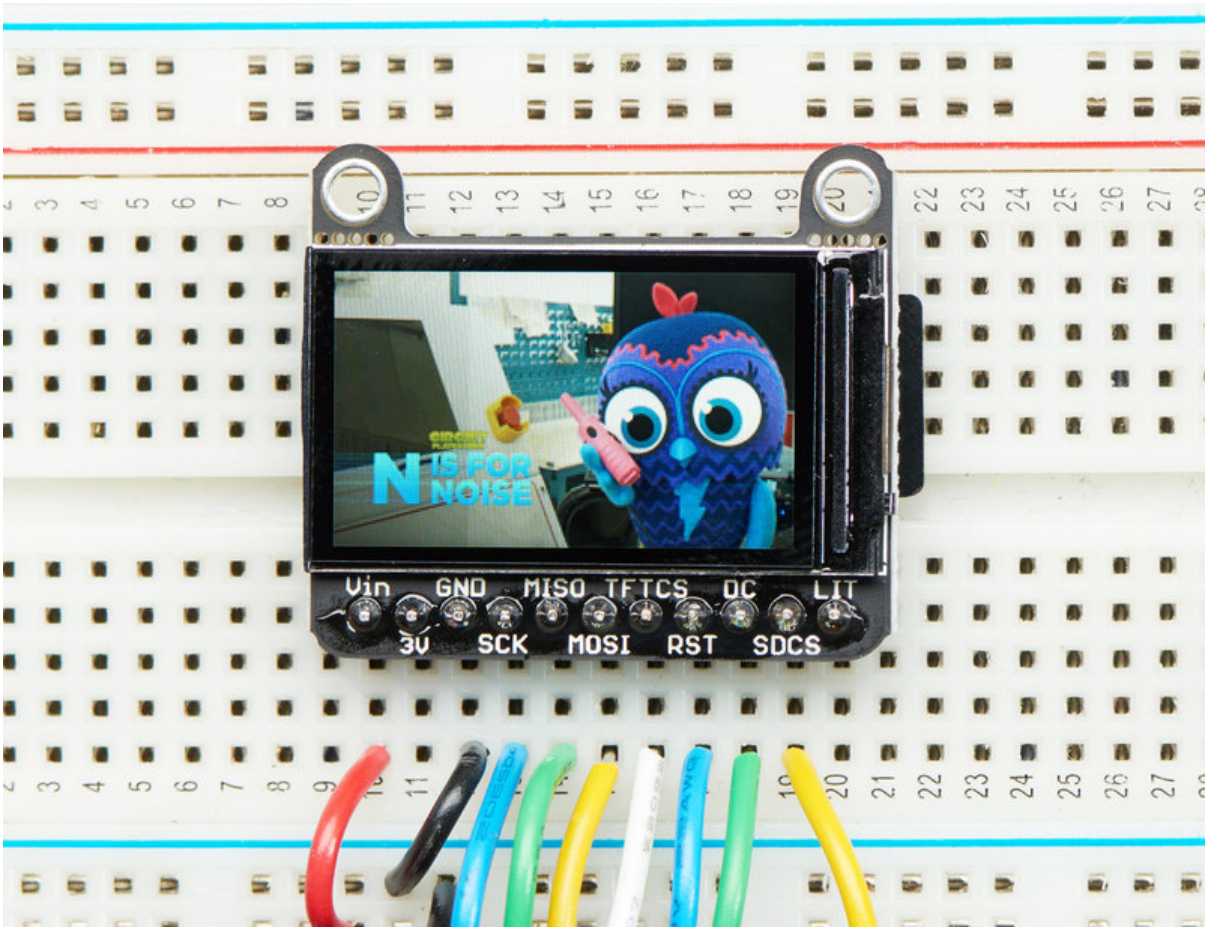
Adafruit 1.14" 240x135 Color TFT Breakout LCD Display

Created by Melissa LeBlanc-Williams



https://learn.adafruit.com/adafruit-1-14-240x135-color-tft-breakout

Last updated on 2024-06-03 02:56:46 PM EDT

Table of Contents

Overview 3

Pinouts 6

Arduino Wiring & Test 7

- Basic Graphics Test Wiring
- Install Arduino Libraries
- Changing Pins

Adafruit GFX library 12

Drawing Bitmaps 13

CircuitPython Displayio Quickstart 16

- Preparing the Breakout
- Required CircuitPython Libraries
- Code Example Additional Libraries
- CircuitPython Code Example
- Where to go from here

Python Wiring and Setup 24

- Wiring
- ILI9341 and HX-8357-based Displays
- ST7789 and ST7735-based Displays
- SSD1351-based Displays
- SSD1331-based Display
- Setup
- Python Installation of RGB Display Library
- DejaVu TTF Font
- Pillow Library

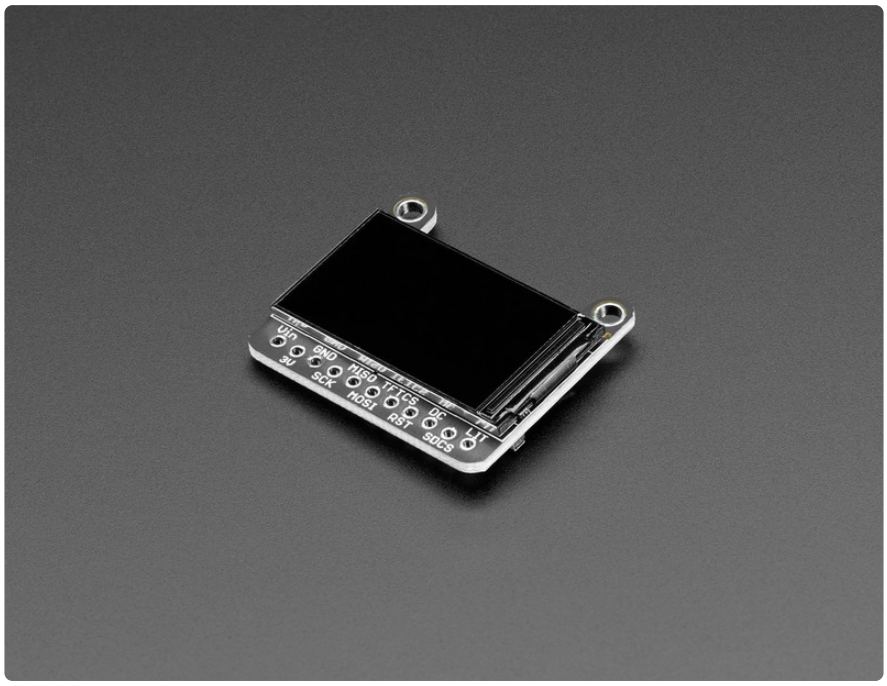
Python Usage 32

- Turning on the Backlight
- Displaying an Image
- Drawing Shapes and Text
- Displaying System Information

Downloads 45

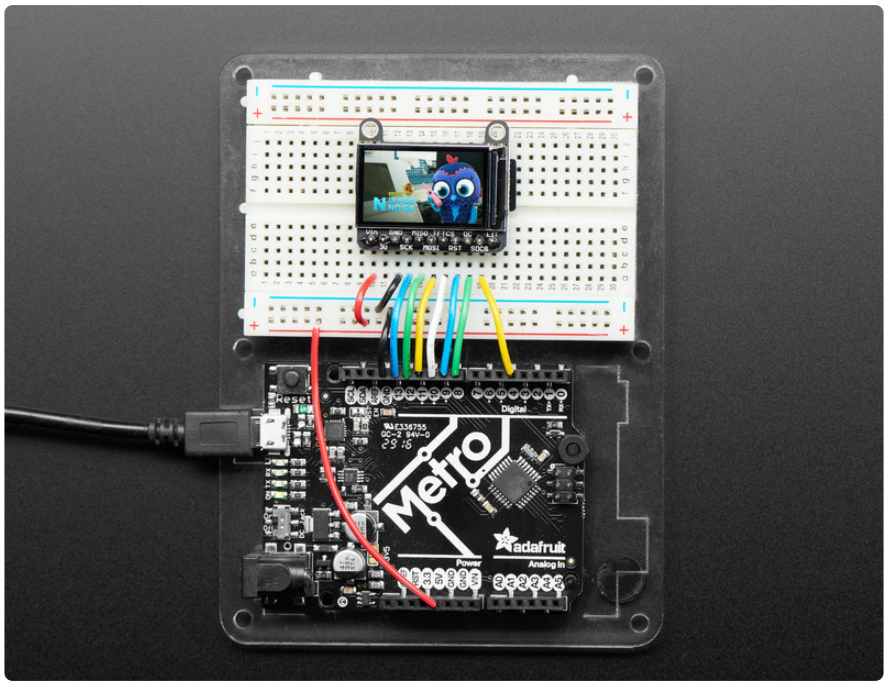
- Files
- Fab Print
- Schematic

Overview

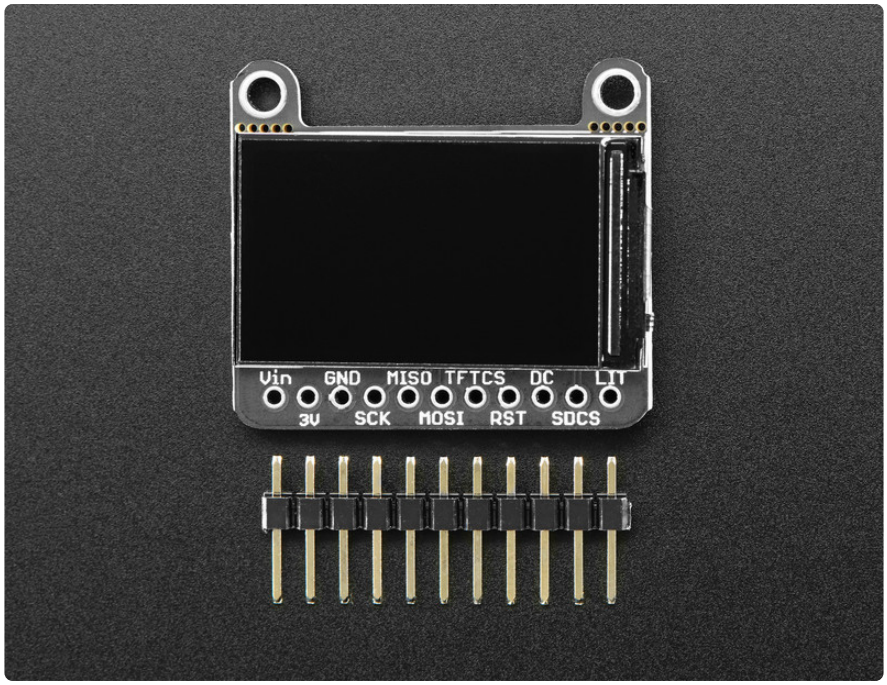


Say hello to our **1.14" 240x135 Color TFT Display w/ MicroSD Card Breakout** – we think it's **T**-**F**-**T**errific! It's the size of your thumbnail, with glorious 240x135 high res pixel color. This very very small display is only 1.14" diagonal, packed with RGB pixels, for making very small, high-density displays.

