0.96" mini Color OLED

Created by lady ada


https://learn.adafruit.com/096-mini-color-oled
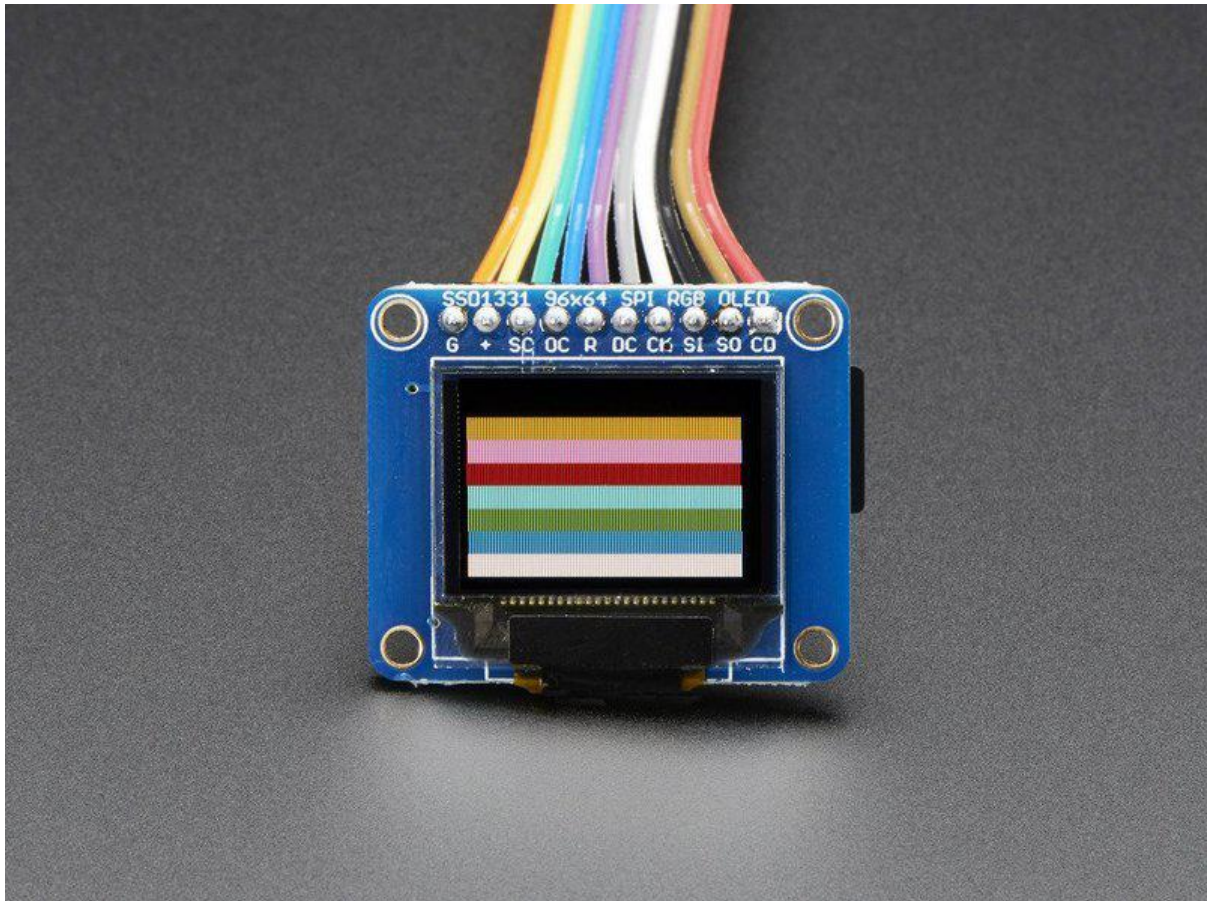
Last updated on 2023-02-06 04:25:09 PM EST

Table of Contents

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!
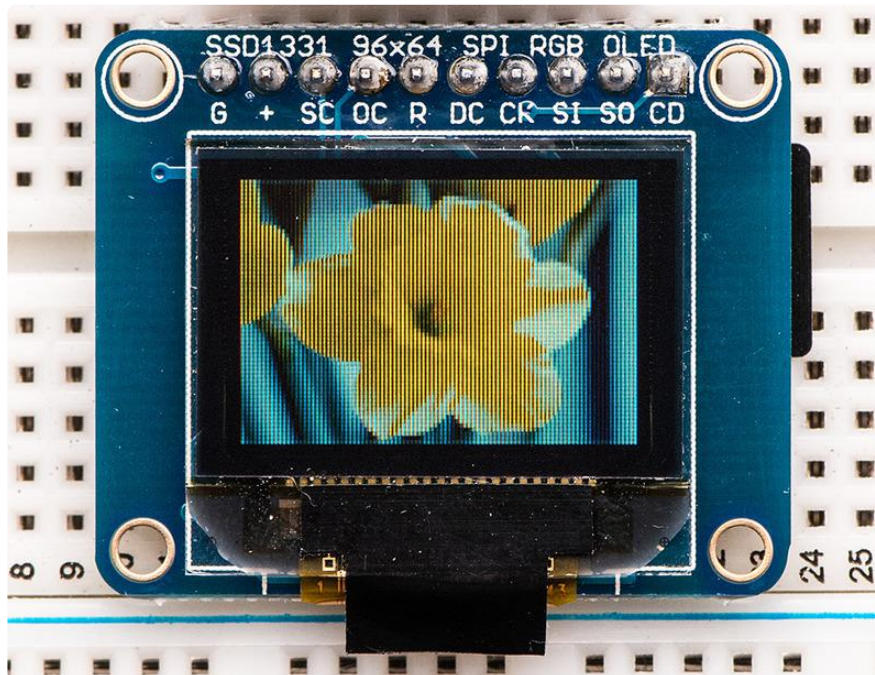
This OLED uses the SSD1331 driver chip, which manages the display. You can talk to the driver chip using either 3 or 4-wire write-only SPI (clock, data, chip select, data/command and an optional reset pin) or standard 8-bit parallel 8080/6800 which also permits reading pixel data from the display. Our 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)

Please note: all OLEDs have a "half life" — their brightness naturally diminishes over time, albeit over many thousands of hours. This makes them a poor choice for always-on 24/7/365 use. Best to turn off the display when inactive, or consider using a color LCD for continuously running projects.

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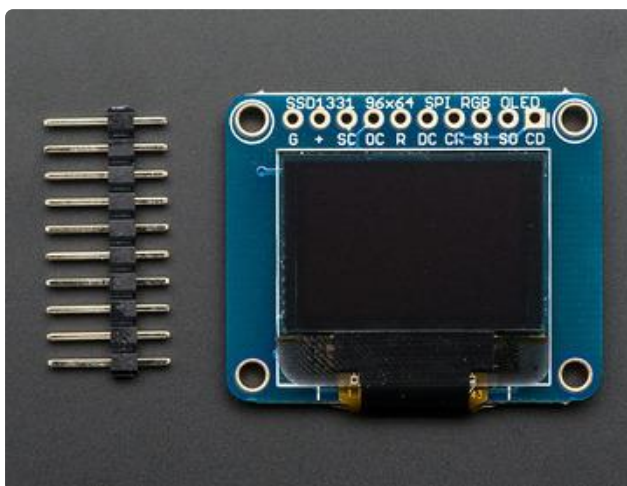
New Model
If your display has a single row header across the top, it is the newer version. For wiring instructions, skip down to "Wiring Up the Newer Version"
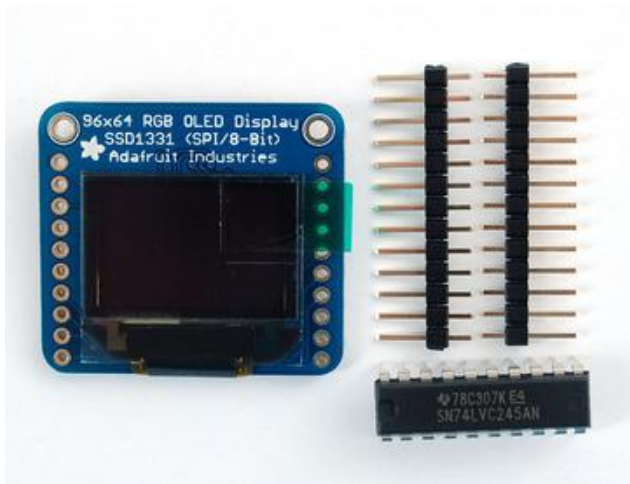
Older Model

