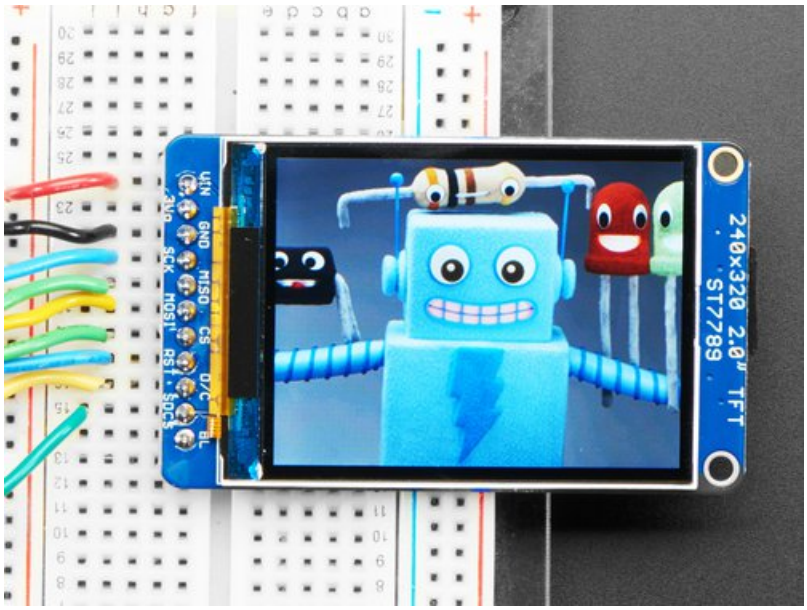
adafruit learning system

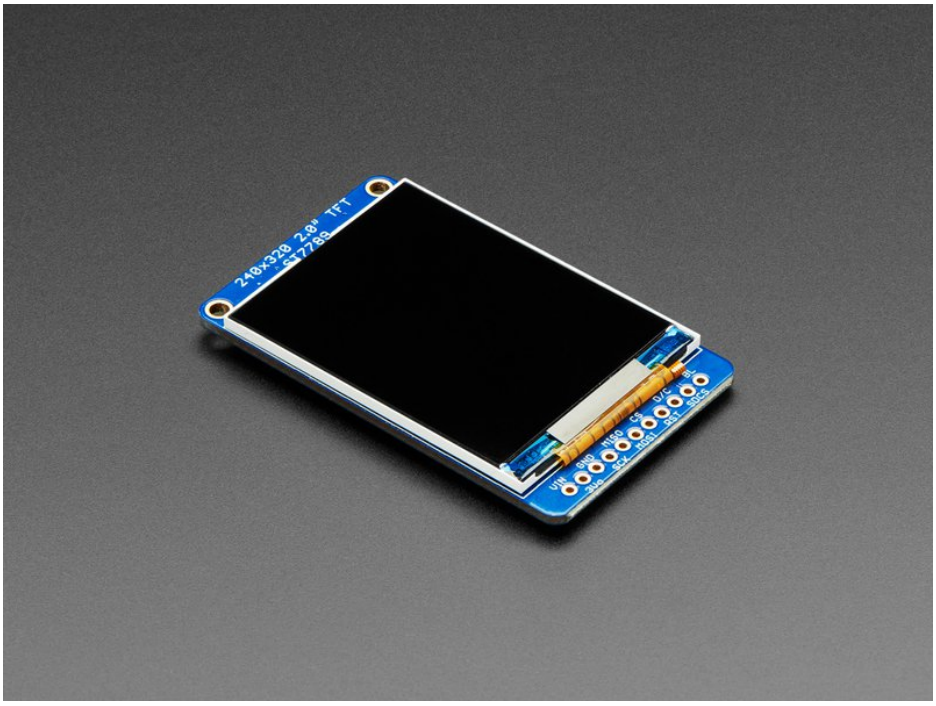
Adafruit 2.0" 320x240 Color IPS TFT Display

