1.8" TFT Display Breakout and Shield
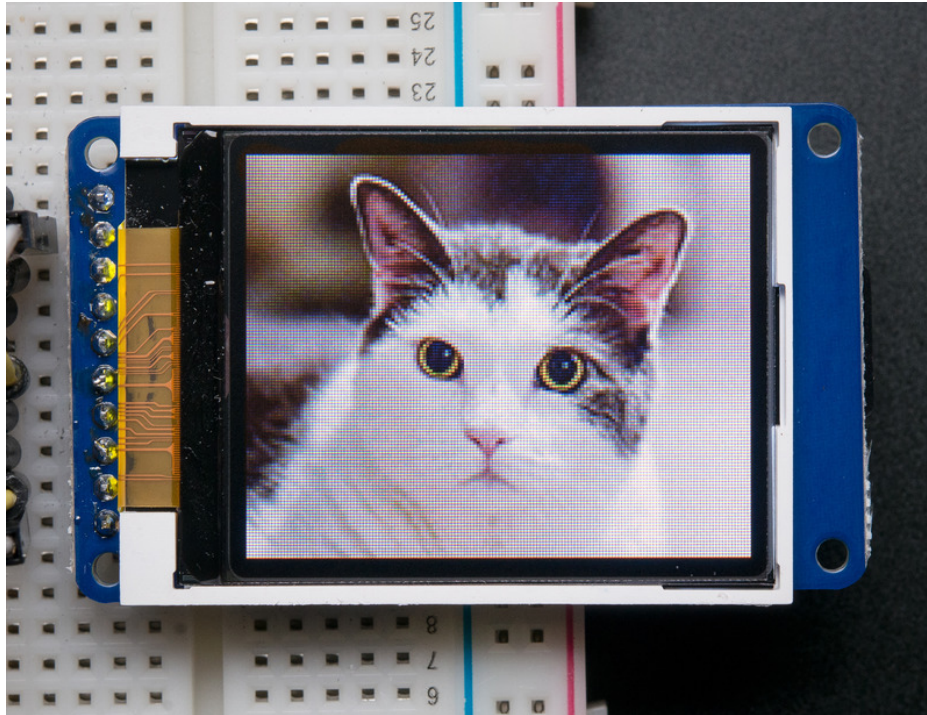Created by lady ada

Last updated on 2020-06-25 12:20:14 PM EDT

This tutorial is for our 1.8" diagonal TFT display. It comes packaged as a breakout or as an Arduino shield. Both styles have a microSD interface for storing files and images. These are both great ways 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 requires little memory and only a few pins. This makes it ideal for use with small microcontrollers.

The shield version plugs directly into an Arduino with no wiring required. The breakout version can be used with every kind of microcontroller.

The 1.8" display has 128x160 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 18-bit color (262,144 shades!). And the LCD will always come with the same driver chip so there's no worries that your code will not work from one to the other.

Both boards have the TFT 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. These also include a microSD card holder so you can easily load full color bitmaps from a FAT16/FAT32 formatted microSD card. And on the Shield version, we've added a nifty 5-way joystick navigation switch!

You can pick up one of these displays in the Adafruit shop!
1.8" 18-bit color TFT breakout (http://adafru.it/358)
1.8" 18-bit Color TFT Shield (http://adafru.it/802)

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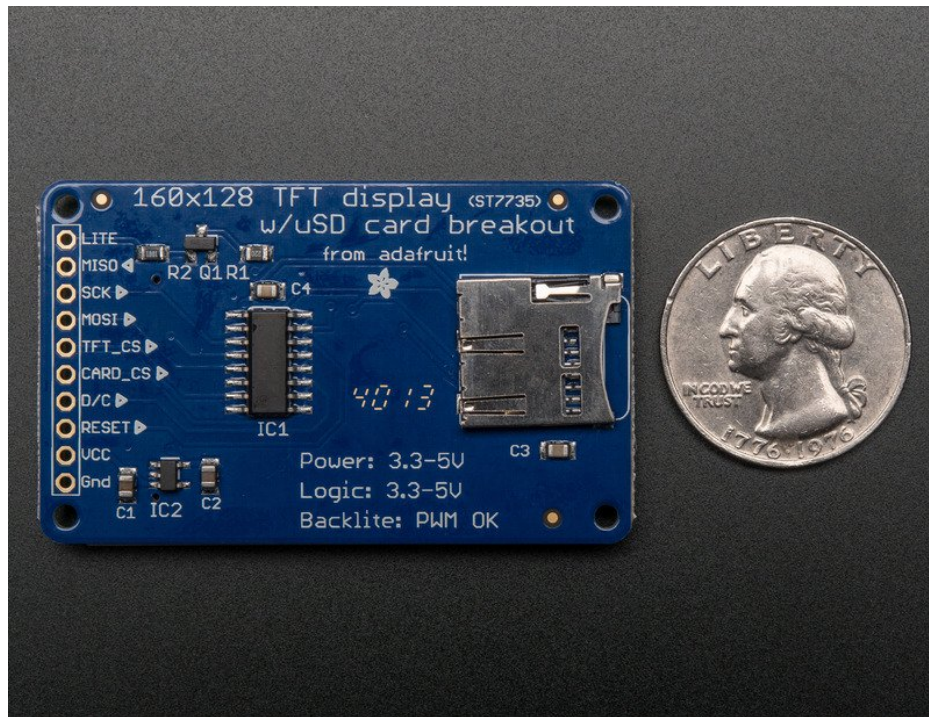
This color display uses SPI to receive image data. That means you need at least 4 pins - clock, data in, tft cs and d/c. If you'd like to have SD card usage too, add another 2 pins - data out and card cs. However, there's a couple other pins you may want to use, lets go thru them all!

- **Lite** - this is the PWM input for the backlight control. Connect to 3-5VDC to turn on the backlight. Connect to ground to turn it off. Or, you can PWM at any frequency.
- **MISO** - this is the SPI Microcontroller In Serial Out pin, its used for the SD card. It isn't used for the TFT display which is write-only
- **SCLK** - this is the SPI clock input pin
- **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
- **TFT_CS** - this is the TFT SPI chip select pin
- **Card CS -** this is the SD card chip select, used if you want to read from the SD card.
- **D/C** - this is the TFT SPI data or command selector 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.
- **Vcc** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **GND** - this is the power and signal ground pin

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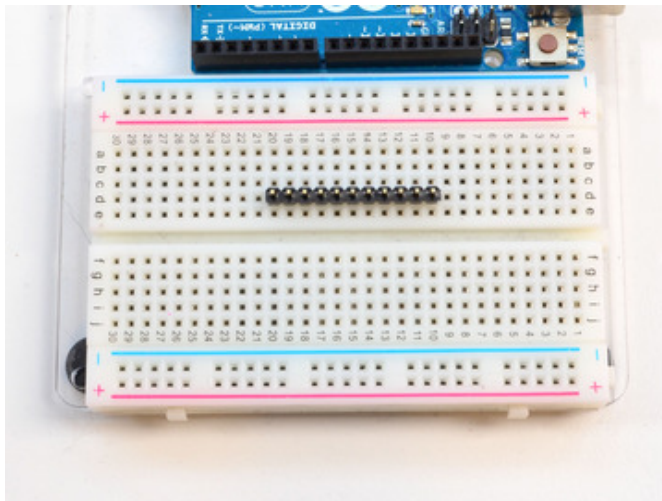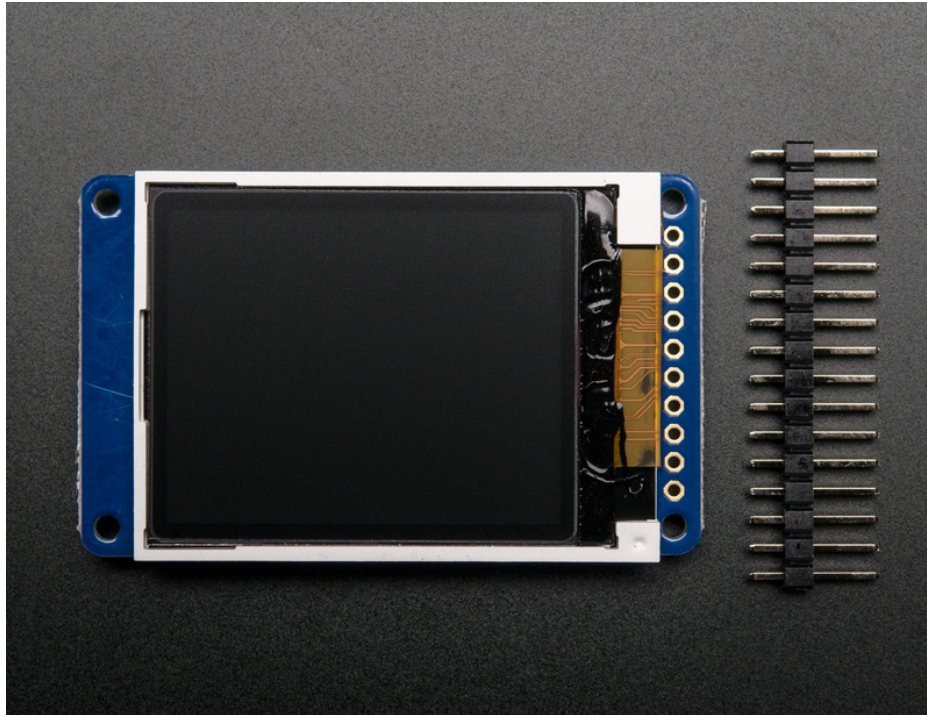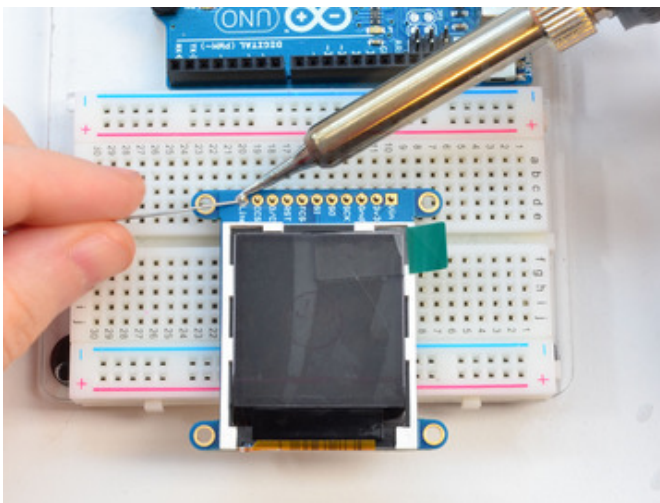
Prepare the header strip:

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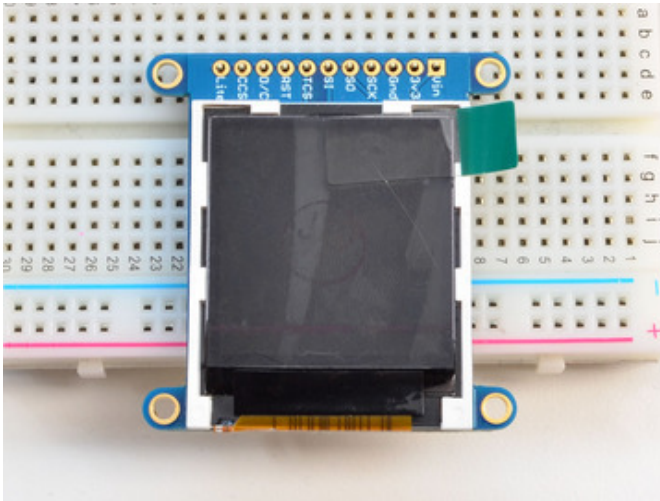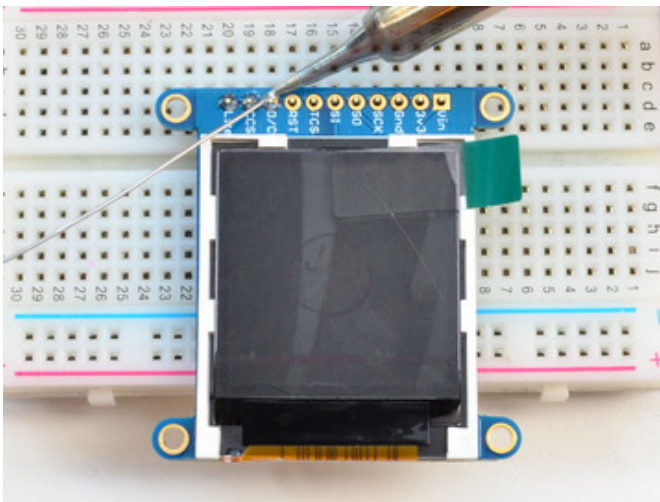
Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads
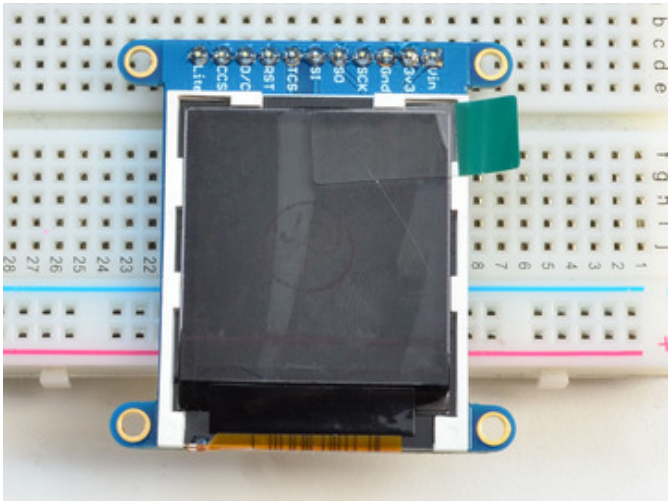
And Solder!

