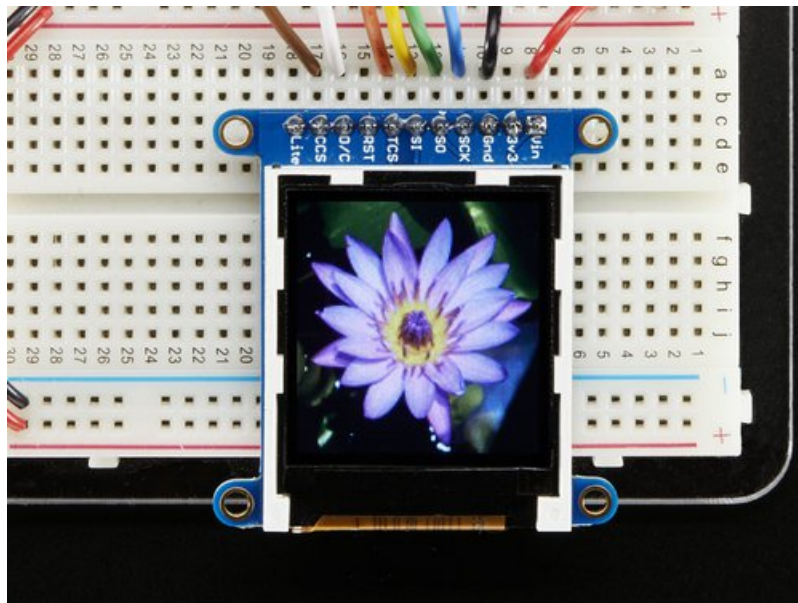


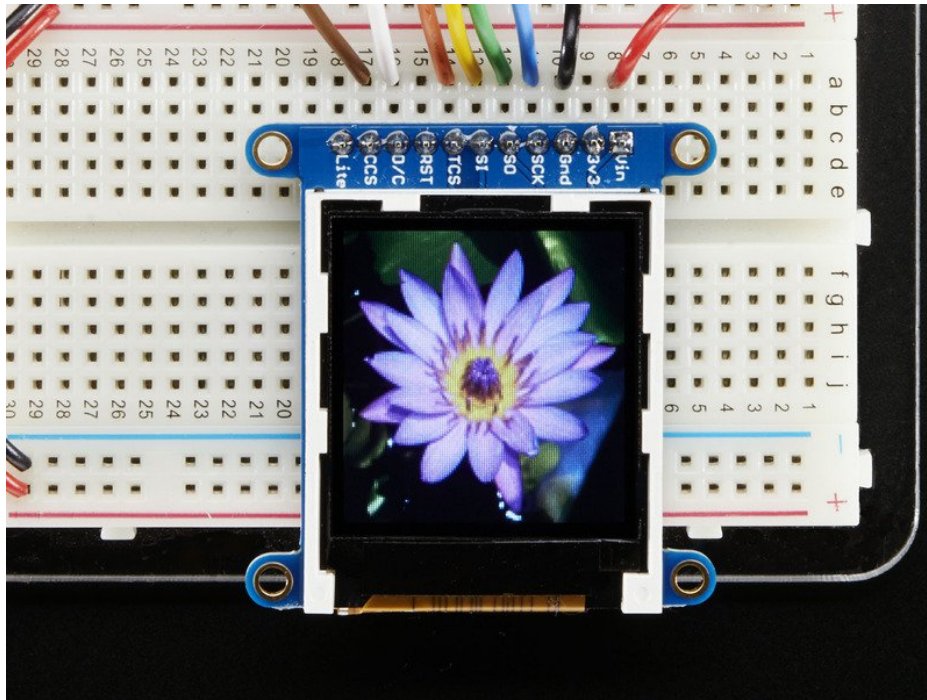
Adafruit 1.44" Color TFT with Micro SD Socket