Created by Kattni Rembor

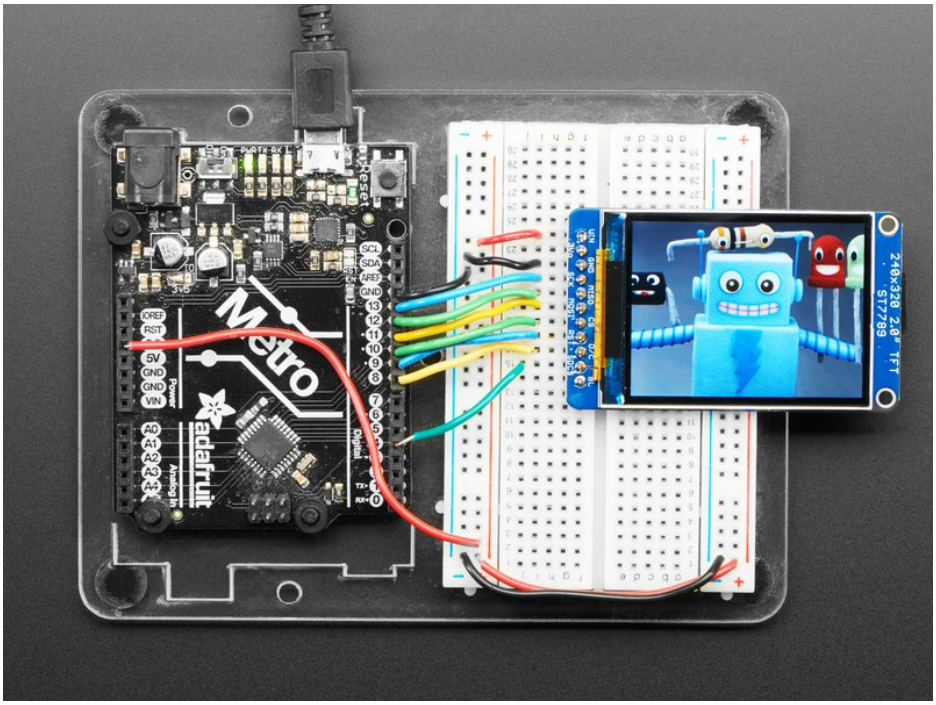


Last updated on 2019-08-15 06:43:09 PM UTC

Overview

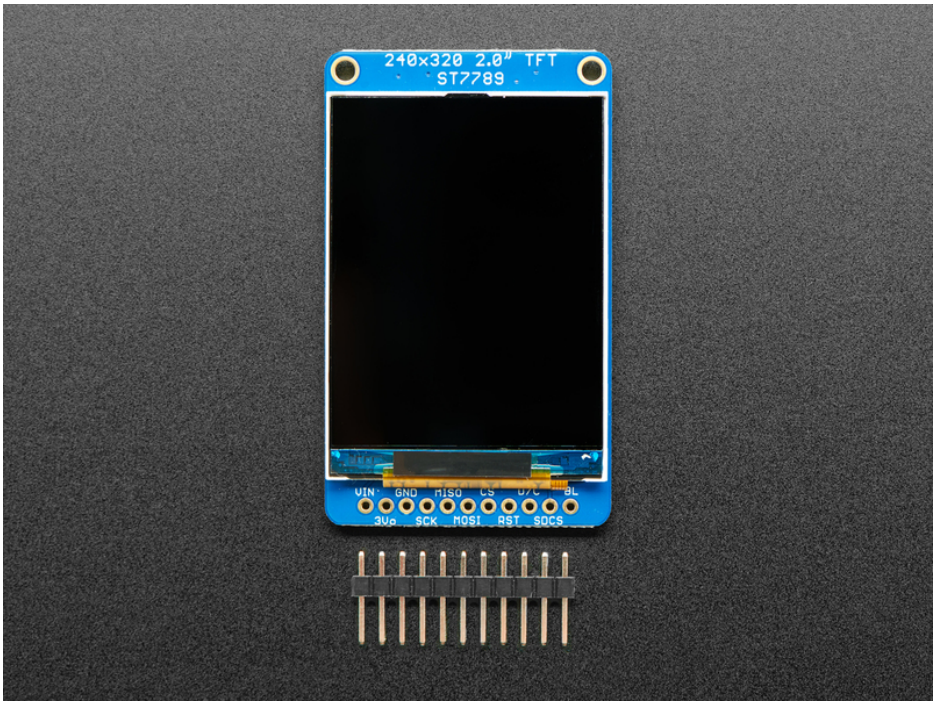


This gorgeous IPS display breakout is the best way to add a small, colorful and bright display to any project, with excellent visibility from any angle. Since the display uses 4-wire SPI to communicate and has its own pixel-addressable frame buffer, it can be used with every kind of microcontroller. Even a very small one with low memory and few pins available!

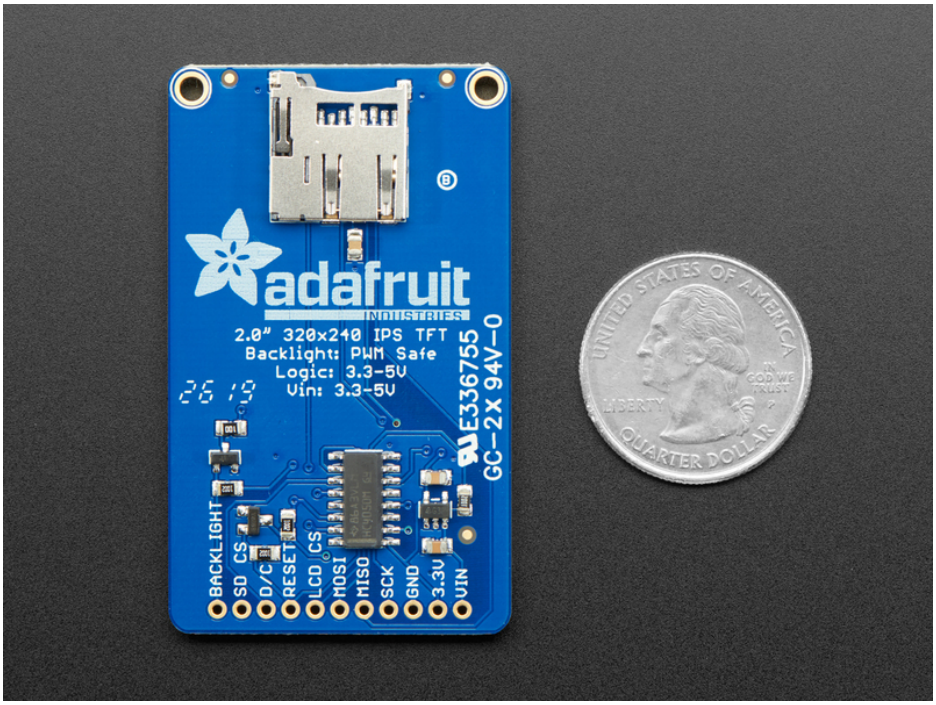


The 2.0" display has 320x240 color pixels. Unlike the low cost "Nokia 6110" and similar LCD displays, which are CSTN type and thus have poor color and slow refresh, this display is a true TFT! Not only that, but its an IPS display for vivid

color and high-angle visibility. The TFT driver (ST7789) can display full 18-bit color (262,144 shades), but almost all drivers will use just 16-bit color. The TFT will always come with the same driver chip so there's no worries that your code will not work from one to the other.