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.
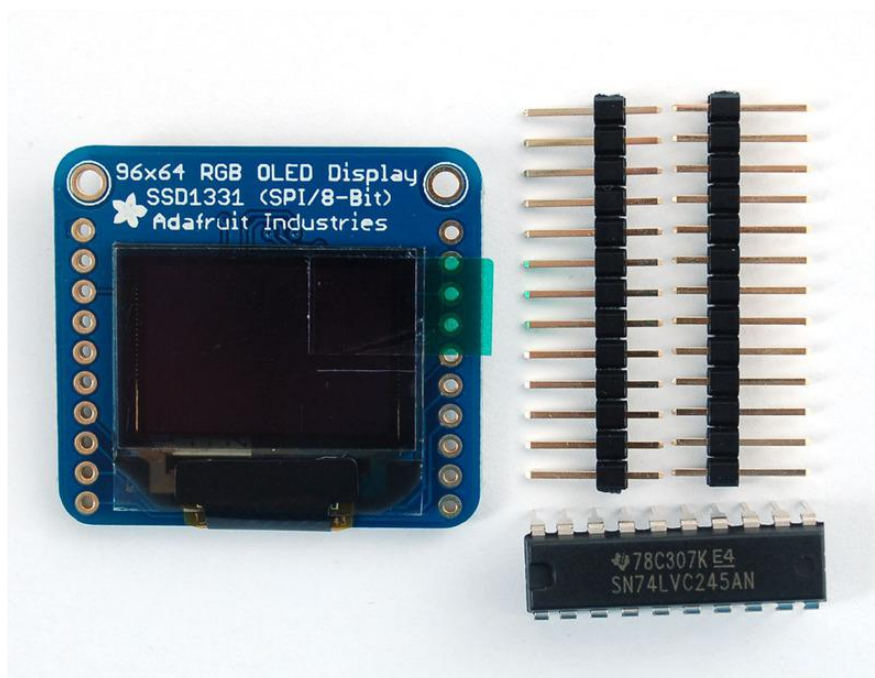
Since the OLED is 3.3V and also uses 3.3V logic, we need to use a logic shifter. We include a DIP logic shifter, the 74LVX245 with the OLED. If you're using a 3.3V logic

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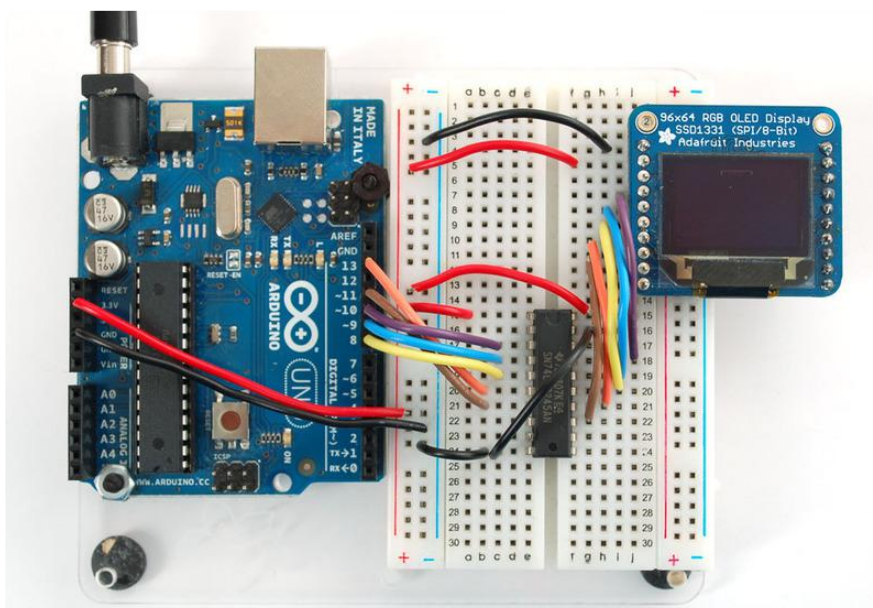
Next we'll connect the remaining 74LVC245 pins to the Arduino

- 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 () 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
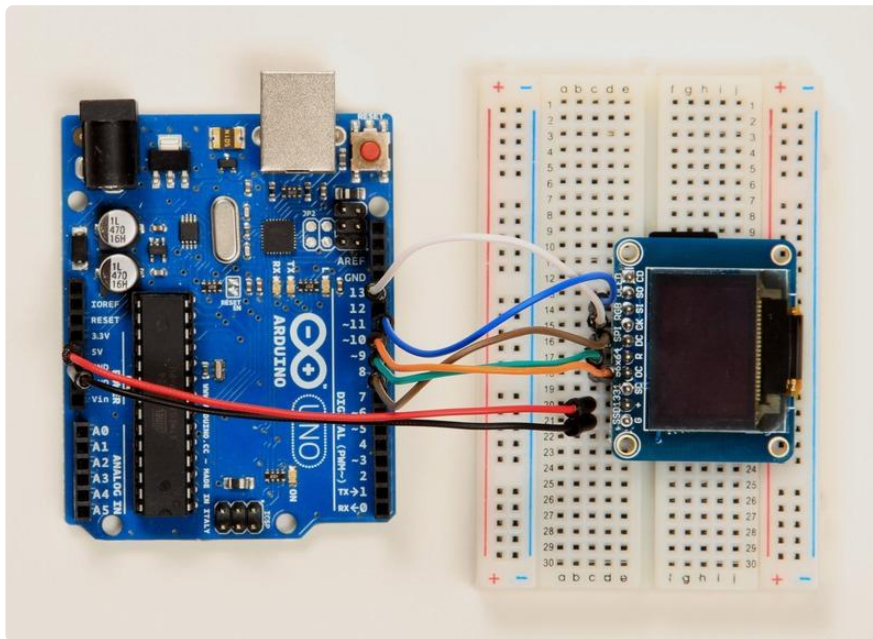
Installing and running Arduino software

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..."
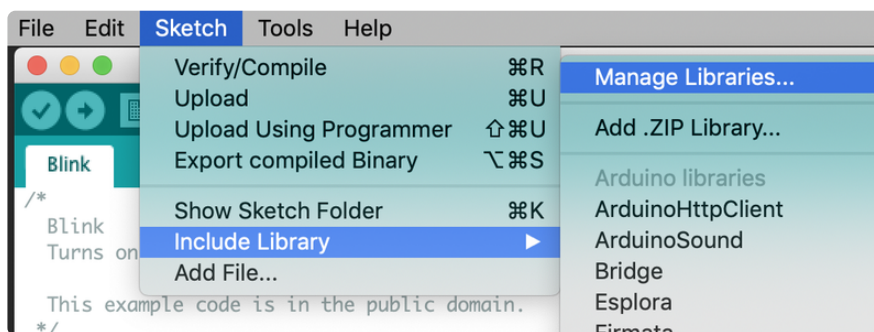
Type "gfx" in the search field to quickly find the first library — Adafruit_GFX:
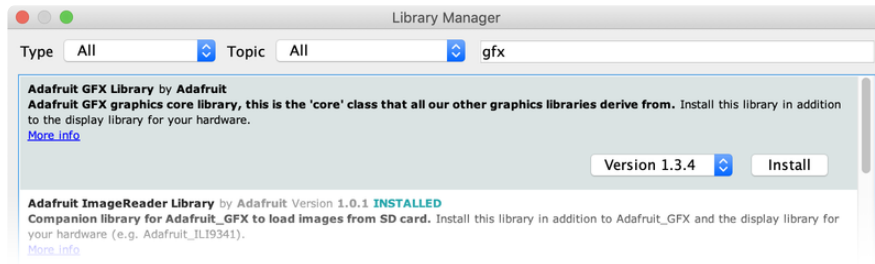
Repeat the search and install steps, looking for the Adafruit_BusIO and Adafruit_SSD1 331 libraries.