We've been looking for a display like this for a long time - it's **so small** only 1.14" diagonal but has a high density 260 ppi, 240x135 pixel display with full-angle viewing. It looks a lot like our 0.96" 160x80 display, but has 2.5x as many pixels. We've seen displays of this caliber used in smartwatches and small electronic devices but they've always used a MIPI interface. Finally, we found one that is SPI, and it has a friendly display driver, so it works with any and all microcontrollers or microcomputers!

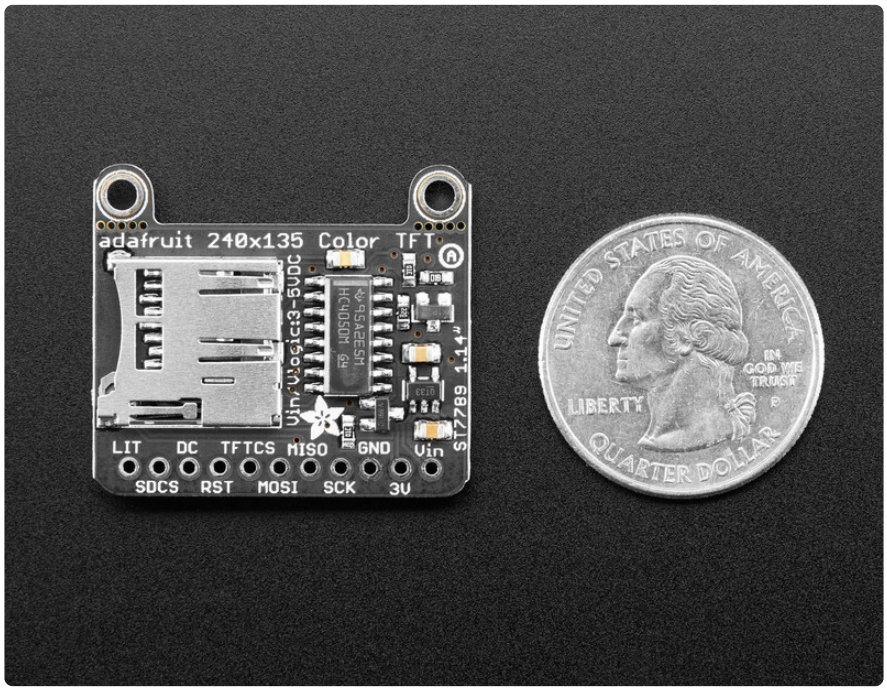


This lovely little display breakout is the best way to add a small, colorful and very bright display to any project. Since the display uses 4-wire SPI to communicate, and has its own pixel-addressable frame buffer, it can be used with every kind of microcontroller. Even a very small one with low memory and few pins available! The 1.14" display has 240x135 16-bit full color pixels and is an **IPS** display, so the color looks great up to 80 degrees off axis in any direction. The TFT driver (ST7789) is very similar to the popular ST7735, and our Arduino library supports it as well.

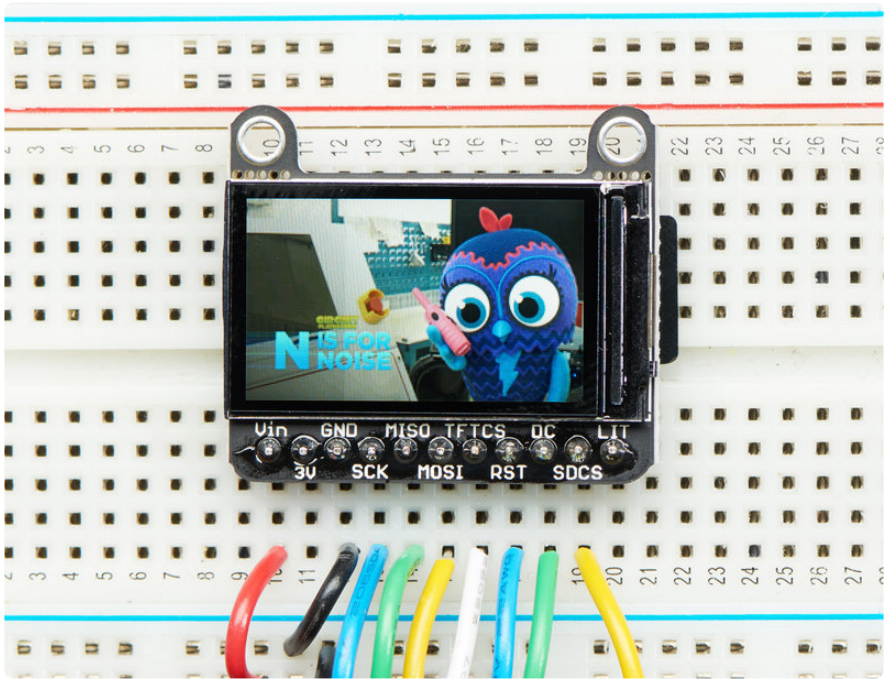


Our breakout has the TFT display soldered on (it uses a delicate flex-circuit connector) as well as a ultra-low-dropout 3.3V regulator and a 3/5V level shifter so you can use it with 3.3V or 5V power and logic. We also had a little space so we

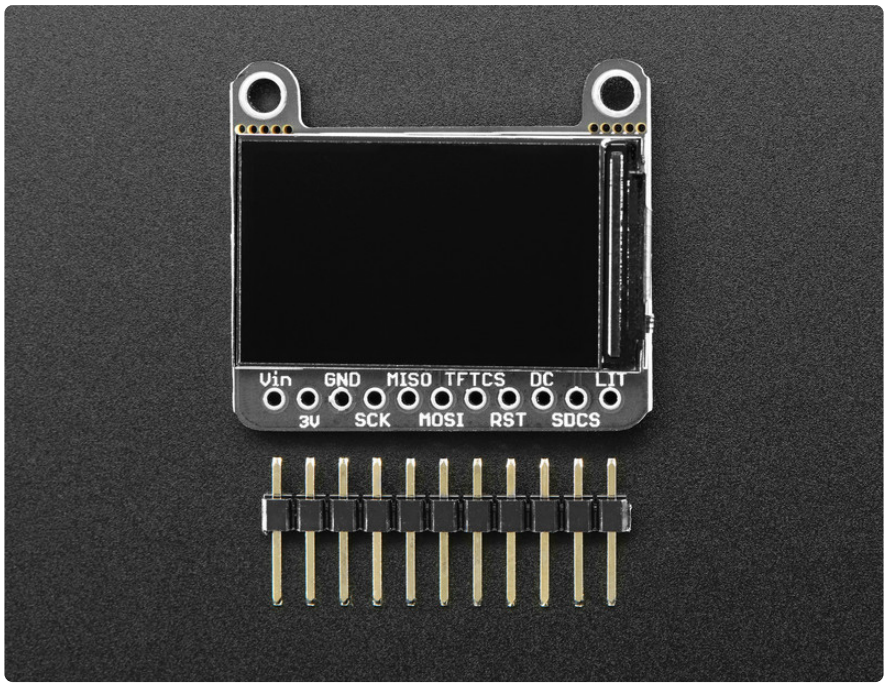
placed a microSD card holder so you can easily load full color bitmaps from a FAT16/FAT32 formatted microSD card. The microSD card is not included, but you can pick one up here (http://adafru.it/102).



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 IDE but can be easily ported to your favorite microcontroller!