Created by lady_ada

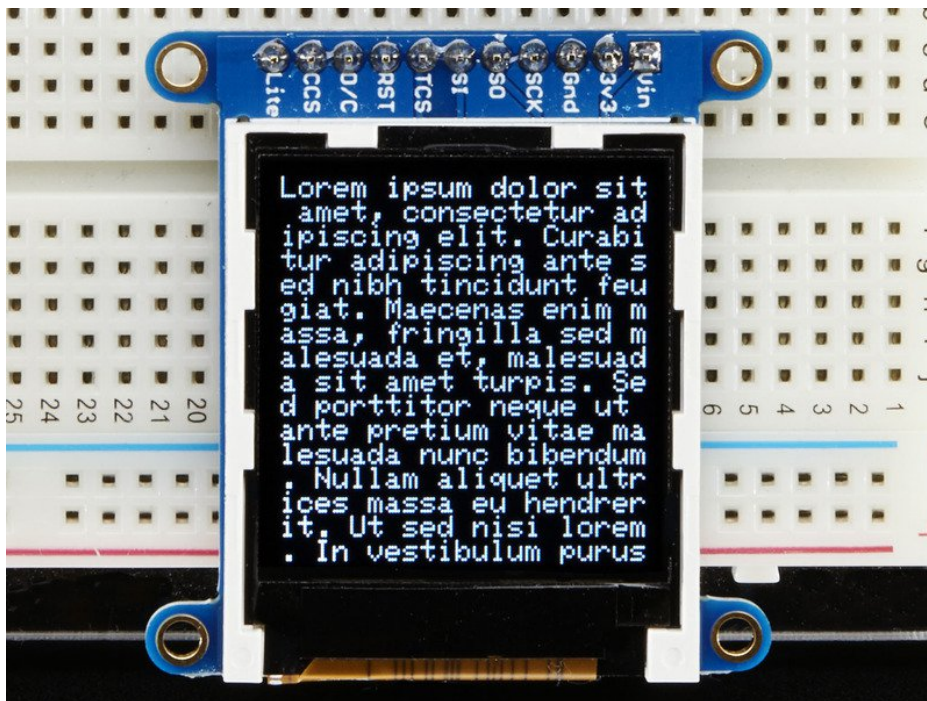


Last updated on 2019-08-15 07:59:48 PM UTC

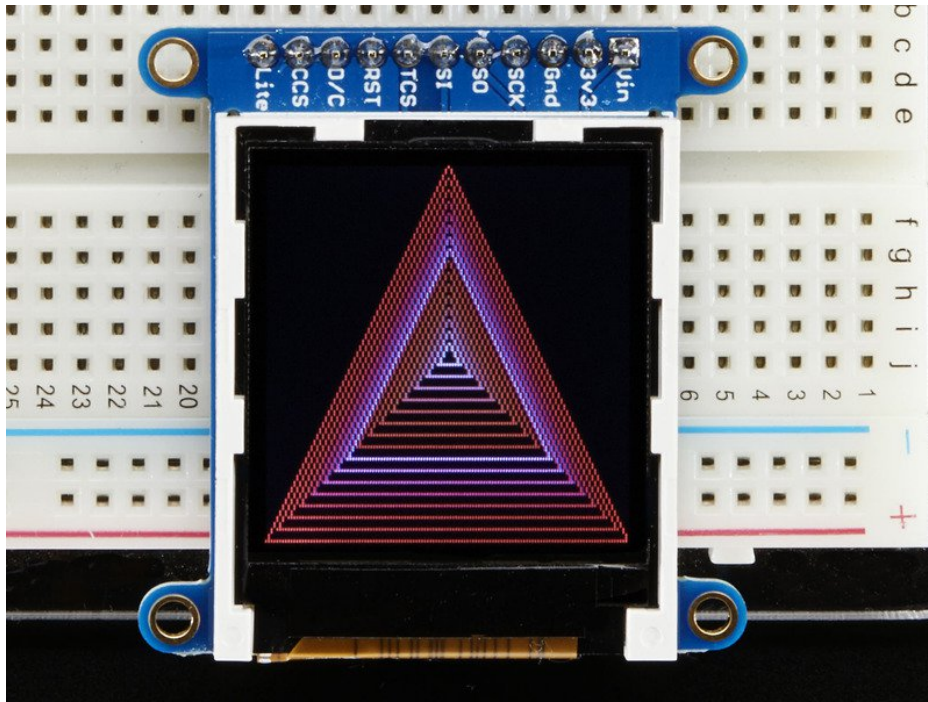
Overview



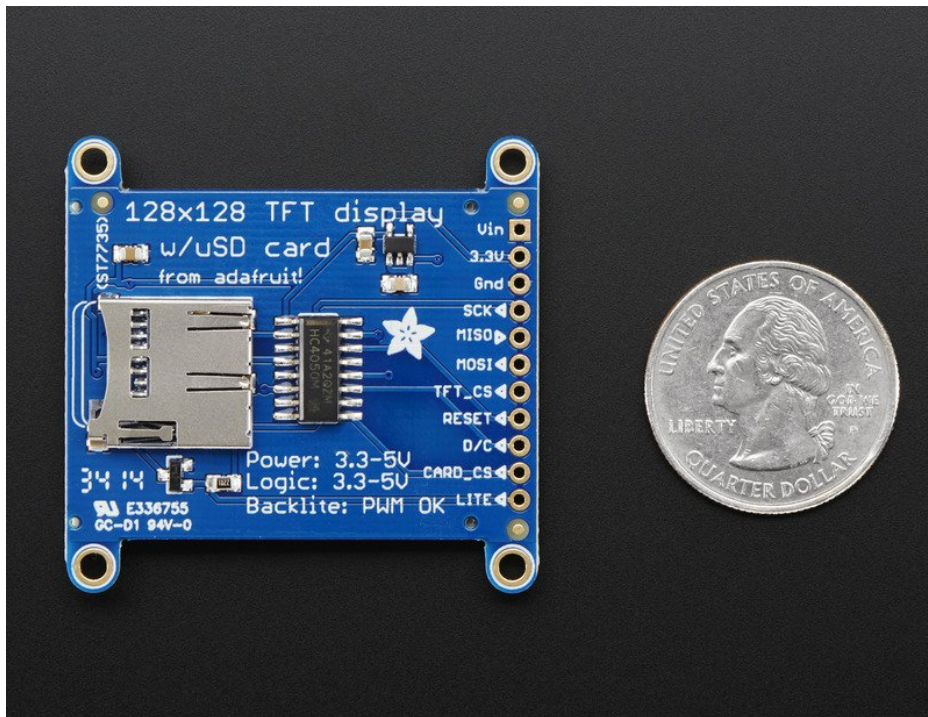
This lovely little display breakout is the best way to add a small, colorful and bright display to any project. Since the display uses 4-wire SPI to communicate and has its own pixel-addressable frame buffer, it can be used with every kind of microcontroller. Even a very small one with low memory and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



The 1.44" display has 128x128 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! The TFT driver (ST7735R) can display full 16-bit color using our library code.

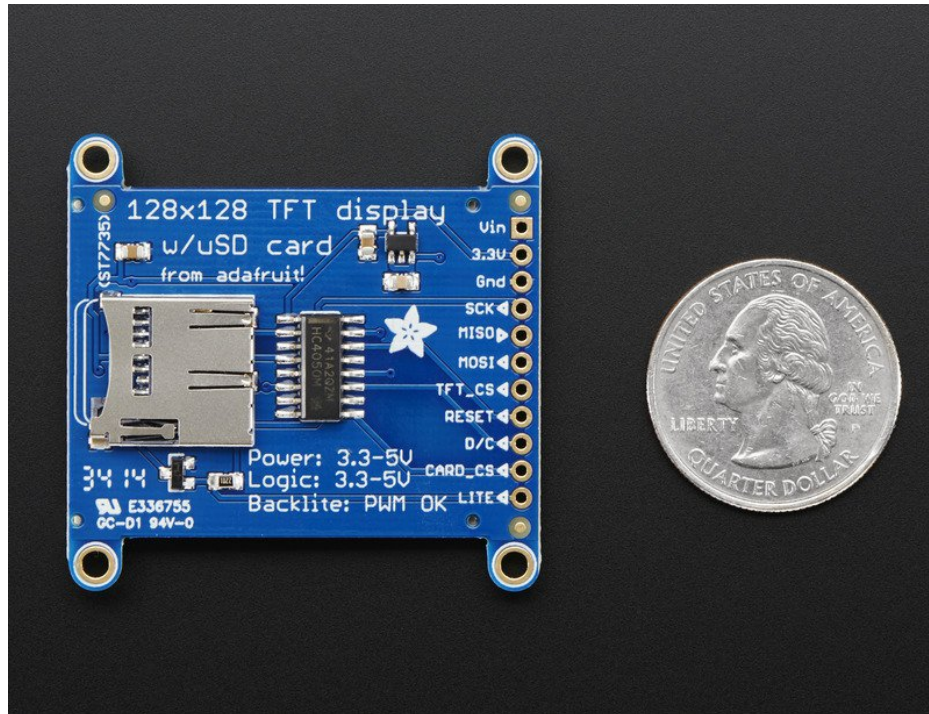


The breakout has the TFT display soldered on (it uses a delicate flex-circuit connector) as well as a ultra-low-dropout 3.3V regulator and a 3/5V level shifter so you can use it with 3.3V or 5V power and logic. We also had a little space so we placed a microSD card holder so you can easily load full color bitmaps from a FAT16/FAT32 formatted microSD card. The microSD card is not included, [but you can pick one up here \(http://adafru.it/102\)](http://adafru.it/102).



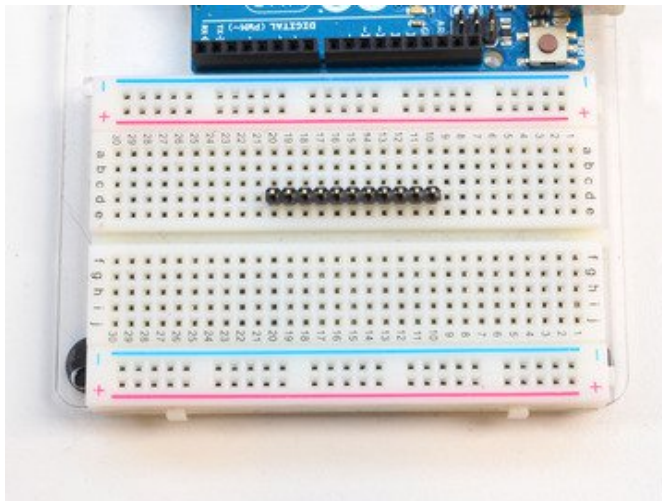
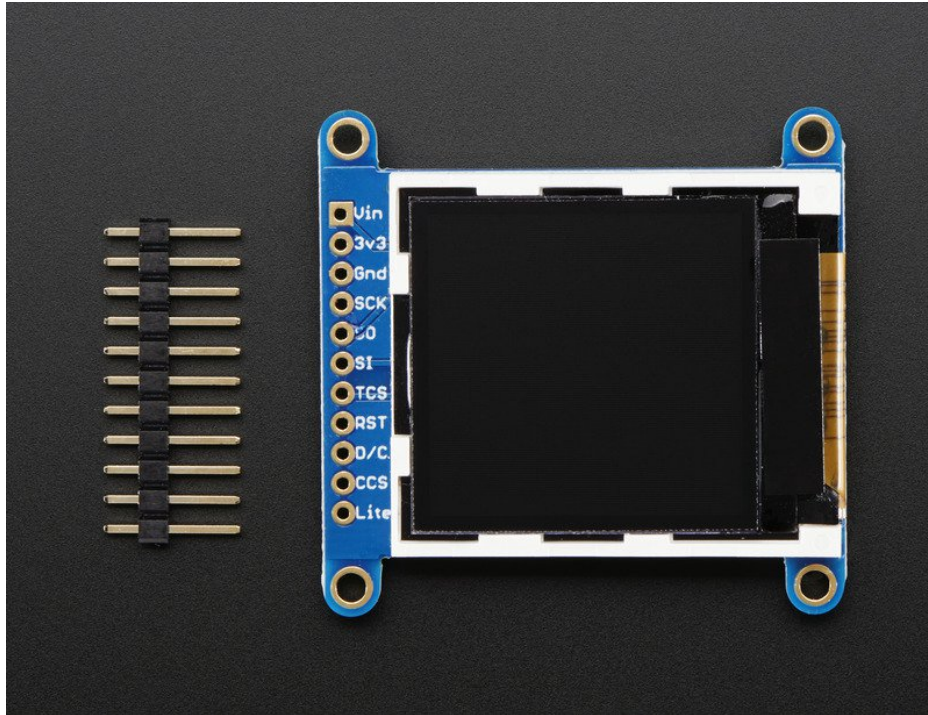
Of course, we wouldn't just leave you with a datasheet and a "good luck!" - [we've written a full open source graphics](#)