After you restart, you should be able to select File→Examples→Adafruit_SSD1331→tes t - this is the example sketch that just tests the display by drawing text and shapes. Upload the sketch and you should see the following:
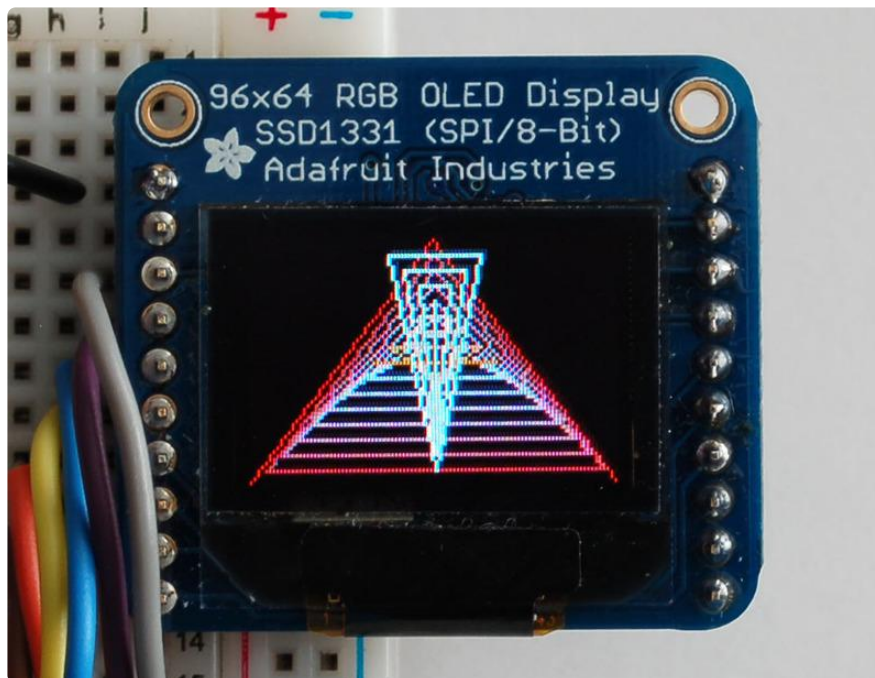
If alls working, then you can start looking through the test sketch for demonstrations on how to print text, circles, lines, etc.

For a detailed tutorial on the Adafruit GFX library, including all the functions available please visit the GFX tutorial page ()

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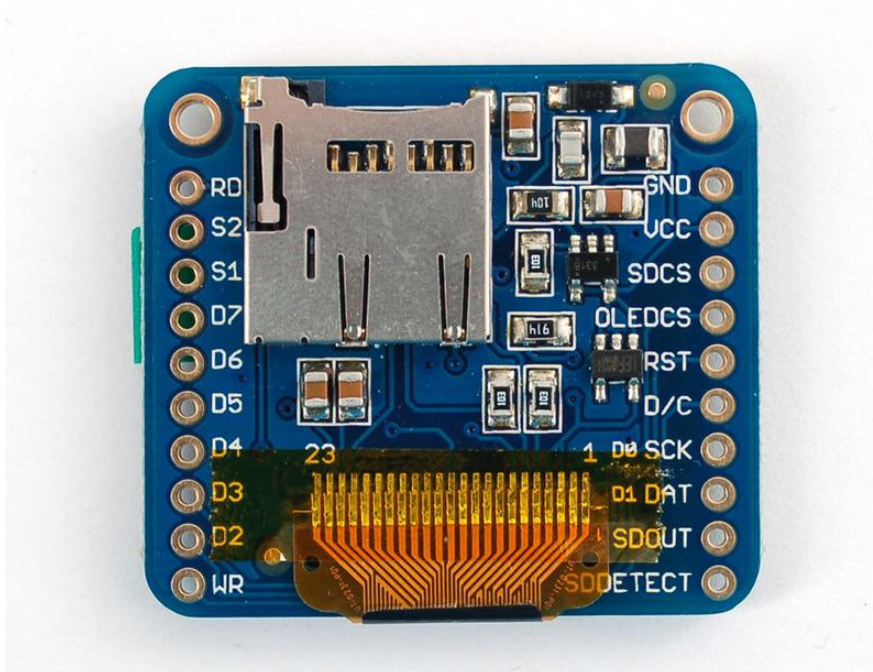
We'll have to add an extra 2 wires so we can 'select' and 'receive data' from the SD card

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.
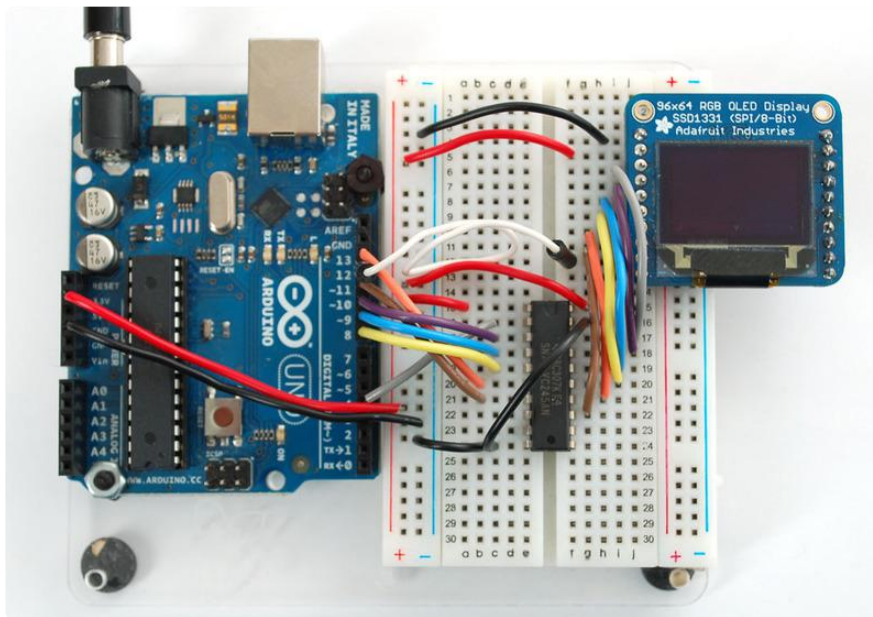
New Style Board: (Single row of pins)