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://adafru.it/aTk)).*

You're done! Check your solder joints visually and continue onto the next steps

There are two ways to wire up these displays - one is a more flexible method (you can use any pins on the Arduino) and the other is much faster (4-8x faster, but you are required to use the hardware SPI pins) We will begin by showing how to use the faster method, you can always change the pins later for flexible 'software SPI'
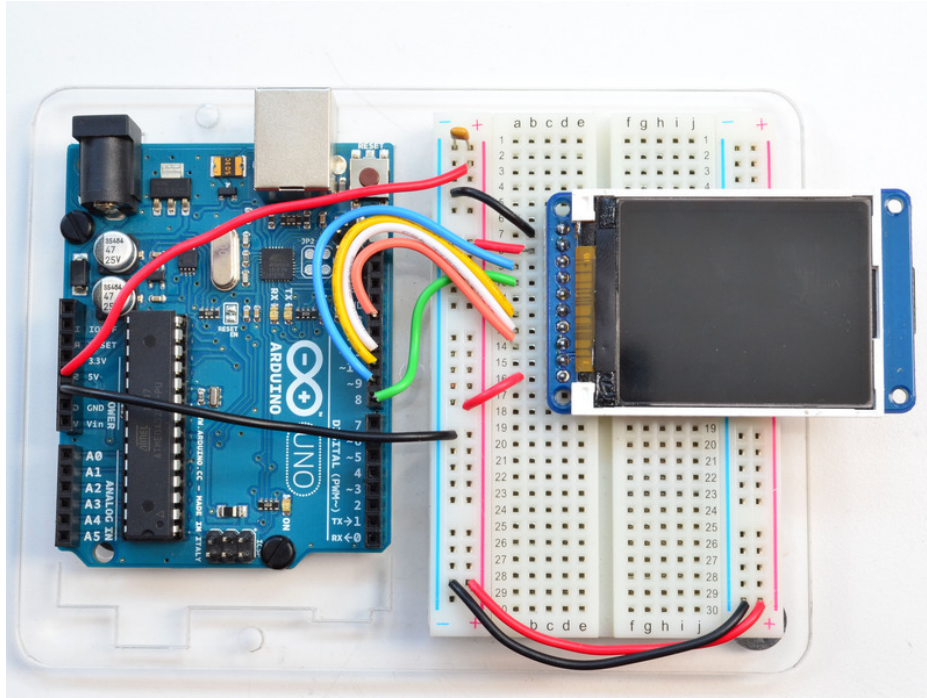

Wiring up the display in SPI mode is pretty easy as there's not that many pins! We'll be using hardware SPI, but you can also use software SPI (any pins) later. Start by connecting the power pins

- **3-5V Vin** connects to the Arduino **5V** pin - red wires
- **GND** connects to Arduino ground - black wires
- **CLK** connects to SPI clock. On Arduino Uno/Duemilanove/328-based, thats **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)) - this is the orange wire
- **MOSI** connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, thats **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)) - this is the white wire
- **CS** connects to our SPI Chip Select pin. We'll be using **Digital 10** but you can later change this to any pin - this is the yellow wire
- **RST** connects to our TFT reset pin. We'll be using **Digital 9** but you can later change this pin too - this is the blue wire
- **D/C** connects to our SPI data/command select pin. We'll be using **Digital 8** but you can later change this pin too - this is the green wire

Install Adafruit ST7735 TFT Library

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..."
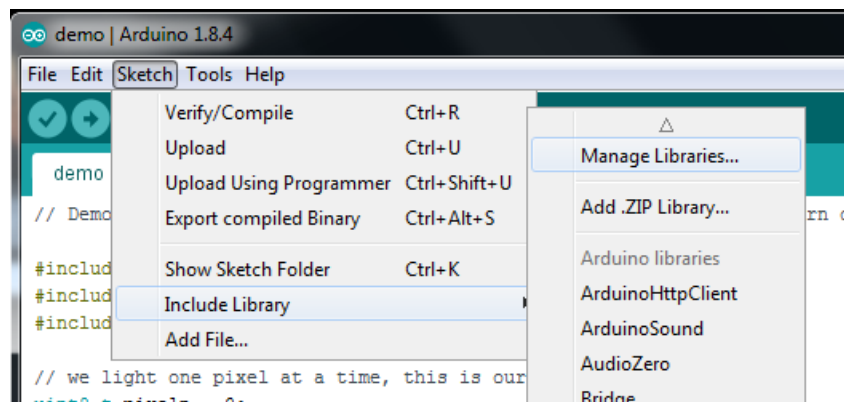
Search for and install the Adafruit GFX library:
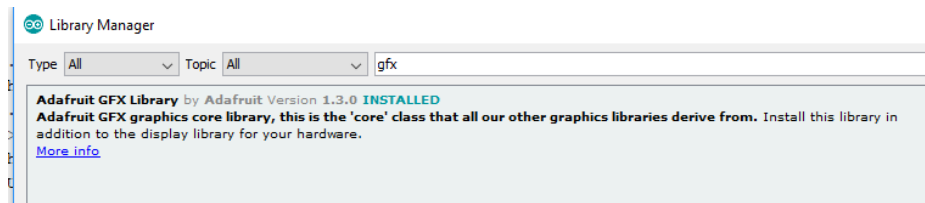
And the Adafruit ST7735 library:
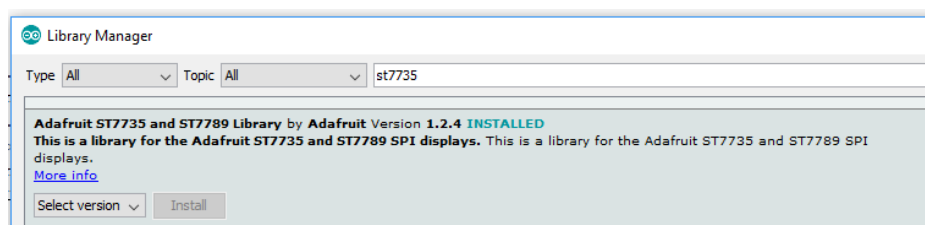
If using an older version of the Arduino IDE (pre-1.8.10), also locate and install the **Adafruit_BusIO** library (newer versions do this automatically when using the Arduino Library Manager).

If this is all unfamiliar, we have a tutorial introducing Arduino library concepts and installation (https://adafru.it/aYM).

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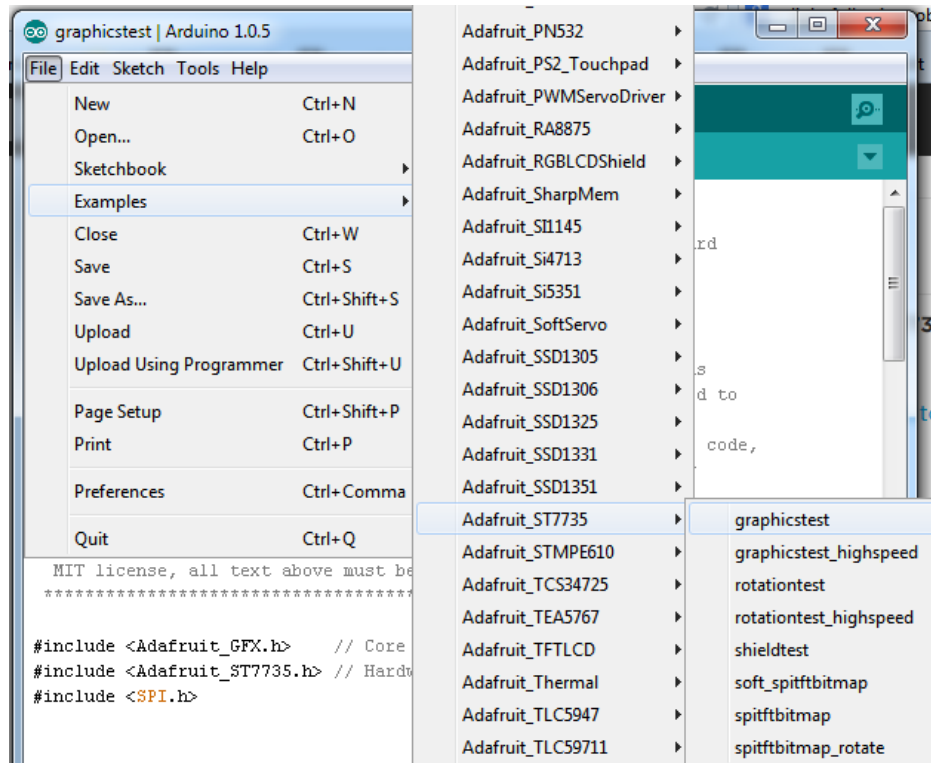
After restarting the Arduino software, you should see a new **example** folder called **Adafruit_ST7735** and inside, an example called **graphicstest**.

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.
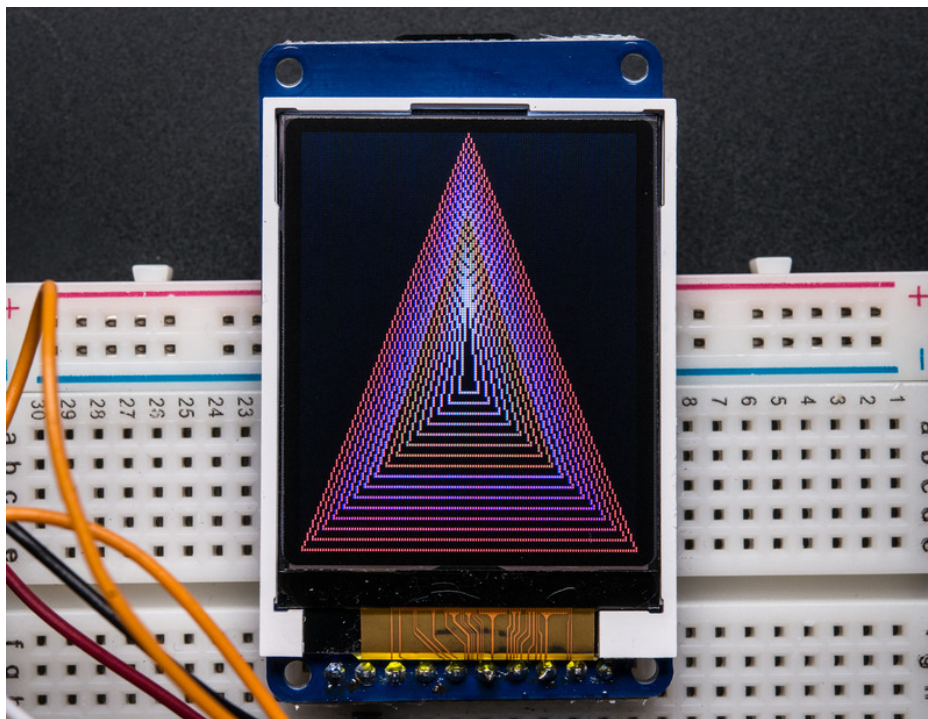
Once uploaded, the Arduino should perform all the test display procedures! If you're not seeing anything - first check if

you have the backlight on, if the backlight is not lit something is wrong with the power/backlight wiring. If the backlight is lit but you see nothing on the display make sure you're using our suggested wiring.

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  -1`

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

In this example, we'll show how to display a 128x160 pixel full color bitmap from a microSD card.

We have an example sketch in the library showing how to display full color bitmap images stored on an SD card. You'll need a microSD card such as this one (http://adafru.it/102).

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:
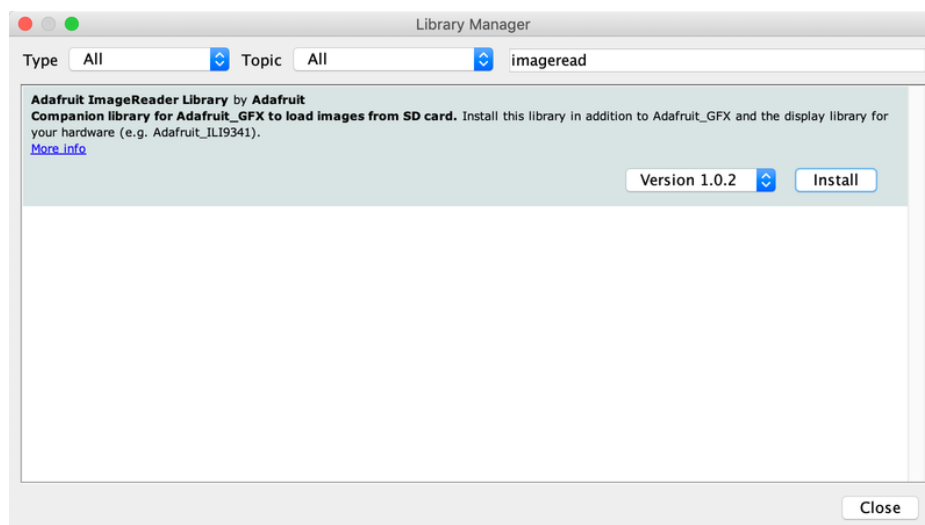
You'll also need an image. We suggest starting with this bitmap of a parrot.

If you want to later use your own image, use an image editing tool and crop your image to no larger than 160 pixels high and 128 pixels wide. Save it as a 24-bit color **BMP** file - it must be 24-bit color format to work, even if it was originally a 16-bit color image - because of the way BMPs are stored and displayed!

