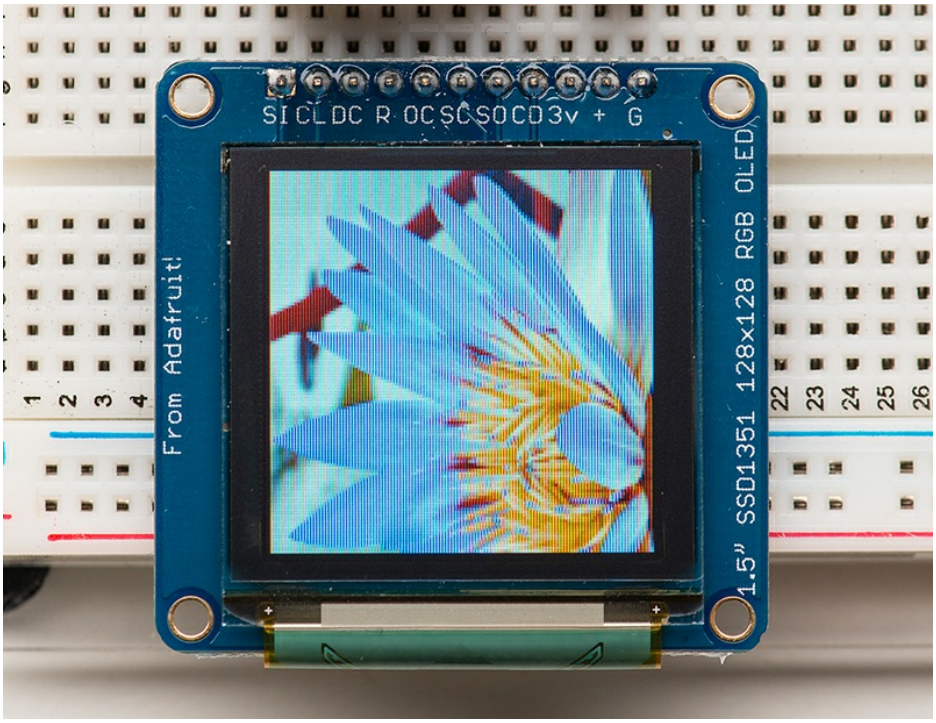
Select **"Examples->Adafruit_ImageReader_Library->BreakoutSSD1351"** and upload it to your Arduino.

In the example, find the following section of code:

```
// Load full-screen BMP file 'rgbwheel.bmp' at position (0,0) (top left).
// Notice the 'reader' object performs this, with 'tft' as an argument.
Serial.print(F("Loading rgbwheel.bmp to screen..."));
stat = reader.drawBMP("/rgbwheel.bmp", tft, 0, 0);
reader.printStatus(stat);   // How'd we do?
```

On the line with `reader.drawBMP()` change `"/rgbwheel.bmp"` to `"/lily128.bmp"`.

After that, upload it to your Arduino. When the Arduino restarts, you should see the flower as below!



To make new bitmaps, make sure they are less than 128 by 128 pixels and save them in **24-bit BMP format**! They must be in 24-bit format, even if they are not 24-bit color as that is the easiest format for the Arduino to decode. You can rotate images using the **setRotation()** procedure.

The BreakoutSSD1351 example sketch shows everything you need to work with BMP images. Here's just the vital bits broken out...

Several header files are included at the top of the sketch. All of these are required...they let us access the SD card and the display, and provide the image-reading functions:

```
#include <SPI.h>
#include <SD.h>
#include <Adafruit_GFX.h>         // Core graphics library
#include <Adafruit_SSD1351.h>     // Hardware-specific library
#include <Adafruit_ImageReader.h> // Image-reading functions
```

Several #defines relate to hardware pin numbers, all fixed values when using the shield.