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

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

fritzing
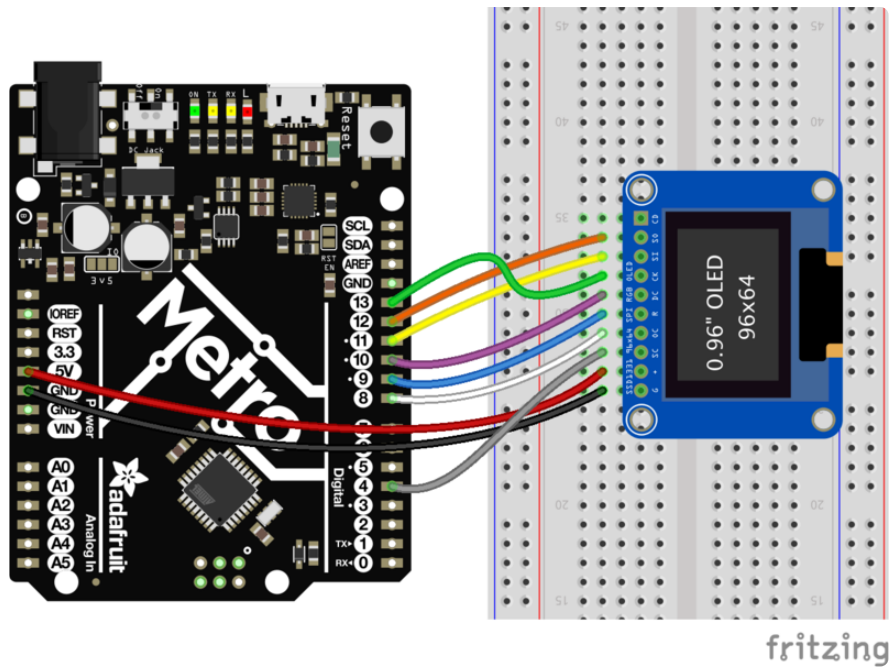
Bitmap Example Sketch

To display bitmaps from the on-board micro SD slot, you will need a micro SD card (htt p://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:
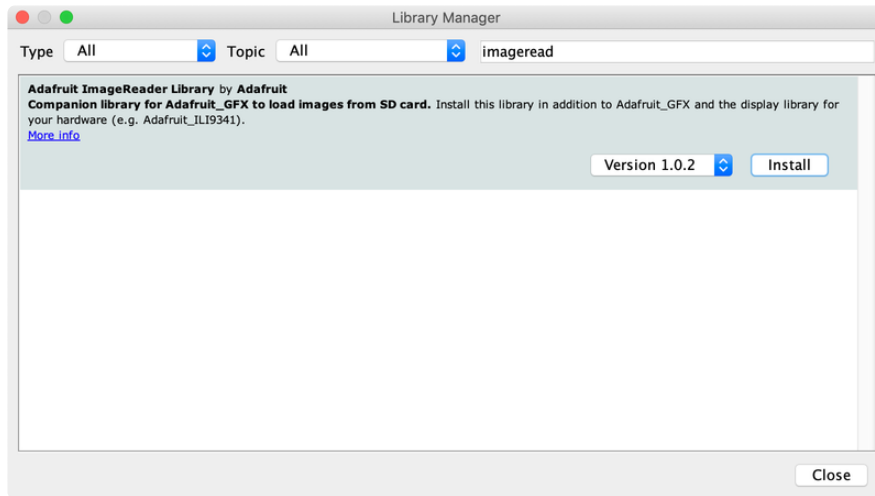
Next you can either download the image here or copy it from the images folder from inside the library files.

Download daffodil.bmp
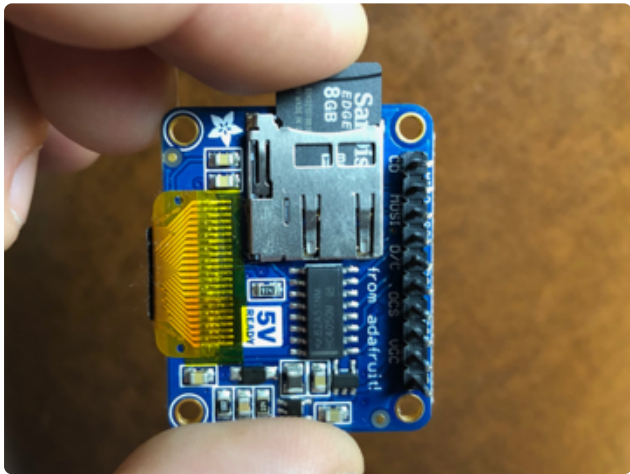
Insert the card
Insert the micro SD card into the slot on the back of the SSD1331 breakout board.
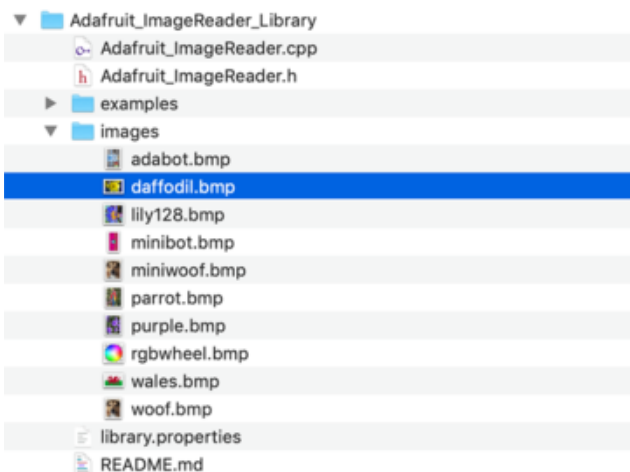
Copy the bitmap file
Copy the file "daffodil.bmp" from the Adafruit_ImageReader_Library\images folder (or wherever you saved it if you downloaded the file) over to the root directory of your micro-SD card.
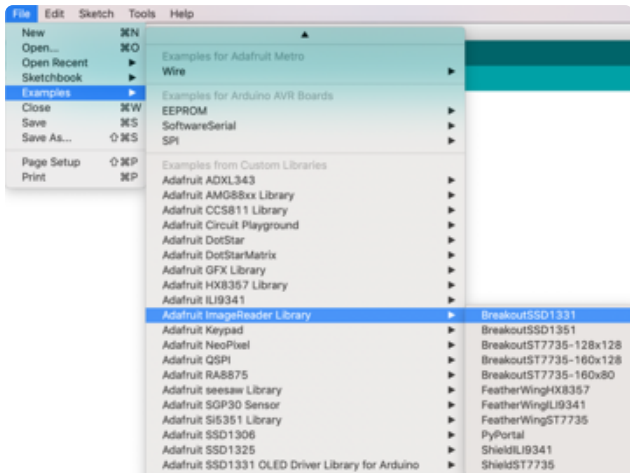
Load the bitmap example sketch

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 () guide.
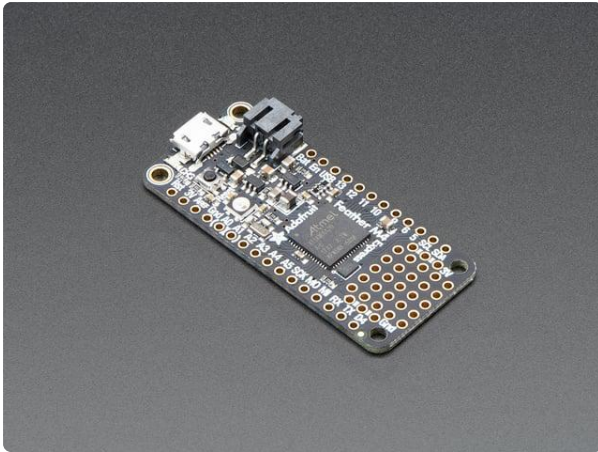
Adafruit Feather M4 Express - Featuring ATSAMD51

It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...

https://www.adafruit.com/product/3857

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 (). After that the breakout should be ready to go.

Required CircuitPython Libraries

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

[Adafruit_CircuitPython_SSD1331]

First, make sure you are running the latest version of Adafruit CircuitPython () 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 (). Our introduction guide has a great page on how to install the library bundle () 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.

Adafruit_CircuitPython_Display_Text

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

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

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

# Release any resources currently in use for the displays
displayio.release_displays()

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

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()
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 release any previously used 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.

```
displayio.release_displays()
```