Copy the **parrot.bmp** to the microSD card and insert it into the micro SD card holder on your shield or breakout board.
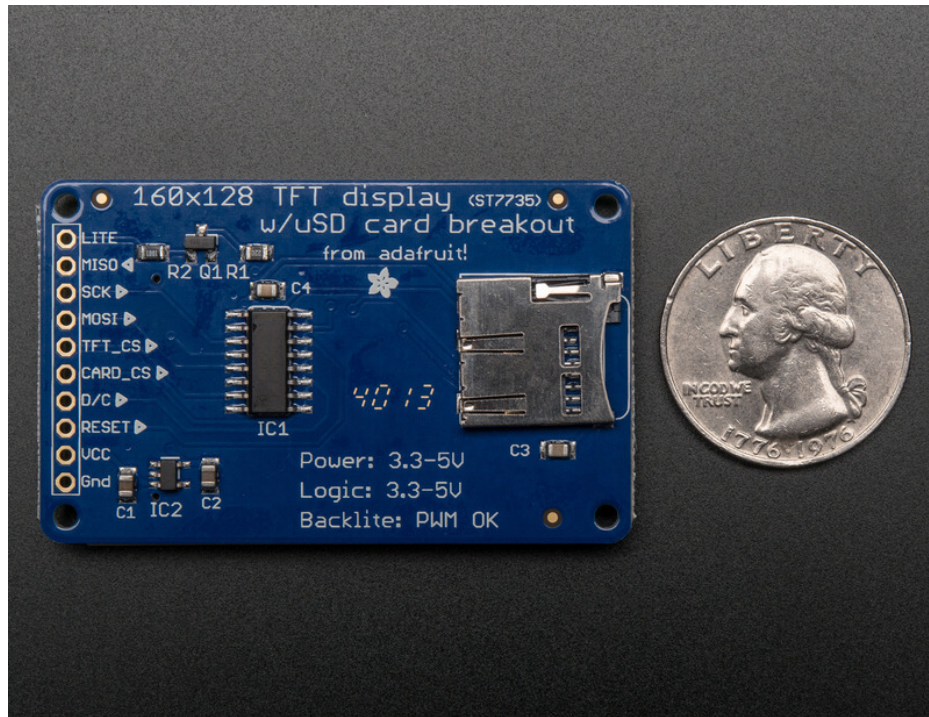
Breakout Wiring

Shield users can skip directly to the "**Example Sketch**" section.

Wire up the TFT as described on the wiring & test page and add the two wires for talking to the SD card.
Connect **CARD_CS** (the unconnected pin in the middle) to digital pin **4** (you can change this later to any pin you want).
Connect **MISO** (second from the right) to the Arduino's hardware SPI **MISO** pin. For Classic arduinos, this is pin **12**. For Mega's this is pin **50**. You can't change the **MISO** pin, it's fixed in the chip hardware.
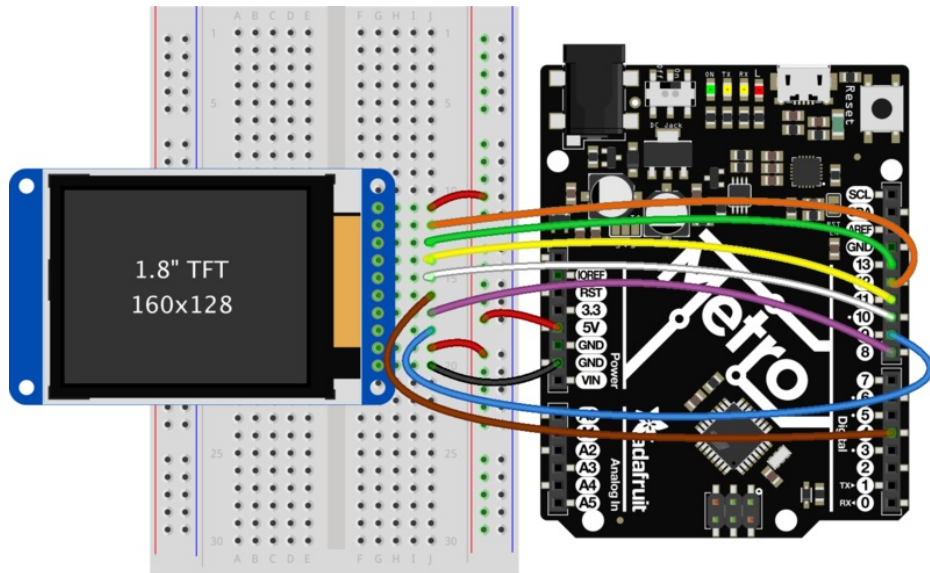
fritzing

Example Sketch
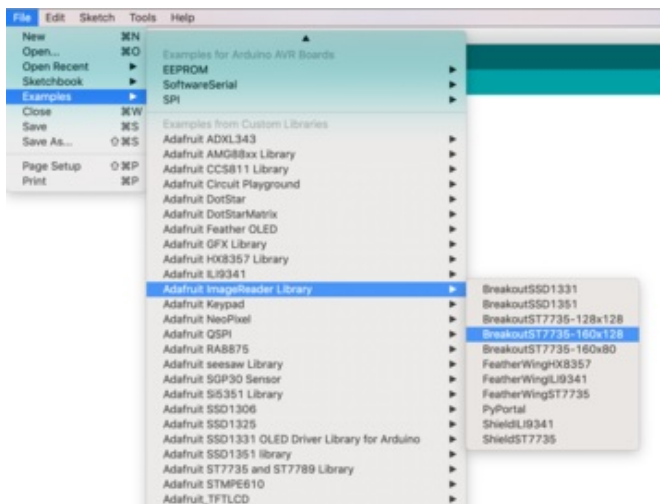
If you have the breakout, open the **File→examples→Adafruit ImageReader Library→BreakoutST7735 - 160x128** example.
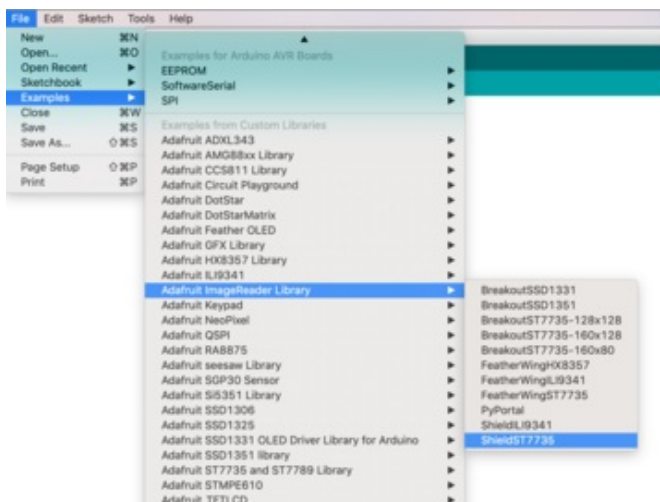
If you have the shield, open the **File→examples→Adafruit ImageReader Library→ShieldST7735** example.

Now upload the example sketch to the Arduino. It should display the parrot image. 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.

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.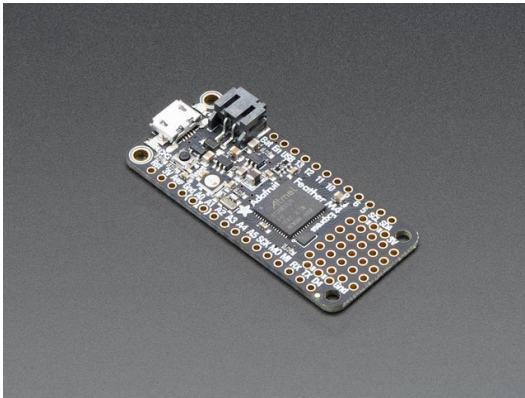 The steps should be about the same for the Feather M0 Express or either of the Metros. If you haven't already, be sure to check out our Feather M4 Express (https://adafru.it/EEm) guide.

Adafruit Feather M4 Express - Featuring ATSAMD51

OUT OF STOCK

Out Of Stock

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.

Wiring the Breakout to the Feather

- **3-5V VCC** connects to the Feather **3V** pin
- **GND** connects to Feather ground
- **SCK** connects to SPI clock. On the Feather that's **SCK**.
- **MISO** connects to SPI MISO. On the Feather that's **MI**
- **MOSI** connects to SPI MOSI. On the Feather that's **MO**
- **TFT_CS** connects to our SPI Chip Select pin. We'll be using **Digital 5** but you can later change this to any pin
- **D/C** connects to our SPI data/command select pin. We'll be using **Digital 6** but you can later change this pin too.
- **RESET** connects to our reset pin. We'll be using **Digital 9** but you can later change this pin too.
- **LITE** connects to the Feather **3V** pin. This is the only display that this pin is required to be connected or the backlight won't work.
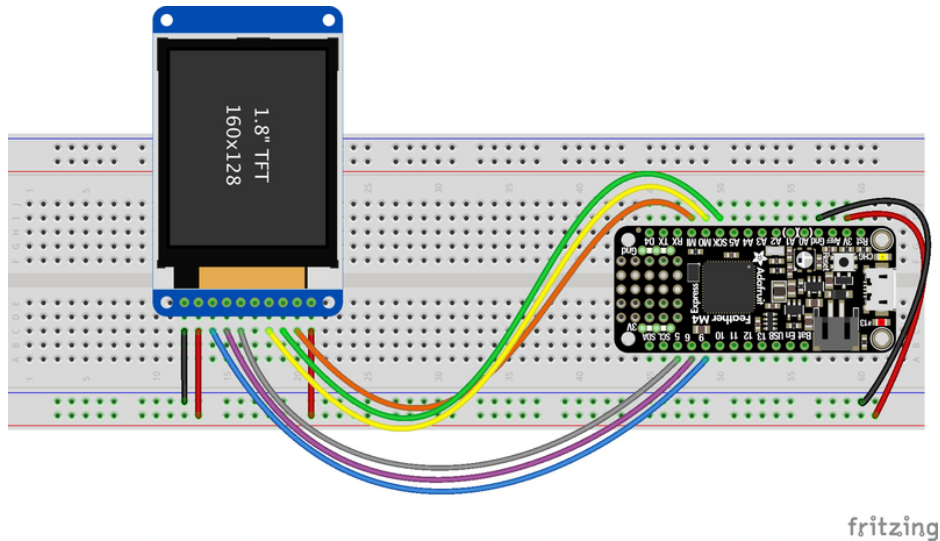
Required CircuitPython Libraries

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_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.

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

CircuitPython Code Example

```python
"""
This test will initialize the display using displayio and draw a solid green
background, a smaller purple rectangle, and some yellow text.
"""

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

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

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

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

display = ST7735R(display_bus, width=160, height=128, rotation=90, bgr=True)

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

color_bitmap = displayio.Bitmap(160, 128, 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(150, 118, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088  # Purple
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=5, y=5)
splash.append(inner_sprite)

# Draw a label
text_group = displayio.Group(max_size=10, scale=2, x=11, y=64)
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_st7735r driver.

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

Next we release any previously used displays. This is important because if the Feather is reset, the display pins are not automatically released and this makes them available for use again.

```
displayio.release_displays()
```

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

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

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

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

Finally, we initialize the driver with a width of 160 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. Because we want to use the display horizontally and the default orientation is vertical, we rotate it 90 degrees. One other parameter that we provide is bgr=True and the reason for this is that the color ordering of certain displays is Blue, Green, Red rather than the usual Red, Green, Blue. It tell displayio the correct color ordering for this particular display.

```
display = ST7735R(display_bus, width=160, height=128, rotation=90, bgr=True)
```
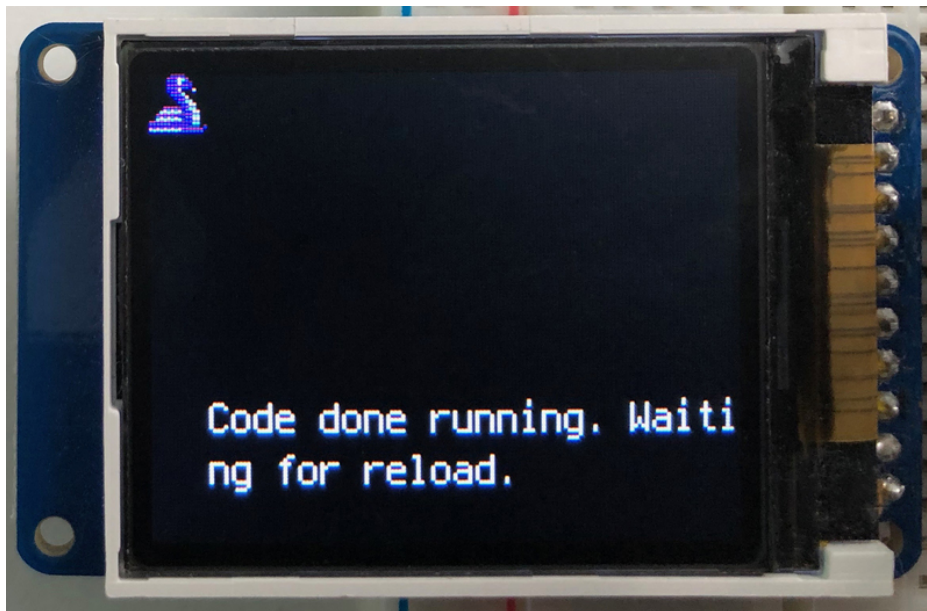
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating 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(160, 128, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

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

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


Next we will create a smaller purple square. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case, we will create a bitmap that is 5 pixels smaller on each side. The screen is **160x128**, so we'll want to subtract 10 from each of those numbers.

We'll also want to place it at the position (5, 5) so that it ends up centered.