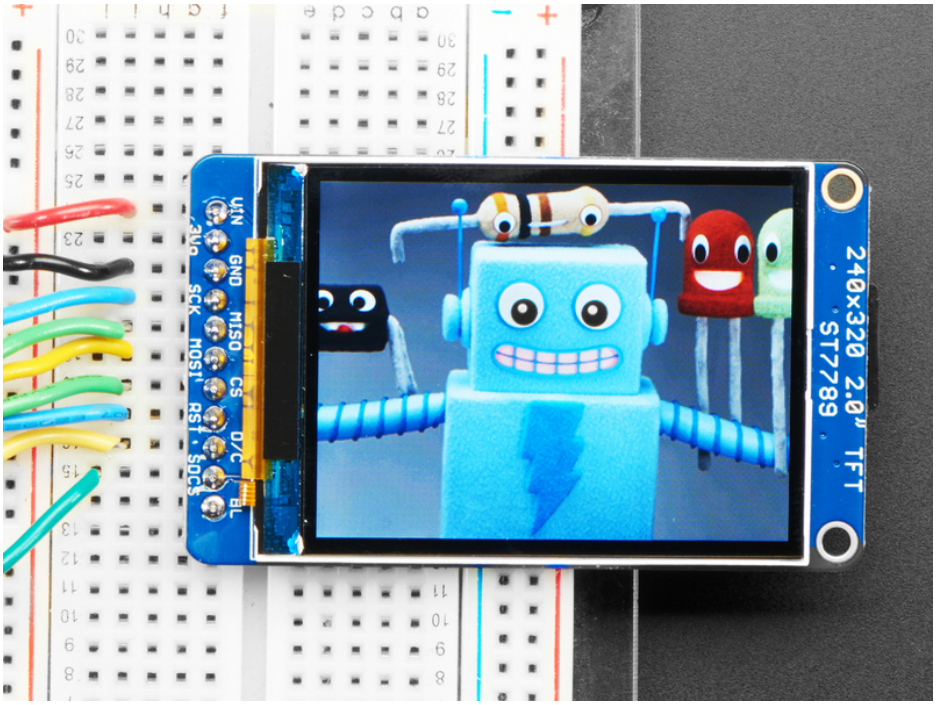


The breakout has the TFT display soldered on (it uses a delicate flex-circuit connector) as well as a ultra-low-dropout 3.3V regulator, auto-reset circuitry, and a 3/5V level shifter so you can use it with 3.3V or 5V power and logic. We also had a little space so we placed a microSD card holder so you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

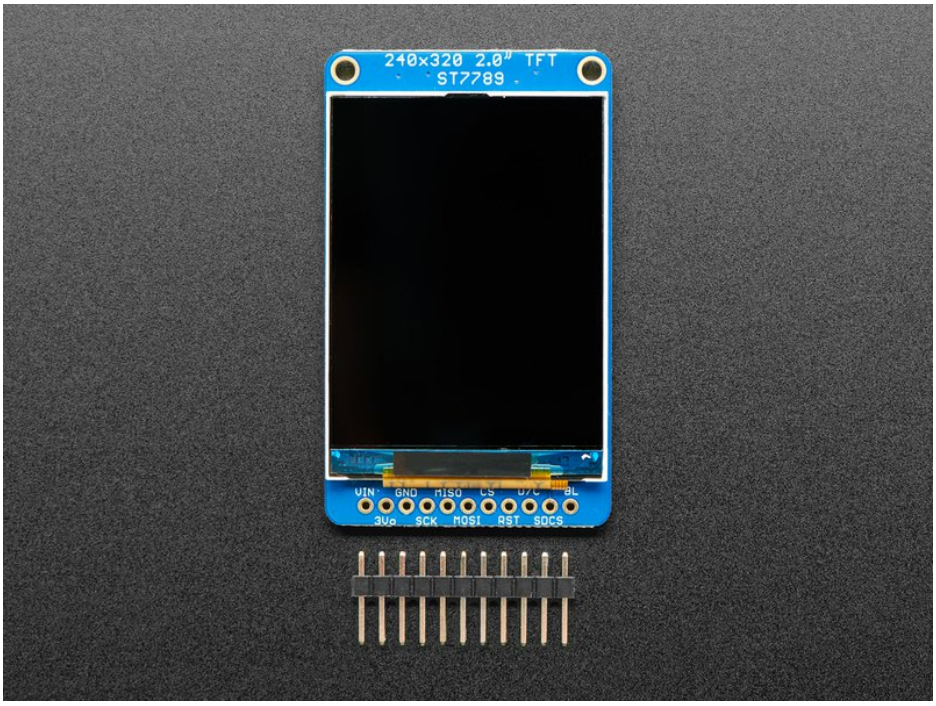


Of course, we wouldn't just leave you with a datasheet and a "good luck!" - we've written a full open source graphics

library that can draw pixels, lines, rectangles, circles, text and bitmaps as well as example code (https://adafru.it/d4d). The code is written for Arduino but can be easily ported to your favorite microcontroller! Wiring is easy, we strongly encourage using the hardware SPI pins of your Arduino as software SPI is noticeably slower when dealing with this size display.



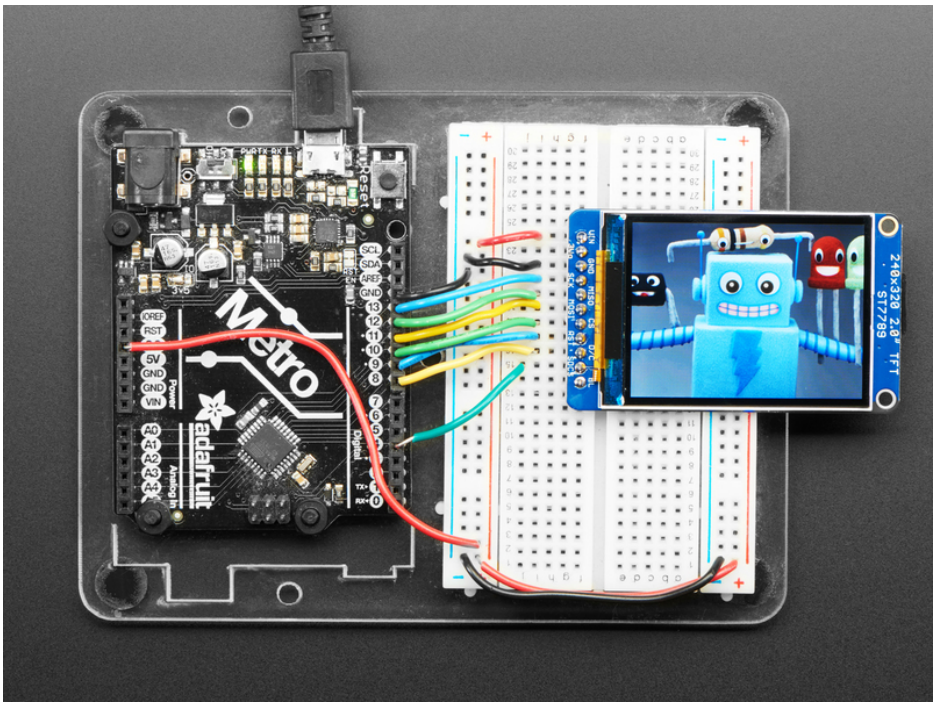
Pinouts



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!
- **3Vo** - 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 Master In Slave 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 Master Out Slave In pin, it is used to send data from the microcontroller to the SD card and/or TFT. Use 3-5V logic level
- **CS** - 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
- **D/C** - 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
- **BL** - 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