Pinouts



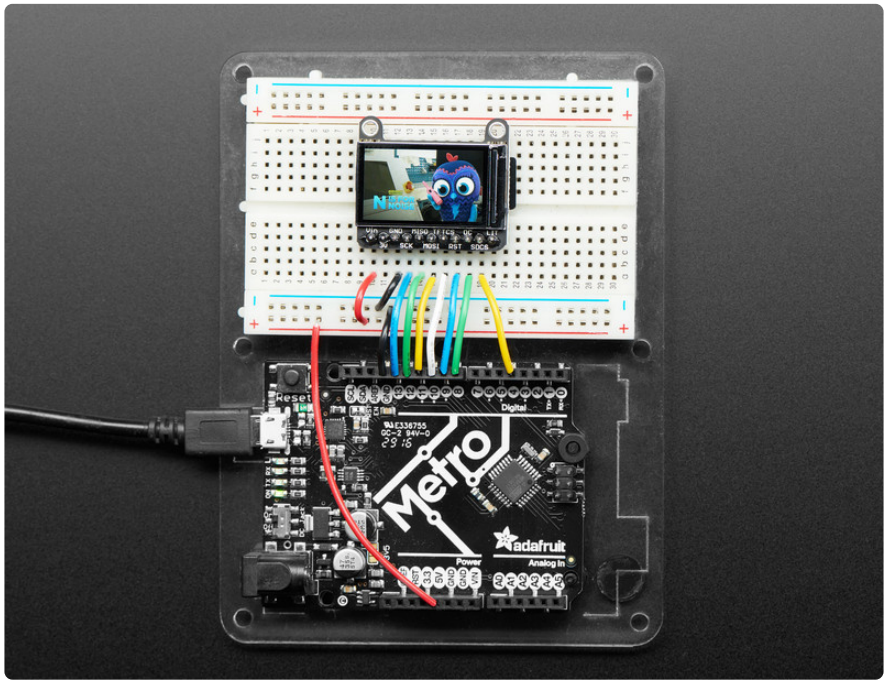
This color display uses SPI to receive image data. That means you need at least 4 pins - clock, data in, TFT cs and d/c. If you'd like to have SD card usage too, add another 2 pins - data out and card cs. However, there's a couple other pins you may want to use, lets go thru them all!

- **3-5V / Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3V** - this is the 3.3V output from the onboard regulator
- **GND** - this is the power and signal ground pin
- **SCK** - this is the SPI clock input pin. Use 3-5V logic level
- **MISO** - this is the SPI Microcontroller In Serial Out pin, it's used for the SD card. It isn't used for the TFT display which is write-only. It is 3.3V logic out (but can be read by 5V logic)
- **MOSI** - this is the SPI Microcontroller Out Serial In pin, it is used to send data from the microcontroller to the SD card and/or TFT. Use 3-5V logic level
- **TFTCS** - this is the TFT SPI chip select pin. Use 3-5V logic level
- **RST** - this is the TFT reset pin. Connect to ground to reset the TFT! It's best to have this pin controlled by the library so the display is reset cleanly, but you can also connect it to the Arduino Reset pin, which works for most cases. There is an automatic-reset chip connected so it will reset on power-up. Use 3-5V logic level
- **DC** - this is the TFT SPI data or command selector pin. Use 3-5V logic level
- **SD Card CS / SDCS -** this is the SD card chip select, used if you want to read from the SD card. Use 3-5V logic level

- **LIT** - this is the PWM input for the backlight control. It is by default pulled high (backlight on) you can PWM at any frequency or pull down to turn the backlight off. Use 3-5V logic level

---

Arduino Wiring & Test



Basic Graphics Test Wiring

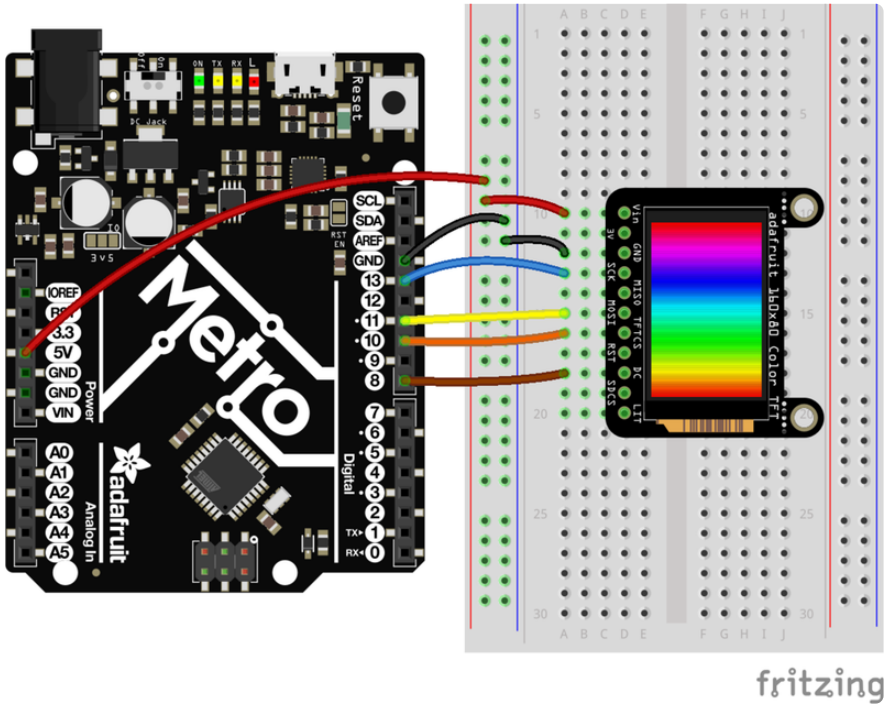
Wiring up the display in SPI mode is pretty easy as there are not that many pins! We'll be using hardware SPI, but you can also use software SPI (any pins) later. Start by connecting the power pins

- **3-5V Vin** connects to the microcontroller **5V** pin
- **GND** connects to Arduino ground
- **SCK** connects to SPI clock. On Arduino Uno/Duemilanove/328-based, thats **Digital 13**. On Mega, its **Digital 52** and on other chips its **ICSP-3** (See SPI Connections for more details (https://adafru.it/d5h))
- **MISO** is not connected
- **MOSI** connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, thats **Digital 11**. On Mega, its **Digital 51** and on other chips its **ICSP-4** (See SPI Connections for more details (https://adafru.it/d5h))
- **TFTCS** connects to our SPI Chip Select pin. We'll be using **Digital 10** but you can later change this to any pin

- **RST** connects to our Display Reset pin. We'll be using **Digital 9** but you can later change this pin too.
- **DC** connects to our SPI data/command select pin. We'll be using **Digital 8** but you can later change this pin too.

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

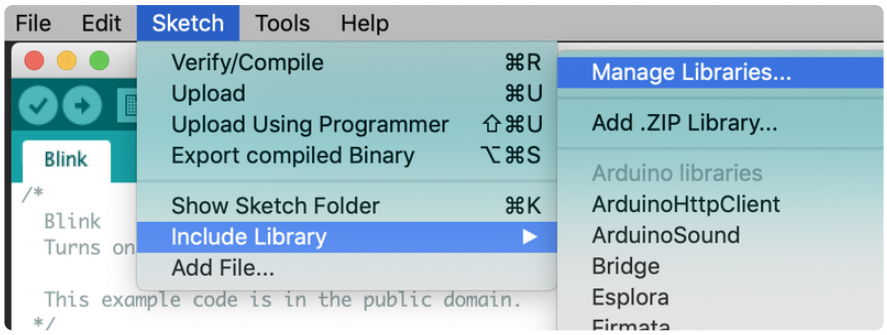
The following diagram uses a 0.96 160x80 display, which is the same size breakout and has the exact same pinout as the 1.14" 240x135 display.



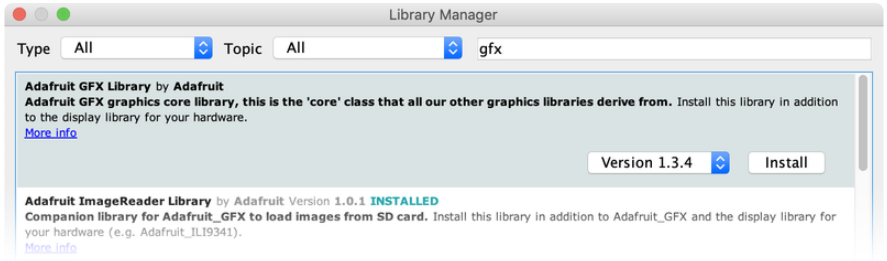
Install Arduino Libraries

We have example code ready to go for use with these TFTs. It's written for Arduino, which should be portable to any microcontroller by adapting the C++ source.

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...”

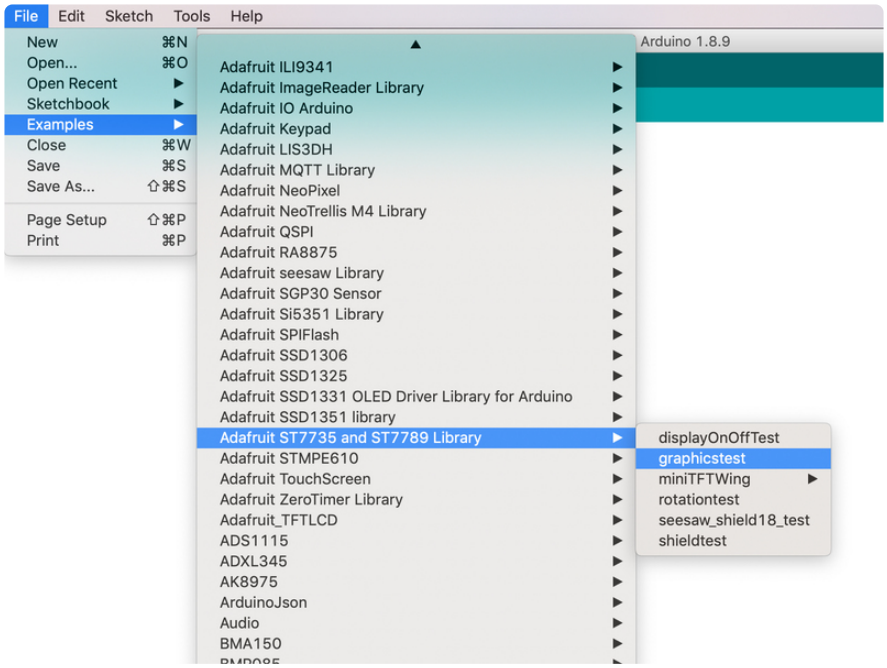


Type “gfx” in the search field to quickly find the first library — **Adafruit_GFX**:



Repeat the search and install steps, looking for the **Adafruit BusIO**, **Adafruit Zero DMA**, **Adafruit ST7735 and ST7789**, **Adafruit SPIFlash**, and **SdFat - Adafruit Fork** libraries.