library that can draw pixels, lines, rectangles, circles, text and bitmaps as well as example code and a wiring tutorial (<https://adafru.it/ckK>). The code is written for Arduino but can be easily ported to your favorite microcontroller!

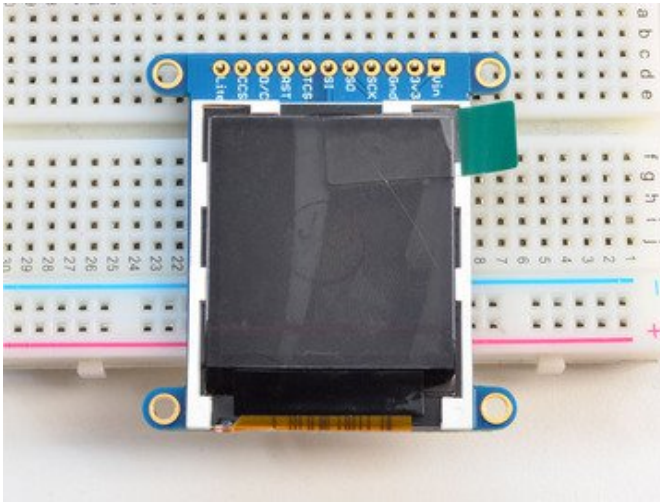


This color display uses SPI to receive image data. That means you need at least 4 pins - clock, data in, tft cs and d/c. If you'd like to have SD card usage too, add another 2 pins - data out and card cs. However, there's a couple other pins you may want to use, lets go thru them all!

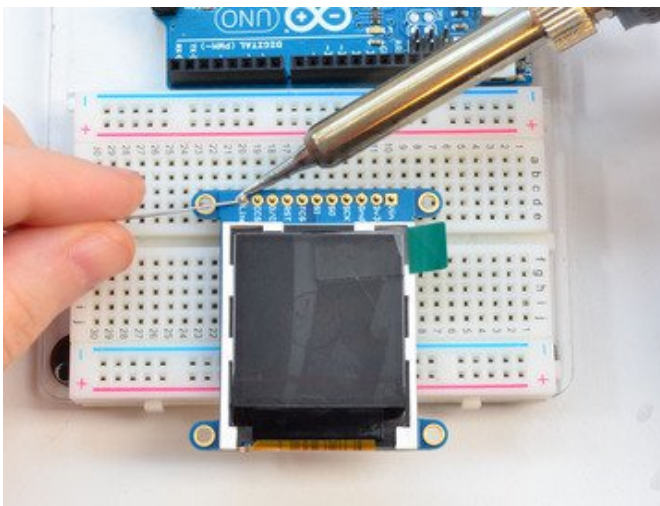
- **3-5V / Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3.3Vout** - this is the 3.3V output from the onboard regulator
- **GND** - this is the power and signal ground pin
- **CLK** - this is the SPI clock input pin
- **MISO** - this is the SPI Master In Slave Out pin, its used for the SD card. It isn't used for the TFT display which is write-only
- **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
- **TFT_CS** - this is the TFT SPI chip select pin
- **RST** - this is the TFT reset pin. Connect to ground to reset the TFT! Its 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.
- **D/C** - this is the TFT SPI data or command selector pin
- **Card CS / CCS** - this is the SD card chip select, used if you want to read from the SD card.
- **Lite** - 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



Prepare the header strip:
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

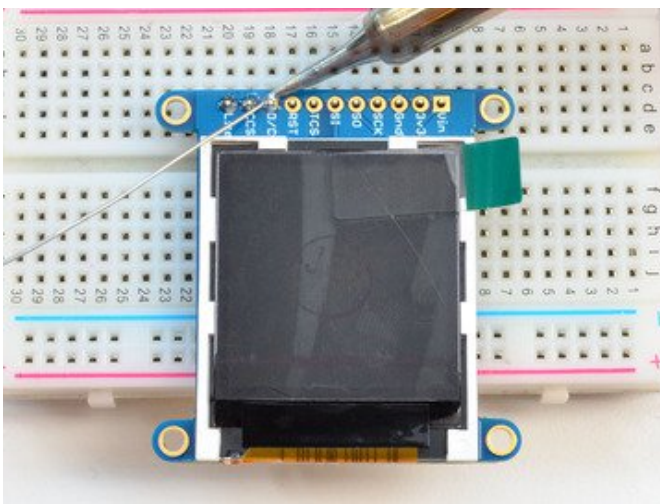


Add the breakout board:
Place the breakout board over the pins so that the short pins poke through the breakout pads



And Solder!
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our [Guide to Excellent Soldering](https://adafruit.it/aTk) (<https://adafruit.it/aTk>)).

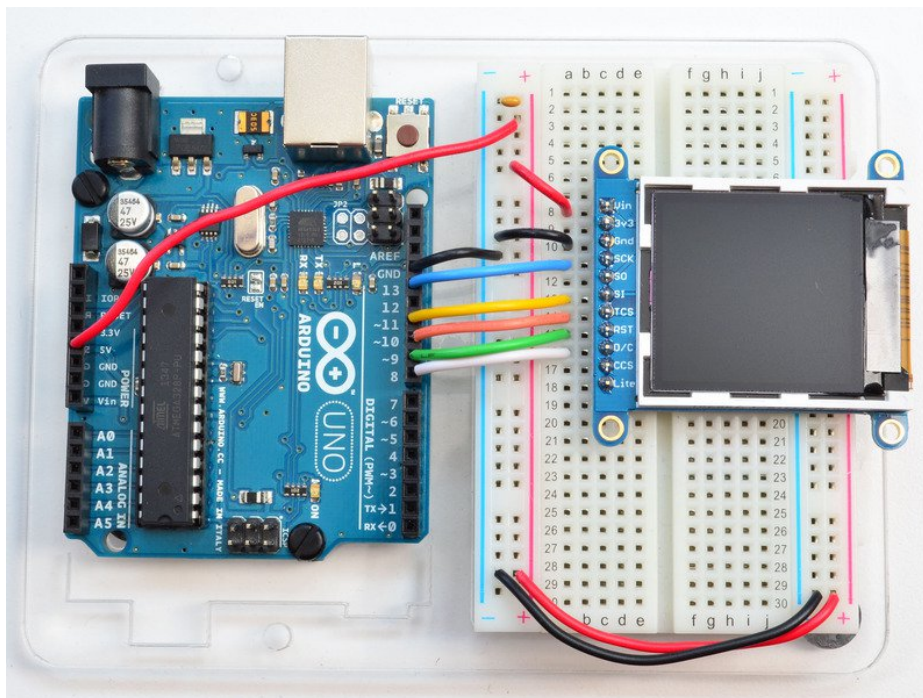


Wiring & Test

Wiring

Wiring up the display in SPI mode is pretty easy as there's 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