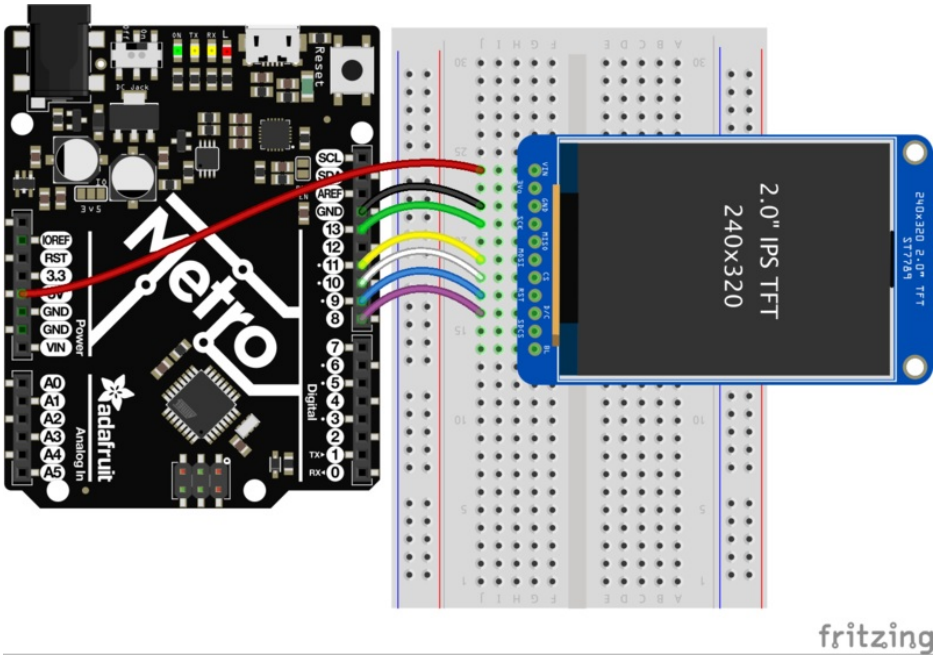


Basic Graphics Test Wiring

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))
- **CS** 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.
- **D/C** 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



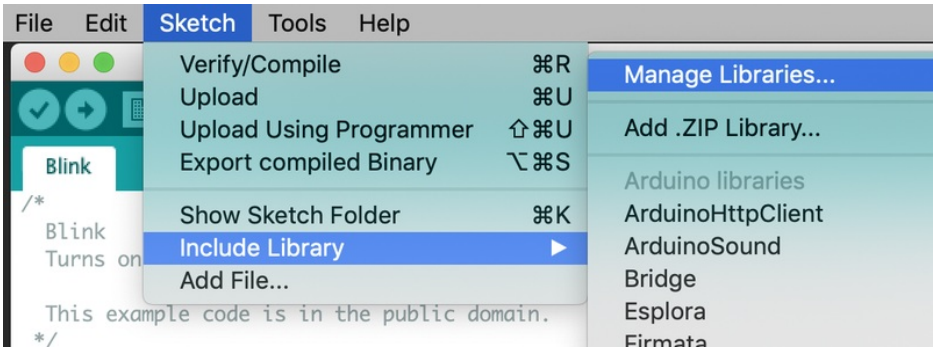
https://adafru.it/FvT

https://adafru.it/FvT

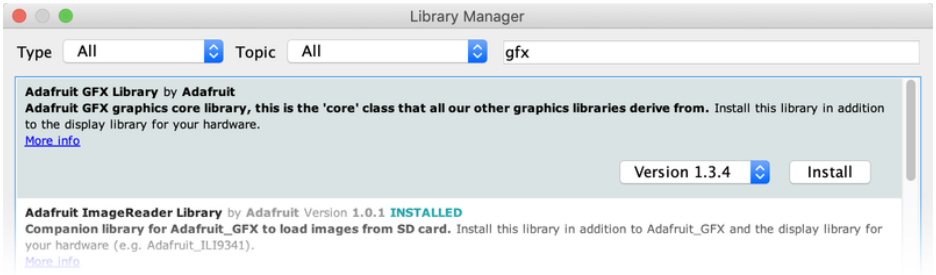
Install Arduino Libraries

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.

Five 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...”