```
inner_bitmap = displayio.Bitmap(150, 118, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=5, y=5)
splash.append(inner_sprite)
```

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

Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font and scale it up by a factor of two. 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 64 for the Y coordinate, and around 11 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=2, x=11, y=64)
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)

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, thats **SLCK**
- **MOSI** connects to SPI MOSI. On the Raspberry Pi, thats also **MOSI**
- **CS** connects to our SPI Chip Select pin. We'll be using **CE0**
- **D/C** connects to our SPI Chip Select pin. We'll be using **GPIO 25**, but this can be changed later.
- **RST** connects to our Reset pin. We'll be using **GPIO 24** but this can be changed later as well.
- **Vin** connects to the Raspberry Pi's **3V** pin
- **GND** connects to the Raspberry Pi's **ground**
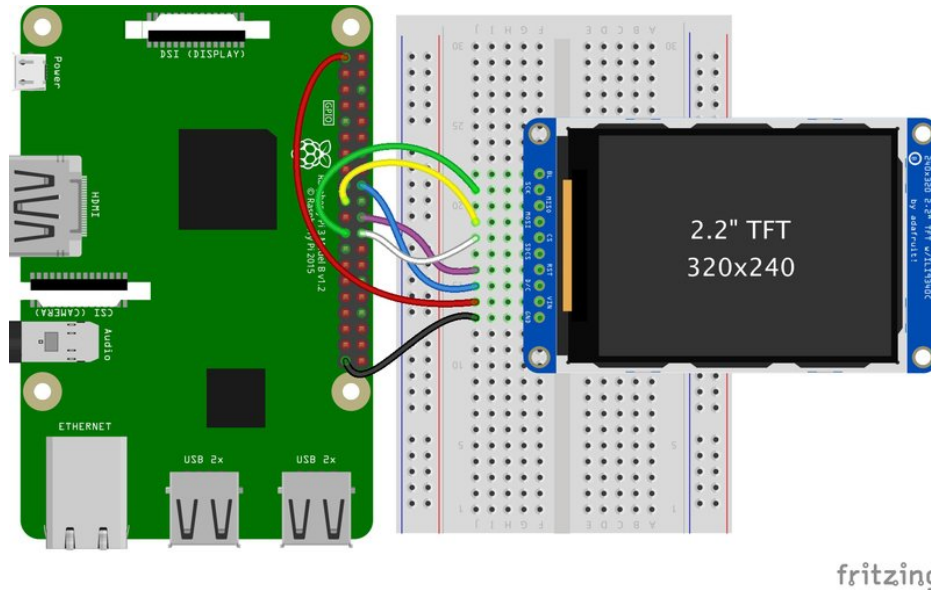
2.4", 2.8", 3.2", and 3.5" Displays

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.
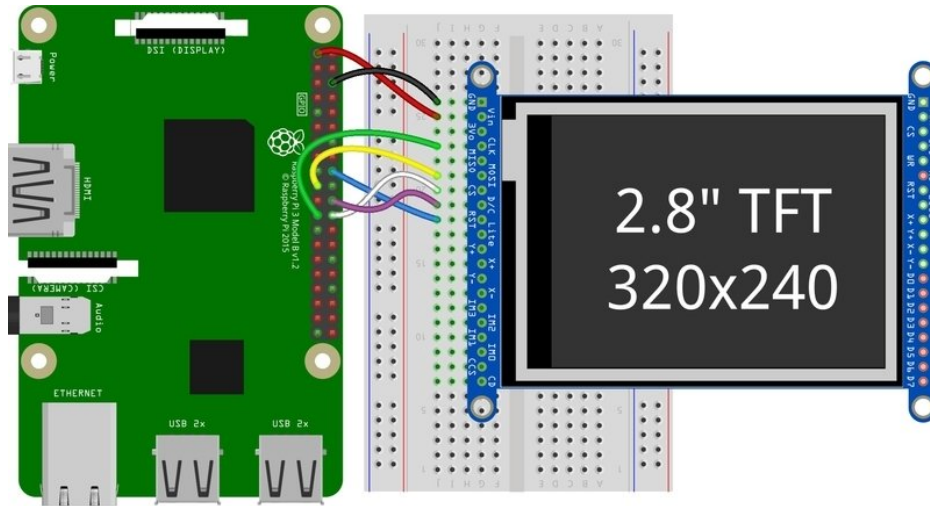
ST7789 and ST7735-based Displays

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.
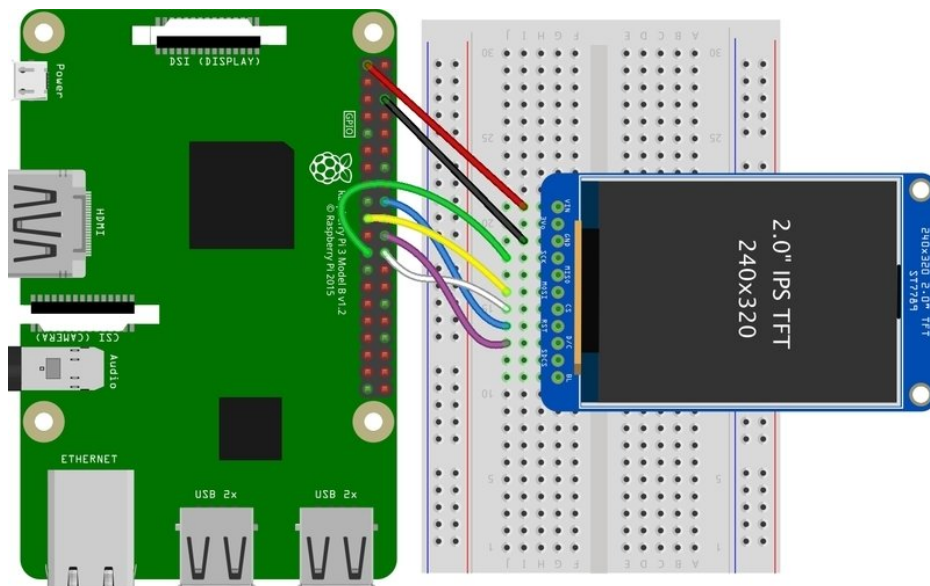
0.96", 1.14", and 1.44" Displays

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

1.8" Display

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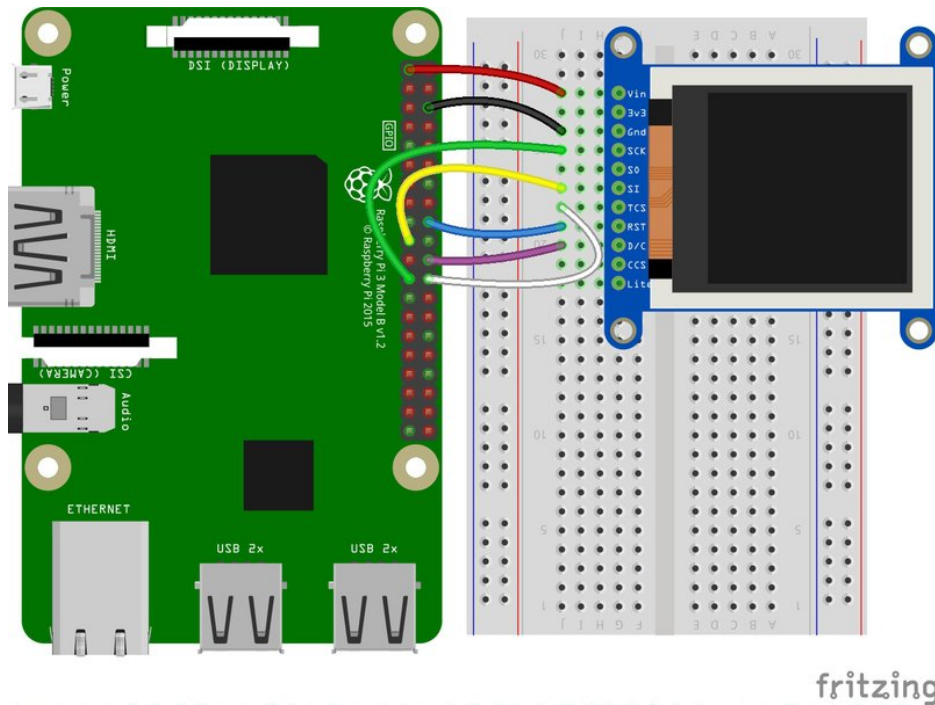- **CS** connects to our SPI Chip Select pin. We'll be using **CE0**
- **MOSI** connects to SPI MOSI. On the Raspberry Pi, thats also **MOSI**
- **CLK** connects to SPI clock. On the Raspberry Pi, thats **SLCK**
- **LITE** connects to the Raspberry Pi's **3V** pin. This can be used to separately control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

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.
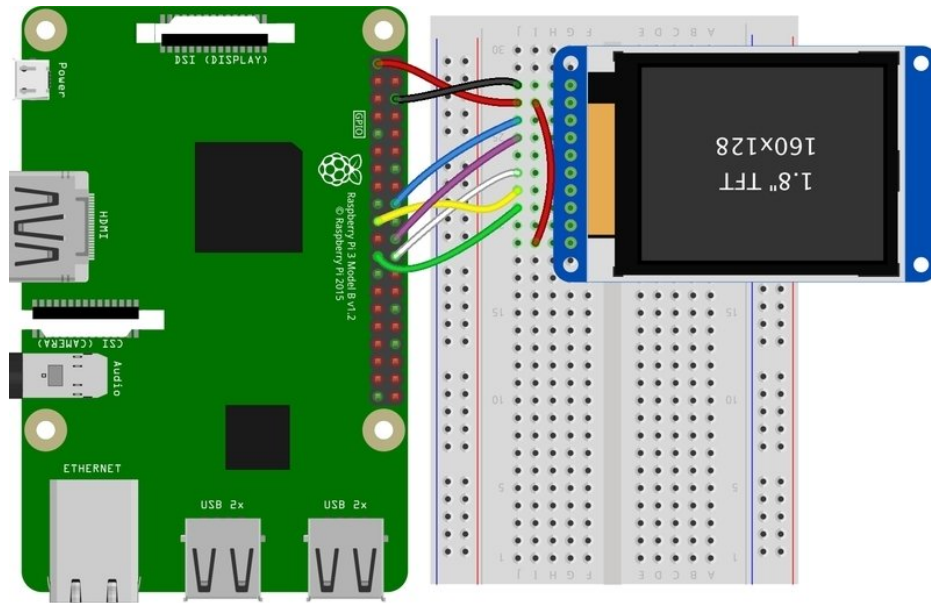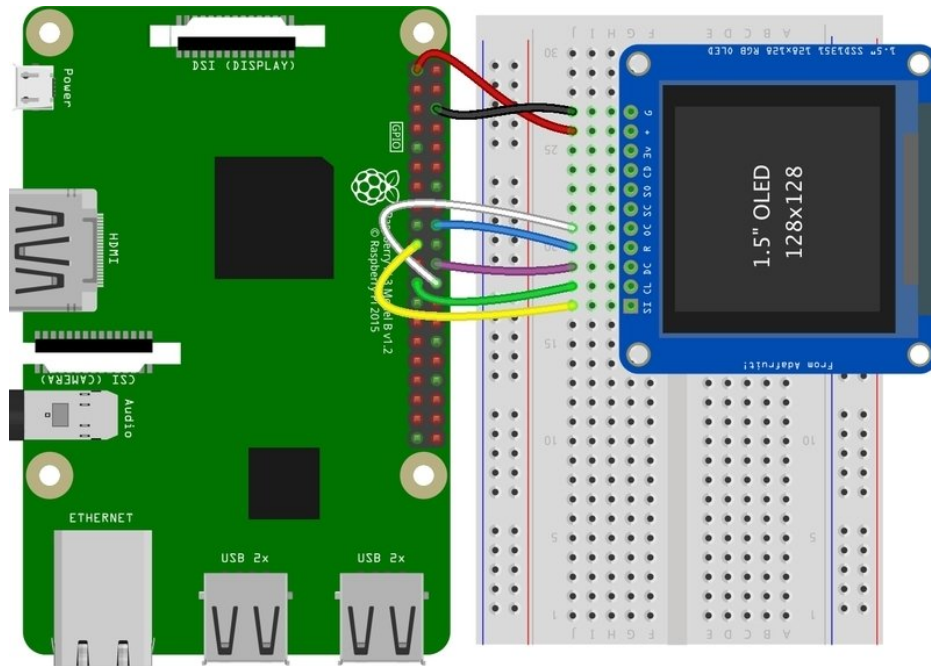
SSD1331-based Display

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.
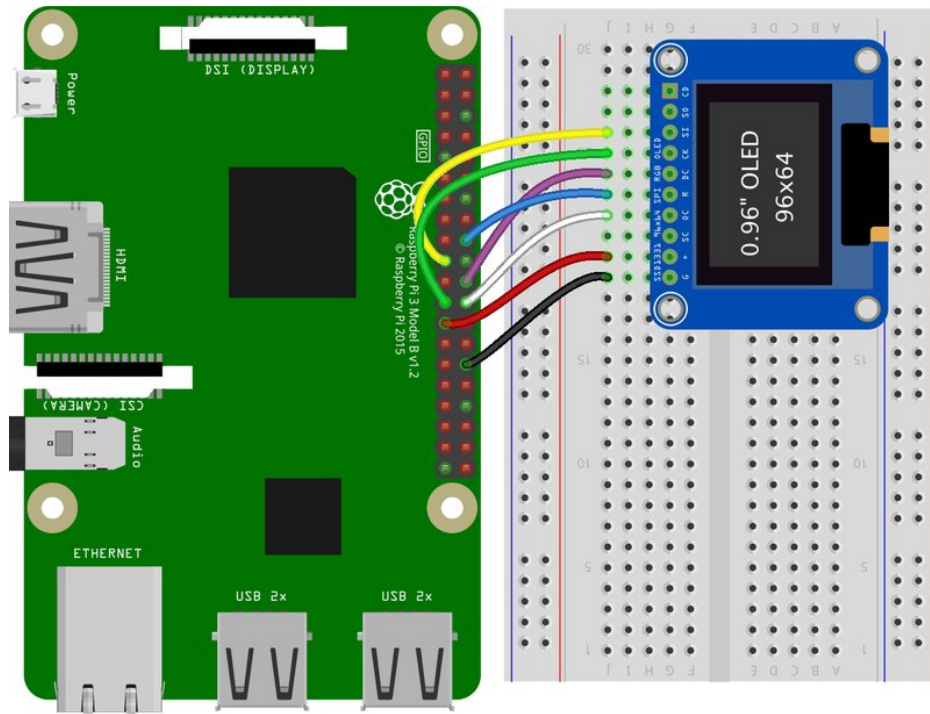- **CS** connects to our SPI Chip Select pin. We'll be using **CE0**
- **Vin** connects to the Raspberry Pi's **3V** pin
- **GND** connects to the Raspberry Pi's **ground**

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 ttf-dejavu

Pillow Library

We also need PIL, the Python Imaging Library, to allow graphics and using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

- sudo apt-get install python3-pil

That's it. You should be ready to go.

Python Usage

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.

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