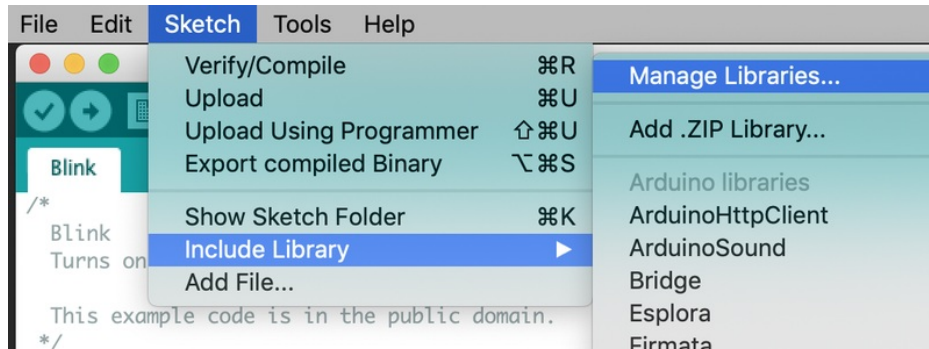
- **3-5V Vin** connects to the Arduino **5V** pin
- **GND** connects to Arduino ground
- **CLK** connects to SPI clock. On Arduino Uno/Duemilanove/328-based, that's **Digital 13**. On Mega's, its **Digital 52** and on Leonardo/Due its **ICSP-3** (See [SPI Connections for more details \(https://adafru.it/d5h\)](https://adafru.it/d5h))
- **MOSI** connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, that's **Digital 11**. On Mega's, its **Digital 51** and on Leonardo/Due its **ICSP-4** (See [SPI Connections for more details \(https://adafru.it/d5h\)](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 TFT 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.



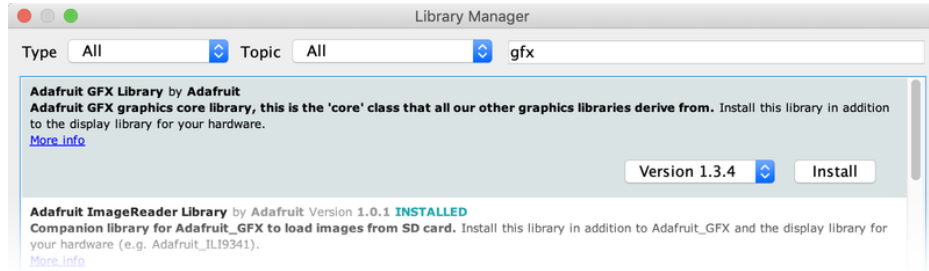
Install Arduino Libraries

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...”

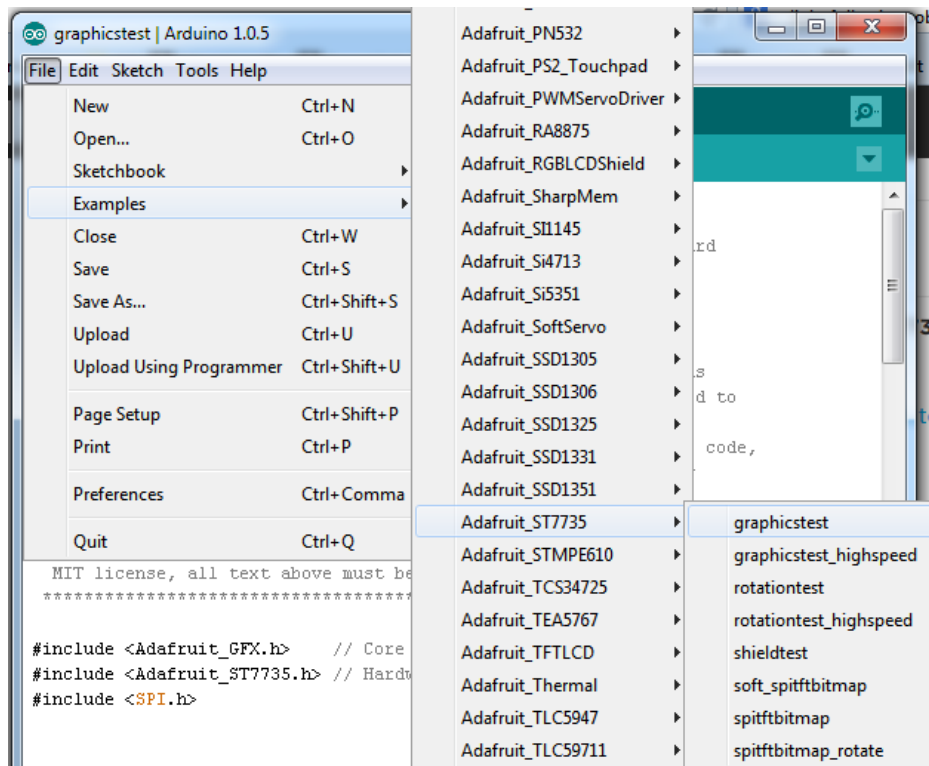


Type “gfx” in the search field to quickly find the first library — **Adafruit_GFX**:



Repeat the search and install steps, looking for the **Adafruit_ZeroDMA** and **Adafruit_ST7735** libraries.

After restarting the Arduino software, you should see a new **example** folder called **Adafruit_ST7735** and inside, an example called **graphicstest**.



In the **graphicstest** source code, look for the lines as follows:

```
// Use this initializer if you're using a 1.8" TFT
tft.initR(INITR_BLACKTAB); // initialize a ST7735S chip, black tab

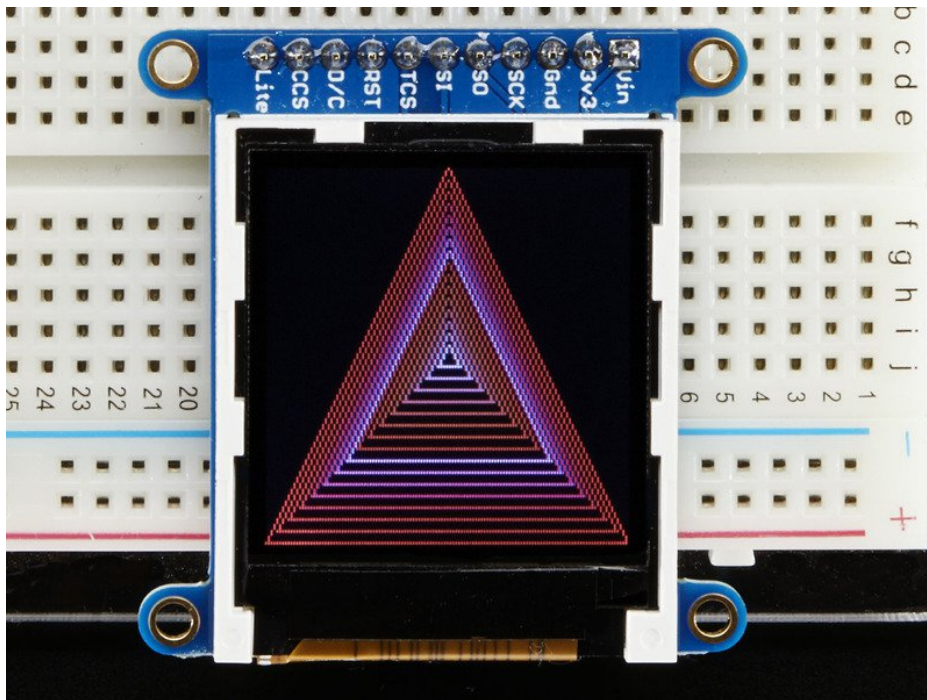
// Use this initializer (uncomment) if you're using a 1.44" TFT
//tft.initR(INITR_144GREENTAB); // initialize a ST7735S chip, black tab
```