After restarting the Arduino software, you should see a new **example** folder called **Adafruit ST7735 and ST7789,** and inside, an example called **graphicstest**.



Since this example is written for several displays, there are two changes we need to make in order to use it with the 1.14" display.

First, in the graphicstest source code, look for the lines as follows:

```
// For 1.44" and 1.8" TFT with ST7735 use:
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

// For 1.14", 1.3", 1.54", and 2.0" TFT with ST7789:
//Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);
```

comment out the line starting with `Adafruit_ST7735` and uncomment the line starting with `Adafruit_ST7789,` so it looks like:

```
// For 1.44" and 1.8" TFT with ST7735 use:
//Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

// For 1.14", 1.3", 1.54", and 2.0" TFT with ST7789:
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);
```

Second, we need to set the correct initializations sequence. In the graphicstest source code, look for the lines as follows:

```
// Use this initializer if using a 1.8" TFT screen:
  tft.initR(INITR_BLACKTAB);      // Init ST7735S chip, black tab

  // OR use this initializer (uncomment) if using a 1.44" TFT:
  //tft.initR(INITR_144GREENTAB); // Init ST7735R chip, green tab

  // OR use this initializer (uncomment) if using a 0.96" 160x80 TFT:
  //tft.initR(INITR_MINI160x80);  // Init ST7735S mini display

  // OR use this initializer (uncomment) if using a 1.3" or 1.54" 240x240 TFT:
  //tft.init(240, 240);           // Init ST7789 240x240

  // OR use this initializer (uncomment) if using a 2.0" 320x240 TFT:
  //tft.init(240, 320);           // Init ST7789 320x240

  // OR use this initializer (uncomment) if using a 1.14" 240x135 TFT:
  //tft.init(135, 240);           // Init ST7789 240x135
```

comment out the first line starting with `tft.initR` and uncomment the sixth line starting with `tft.init`, so it looks like:

```
// Use this initializer if using a 1.8" TFT screen:
  //tft.initR(INITR_BLACKTAB);      // Init ST7735S chip, black tab

  // OR use this initializer (uncomment) if using a 1.44" TFT:
  //tft.initR(INITR_144GREENTAB); // Init ST7735R chip, green tab

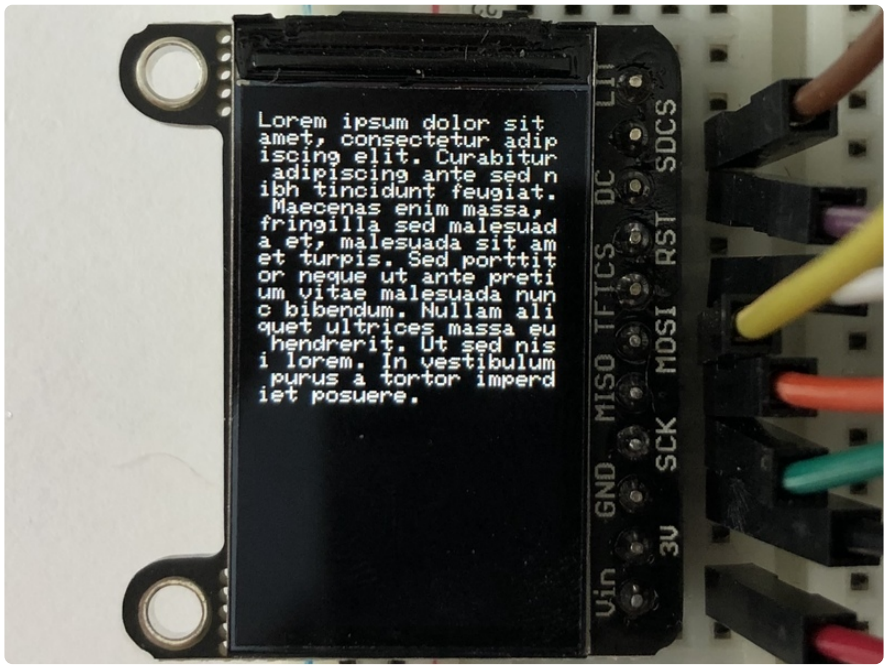
  // OR use this initializer (uncomment) if using a 0.96" 160x80 TFT:
  //tft.initR(INITR_MINI160x80);  // Init ST7735S mini display

  // OR use this initializer (uncomment) if using a 1.3" or 1.54" 240x240 TFT:
  //tft.init(240, 240);           // Init ST7789 240x240

  // OR use this initializer (uncomment) if using a 2.0" 320x240 TFT:
  //tft.init(240, 320);           // Init ST7789 320x240
```

```
  // OR use this initializer (uncomment) if using a 1.14" 240x135 TFT:
  tft.init(135, 240);           // Init ST7789 240x135
```

Now upload the sketch to your Arduino. You may need to press the Reset button to reset the Arduino and TFT. You should see a collection of graphical tests draw out on the TFT.



Changing Pins

Now that you have it working, there's a few things you can do to change around the pins.

If you're using Hardware SPI, the CLOCK and MOSI pins are 'fixed' and can't be changed. But you can change to software SPI, which is a bit slower, and that lets you pick any pins you like. Find these lines:

```
// OPTION 1 (recommended) is to use the HARDWARE SPI pins, which are unique
// to each board and not reassignable. For Arduino Uno: MOSI = pin 11 and
// SCLK = pin 13. This is the fastest mode of operation and is required if
// using the breakout board's microSD card.

// For 1.44" and 1.8" TFT with ST7735 use:
//Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

// For 1.14", 1.3", 1.54", and 2.0" TFT with ST7789:
Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_RST);


// OPTION 2 lets you interface the display using ANY TWO or THREE PINS,
// tradeoff being that performance is not as fast as hardware SPI above.
//#define TFT_MOSI 11  // Data out
//#define TFT_SCLK 13  // Clock out
```

```
// For ST7735-based displays, we will use this call
//Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
TFT_RST);

// OR for the ST7789-based displays, we will use this call
//Adafruit_ST7789 tft = Adafruit_ST7789(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
TFT_RST);
```

Comment out option 1, and uncomment option 2 for the ST7789. Then you can change the **TFT_** pins to whatever pins you'd like!

The 1.14" TFT display has an auto-reset circuit on it so you probably don't need to use the **RST** pin. You can change

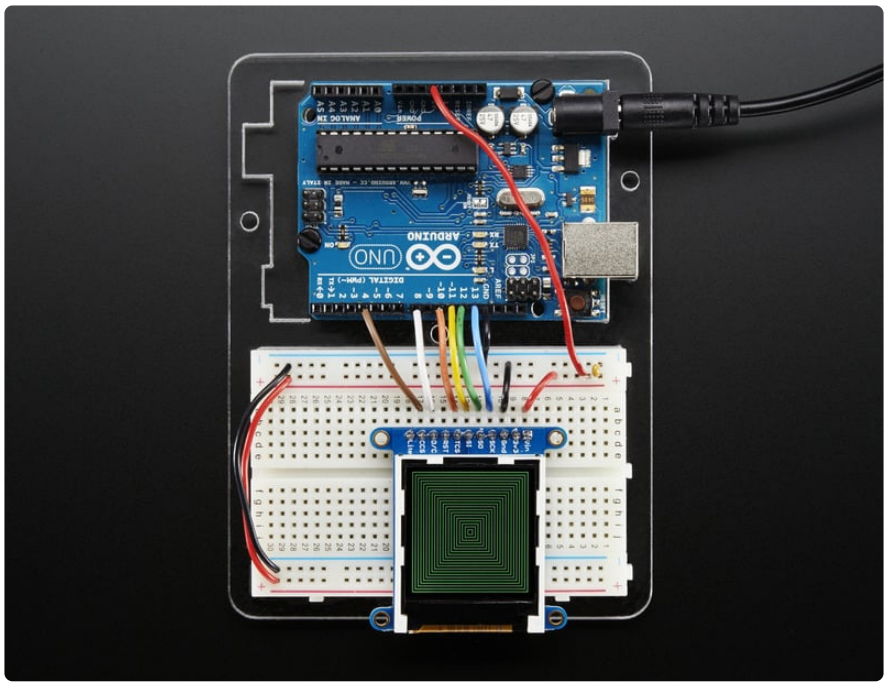
```
#define TFT_RST    9
```

to

```
#define TFT_RST   -1
```

so that pin isn't used either. Or connect it up for manual TFT resetting!

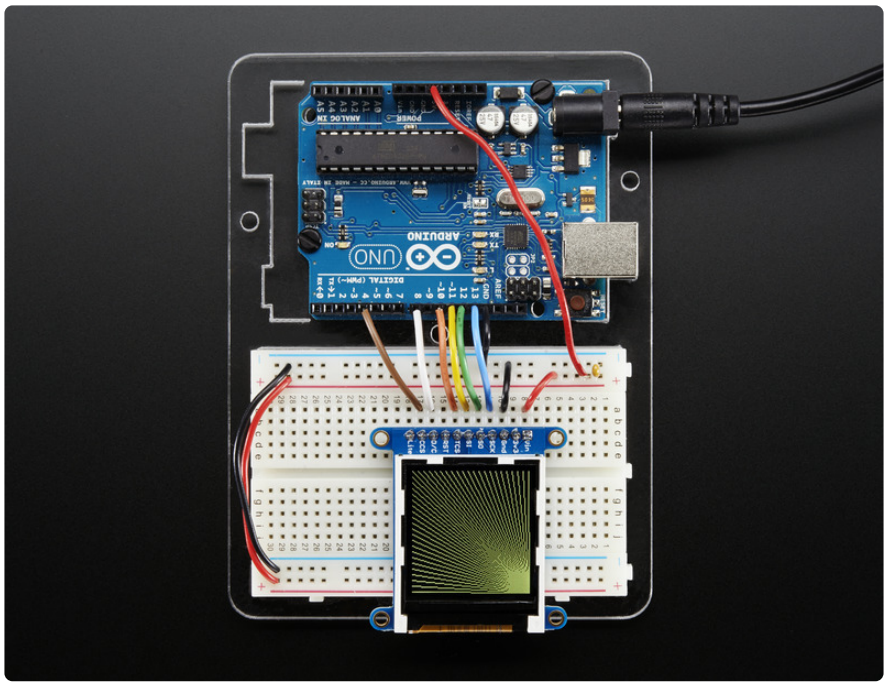
Adafruit GFX library



The Adafruit_GFX library for Arduino provides a common syntax and set of graphics functions for all of our TFT, LCD and OLED displays. This allows Arduino sketches to easily be adapted between display types with minimal fuss...and any new features, performance improvements and bug fixes will immediately apply across our complete

offering of color displays.