```python
import adafruit_rgb_display.hx8357 as hx8357  # pylint: disable=unused-import
import adafruit_rgb_display.st7735 as st7735  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 as ssd1351  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1331 as 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 = 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,                 # 0.96" MiniTFT ST7735R
# 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
```

```
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 the line that starts with:

`disp = ili9341.ILI9341(spi,`

and uncomment the line appropriate for your display. The displays have a rotation property so that it can be set in just one place.

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

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

#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,               # 0.96" MiniTFT ST7735R
#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.

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

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

```python
# 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.

```python
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.

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

```python
# 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,                # 0.96" MiniTFT ST7735R
# 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:

```
import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789  # pylint: disable=unused-import
import adafruit_rgb_display.hx8357 as hx8357  # pylint: disable=unused-import
import adafruit_rgb_display.st7735 as st7735  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1351 as ssd1351  # pylint: disable=unused-import
import adafruit_rgb_display.ssd1331 as 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, 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,                # 0.96" MiniTFT ST7735R
# 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)
```
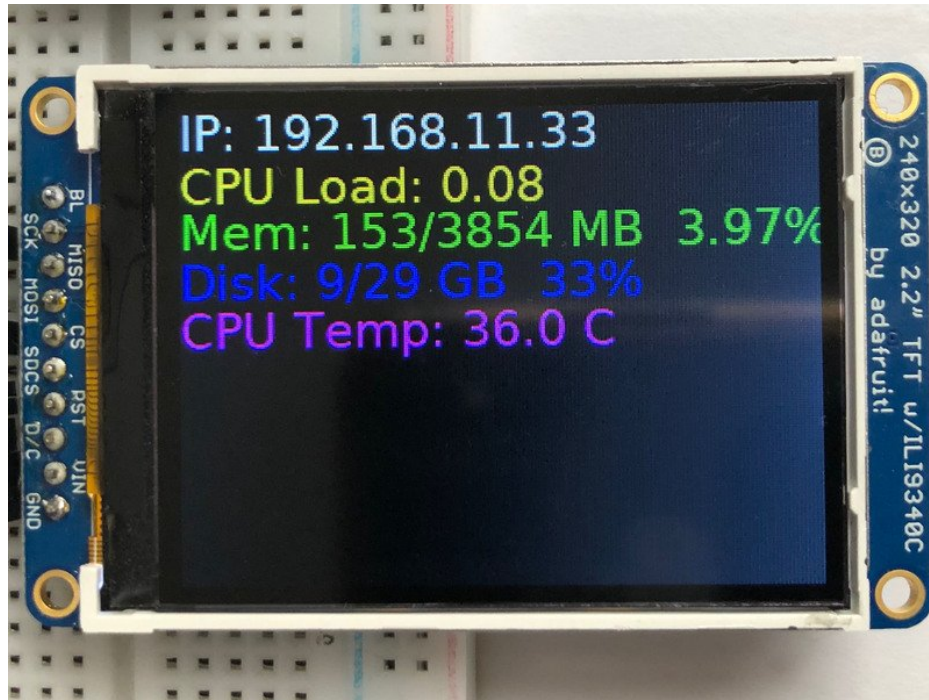
1.8" TFT Shield V2

Let's take a tour of the 1.8" TFT Shield
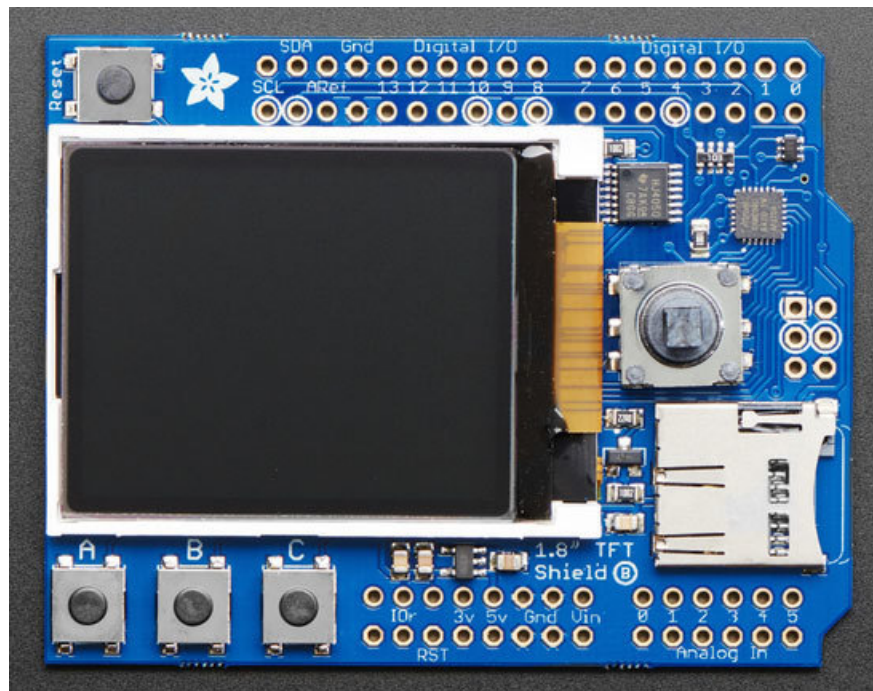

TFT Display
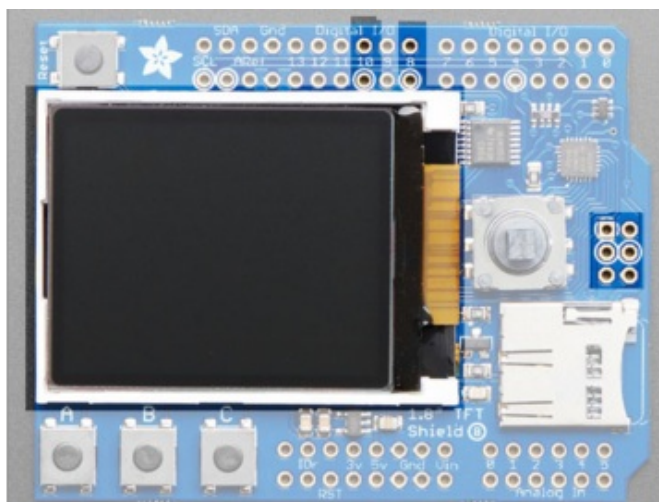
In the center is the 1.8" TFT display. This display is full color (16-bit RGB), 128x160 pixels, and has a backlight. The display receives data over SPI plus two pins:

- SCK - SPI Clock
- MOSI - SPI Data
- Digital 10 - Chip Select
- Digital 8 - Data/Command Select

The TFT reset is connected to the seesaw chip. The backlight is also PWM controlled by the seesaw chip. The 4 SPI+control pins, however, must be controlled directly by the Arduino

Buttons & Joystick
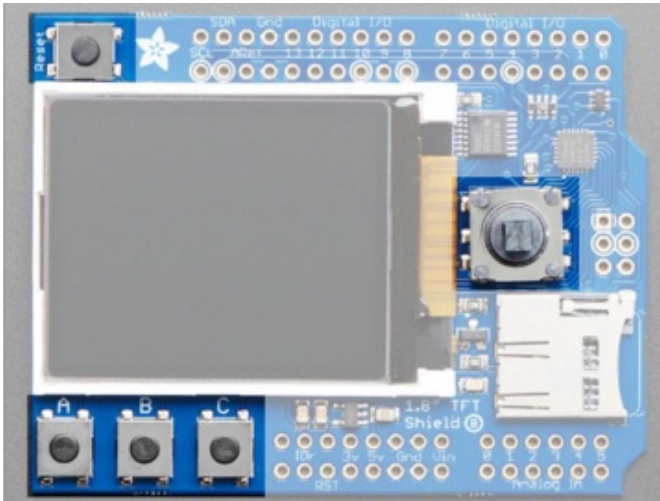
In addition of the display, you also get a bunch of user-interface buttons.

In the top left is the **Reset** button, this will reset the shield and Arduino when pressed. It is connected directly to the Reset pins

There are three buttons labeled **A B C** below the TFT, these are connected to the seesaw chip. You can read the values over I2C

To the right of the TFT is a 5-way joystick. It can be pushed up/down/left/right and select (in). It is connected to the seesaw chip, you can read the joystick over I2C

SD Card
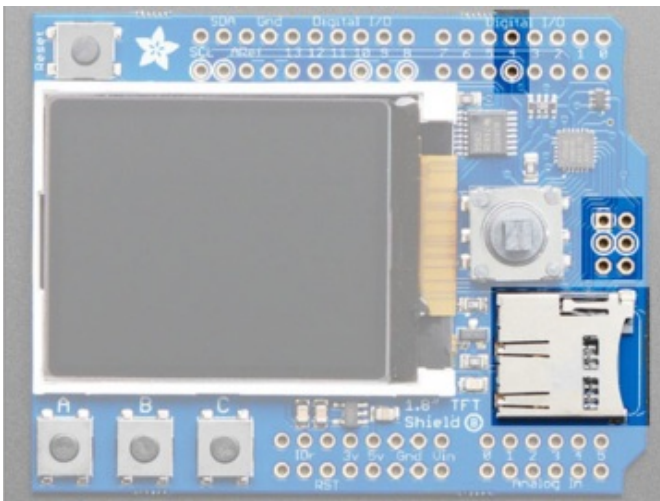
The micro SD card slot can be used to read/write data from any micro SD card using the Arduino libraries. The SD card is connected to the SPI pins as well as **Digital #4** for **Chip Select**

The SD card is not required for use, but it's handy for storing images

seesaw I2C Expander
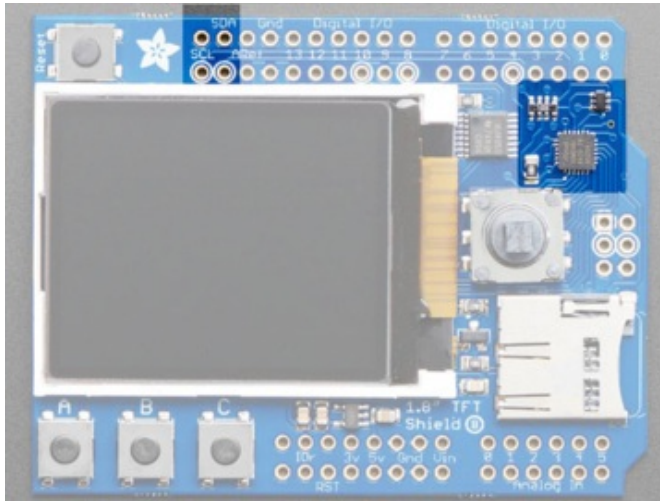
Instead of taking up a bunch of GPIO pins to read the buttons and joystick, as well as controlling the TFT backlight, we use an I2C expander chip called the **seesaw**. It is connected to the SDA/SCL pins and can read/write pins with our library. This saves a ton of pins and then you can always use the I2C pins for other sensors, as long as the address doesnt conflict

Don't forget! Since the seesaw chip is used for the TFT backlight and reset, **you need to activate it even if you are not reading the buttons or joystick.**

Testing the Shield

You can test your assembled shield using the example code from the library.

Start by installing a bunch of libraries!

Open the Arduino Library manager
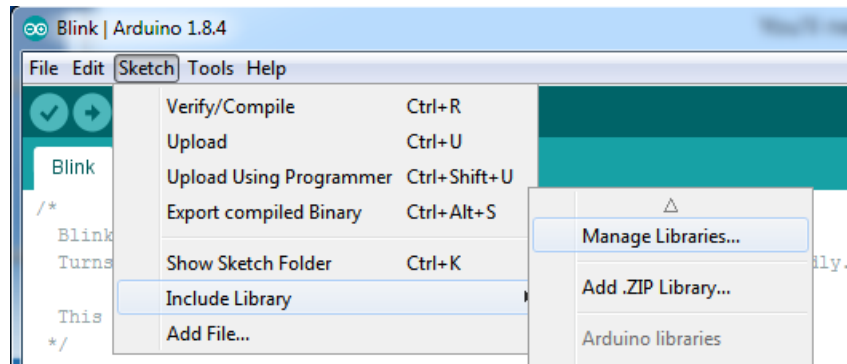
Install the **Adafruit GFX Library**
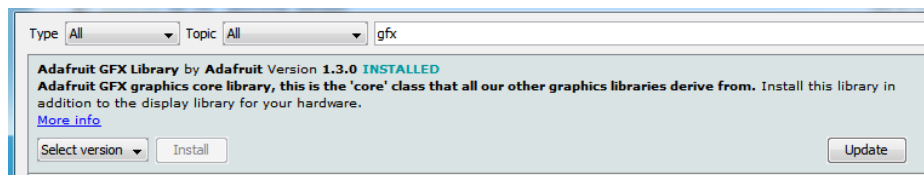
If using an older version of the Arduino IDE (pre-1.8.10), also locate and install the **Adafruit_BusIO** library (newer versions do this automatically when using the Arduino Library Manager).

Adafruit ST7735 Library

Adafruit seesaw Library
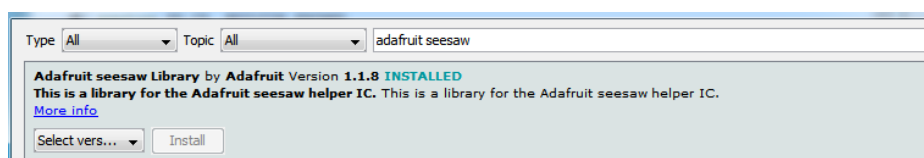
You can read more about installing libraries in our tutorial (https://adafru.it/aYG).

Restart the Arduino IDE.

1.8" Shield with seesaw
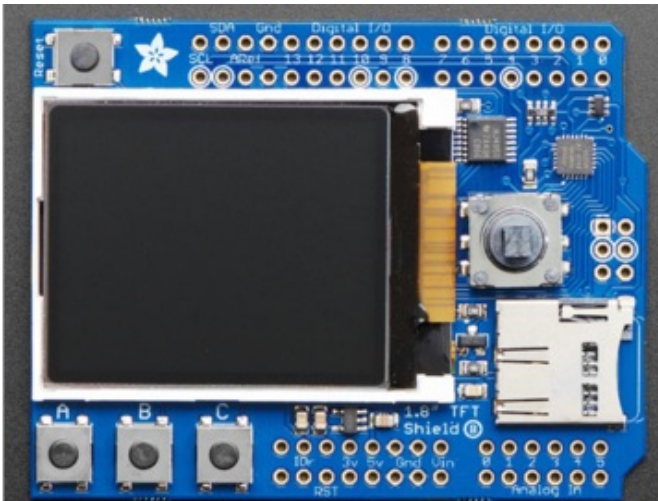
If your shield looks like this, you have the 1.8" seesaw version (the most recent) which will work with just about any/all boards. For this version load up the **seesaw_shield18_test example**
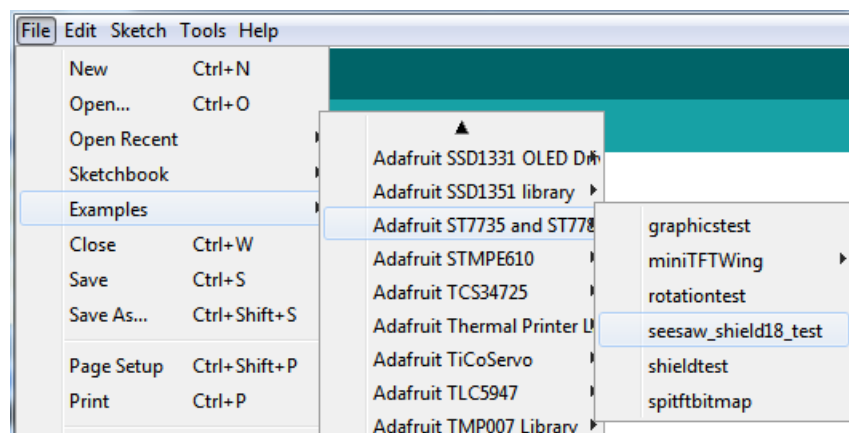
Upload to your microcontroller, and open the serial port watcher at **9600** baud:
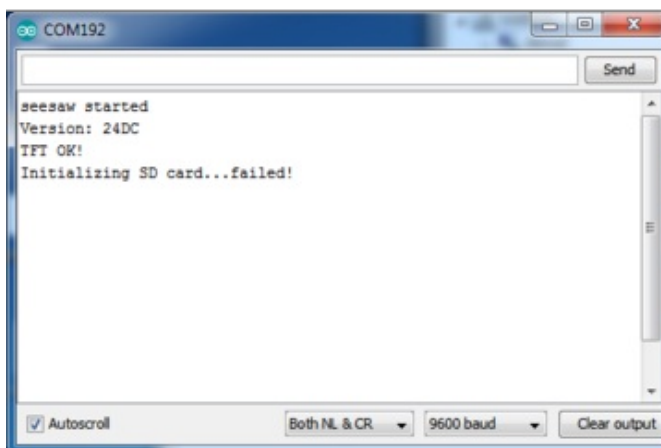
The sketch waits until the serial port is opened (you can make it auto-start once you know things are working by removing the `while (!Serial);` line

Check that the seesaw chip is detected, you should see text display on the TFT after a quick draw test.

If you don't have an SD card inserted, it will fail to init the SD card, that's ok you can continue with the test

Once you've gotten this far try pressing all the buttons on the board (except for RESET) to activate the invert-blinking loop.
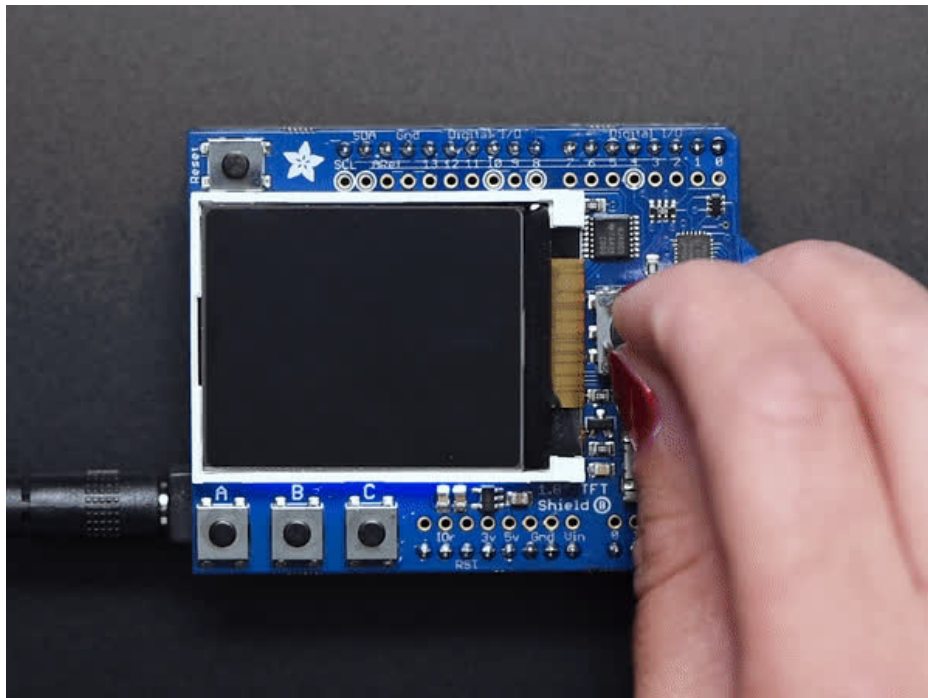
For more details about seesaw, check out our guide (https://adafru.it/Cmk) - we made a nice wrapper for the 1.8" TFT to control the backlight and read buttons but it still might be useful to know the underlying protocol

Displaying a Bitmap

If you have parrot.bmp (https://adafru.it/cmn) stored on the SD card you will get a nice parrot display once the buttons have all been pressed