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 line, we set the display bus to FourWire which makes use of the SPI bus.

```
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)
```
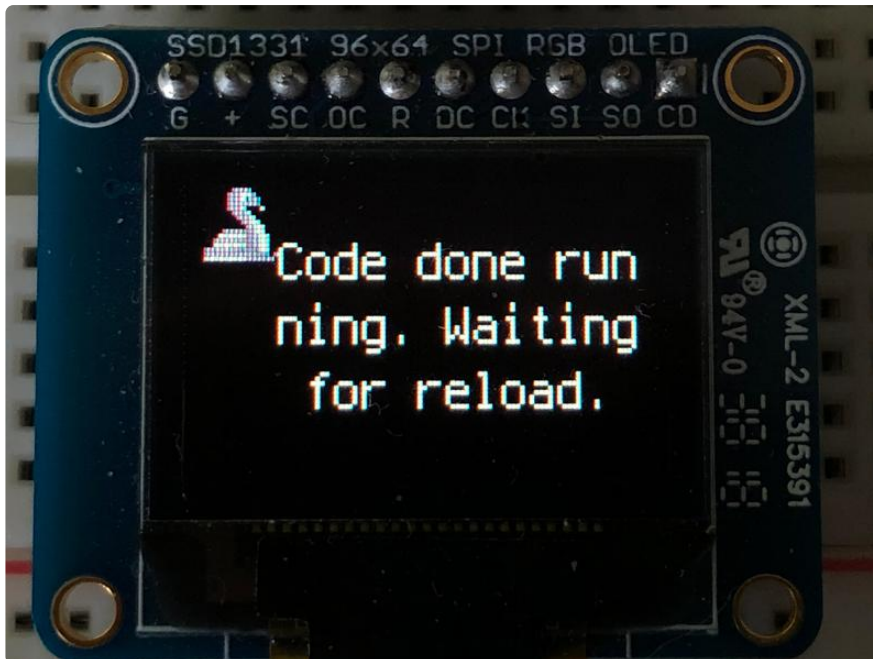
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating 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)
```
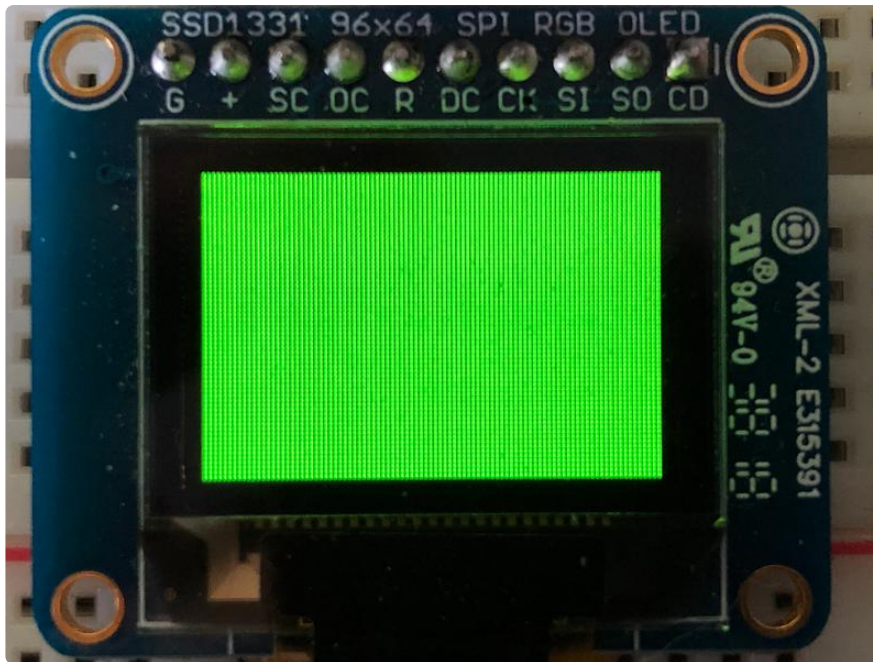
Next we will create a smaller purple square. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case, we will create a bitmap that is 5 pixels smaller on each side. The screen is 96x64, so 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.
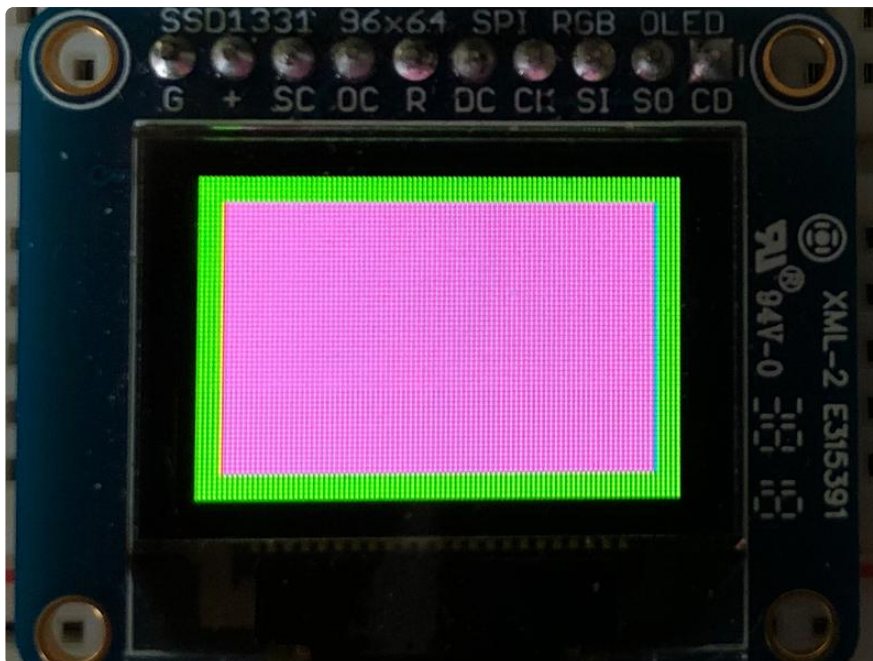
Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font. In this example, we won't be doing any scaling because of the small resolution, so we'll add the label directly the main group. If we were scaling, we would have used a subgroup.

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 ()

Python Wiring and Setup

Wiring

It's easy to use display breakouts with Python and the Adafruit CircuitPython RGB Display () module. This module allows you to easily write Python code to control the display.

We'll cover how to wire the display to your Raspberry Pi. First assemble your 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 ().

Connect the display as shown below to your Raspberry Pi.

> Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbtft driver, and want to use the TFT purely from 'user Python' code!

> You can only use this technique with Linux/computer devices that have hardware SPI support, and not all single board computers have an SPI device so check before continuing

ILI9341 and HX-8357-based Displays

2.2" Display

- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CE0
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later.
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later as well.
- Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground

Download the Fritzing Diagram

2.4", 2.8", 3.2", and 3.5" Displays

These displays are set up to use the 8-bit data lines by default. We want to use them for SPI. To do that, you'll need to either solder bridge some pads on the back or connect the appropriate IM lines to 3.3V with jumper wires. Check the back of your display for the correct solder pads or IM lines to put it in SPI mode.

- Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground
- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CE0
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later.
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later as well.

> These larger displays are set to use 8-bit data lines by default and may need to be modified to use SPI.

Download the Fritzing Diagram

ST7789 and ST7735-based Displays

1.3", 1.54", and 2.0" IPS TFT Display

- Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground

- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CE0
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later.
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later as well.

Download the Fritzing Diagram

0.96", 1.14", and 1.44" Displays

- Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground
- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CE0
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later.
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later as well.

1.8" Display

- GND connects to the Raspberry Pi's ground
- Vin connects to the Raspberry Pi's 3V pin
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later.
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later as well.
- CS connects to our SPI Chip Select pin. We'll be using CE0
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- LITE connects to the Raspberry Pi's 3V pin. This can be used to separately control the backlight.
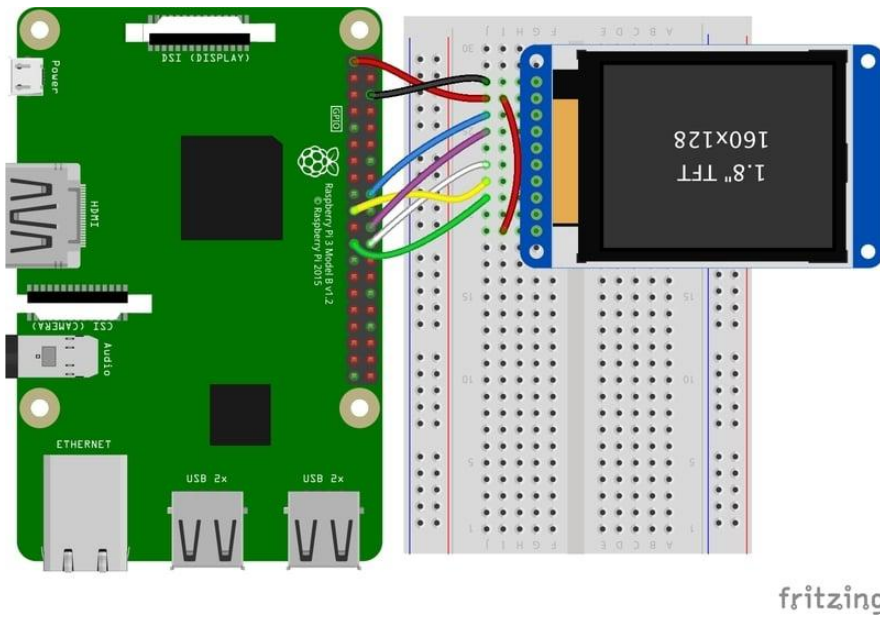
fritzing

Download the Fritzing Diagram

SSD1351-based Displays

1.27" and 1.5" OLED Displays

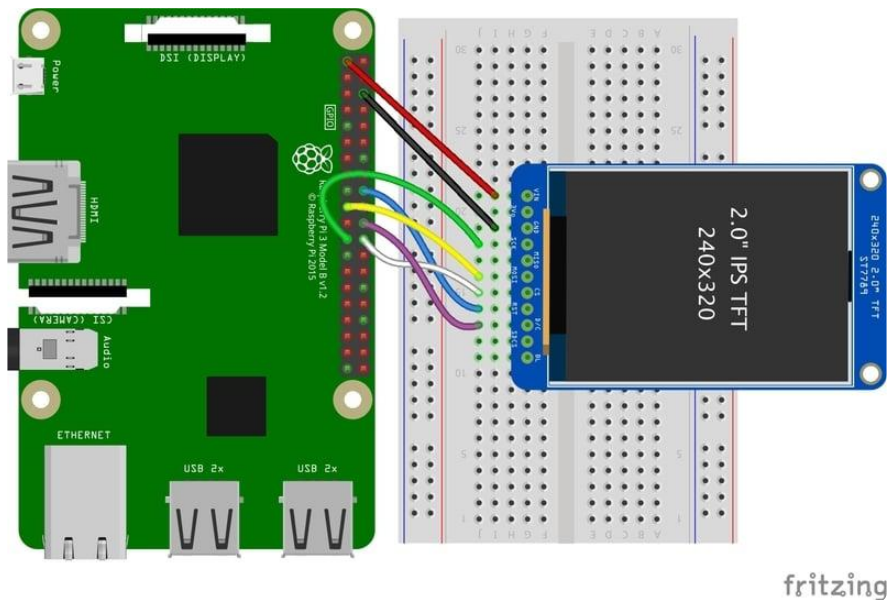- GND connects to the Raspberry Pi's ground
- Vin connects to the Raspberry Pi's 3V pin
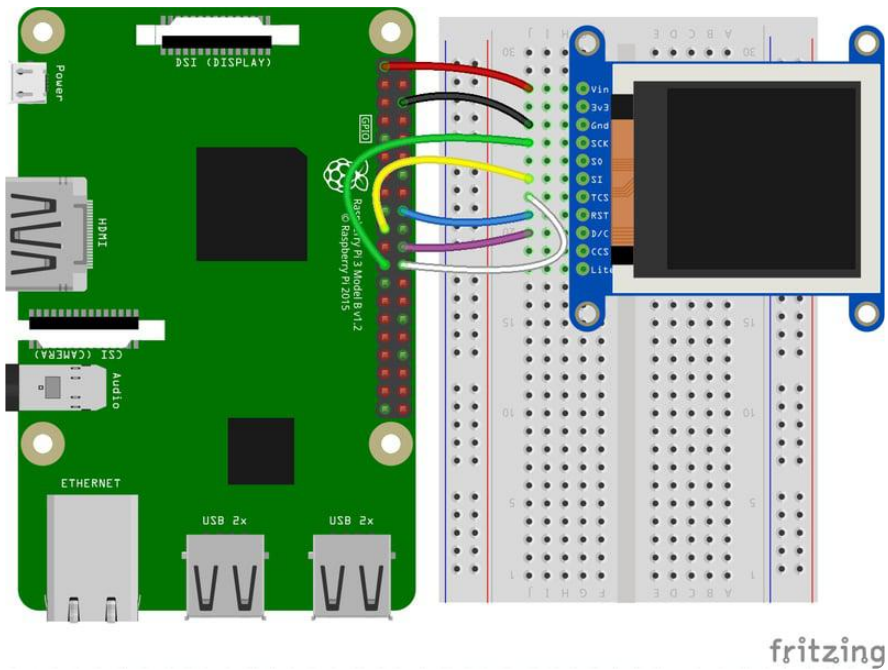- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CS connects to our SPI Chip Select pin. We'll be using CE0
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later.
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later as well.
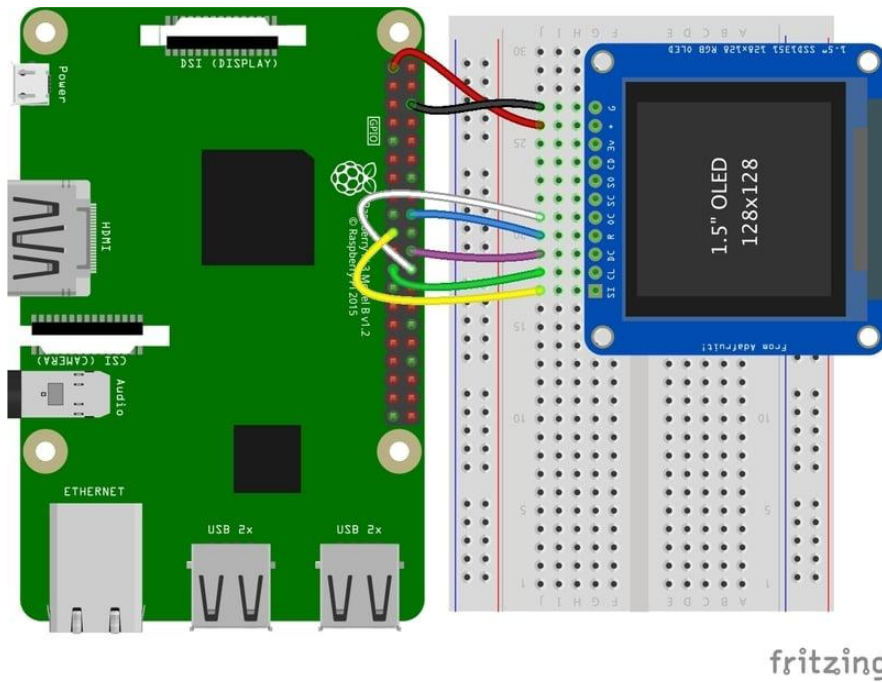
Download the Fritzing Diagram

SSD1331-based Display

0.96" OLED Display

- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CLK connects to SPI clock. On the Raspberry Pi, thats SLCK
- D/C connects to our SPI Chip Select pin. We'll be using GPIO 25, but this can be changed later.
- RST connects to our Reset pin. We'll be using GPIO 24 but this can be changed later as well.
- CS connects to our SPI Chip Select pin. We'll be using CE0
- Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground

Download the Fritzing Diagram

Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI 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 ()!

> If you have previously installed the Kernel Driver with the PiTFT Easy Setup, you will need to remove it first in order to run this example.

Python Installation of RGB Display Library

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