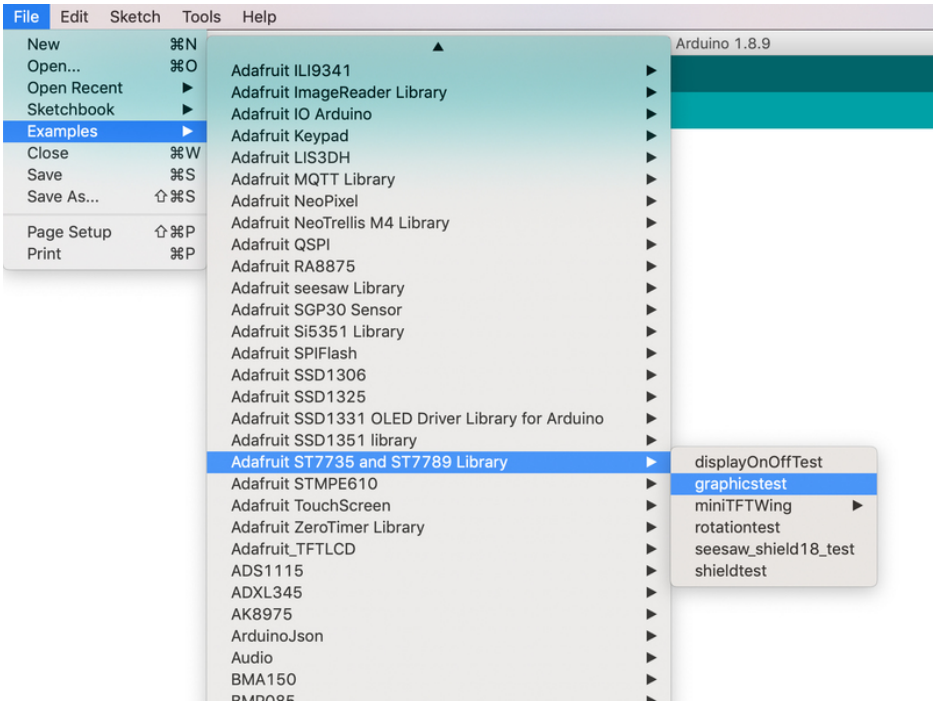


Type “gfx” in the search field to quickly find the first library — **Adafruit_GFX**:



Repeat the search and install steps, looking for the **Adafruit 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 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 2.0" display.

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

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

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

comment out the first line, and uncomment the second, so it looks like:

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

    // For 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" 180x60 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
```

comment out the first line, and uncomment the fifth, 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" 180x60 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
```

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.

#if defined(ADAFRUIT_PYBADGE_M4_EXPRESS) || defined(ADAFRUIT_PYGAMER_M4_EXPRESS)
    // For PyBadge and PyGamer
    Adafruit_ST7735 tft = Adafruit_ST7735(&SPI1, TFT_CS, TFT_DC, TFT_RST);
#else
    // For 1.44" and 1.8" TFT with ST7735 (including HalloWing) use:
    //Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

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

// 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 2.0" IPS TFT display has an auto-reset circuit on it so you probably don't need to use the **RST** pin. You can change

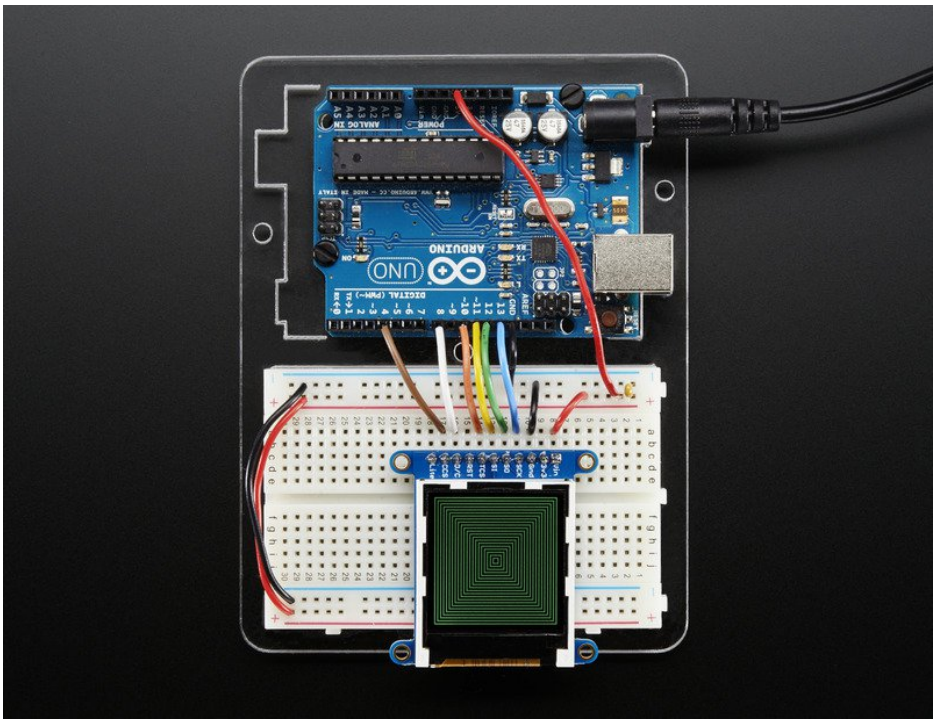
`#define TFT_RST  9`

to

`#define TFT_RST  -1`

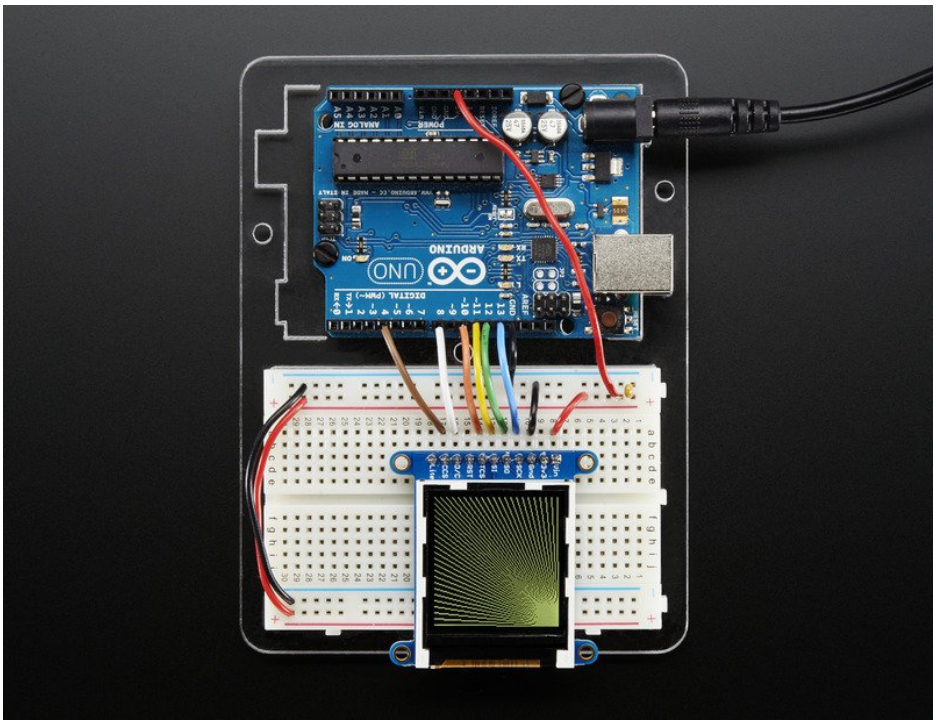
so that pin isn't used either. Or connect it up for manual TFT resetting!