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

```
// Use this initializer if you're using a 1.8" TFT
//tft.initR(INITR_BLACKTAB); // initialize a ST7735S chip, black tab

// Use this initializer (uncomment) if you're using a 1.44" TFT
tft.initR(INITR_144GREENTAB); // initialize a ST7735S chip, black tab
```

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 cant 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): must use the hardware SPI pins
// (for UNO thats sclk = 13 and sid = 11) and pin 10 must be
// an output. This is much faster - also required if you want
// to use the microSD card (see the image drawing example)
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);

// Option 2: use any pins but a little slower!
#define TFT_SCLK 13 // set these to be whatever pins you like!
#define TFT_MOSI 11 // set these to be whatever pins you like!
//Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK, TFT_RST);
```

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

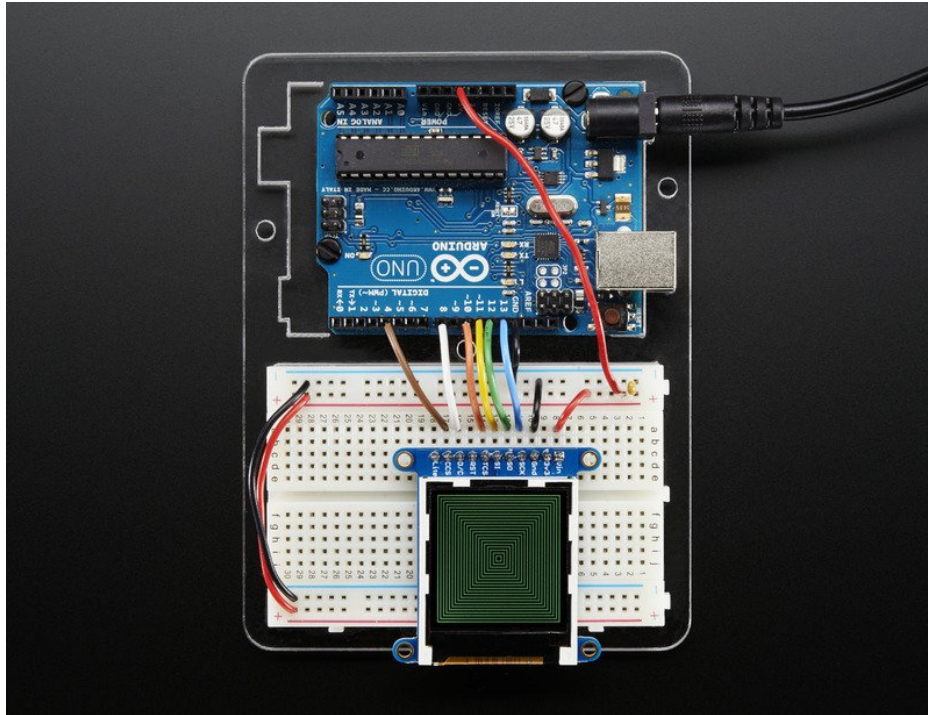
You can also save a pin by setting