- ```
  sudo pip3 install adafruit-circuitpython-rgb-display
  ```

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`

DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

- `sudo apt-get install fonts-dejavu`

This package was previously calls ttf-dejavu, so if you are running an older version of Raspberry Pi OS, it may be called that.

Pillow Library

We also need PIL, the Python Imaging Library, to allow graphics and 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.

Python Usage

If you have previously installed the Kernel Driver with the PiTFT Easy Setup, you will need to remove it first in order to run this example.

Now that you have everything setup, we're going to look over three different examples. For the first, we'll take a look at automatically scaling and cropping an image and then centering it on the display.

Turning on the Backlight

On some displays, the backlight is controlled by a separate pin such as the 1.3" TFT Bonnet with Joystick. On such displays, running the below code will likely result in the

display remaining black. To turn on the backlight, you will need to add a small snippet of code. If your backlight pin number differs, be sure to change it in the code:

```
# Turn on the Backlight
backlight = DigitalInOut(board.D26)
backlight.switch_to_output()
backlight.value = True
```

Displaying an Image

Here's the full code to the example. We will go through it section by section to help you better understand what is going on. Let's start by downloading an image of Blinka. This image has enough border to allow resizing and cropping with a variety of display sizes and rations to still look good.

Make sure you save it as blinka.jpg and place it in the same folder as your script. Here's the code we'll be loading onto the Raspberry Pi. We'll go over the interesting parts.

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

"""
Be sure to check the learn guides for more usage information.

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries
"""

import digitalio
import board
from PIL import Image, ImageDraw
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
```

```python
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

# Config for display baudrate (default max is 24mhz):
BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                              # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,   # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                             # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                             # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,        # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,     # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                           # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                           # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height
image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)

image = Image.open("blinka.jpg")

# Scale the image to the smaller screen dimension
```

```
image_ratio = image.width / image.height
screen_ratio = width / height
if screen_ratio < image_ratio:
    scaled_width = image.width * height // image.height
    scaled_height = height
else:
    scaled_width = width
    scaled_height = image.height * width // image.width
image = image.resize((scaled_width, scaled_height), Image.Resampling.BICUBIC)

# Crop and center the image
x = scaled_width // 2 - width // 2
y = scaled_height // 2 - height // 2
image = image.crop((x, y, x + width, y + height))

# Display image.
disp.image(image)
```