The GFX library is what lets you draw points, lines, rectangles, round-rects, triangles, text, etc.



Check out our detailed tutorial here http://learn.adafruit.com/adafruit-gfx-graphics-library (https://adafru.it/aPx) It covers the latest and greatest of the GFX library!

Drawing Bitmaps

There is a built in microSD card slot into the breakout, and we can use that to load bitmap images! You will need a microSD card formatted **FAT16 or FAT32** (they almost always are by default).

It's really easy to draw bitmaps! Let's start by downloading this image of **Minerva**

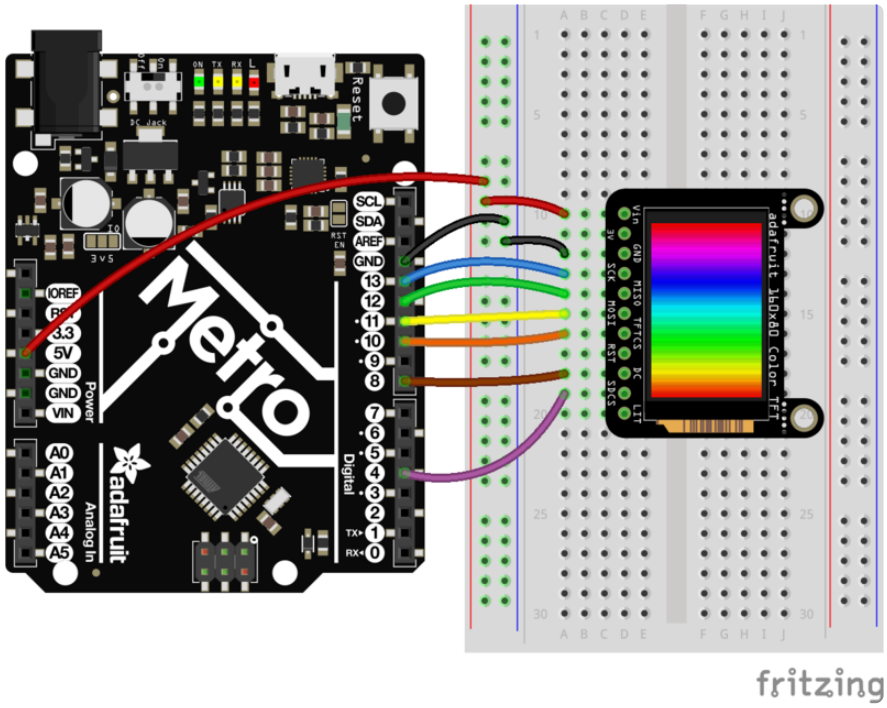


Copy **minerva.bmp** into the base directory of a microSD card and insert it into the microSD socket in the breakout.

Two more wires are required to interface with the onboard SD card:

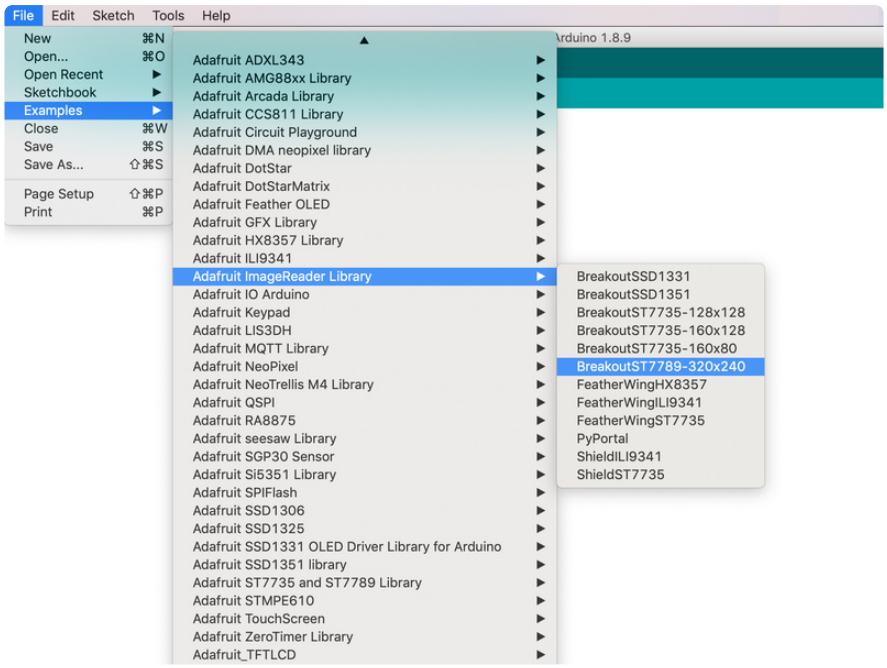
- You'll need to connect up the **SO** pin to the SPI MISO line on your microcontroller. On Arduino Uno/Duemilanove/328-based, thats **Digital 12**. On Mega's, its **Digital 50** and on Leonardo/Due its **ICSP-1** (See SPI Connections for more details (https://adafru.it/d5h))
- Also, the **CCS** or **CC** pin to **Digital 4** on your Arduino as well. You can change this pin later, but stick with this for now.

The following diagram uses a 0.96 160x80 display, which is the same size breakout and has the exact same pinout as the 1.14" 240x135 display.



You may want to try the **SD library** examples before continuing, especially one that lists all the files on the SD card

Open the **File→examples→Adafruit ImageReader Library→BreakoutST7789 - 320x240** example:



You will need to change a couple of lines for this to work with the 240x135 display. First, we need to set this to the correct display size, so look for the following code:

```
tft.init(240, 320);           // Init ST7789 320x240
```

and change it to this:

```
tft.init(135, 240);           // Init ST7789 240x135
```

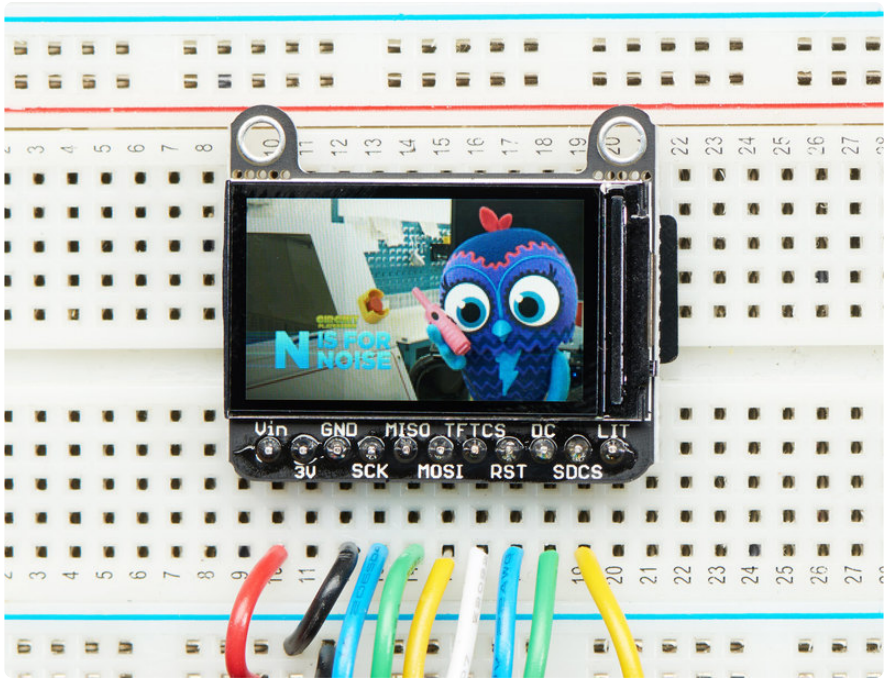
Second, we need to change the filename that we are loading, so look for the following lines of code.

```
Serial.print(F("Loading purple.bmp to screen..."));
stat = reader.drawBMP("/purple.bmp", tft, 0, 0);
```

and change them to this:

```
Serial.print(F("Loading minerva.bmp to screen..."));
stat = reader.drawBMP("/minerva.bmp", tft, 0, 0);
```

Now upload the example sketch to the Arduino. You should see ADABOT appear! If you have any problems, check the serial console for any messages such as not being able to initialize the microSD card or not finding the image.