```
version: 24DC
TFT OK!
Initializing SD card...OK!
SYSTEM~1/
        WPSETT~1.DAT              12
        INDEXE~1                 76
SINE.MP3               80666
TEST.MP3               869590
TEST3.MP3              3940145
TEST2.MP3              443904
T2.MP3         1504117
T3.MP3         198658
PARROT.BMP             61496

Loading image '/parrot.bmp'
File size: 61496
Image Offset: 54
Header size: 40
Bit Depth: 24
Image size: 128x160
Loaded in 1117 ms
```

Autoscroll   Both NL & CR   9600 baud   Clear output
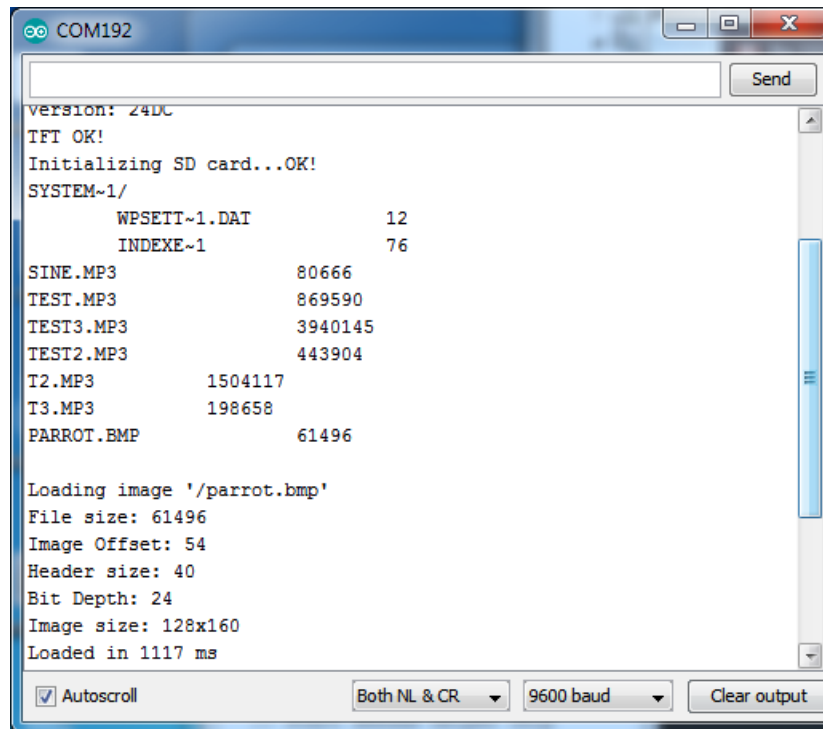
CircuitPython Displayio Quickstart

You will need a Metro capable of running CircuitPython such as the Metro M0 Express or the Metro M4 Express. We recommend the Metro M4 Express because it's much faster and works better for driving a display. The steps should be about the same for the Metro M0 Express. If you haven't already, be sure to check out our Adafruit Metro M4 Express featuring ATSAMD51 (https://adafru.it/Fkt) guide.
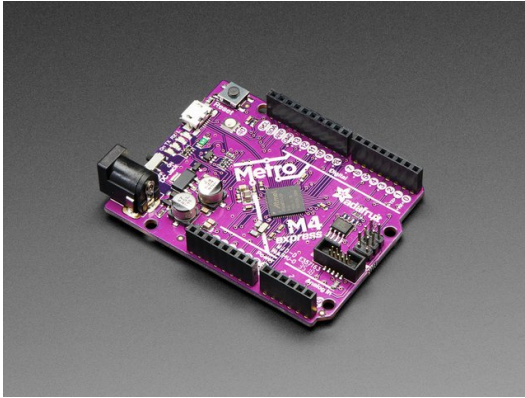
Adafruit Metro M4 feat. Microchip ATSAMD51

$27.50
IN STOCK

Add To Cart

You could use a Grand Central which also has an M4 Processor. For this board, be sure to check out our Introducing the Adafruit Grand Central M4 Express (https://adafru.it/DK7) guide.
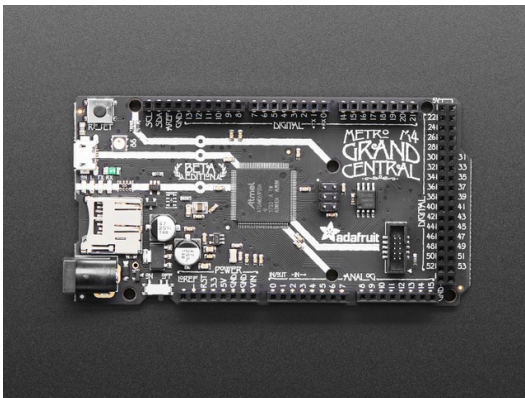
Adafruit Grand Central M4 Express featuring the SAMD51

$37.50
IN STOCK

Add To Cart

If you need WiFi capabilities for your project, you could also use the Metro M4 Airlift Lite. For this board, be sure to check out our Adafruit Metro M4 Express AirLift (https://adafru.it/EZh) guide.
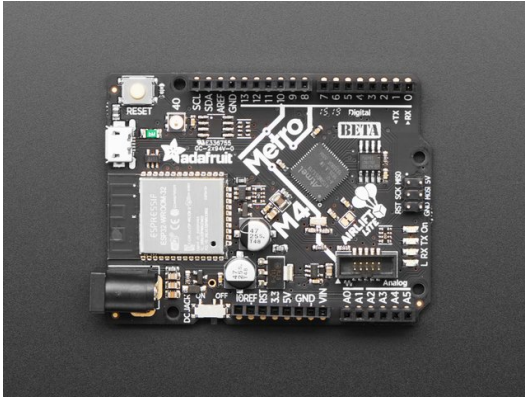
Adafruit Metro M4 Express AirLift (WiFi) - Lite

$34.95
IN STOCK

Add To Cart

Preparing the Shield

Before using the TFT Shield, you will need to solder the headers on. Be sure to check out the Adafruit Guide To Excellent Soldering (https://adafru.it/drI). After that the shield should be ready to go.

Required CircuitPython Libraries

To use this display with `displayio`, there are a few required libraries. You will need the display driver and since this is no ordinary display and has some additional controls, you will also need the seesaw and busdevice libraries.

https://adafru.it/u0d

https://adafru.it/u0d

https://adafru.it/D5O

https://adafru.it/D5O

https://adafru.it/EGk

https://adafru.it/EGk

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_st7735r**
- **adafruit_seesaw**
- **adafruit_bus_device**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_st7735r**, **adafruit_seesaw** and **adafruit_bus_device** files and folders copied over.

CircuitPython Code Example