So we start with our usual imports including a couple of Pillow modules and the display drivers. That is followed by defining a few pins here. The reason we chose these is because they allow you to use the same code with the PiTFT if you chose to do so.

```
import digitalio
import board
from PIL import Image, ImageDraw
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789
import adafruit_rgb_display.hx8357 as hx8357
import adafruit_rgb_display.st7735 as st7735
import adafruit_rgb_display.ssd1351 as ssd1351
import adafruit_rgb_display.ssd1331 as ssd1331

# Configuration for CS and DC pins
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)
```

Next we'll set the baud rate from the default 24 MHz so that it works on a variety of displays. The exception to this is the SSD1351 driver, which will automatically limit it to 16MHz even if you pass 24MHz. We'll set up out SPI bus and then initialize the display.

We wanted to make these examples work on as many displays as possible with very few changes. The ILI9341 display is selected by default. For other displays, go ahead and comment out these lines:

```
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
```

and uncomment the line appropriate for your display and possibly the line below in the case of longer initialization sequences. The displays have a rotation property so that it can be set in just one place.

```
#disp = st7789.ST7789(spi, rotation=90,                              # 2.0" ST7789
#disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
#disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
#disp = hx8357.HX8357(spi, rotation=180,                            # 3.5" HX8357
#disp = st7735.ST7735R(spi, rotation=90,                            # 1.8" ST7735R
#disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
#disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,        # 0.96" MiniTFT
Rev A ST7735R
#disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,     # 0.96" MiniTFT
Rev B ST7735R
#x_offset=26, y_offset=1,#disp = ssd1351.SSD1351(spi,
rotation=180,                               # 1.5" SSD1351
#disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
#disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE
)
```

Next we read the current rotation setting of the display and if it is 90 or 270 degrees, we need to swap the width and height for our calculations, otherwise we just grab the width and height. We will create an `image` with our dimensions and use that to create a `draw` object. The `draw` object will have all of our drawing functions.

```
# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width   # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width    # we swap height/width to rotate it to landscape!
    height = disp.height
image = Image.new('RGB', (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)
```

Next we clear whatever is on the screen by drawing a black rectangle. This isn't strictly necessary since it will be overwritten by the image, but it kind of sets the stage.

```
# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)
```

Next we open the Blinka image, which we've named blinka.jpg, which assumes it is in the same directory that you are running the script from. Feel free to change it if it doesn't match your configuration.

```
image = Image.open("blinka.jpg")
```

Here's where it starts to get interesting. We want to scale the image so that it matches either the width or height of the display, depending on which is smaller, so that we have some of the image to chop off when we crop it. So we start by calculating the width to height ration of both the display and the image. If the height is the closer of the dimensions, we want to match the image height to the display height and let it be a bit wider than the display. Otherwise, we want to do the opposite.

Once we've figured out how we're going to scale it, we pass in the new dimensions and using a Bicubic rescaling method, we reassign the newly rescaled image back to `image`. Pillow has quite a few different methods to choose from, but Bicubic does a great job and is reasonably fast.

```
# Scale the image to the smaller screen dimension
image_ratio = image.width / image.height
screen_ratio = width / height
if screen_ratio < image_ratio:
    scaled_width = image.width * height // image.height
    scaled_height = height
else:
    scaled_width = width
    scaled_height = image.height * width // image.width
image = image.resize((scaled_width, scaled_height), Image.BICUBIC)
```

Next we want to figure the starting x and y points of the image where we want to begin cropping it so that it ends up centered. We do that by using a standard centering function, which is basically requesting the difference of the center of the display and the center of the image. Just like with scaling, we replace the `image` variable with the newly cropped image.

```
# Crop and center the image
x = scaled_width // 2 - width // 2
y = scaled_height // 2 - height // 2
image = image.crop((x, y, x + width, y + height))
```

Finally, we take our image and display it. At this point, the image should have the exact same dimensions at the display and fill it completely.

```
disp.image(image)
```

Drawing Shapes and Text

In the next example, we'll take a look at drawing shapes and text. This is very similar to the displayio example, but it uses Pillow instead. Here's the code for that.

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

"""
This demo will draw a few rectangles onto the screen along with some text
on top of that.

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries
"""

import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# First define some constants to allow easy resizing of shapes.
BORDER = 20
FONTSIZE = 24

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

# Config for display baudrate (default max is 24mhz):
```

```
BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                           # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                           # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,       # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,    # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                         # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height

image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a green filled box as the background
draw.rectangle((0, 0, width, height), fill=(0, 255, 0))
disp.image(image)

# Draw a smaller inner purple rectangle
draw.rectangle(
    (BORDER, BORDER, width - BORDER - 1, height - BORDER - 1), fill=(170, 0, 136)
)

# Load a TTF Font
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
FONTSIZE)

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text(
```

```
        (width // 2 - font_width // 2, height // 2 - font_height // 2),
        text,
        font=font,
        fill=(255, 255, 0),
)

# Display image.
disp.image(image)
```

Just like in the last example, we'll do our imports, but this time we're including the
`ImageFont` Pillow module because we'll be drawing some text this time.

```
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
```

Next we'll define some parameters that we can tweak for various displays. The `BORDER` will be the size in pixels of the green border between the edge of the display and the inner purple rectangle. The `FONTSIZE` will be the size of the font in points so that we can adjust it easily for different displays.

```
BORDER = 20
FONTSIZE = 24
```

Next, just like in the previous example, we will set up the display, setup the rotation, and create a draw object. If you have are using a different display than the ILI9341, go ahead and adjust your initializer as explained in the previous example. After that, we will setup the background with a green rectangle that takes up the full screen. To get green, we pass in a tuple that has our Red, Green, and Blue color values in it in that order which can be any integer from `0` to `255`.

```
draw.rectangle((0, 0, width, height), fill=(0, 255, 0))
disp.image(image)
```

Next we will draw an inner purple rectangle. This is the same color value as our example in displayio quickstart, except the hexadecimal values have been converted to decimal. We use the `BORDER` parameter to calculate the size and position that we want to draw the rectangle.

```
draw.rectangle((BORDER, BORDER, width - BORDER - 1, height - BORDER - 1),
               fill=(170, 0, 136))
```

Next we'll load a TTF font. The `DejaVuSans.ttf` font should come preloaded on your Pi in the location in the code. We also make use of the `FONTSIZE` parameter that we discussed earlier.

```
# Load a TTF Font
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf',
FONTSIZE)
```

Now we draw the text Hello World onto the center of the display. You may recognize the centering calculation was the same one we used to center crop the image in the previous example. In this example though, we get the font size values using the `getsize()` function of the font object.

```
# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text((width//2 - font_width//2, height//2 - font_height//2),
          text, font=font, fill=(255, 255, 0))
```

Finally, just like before, we display the image.