To make new bitmaps, make sure they are less than 240 by 135 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. You can rotate images using the **setRotation()** procedure

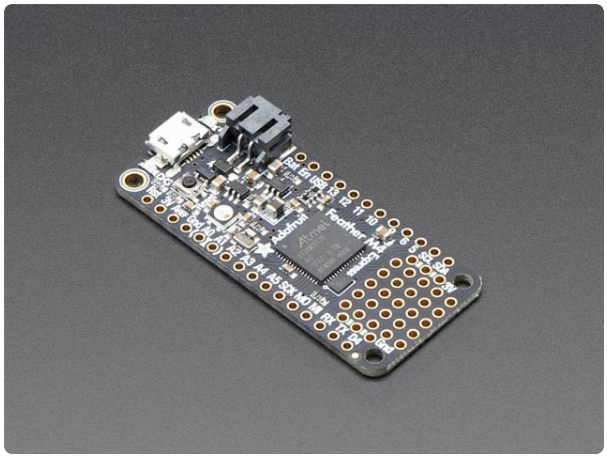
You can draw as many images as you want - don't forget the names must be less than 8 characters long. Just copy the BMP drawing routines below loop() and call

bmpDraw(bmpfilename, x, y);

For each bitmap. They can be smaller than 240x135 and placed in any location on the screen.

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
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 (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.

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_st7789**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_st7789** 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. You'll also need to copy over the following library from the bundle:

- **adafruit_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

```
# 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

# Starting in CircuitPython 9.x fourwire will be a seperate internal library
# rather than a component of the displayio library
try:
    from fourwire import FourWire
except ImportError:
    from displayio import FourWire
from adafruit_display_text import label
from adafruit_st7789 import ST7789

# First set some parameters used for shapes and text
BORDER = 20
FONTSCALE = 2
BACKGROUND_COLOR = 0x00FF00  # Bright Green
FOREGROUND_COLOR = 0xAA0088  # Purple
TEXT_COLOR = 0xFFFF00

# 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 = FourWire(spi, command=tft_dc, chip_select=tft_cs)
display = ST7789(
    display_bus, rotation=270, width=240, height=135, rowstart=40, colstart=53
)

# Make the display context
splash = displayio.Group()
display.root_group = splash

color_bitmap = displayio.Bitmap(display.width, display.height, 1)
```

```
color_palette = displayio.Palette(1)
color_palette[0] = BACKGROUND_COLOR

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(
    display.width - BORDER * 2, display.height - BORDER * 2, 1
)
inner_palette = displayio.Palette(1)
inner_palette[0] = FOREGROUND_COLOR
inner_sprite = displayio.TileGrid(
    inner_bitmap, pixel_shader=inner_palette, x=BORDER, y=BORDER
)
splash.append(inner_sprite)

# Draw a label
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=TEXT_COLOR)
text_width = text_area.bounding_box[2] * FONTSCALE
text_group = displayio.Group(
    scale=FONTSCALE,
    x=display.width // 2 - text_width // 2,
    y=display.height // 2,
)
text_group.append(text_area)  # Subgroup for text scaling
splash.append(text_group)

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_st7789` driver.

```
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_st7789 import ST7789
```

Next we define some parameters so that making changes to the shapes and text are easy. `BORDER` will be the distance between the background and foreground rectangles. `FONTSCALE` will be the multiplier for the font size. `BACKGROUND_COLOR` is the color of the larger outer rectangle and is set to `0x00FF00`, which is bright green by default. Colors are Hexadecimal values in the format of **RRGGBB**. `FOREGROUND_COLOR` is the color of the smaller inner rectangle and is set to purple by default. `TEXT_COLOR` is the color of the text that appears in the label and by default is yellow.

```
BORDER = 20
FONTSCALE = 2
BACKGROUND_COLOR = 0x00FF00  # Bright Green
FOREGROUND_COLOR = 0xAA0088  # Purple
TEXT_COLOR = 0xFFFF00
```

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. Next we set the Chip Select and Data/Command pins that will be used.

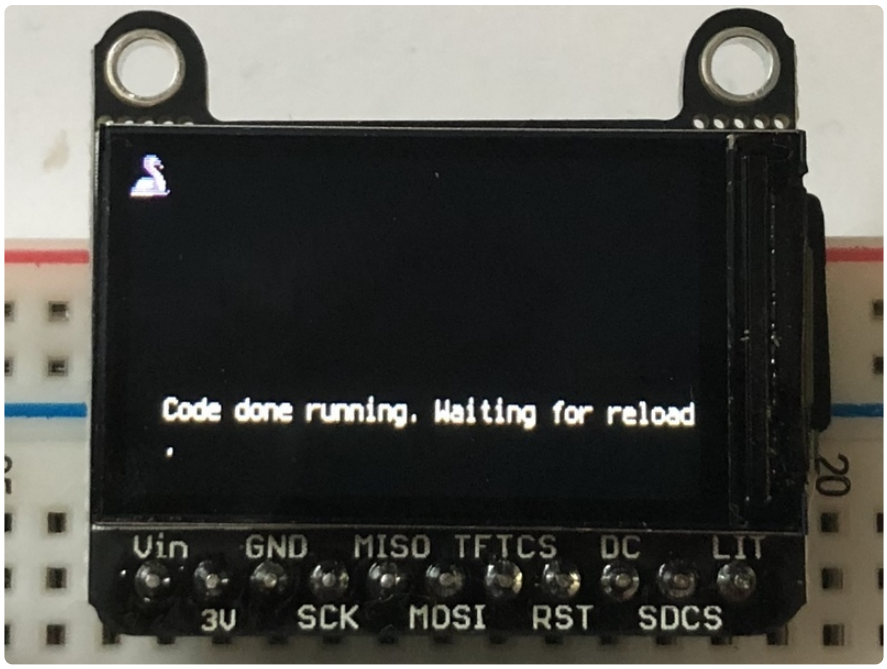
```
spi = board.SPI()
tft_cs = board.D5
tft_dc = board.D6
```

In the next line, we set the display bus to FourWire which makes use of the SPI bus. We would normally pass in reset with other displays, but this one has an automatic reset circuit built in.

```
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs)
```

Finally, we initialize the driver with a width of 240 and a height of 135. Because the ST7789 chip is capable of driving both 320 and 240 pixel width displays. With the 240x135 display, the rectangle is in the middle of that space, we will need to tell the display that our row starts at 40 pixels over and our columns start 53 pixels down. 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 = ST7789(display_bus, rotation=270, width=240, height=135, rowstart=40,
colstart=53)
```



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.

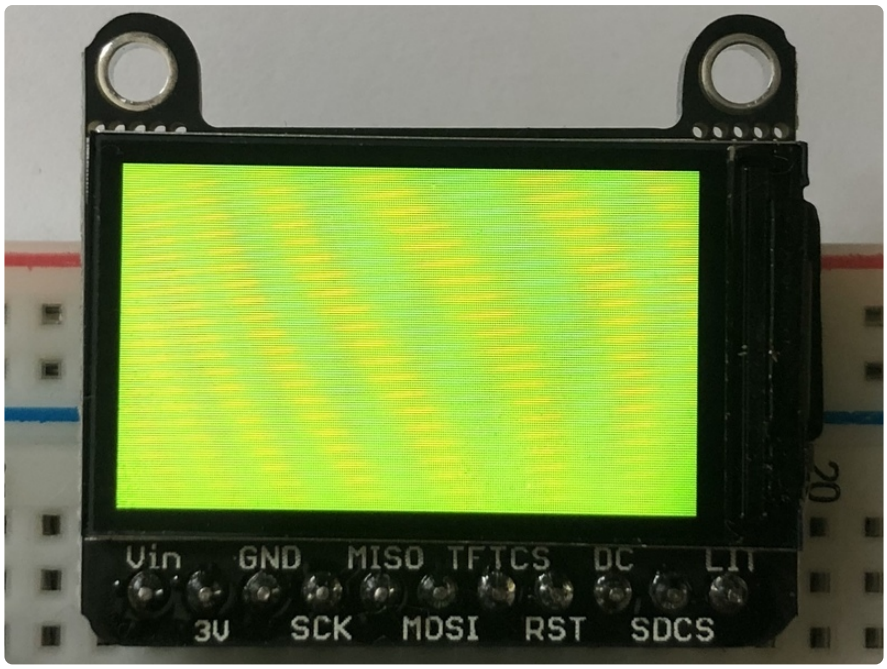
```
splash = displayio.Group(max_size=10)
display.show(splash)
```

Next we create a Bitmap, which is like a canvas that we can draw on, and set it to the same size as the display. 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 the value of `BACKGROUND_COLOR`. 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(display.width, display.height, 1)
color_palette = displayio.Palette(1)
color_palette[0] = BACKGROUND_COLOR
```

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)
```

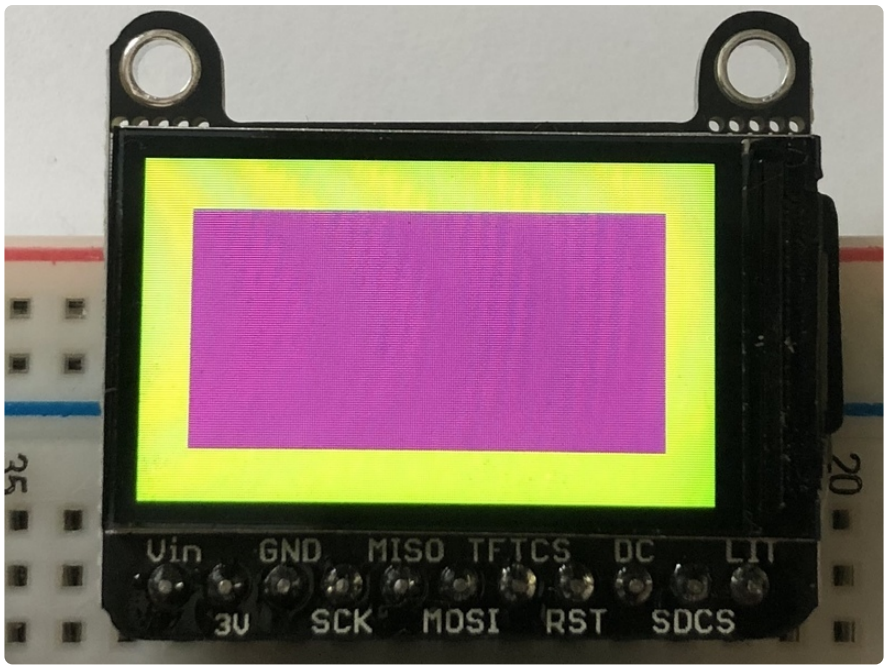


Next we will create a smaller purple square. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case, we will create a bitmap that is the size of the display with the value of `BORDER`, which is 20 pixels, subtracted from each side. The screen is **240x135**, so we'll end up subtracting 40 from each of those numbers.