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

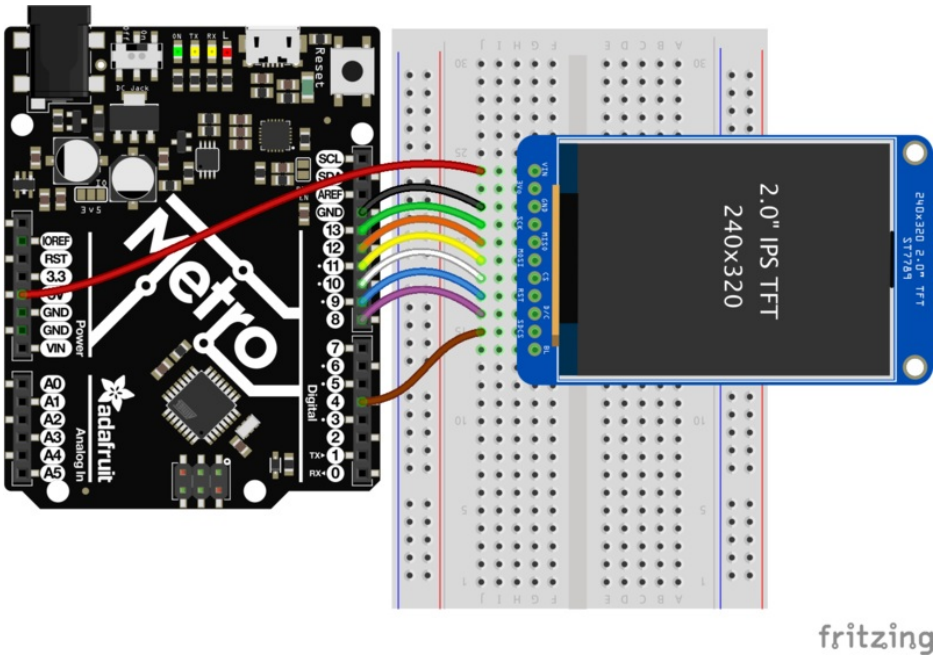
There is a built in microSD card slot into the breakout, and we can use that to load bitmap images! You will need a microSD card formatted **FAT16 or FAT32** (they almost always are by default).

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

Copy **purple.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, 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.

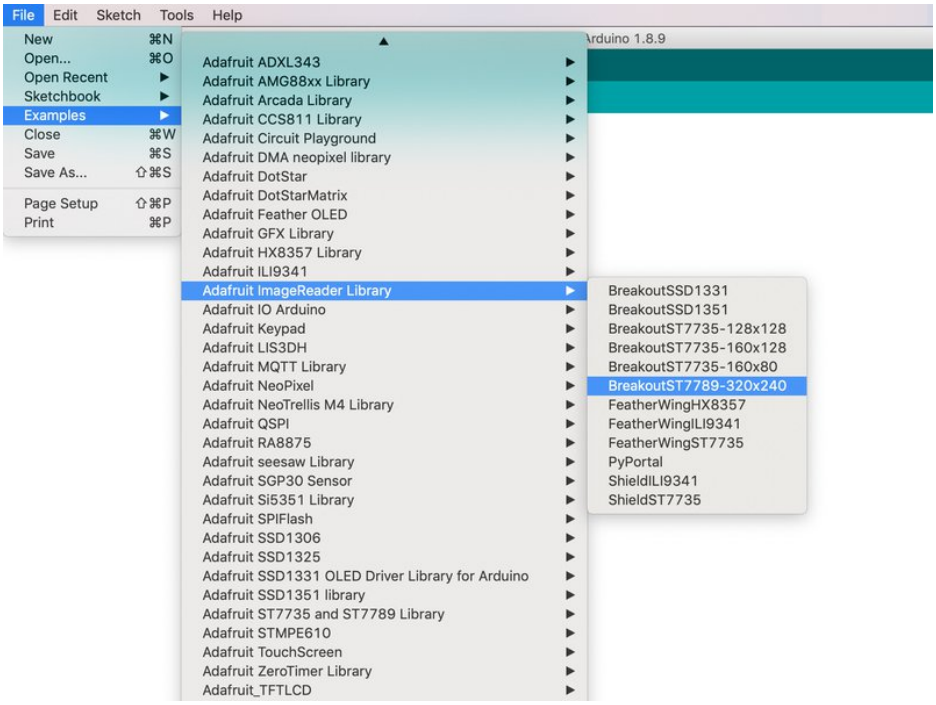


https://adafru.it/FvV

https://adafru.it/FvV

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:



Now upload the example sketch to the Arduino. You should see **purple flowers** 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 320 by 240 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

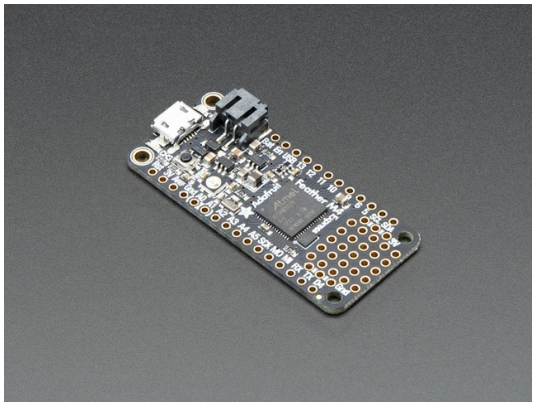
You can draw as many images as you want - don't forget the names must be less than 8 characters long. Just copy the BMP drawing routines below loop() and call

```
bmpDraw(bmpfilename, x, y);
```