```
#define TFT_RST 9
```

to

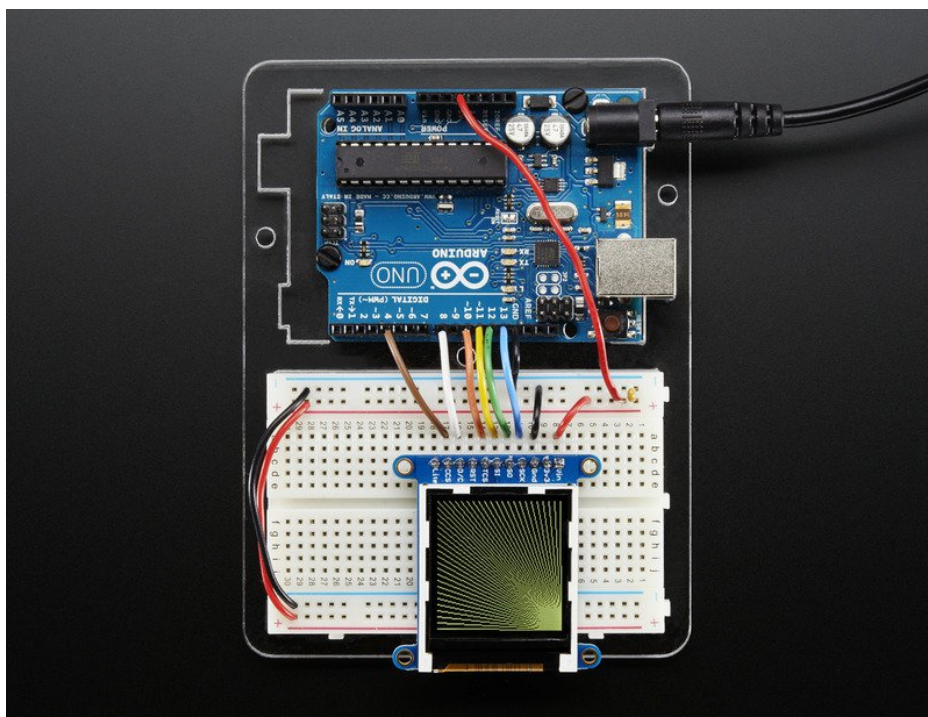
```
#define TFT_RST 0
```

and connecting the RST line to the Arduino Reset pin. That way the Arduino will auto-reset the TFT as well.



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.

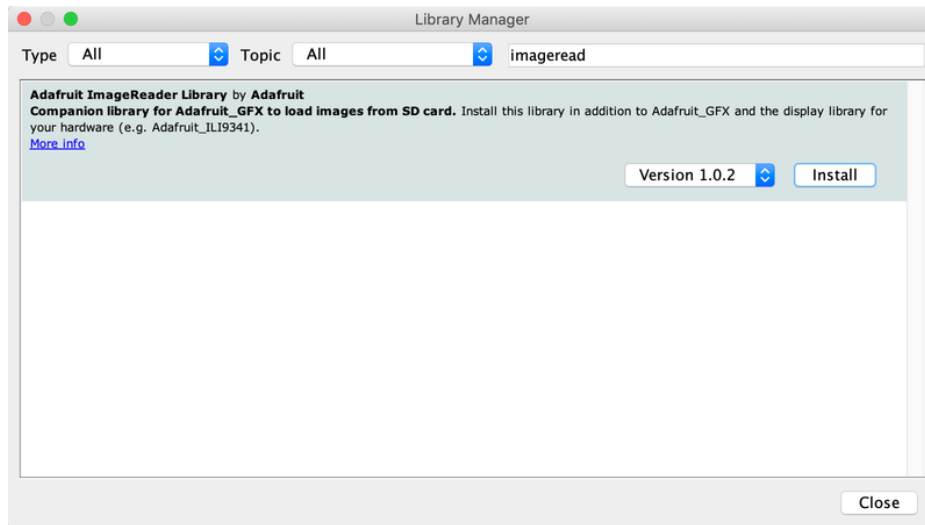


Check out our detailed tutorial here <http://learn.adafruit.com/adafruit-gfx-graphics-library> (<https://adafru.it/aPx>) It covers the latest and greatest of the GFX library!

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



Let's start by downloading this image of a lily:

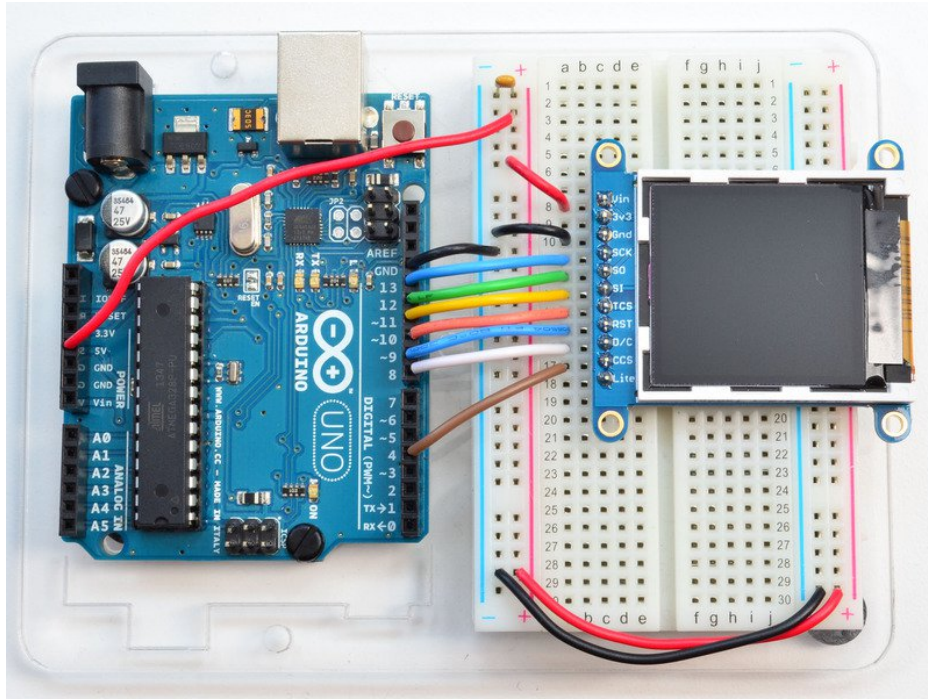


Copy `lily128.bmp` into the base directory of a microSD card and insert it into the microSD socket in the breakout.

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

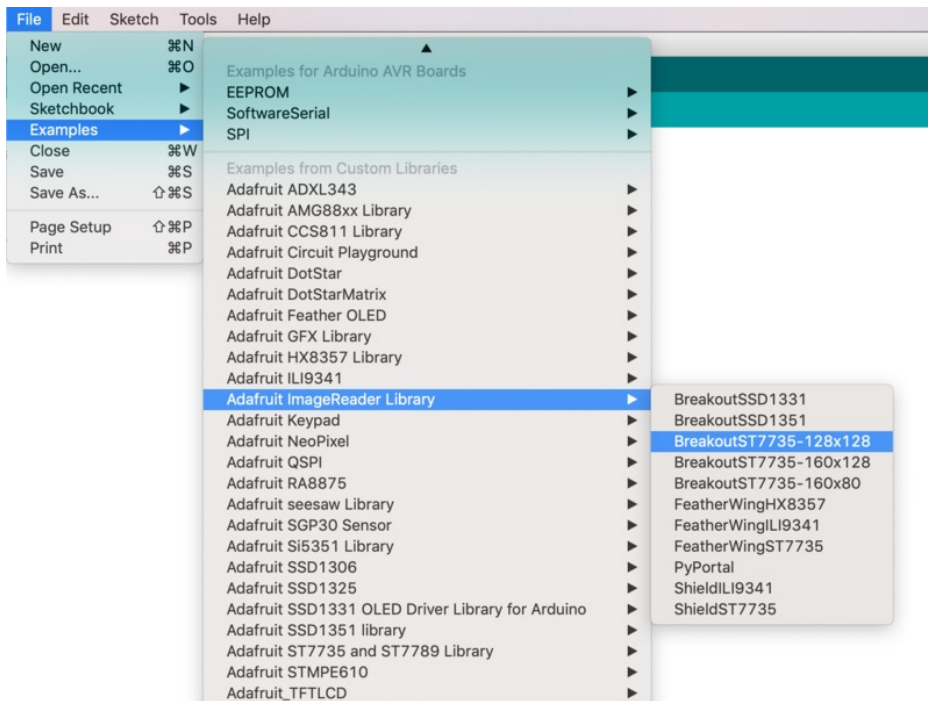
You'll need to connect up the MISO pin to the SPI MISO line on your microcontroller. On Arduino Uno/Duemilanove/328-based, that's **Digital 12**. On Mega's, its **Digital 50** and on Leonardo/Due its **ICSP-1** ([See SPI Connections for more details \(https://adafru.it/d5h\)](https://adafru.it/d5h))

Also, **CCS** pin to **Digital 4** on your Arduino as well. You can change this pin later, but stick with this for now.



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**→**BreakoutST7735 - 128x128** example

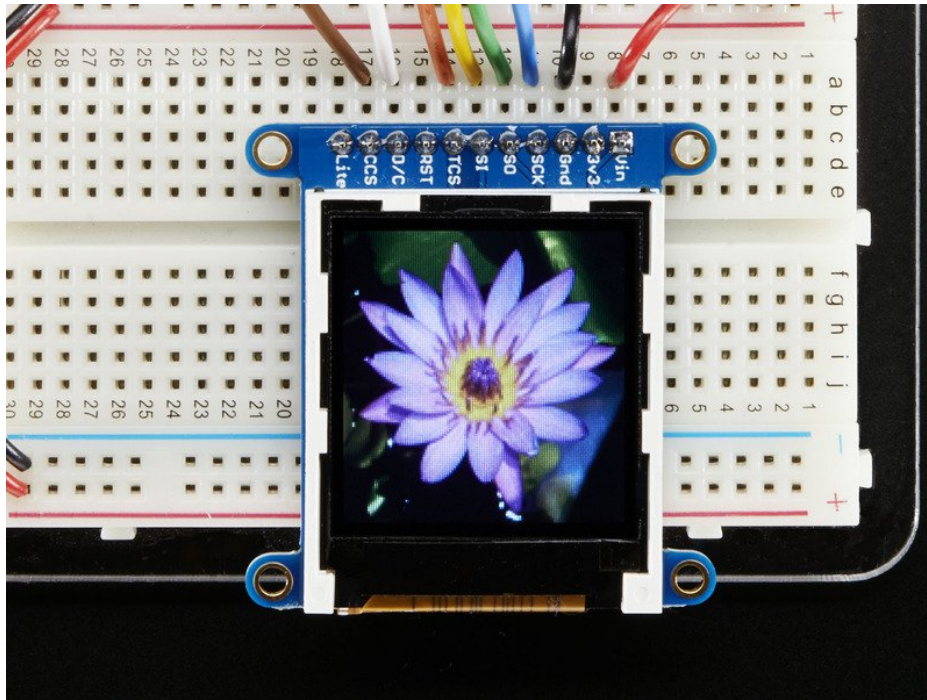


In the example, find the following section of code:


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

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

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



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

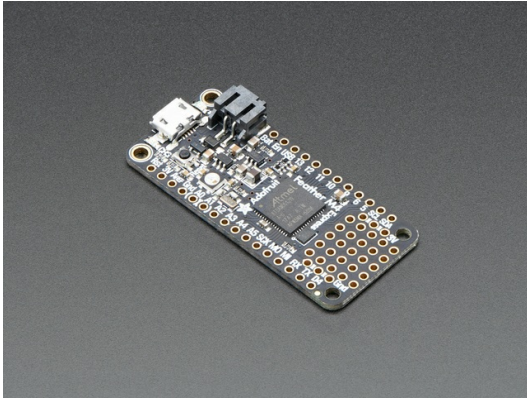
You can draw as many images as you want - dont forget the names must be less than 8 characters long. Just copy the BMP drawing routines below `loop()` and call

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

For each bitmap. They can be smaller than 128x128 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 \(https://adafru.it/EEem\)](https://adafru.it/EEem) guide.



[Adafruit Feather M4 Express - Featuring ATSAMD51](#)

\$22.95
IN STOCK

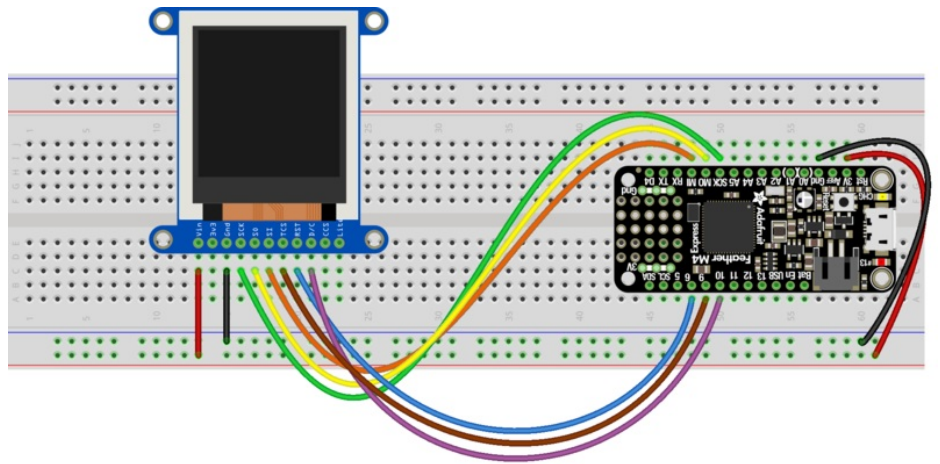
ADD TO CART

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 \(https://adafru.it/drl\)](https://adafru.it/drl). After that the breakout should be ready to go.

Wiring the Breakout to the Feather

- **3-5V VIN** connects to the Feather **3V** pin
- **GND** connects to Feather ground
- **SCK** connects to SPI clock. On the Feather that's **SCK**.
- **SO** connects to SPI MISO. On the Feather that's **MI**
- **SI** connects to SPI MOSI. On the Feather that's **MO**
- **TCS** connects to our SPI Chip Select pin. We'll be using **Digital 9** but you can later change this to any pin
- **D/C** connects to our SPI data/command select pin. We'll be using **Digital 10** but you can later change this pin too.
- **RST** connects to our reset pin. We'll be using **Digital 6** but you can later change this pin too.



fritzing

<https://adafru.it/FyY>

<https://adafru.it/FyY>

Required CircuitPython Libraries

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

<https://adafru.it/EGk>

<https://adafru.it/EGk>

First, make sure you are running the [latest version of Adafruit CircuitPython](https://adafru.it/Amd) (<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) (<https://adafru.it/zdx>). Our introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<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_st7735r`

Before continuing make sure your board's lib folder or root filesystem has the `adafruit_st7735r` 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

Temporarily unable to load content:

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_st7735r` driver.