Then we declare the tft screen object, and the image-reader object like so:

```
#define SD_CS    7 // SD card select pin
#define TFT_CS   5 // TFT select pin
#define TFT_DC   4 // TFT display/command pin
#define TFT_RST  6 // Or set to -1 and connect to Arduino RESET pin

Adafruit_SSD1351 tft = Adafruit_SSD1351(SCREEN_WIDTH, SCREEN_HEIGHT, &SPI, TFT_CS, TFT_DC, TFT_RST);

Adafruit_ImageReader reader;     // Class w/image-reading functions
```

After the SD and TFT's `begin()` functions have been called (see the example sketch again, in the `setup()` function),

you can then call `reader.drawBMP()` to load an image from the card to the screen:

```
ImageReturnCode stat; // Status from image-reading functions
stat = reader.drawBMP("/lily128.bmp", tft, 0, 0);
```

You can draw as many images as you want — though remember the names must be less than 8 characters long. Call like so:

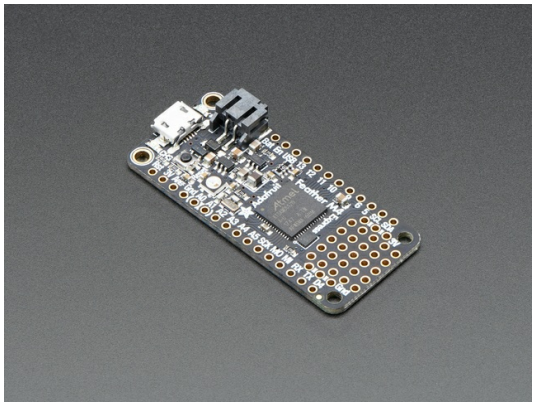
```
reader.drawBMP(filename, tft, x, y);
```

'x' and 'y' are pixel coordinates where top-left corner of the image will be placed. Images can be placed anywhere on screen...even partially off screen, the library will clip the section to load.

Image loading is explained in greater depth in the Adafruit_GFX library guide. (https://adafru.it/DpM)

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/FxO

https://adafru.it/FxO

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_ssd1351**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_ssd1351** 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

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

```
Temporarily unable to load content:
```

Let's take a look at the sections of code one by one. We're going to take a look at the code for the 1.5" display, but there is also an example available for the 128x96 1.27" display. The example is the same as we are covering in here but with the numbers adjusted for the different height.

We start by importing the board so that we can initialize `SPI`, `displayio`, `terminalio` for the font, a `label`, and the `adafruit_ssd1351` driver.

```
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_ssd1351 import SSD1351
```

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. Additionally, we need to set the baudrate to 16MHz since that is the maximum speed that the SSD1351 chip will run at. Anything higher creates strange artifacts on the screen.

```
displayio.release_displays()
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs,
                                 reset=board.D9, baudrate=16000000)
```

Finally, we initialize the driver with a width of 128 and a height of 128. 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 = SSD1351(display_bus, width=128, height=128)
```

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.

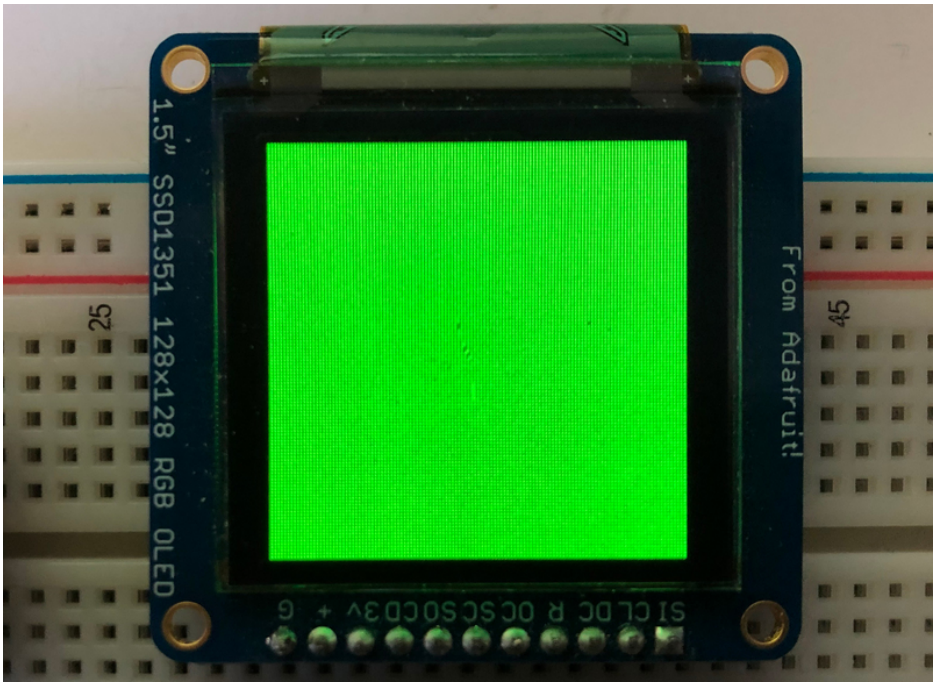
```
splash = displayio.Group(max_size=10)
display.show(splash)
```

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(128, 128, 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)
```