For each bitmap. They can be smaller than 320x240 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

$22.95
IN STOCK

ADD TO CART

Preparing the Breakout

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

Required CircuitPython Libraries

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

https://adafru.it/FvR

https://adafru.it/FvR

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.

https://adafru.it/FiA

https://adafru.it/FiA

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

CircuitPython Code Example

```
"""
This test will initialize the display using displayio and draw a solid green
background, a smaller purple rectangle, and some yellow text.
"""
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_st7789 import ST7789

spi = board.SPI()
while not spi.try_lock():
    pass
spi.configure(baudrate=24000000) # Configure SPI for 24MHz
spi.unlock()
tft_cs = board.D5
tft_dc = board.D6

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

display = ST7789(display_bus, width=320, height=240, rotation=90)

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

color_bitmap = displayio.Bitmap(320, 240, 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(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=20, y=20)
splash.append(inner_sprite)

# Draw a label
text_group = displayio.Group(max_size=10, scale=3, x=57, y=120)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
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 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.

For the ST7789, because this is a nice zippy display, we'll change the baud rate to 24MHz. In order to do that, first we enter a loop until we can lock the SPI bus for our exclusive use. After that we set it, and then unlock it again. Next we set the Chip Select and Data/Command pins that will be used.

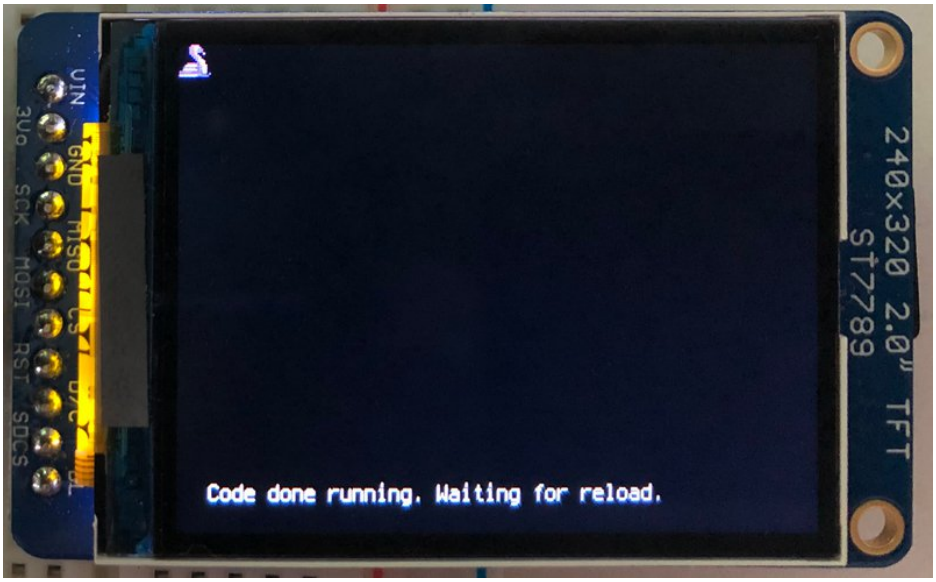
```
spi = board.SPI()
while not spi.try_lock():
    pass
spi.configure(baudrate=24000000) # Configure SPI for 24MHz
spi.unlock()
tft_cs = board.D5
tft_dc = board.D6
```

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

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

Finally, we initialize the driver with a width of 320 and a height of 240. Because we want the display to start in a horizontal orientation, we tell it to start with a rotation of 90 degrees. 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, width=320, height=240, rotation=90)
```



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(320, 240, 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.

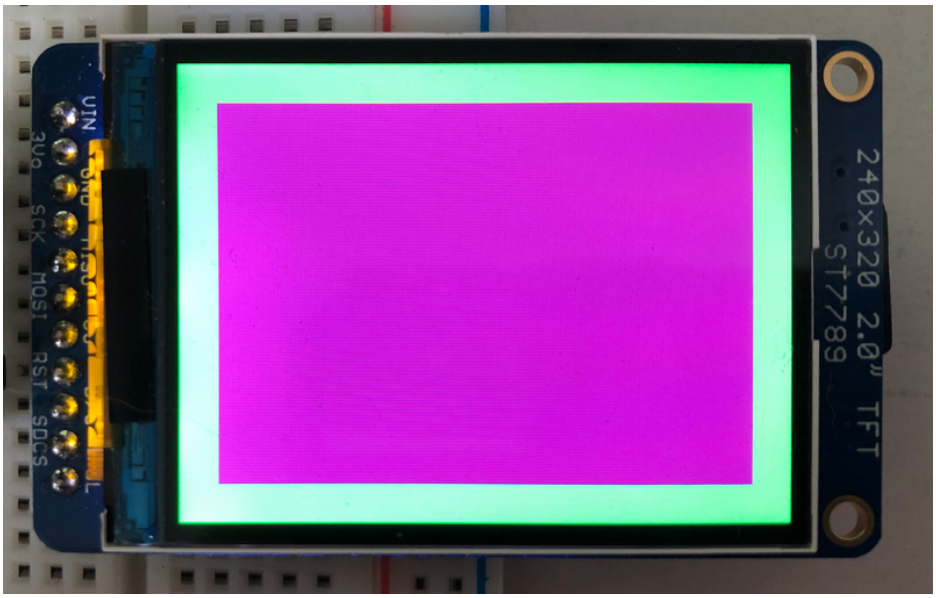
```
bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)
splash.append(bg_sprite)
```

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 20 pixels smaller on each side. The screen is **320x240**, so we'll want to subtract 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(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=20, y=20)
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 three. To scale the label only, we will make use of a subgroup, which we will then add to the main group.