```
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_st7735r import ST7735R
```

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

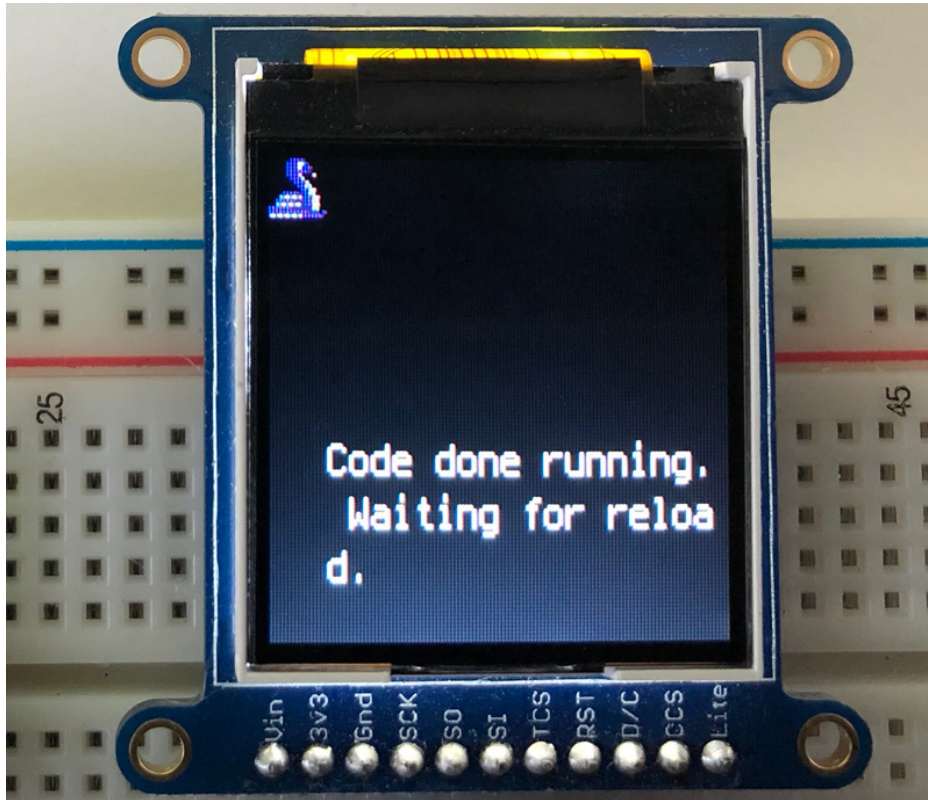
```
spi = board.SPI()
tft_cs = board.D5
tft_dc = board.D6
```

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

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

Finally, we initialize the driver with a width of 128 and a height of 128. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update. This display has a couple of empty columns and an unused row, so we pass it the `colstart` and `rowstart` parameters.