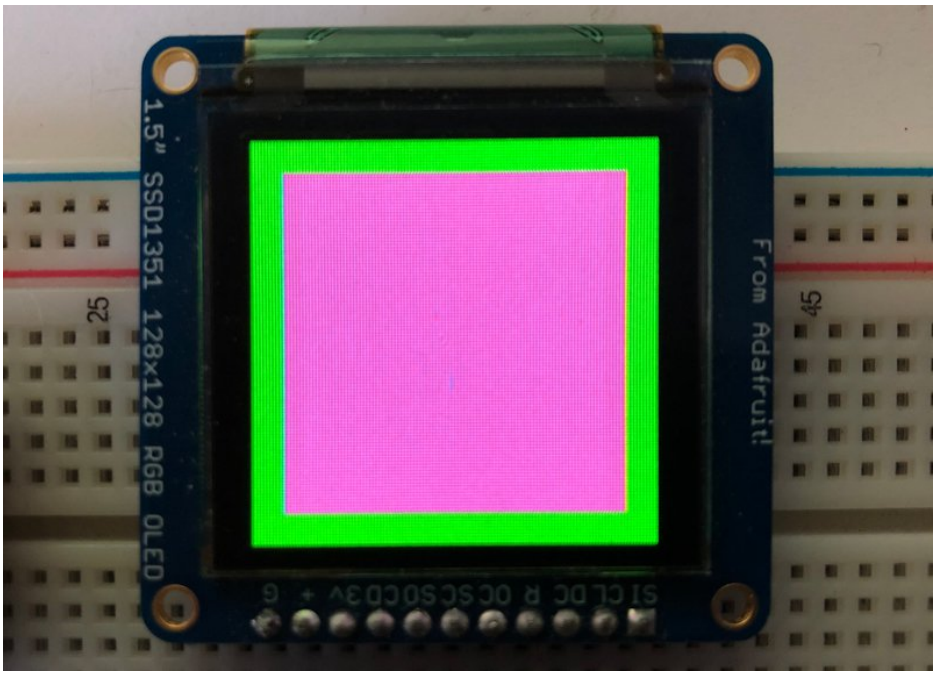


Next we will create a smaller purple square. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case, we will create a bitmap that is 10 pixels smaller on each side. The screen is **128x128**, so we'll want to subtract 20 from each of those numbers.