Labels are centered vertically, so we'll place it at 120 for the Y coordinate, and around 57 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_group = displayio.Group(max_size=10, scale=3, x=57, y=120)
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00)
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

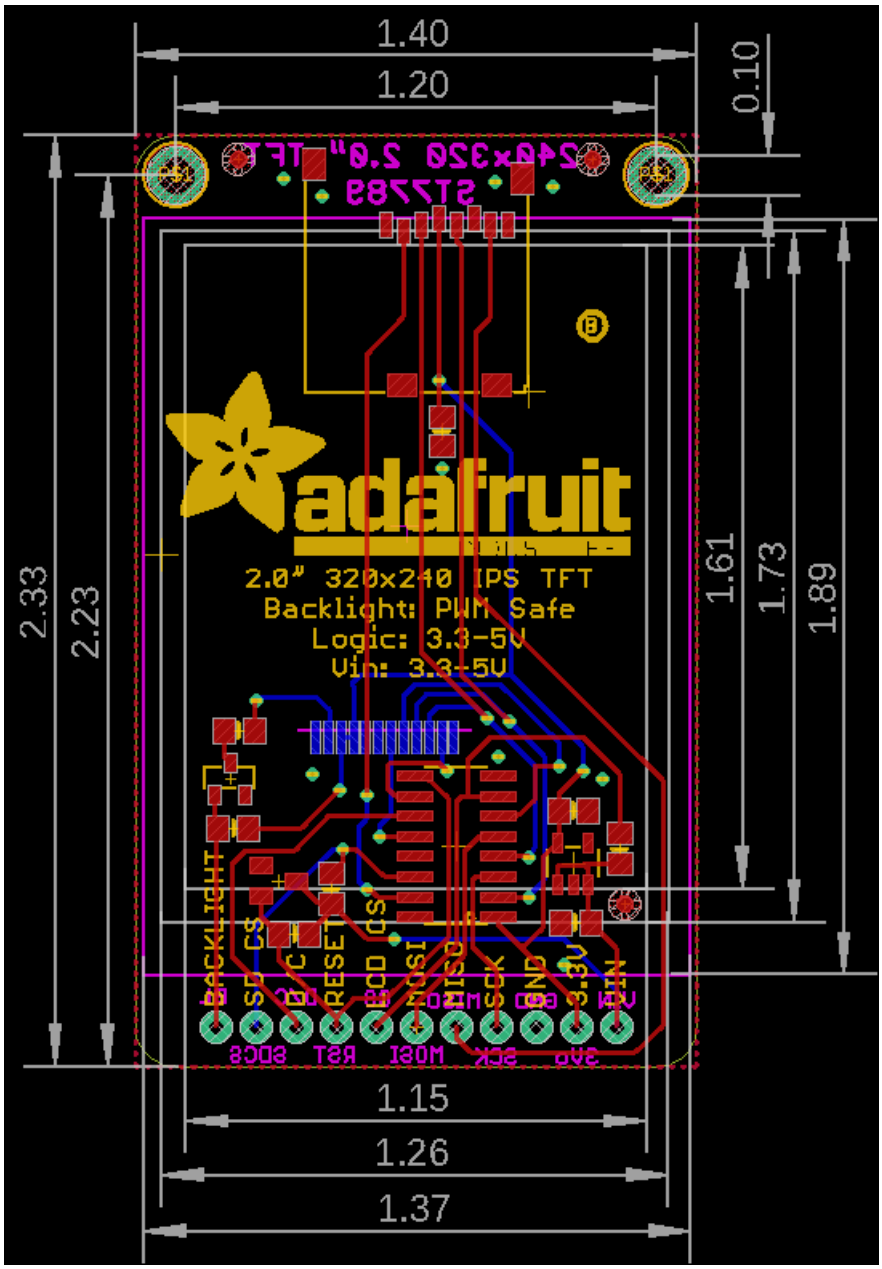
Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafru.it/EGh)

Downloads

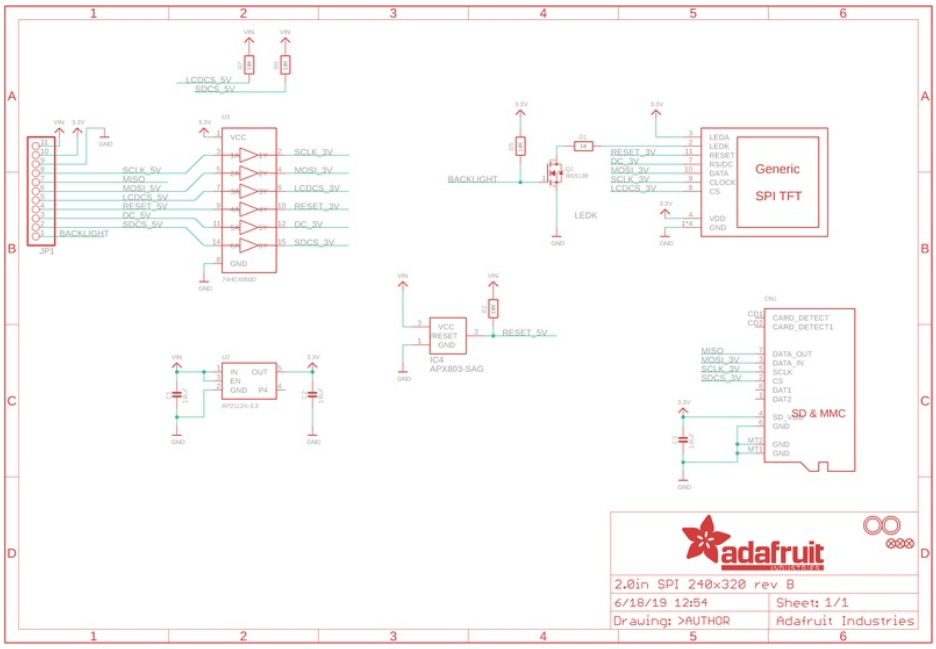
Files

- 2.0" display EagleCAD files on GitHub (https://adafru.it/FvO)
- 2.0" display Fritzing object in the Adafruit Fritzing Library (https://adafru.it/FvP)

Fab Print



Schematic


Generic
SPI TFT
SD & MMC
adafruit
2.0in SPI 240x320 rev B
6/18/19 12:54
Sheet: 1/1
Drawing: >AUTHOR
Adafruit Industries