```
display = ST7735R(display_bus, width=128, height=128, colstart=2, rowstart=1)
```



Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

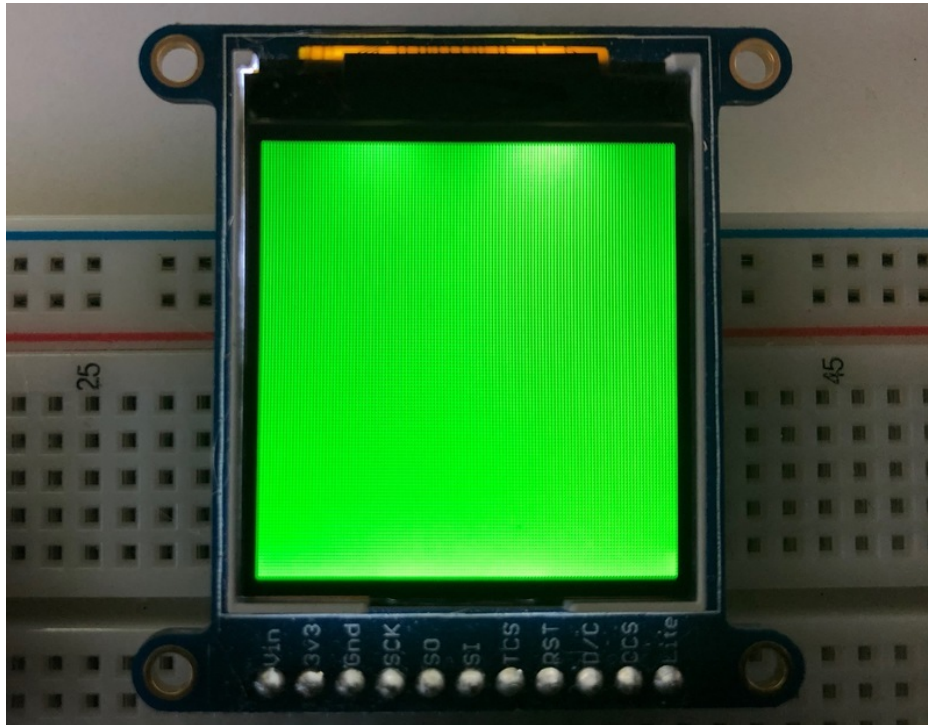
```
splash = displayio.Group(max_size=10)
display.show(splash)
```

Next we create a Bitmap which is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. The Bitmaps can currently handle up to 256 different colors. We create a Palette with one color and set that color to 0x00FF00 which happens to be green. Colors are Hexadecimal values in the format of RRGGBB. Even though the Bitmaps can only handle 256 colors at a time, you get to define what those 256 different colors are.

```
color_bitmap = displayio.Bitmap(128, 128, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

With all those pieces in place, we create a TileGrid by passing the bitmap and palette and draw it at (0, 0) which represents the display's upper left.

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

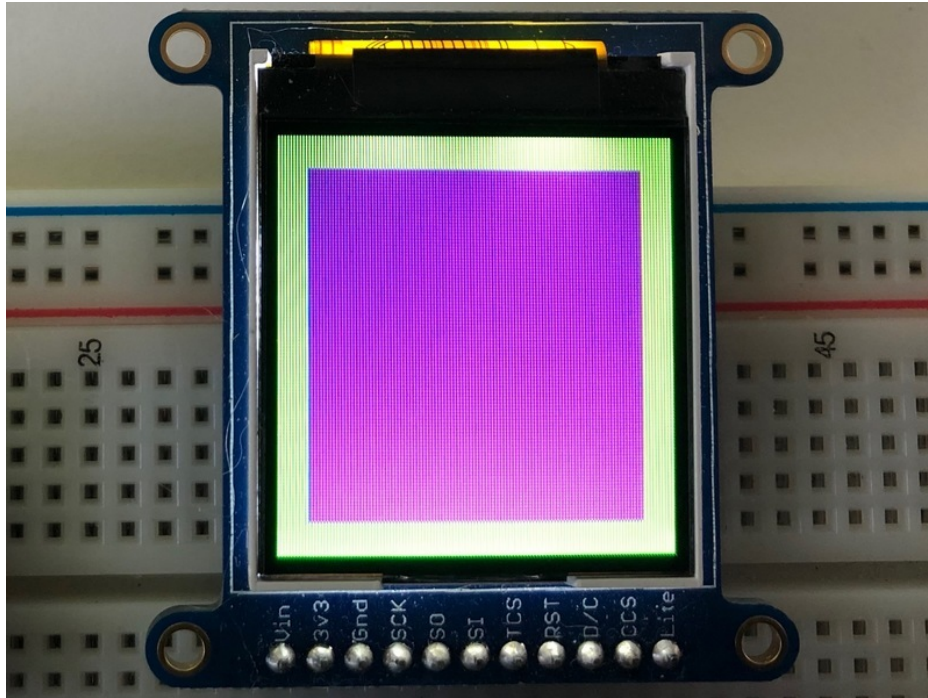


Next we will create a smaller purple square. The easiest way to do this is to create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case, we will create a bitmap that is 10 pixels smaller on each side. The screen is **128x128**,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

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

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

Since we are adding this after the first square, it's automatically drawn on top. Here's what it looks like now.



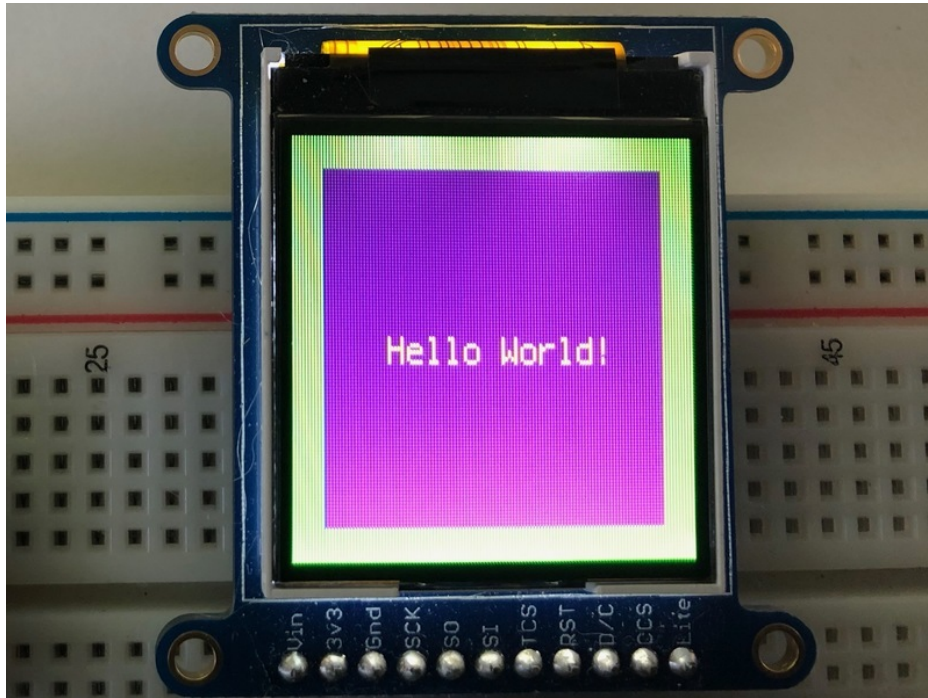
Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font. In this example, we won't be doing any scaling because of the small resolution compared to some of the other displays, so we'll add the label directly the main group. If we were scaling, we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

Labels are centered vertically, so we'll place it at 64 for the Y coordinate, and around 30 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some yellow text, so we'll pass it a value of `0xFFFF00`.

```
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=30, y=64)
splash.append(text_area)
```

Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

```
while True:
    pass
```



Where to go from here

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

Troubleshooting

- Display does not work on initial power but does work after a reset.

The display driver circuit needs a small amount of time to be ready after initial power. If your code tries to write to the display too soon, it may not be ready. It will work on reset since that typically does not cycle power. If you are having this issue, try adding a small amount of delay before trying to write to the display.

In Arduino, use `delay()` to add a few milliseconds before calling `tft.begin()`. Adjust the amount of delay as needed to see how little you can get away with for your specific setup.

Downloads

Files & Datasheets

- [ST7735R display driver datasheet \(https://adafru.it/aP9\)](https://adafru.it/aP9)
- [Raw 1.44" TFT Datasheet \(https://adafru.it/dYA\)](https://adafru.it/dYA)
- [Fritzing object in Adafruit Fritzing library \(https://adafru.it/aP3\)](https://adafru.it/aP3)
- [EagleCAD PCB files in GitHub \(https://adafru.it/pAu\)](https://adafru.it/pAu)

Schematics & PCB Print