We'll also want to place it at the position `(10, 10)` so that it ends up centered.

```
inner_bitmap = displayio.Bitmap(108, 108, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=10, y=10)
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.



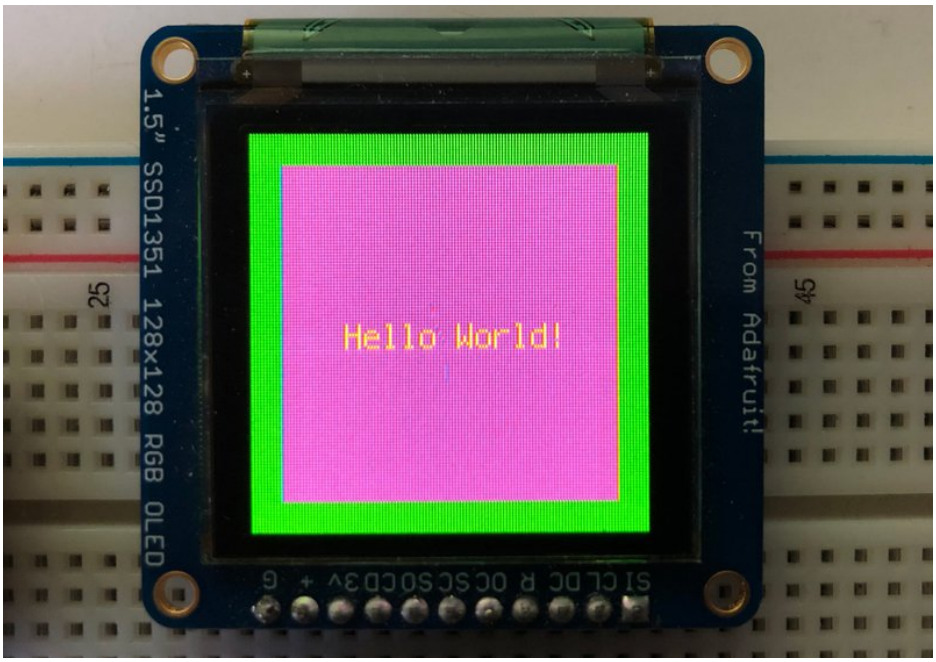
Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font. In this example, we won't be doing any scaling because of the small resolution compared to some of the other displays, so we'll add the label directly the main group. If we were scaling, we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

Labels are centered vertically, so we'll place it at 64 for the Y coordinate, and around 30 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=30, y=64)
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)

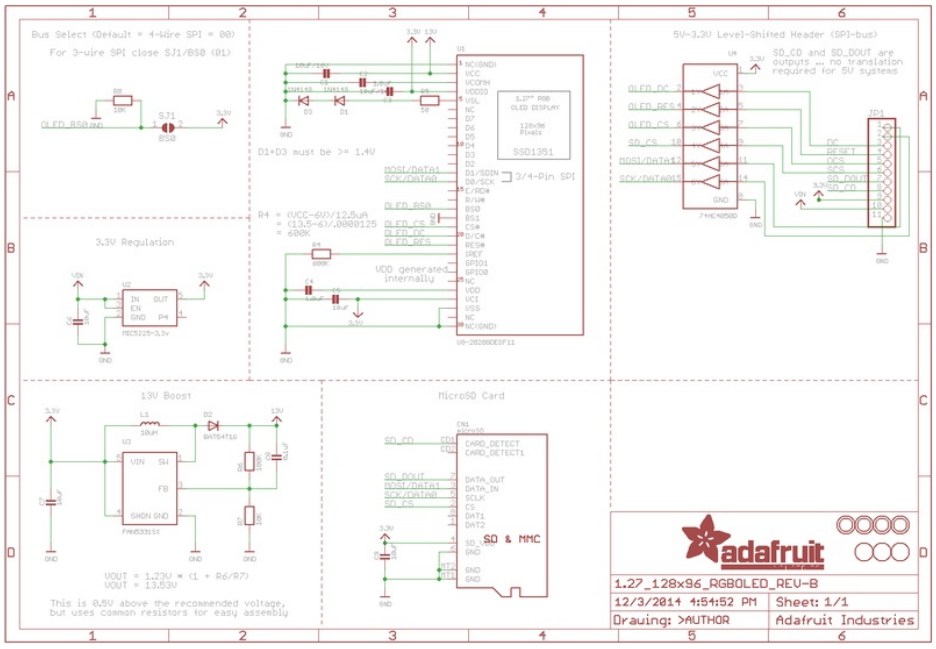
Downloads and Links

Data Sheets:

- SSD1351 Display Controller Datasheet (https://adafru.it/sVb)
- 1.5" OLED Display Module datasheet (https://adafru.it/cBE)
- Fritzing objects in the Adafruit Fritzing library (https://adafru.it/aP3)
- EagleCAD PCB for 1.27" Color OLED (https://adafru.it/rqB)
- EagleCAD PCB for the 1.5" Color OLED (https://adafru.it/rqC)

Schematic

Click to enlarge



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