We'll also want to place it at the position `(20, 20)` so that it ends up centered.

```
inner_bitmap = displayio.Bitmap(display.width - BORDER * 2, display.height - BORDER
* 2, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = FOREGROUND_COLOR
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=BORDER, y=BORDER)
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 and scale it up by a factor of two, which is what we have `FONTSCALE` set to. To scale the label only, we will make use of a subgroup, which we will then add to the main group.

We create the label first so that we can get the width of the bounding box and multiply it by the `FONTSCALE`. This gives us the actual with of the text.

Labels are automatically centered vertically, so we'll place it at half the display height for the Y coordinate, and we calculate the X coordinate to horizontally center the label. For the color, we just use the value inside of `TEXT_COLOR`.

```
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=TEXT_COLOR)
text_width = text_area.bounding_box[2] * FONTSCALE
text_group = displayio.Group(max_size=10, scale=FONTSCALE, x=display.width // 2 -
text_width // 2,
                             y=display.height // 2)
text_group.append(text_area) # Subgroup for text scaling
splash.append(text_group)
```

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)

Python Wiring and Setup

Wiring

It's easy to use display breakouts with Python and the Adafruit CircuitPython RGB Display (https://adafru.it/u1C) 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 (https://adafru.it/BSN).

Connect the display as shown below to your Raspberry Pi.

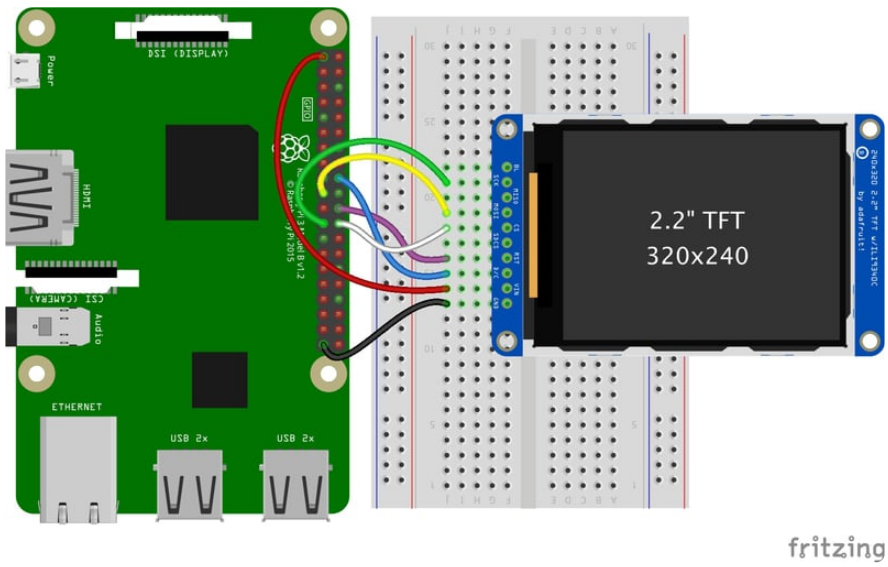
Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbtft driver, and want to use the TFT purely from 'user Python' code!

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, that's **SCLK**
- **MOSI** connects to SPI MOSI. On the Raspberry Pi, that's 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

https://adafru.it/H6C

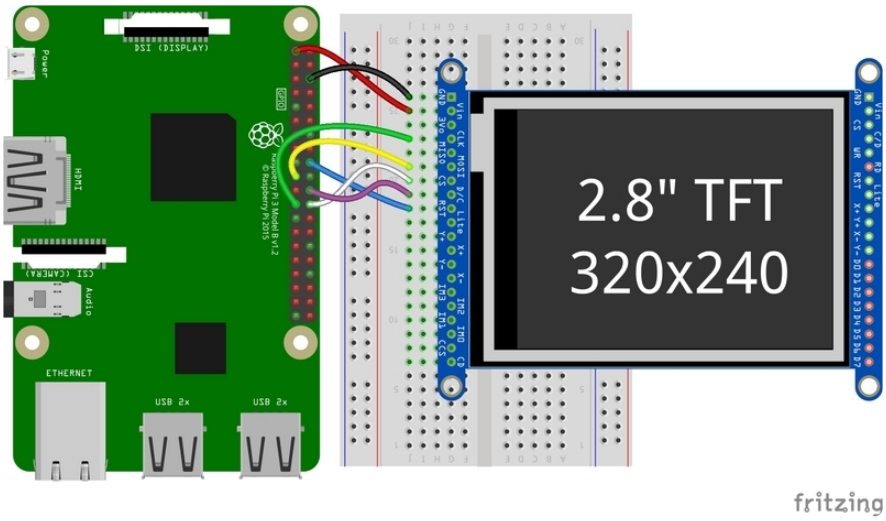
2.4", 2.8", 3.2", and 3.5" Displays

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

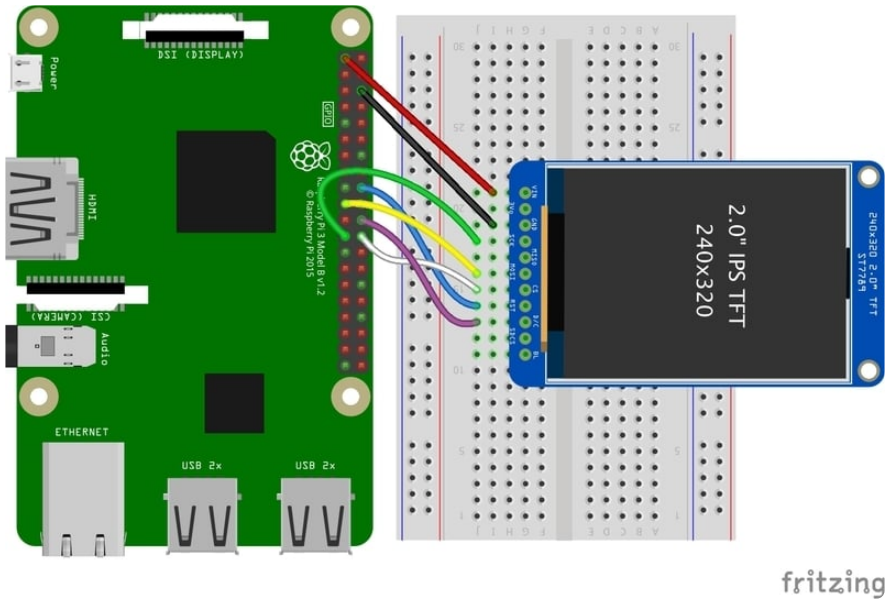
https://adafru.it/H7a

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.

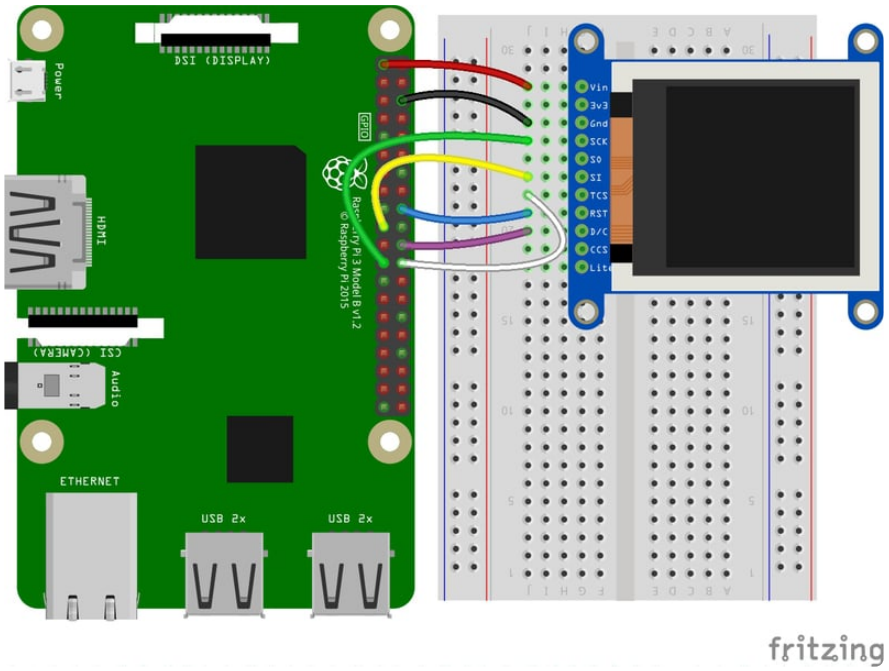


Download the Fritzing Diagram

https://adafru.it/H7A

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.

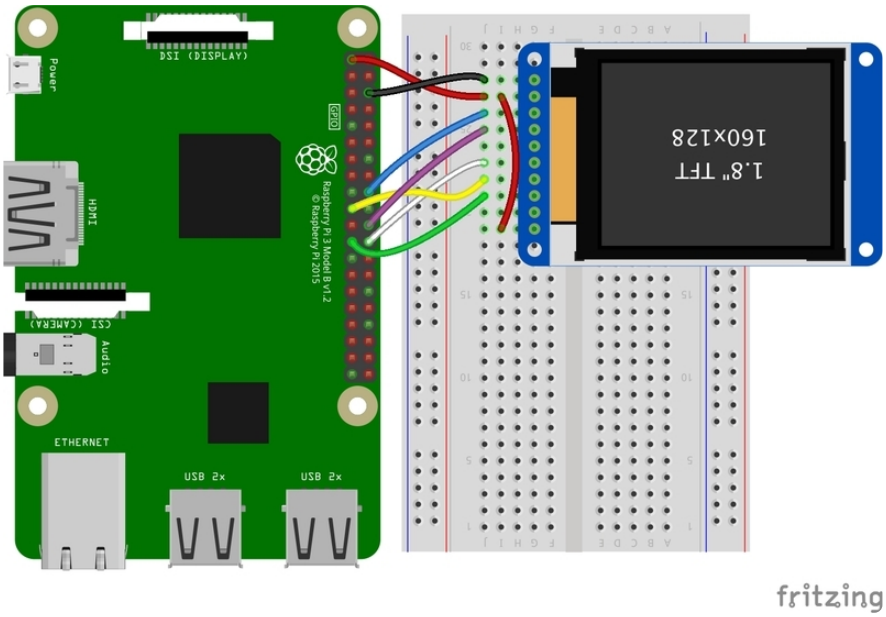


Download the Fritzing Diagram

https://adafru.it/H7B

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.



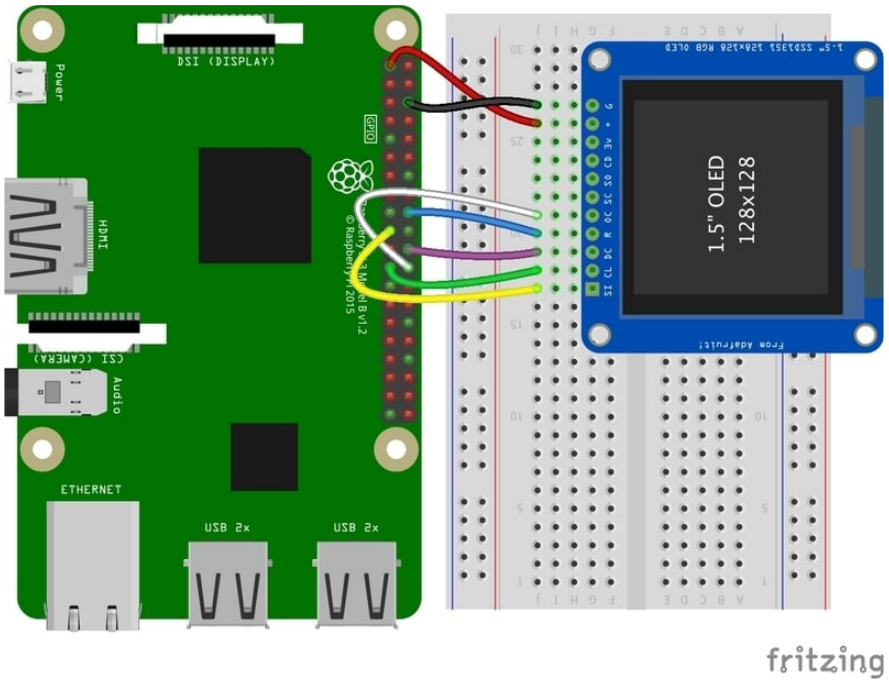
Download the Fritzing Diagram

https://adafru.it/H8a

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.



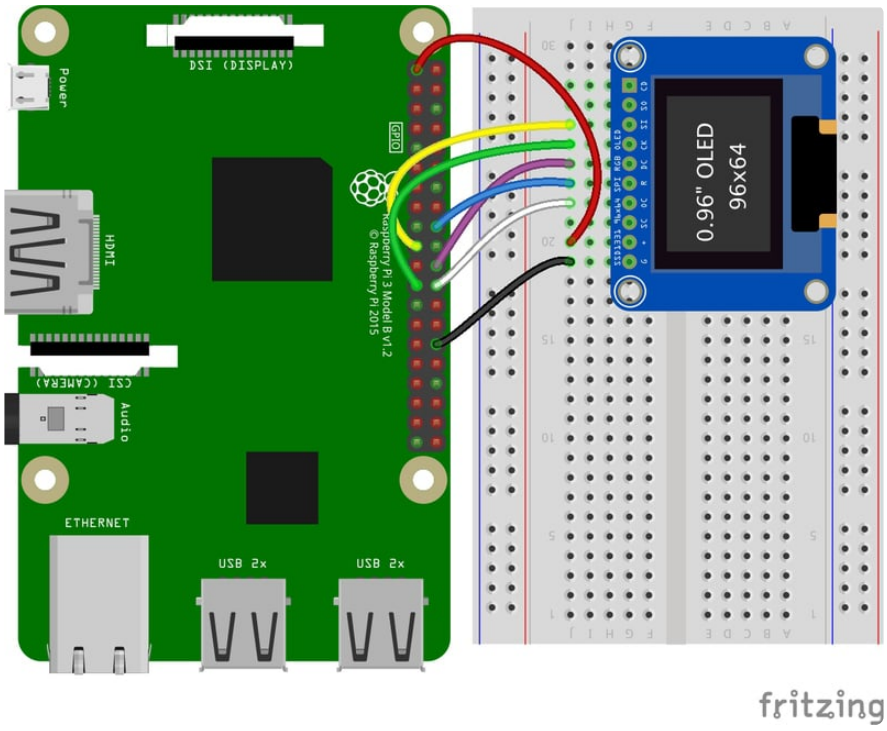
Download the Fritzing Diagram

https://adafru.it/H8A

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

https://adafru.it/OaF

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 (https://adafru.it/BSN)!

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`

If you installed the PIL through PIP, you may need to install some additional libraries:

- `sudo apt-get install libopenjp2-7 libtiff5 libatlas-base-dev`

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
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.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 &lt; 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.