```
disp.image(image)
```


Displaying System Information

In this last example we'll take a look at getting the system information and displaying it. This can be very handy for system monitoring. Here's the code for that example:

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

"""
This will show some Linux Statistics on the attached display. Be sure to adjust
```

```
to the display you have connected. Be sure to check the learn guides for more
usage information.

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!
"""

import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

# Config for display baudrate (default max is 24mhz):
BAUDRATE = 24000000

# Setup SPI bus using hardware SPI:
spi = board.SPI()

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                          # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                          # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                          # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,       # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,    # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                         # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
```

```
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height

image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)

# First define some constants to allow easy positioning of text.
padding = -2
x = 0

# Load a TTF font.  Make sure the .ttf font file is in the
# same directory as the python script!
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 24)

while True:
    # Draw a black filled box to clear the image.
    draw.rectangle((0, 0, width, height), outline=0, fill=0)

    # Shell scripts for system monitoring from here:
    # https://unix.stackexchange.com/questions/119126/command-to-display-memory-
usage-disk-usage-and-cpu-load
    cmd = "hostname -I | cut -d' ' -f1"
    IP = "IP: " + subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\",
$3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = 'df -h | awk \'$NF=="/"{printf "Disk: %d/%d GB  %s", $3,$2,$5}\''
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "cat /sys/class/thermal/thermal_zone0/temp |  awk '{printf \"CPU Temp: %.
1f C\", $(NF-0) / 1000}'"  # pylint: disable=line-too-long
    Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.
    y = padding
    draw.text((x, y), IP, font=font, fill="#FFFFFF")
    y += font.getsize(IP)[1]
    draw.text((x, y), CPU, font=font, fill="#FFFF00")
    y += font.getsize(CPU)[1]
    draw.text((x, y), MemUsage, font=font, fill="#00FF00")
    y += font.getsize(MemUsage)[1]
    draw.text((x, y), Disk, font=font, fill="#0000FF")
    y += font.getsize(Disk)[1]
    draw.text((x, y), Temp, font=font, fill="#FF00FF")

    # Display image.
    disp.image(image)
    time.sleep(0.1)
```

Just like the last example, we'll start by importing everything we imported, but we're adding two more imports. The first one is `time` so that we can add a small delay and the other is `subprocess` so we can gather some system information.

```
import time
import subprocess
import digitalio
```

```
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
```

Next, just like in the first two examples, we will set up the display, setup the rotation, and create a draw object. If you have are using a different display than the ILI9341, go ahead and adjust your initializer as explained in the previous example.

Just like in the first example, we're going to draw a black rectangle to fill up the screen. After that, we're going to set up a couple of constants to help with positioning text. The first is the `padding` and that will be the Y-position of the top-most text and the other is `x` which is the X-Position and represents the left side of the text.

```
# First define some constants to allow easy positioning of text.
padding = -2
x = 0
```

Next, we load a font just like in the second example.

```
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', 24)
```

Now we get to the main loop and by using `while True:`, it will loop until Control+C is pressed on the keyboard. The first item inside here, we clear the screen, but notice that instead of giving it a tuple like before, we can just pass `0` and it will draw black.

```
draw.rectangle((0, 0, width, height), outline=0, fill=0)
```

Next, we run a few scripts using the `subprocess` function that get called to the Operating System to get information. The in each command is passed through awk in order to be formatted better for the display. By having the OS do the work, we don't have to. These little scripts came from https://unix.stackexchange.com/questions/119126/command-to-display-memory-usage-disk-usage-and-cpu-load

```
cmd = "hostname -I | cut -d\' \' -f1"
IP = "IP: "+subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\", $3,$2,$3*100/$2 }'"
MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "df -h | awk '$NF==\"/\"{printf \"Disk: %d/%d GB  %s\", $3,$2,$5}'"
Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "cat /sys/class/thermal/thermal_zone0/temp |  awk \'{printf \"CPU Temp: %.1f
C\", $(NF-0) / 1000}\'" # pylint: disable=line-too-long
Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")
```

Now we display the information for the user. Here we use yet another way to pass color information. We can pass it as a color string using the pound symbol, just like we would with HTML. With each line, we take the height of the line using `getsize()` and move the pointer down by that much.

```
y = padding
draw.text((x, y), IP, font=font, fill="#FFFFFF")
y += font.getsize(IP)[1]
draw.text((x, y), CPU, font=font, fill="#FFFF00")
y += font.getsize(CPU)[1]
draw.text((x, y), MemUsage, font=font, fill="#00FF00")
y += font.getsize(MemUsage)[1]
draw.text((x, y), Disk, font=font, fill="#0000FF")
y += font.getsize(Disk)[1]
draw.text((x, y), Temp, font=font, fill="#FF00FF")
```

Finally, we write all the information out to the display using `disp.image()`. Since we are looping, we tell Python to sleep for `0.1` seconds so that the CPU never gets too busy.

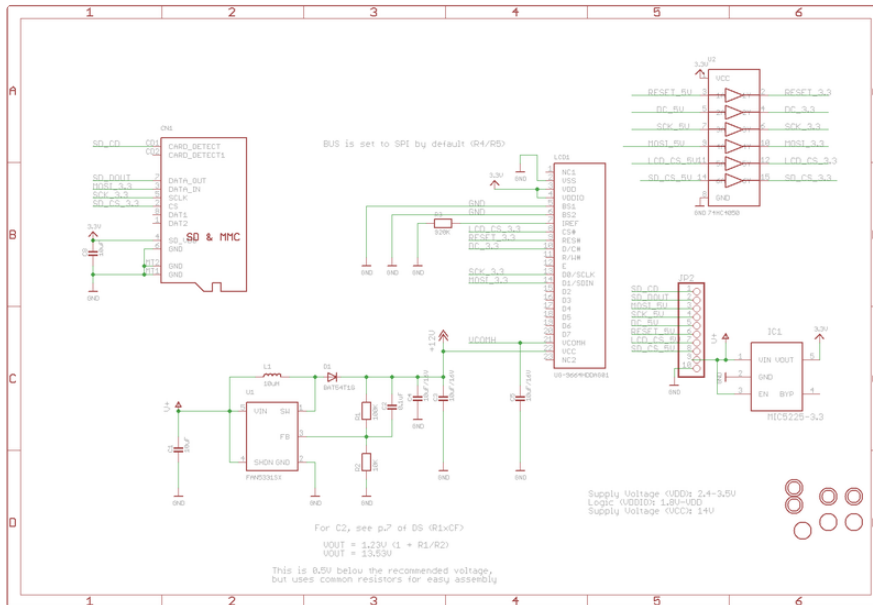```
disp.image(image)
time.sleep(.1)
```
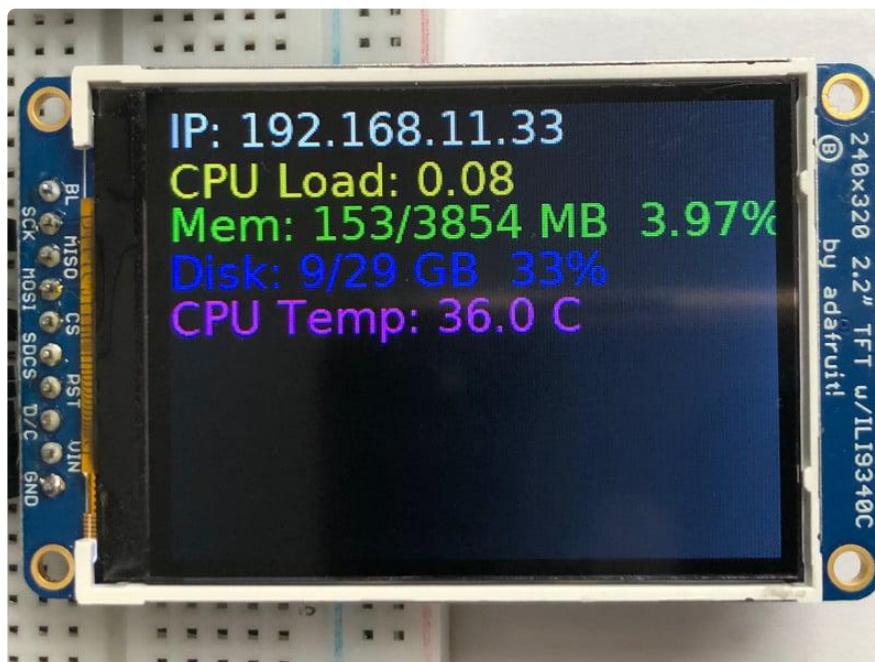
Downloads

Files

- [Datasheet for the SSD1331 driver ()](#)
- [Datasheet for the raw OLED module itself ()](#)

- Adafruit SSD1331 Arduino Library repository ()
- You'll also have to install the Adafruit GFX graphics core library at this github repo () and install it after you've gotten the OLED driver library.
- Adafruit breakout board PCB files ()
- Fritzing object in the Adafruit Fritzing Library ()

Schematic & Fabrication Print

For the level shifter we use the CD74HC4050 () which has a typical propagation delay of ~10ns