```
# 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")
```

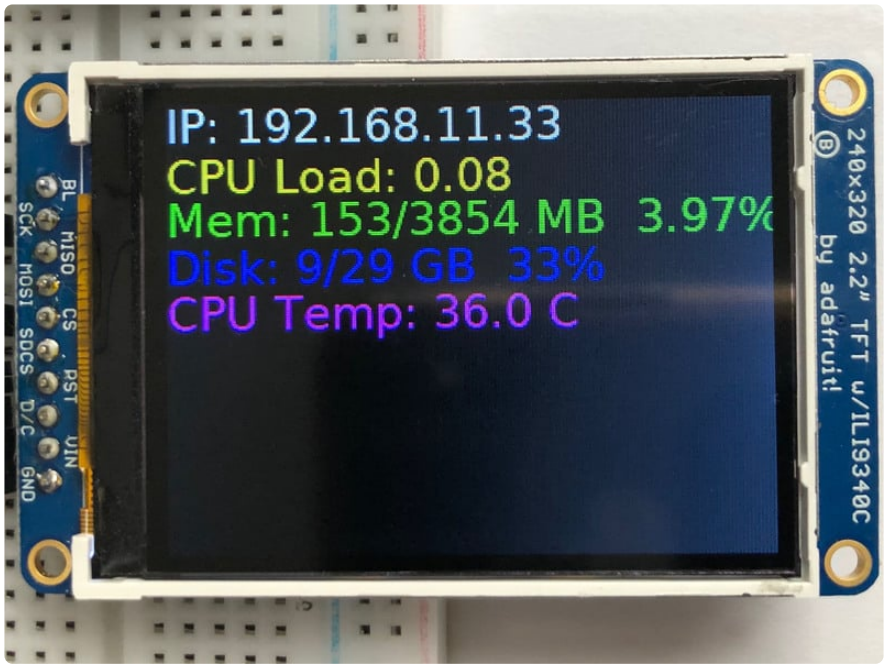
Now we display the information for the user. Here we use yet another way to pass color information. We can pass it as a color string using the pound symbol, just like we

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)
```

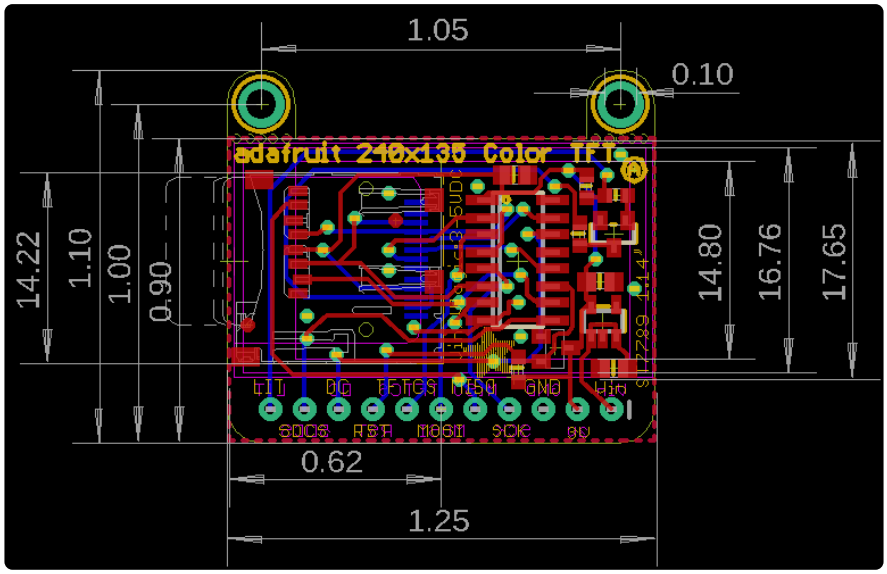


Downloads

Files

- 1.14" display EagleCAD files on GitHub (https://adafru.it/GfU)
- 3D models on GitHub (https://adafru.it/11vB)
- Display Module datasheet (https://adafru.it/GfV)
- ST7789VW datasheet 1.0 (https://adafru.it/GfW)

Fab Print



Schematic