```
"""
This example will test out the display on the 1.8" TFT Shield
"""
import time
import board
import displayio
from adafruit_seesaw.tftshield18 import TFTShield18
from adafruit_st7735r import ST7735R

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

ss = TFTShield18()

spi = board.SPI()
tft_cs = board.D10
tft_dc = board.D8

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

ss.tft_reset()
display = ST7735R(display_bus, width=160, height=128, rotation=90, bgr=True)

ss.set_backlight(True)

while True:
    buttons = ss.buttons

    if buttons.right:
        print("Button RIGHT!")

    if buttons.down:
        print("Button DOWN!")

    if buttons.left:
        print("Button LEFT!")

    if buttons.up:
        print("Button UP!")

    if buttons.select:
        print("Button SELECT!")

    if buttons.a:
        print("Button A!")

    if buttons.b:
        print("Button B!")

    if buttons.c:
        print("Button C!")

    time.sleep(0.001)
```

Let's take a look at the sections of code one by one. We start by importing `time`, so we can pause, the `board` so that we can initialize SPI, `displayio`, the `tftshield18` seesaw library, and the `adafruit_ili9341` driver.

```
import time
import board
import displayio
from adafruit_seesaw.tftshield18 import TFTShield18
from adafruit_st7735r import ST7735R
```

Next we release any previously used displays. This is important because if the Metro is reset, the display pins are not automatically released and this makes them available for use again.

```
displayio.release_displays()
```

We set up seesaw using the TFTShield18, which was written specifically for this shield to make things very easy.

```
ss = TFTShield18()
```

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.D10
tft_dc = board.D8
```

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

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

Finally, we reset the display, initialize the driver with a width of 160 and a height of 128, and turn on the backlight. 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.

```
ss.tft_reset()
display = ST7735R(display_bus, width=160, height=128, rotation=90, bgr=True)

ss.set_backlight(True)
```
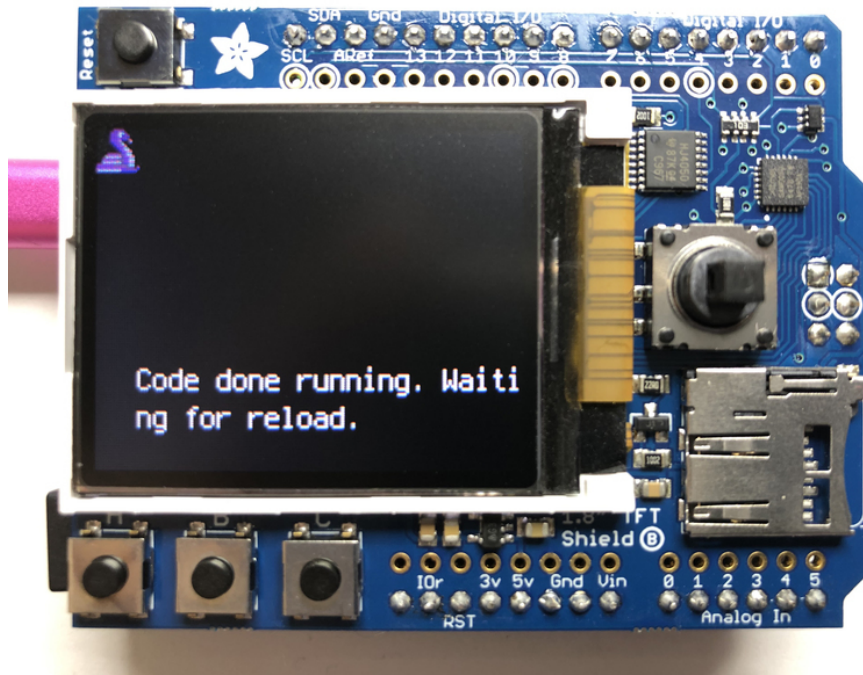
Finally, we place an infinite loop at the end and constantly read the buttons. If a button is detected as being pressed, a message specifies which one. Multiple buttons can be pressed at the same time. We also provide an optional small delay to allow you to adjust how quickly you want the buttons to read in case you want to debounce the output.

```python
while True:
    buttons = ss.buttons

    if buttons.right:
        print("Button RIGHT!")

    if buttons.down:
        print("Button DOWN!")

    if buttons.left:
        print("Button LEFT!")

    if buttons.up:
        print("Button UP!")

    if buttons.select:
        print("Button SELECT!")

    if buttons.a:
        print("Button A!")

    if buttons.b:
        print("Button B!")

    if buttons.c:
        print("Button C!")

    time.sleep(.001)
```

Now go ahead and run the code. Once it's running, try pushing a few buttons and see what happens.
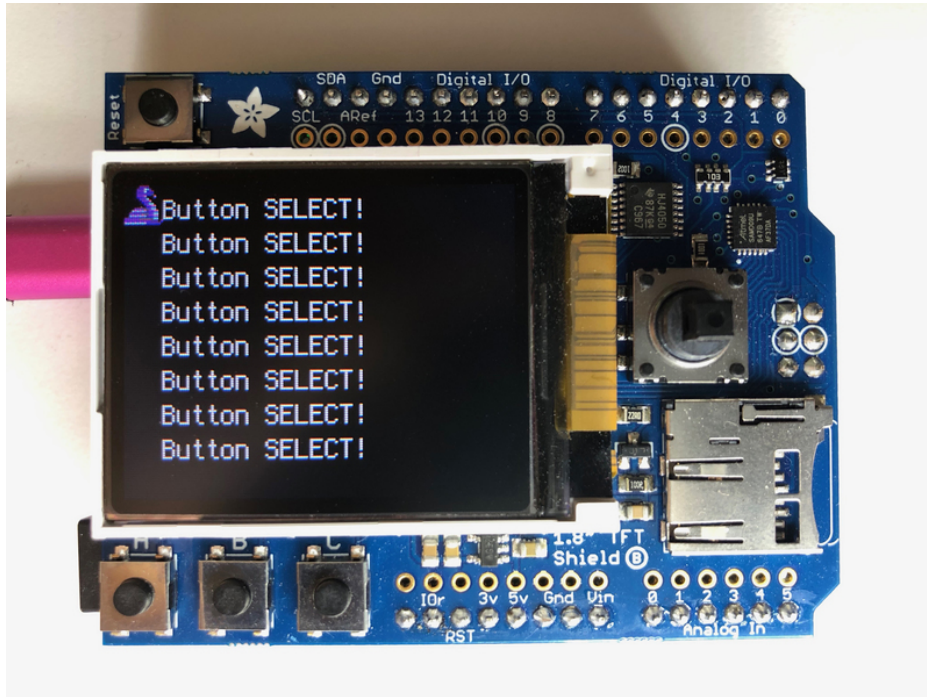
Where to go from here

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

Original V1 Shield

Original V1.0 Shield

If your shield looks like this, you have the original 1.8" TFT shield which does not have a helper seesaw chip

The shield uses the "Classic Arduino" SPI wiring and will perform best with Atmega 328-based Arduinos such as the Uno. It can work with other Arduinos but not very well.

Load up the **shieldtest** demo
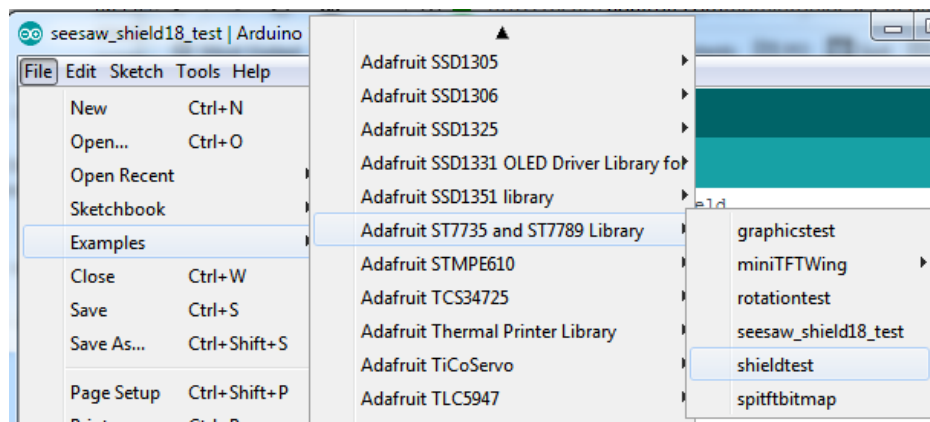
If you are using an Arduino UNO, Duemilanove or compatible with the ATmega328 chipset, you don't have to do anything! If you're using a Mega, Leonardo, Due or other non-ATmega328 chipset, you'll have to make a modification

To use with the shield, modify the example code pin definitions as follows.

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);
```

> This is only required for the V1 shield, the V2 shield uses the hardware SPI port so it's not necessary to use software SPI and in fact it won't work!

The Example code has 2 options for defining the display object. **Uno, Duemilanove** and other Atmega 328-based processors can use the "Option 1" version of the constructor for best performance:

```
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS,  TFT_DC, TFT_RST);
```

Mega and **Leonardo** users should use the "Option 2" version of the constructor for compatibility:

```
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK, TFT_RST);
```
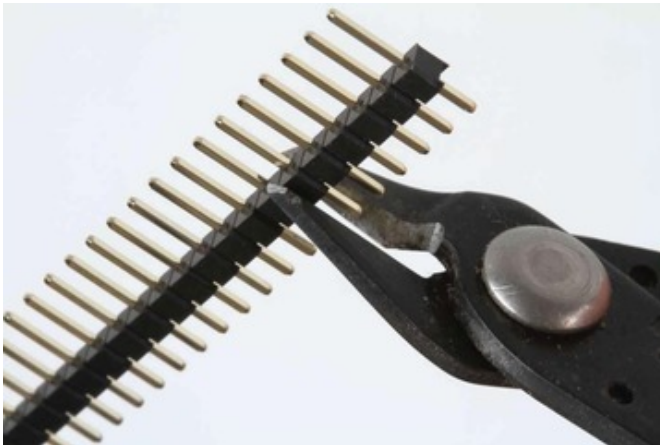
Be sure to select only one option and comment out the other with a pair of //'s.

Now upload the sketch to see the graphical display!

Assembling the Shield

The shield comes with all surface mount parts pre-soldered. All that remains is to install the headers!

Cut the Header Sections

Cut the breakaway header strip into sections to fit the holes on the edge of the shield. You will need 2 sections of 6-pins and 2 sections of 8 pins.

You can use wire-cutters as shown or pliers to snap them apart between pins.
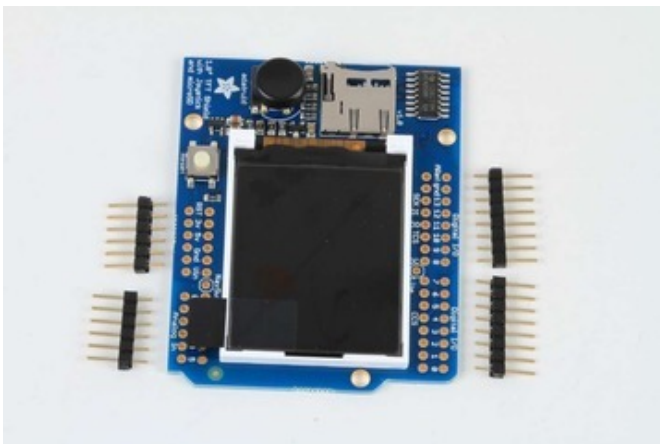
Insert the Headers into an Arduino

To align the header strips for soldering, insert them (long pins down) into the headers of an Arduino.

Note that for R3 and later Arduinos, there will be an extra 2 unused pins on the end closest the USB and DC power jacks.
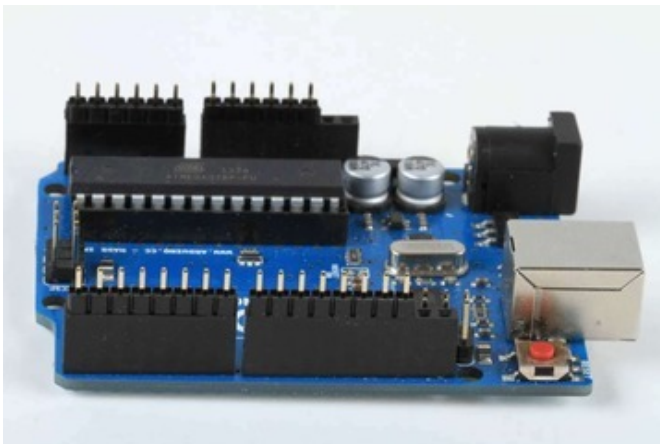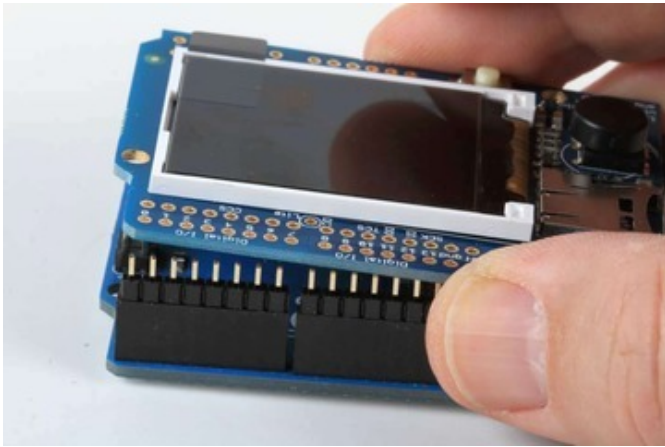
Add the Shield

Place the shield over the header strips so that the short pins stick up through the holes.
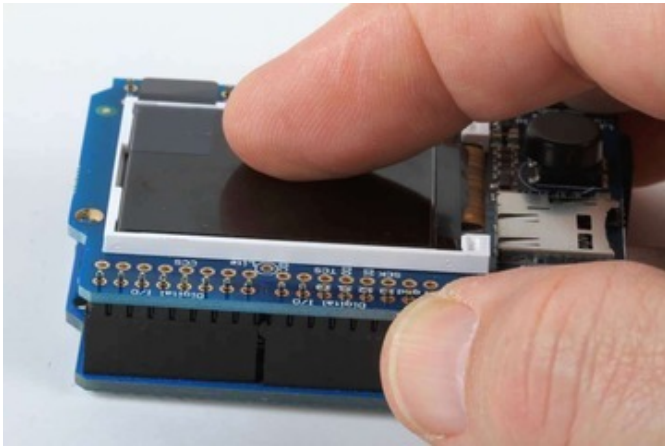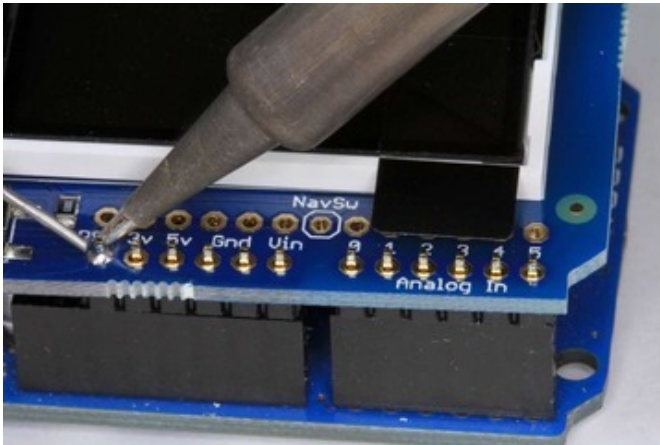
## And Solder!
Solder each pin to assure good electrical contact.

For tips on soldering see the Adafruit Guide to Excellent Soldering (https://adafru.it/aTk).
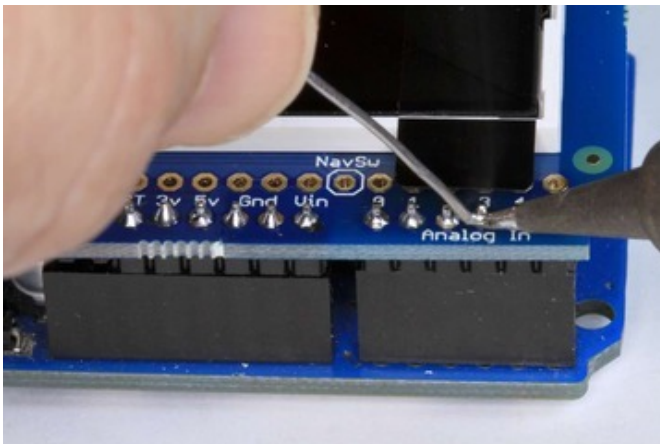
Reading the Joystick
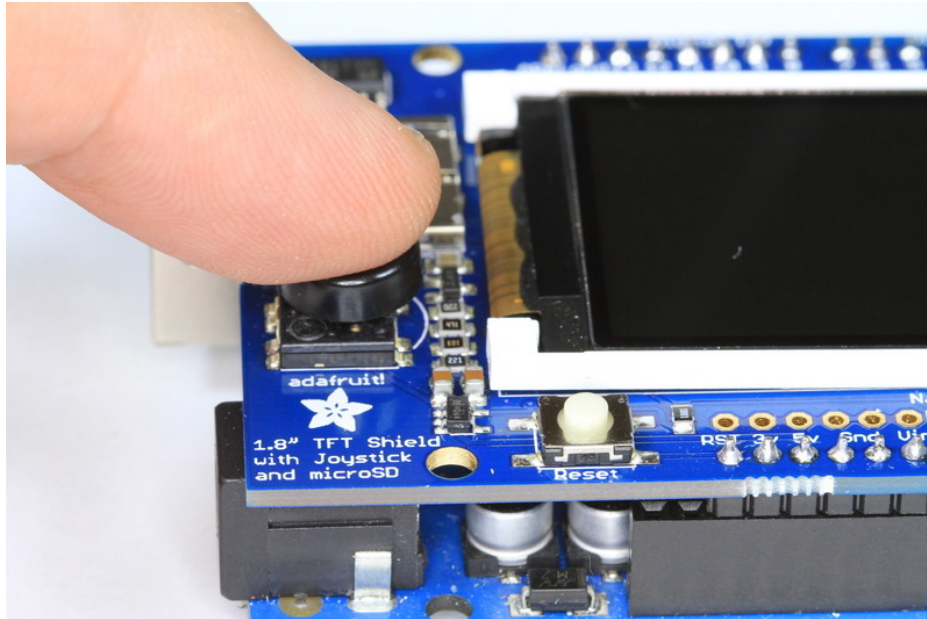
The 5-way joystick on the shield is great for implementing menu navigation or even for use as a tiny game controller. To minimize the number of pins required, the joystick uses a different resistor on each leg of the control to create a variable voltage divider that can be monitored with a single analog pin. Each movement of the joystick control connects a different resistor and results in a different voltage reading.
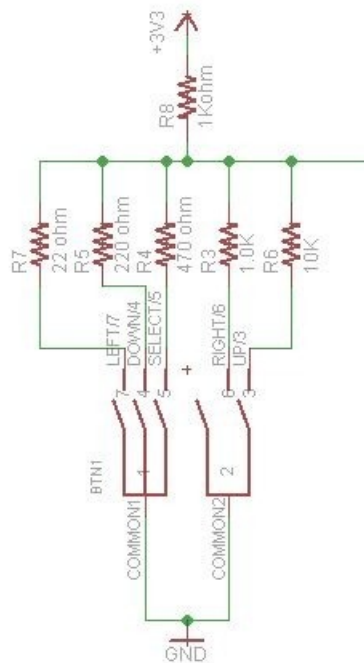
In the code example below, the CheckJoystick() function reads the analog pin and compares the result with 5 different ranges to determine which (if any) direction the stick has been moved. If you upload this to your Arduino and open the Serial Monitor, you will see the current joystick state printed to the screen.

You can use this code as the input method for your menu system or game:

```
void setup()
{
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

#define Neutral 0
#define Press 1
#define Up 2
#define Down 3
#define Right 4
#define Left 5

// Check the joystick position
int CheckJoystick()
{
  int joystickState = analogRead(3);

  if (joystickState < 50) return Left;
  if (joystickState < 150) return Down;
  if (joystickState < 250) return Press;
  if (joystickState < 500) return Right;
  if (joystickState < 650) return Up;
  return Neutral;
}

void loop()
{
  int joy = CheckJoystick();
  switch (joy)
  {
    case Left:
      Serial.println("Left");
      break;
    case Right:
      Serial.println("Right");
      break;
    case Up:
      Serial.println("Up");
      break;
    case Down:
      Serial.println("Down");
      break;
    case Press:
      Serial.println("Press");
      break;
  }
}
```

Graphics Library

We've written a full graphics library specifically for this display which will get you up and running quickly. The code is written in C/C++ for Arduino but is easy to port to any microcontroller by rewritting the low level pin access functions.

The TFT LCD library is based off of the Adafruit GFX graphics core library. GFX has many ready to go functions that should help you start out with your project. It's not exhaustive and we'll try to update it if we find a really useful function. Right now it supports pixels, lines, rectangles, circles, round-rects, triangles and printing text as well as rotation.

Two libraries need to be downloaded and installed: first is the ST7735 library (https://adafru.it/aHm) (this contains the low-level code specific to this device), and second is the Adafruit GFX Library (https://adafru.it/aJa) (which handles graphics operations common to many displays we carry). You can install these with the Arduino library manager.
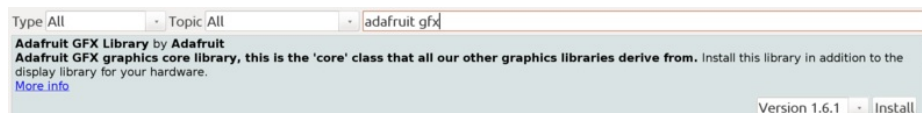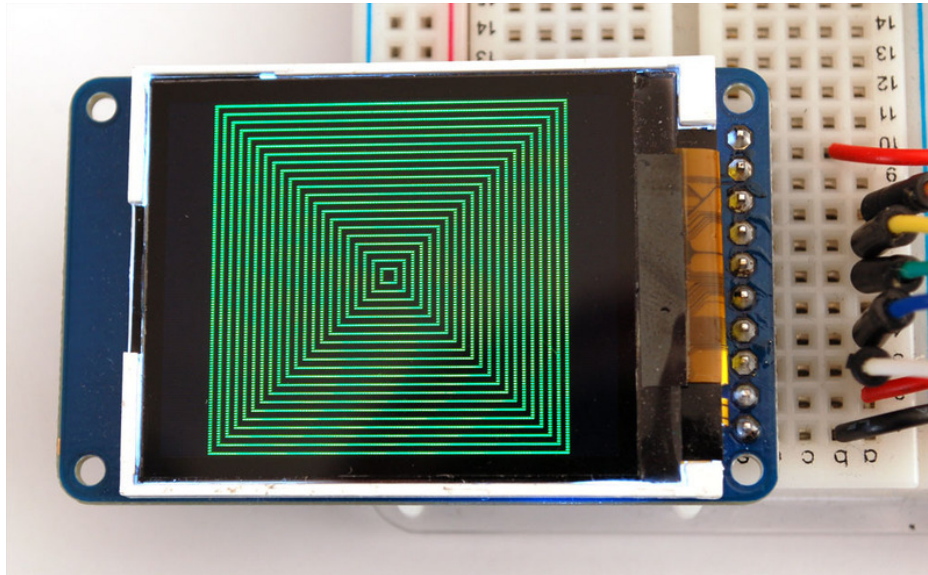
Open up the Arduino library manager:
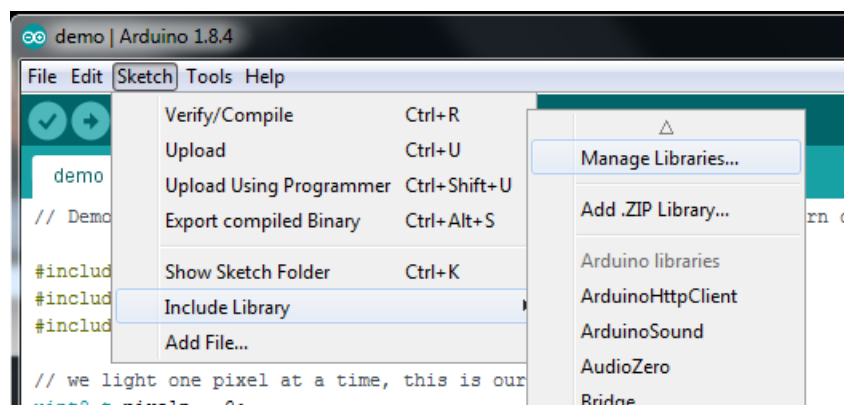
Search for the **Adafruit ST7735** library and install it
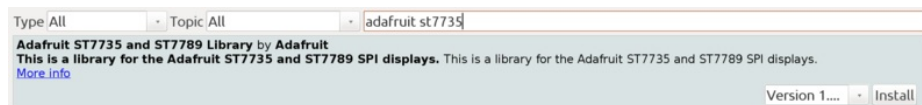
Search for the **Adafruit GFX** library and install it

Check out the GFX tutorial for detailed information about what is supported and how to use it (https://adafru.it/aPx)!

☐ 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

- Adafruit GFX library (https://adafru.it/aJa)
- Adafruit ST7735 library (https://adafru.it/aHm)(See our detailed tutorial for installation assistance (https://adafru.it/aYG))
- Fritzing object in the Adafruit library (https://adafru.it/aP3)
- Datasheet for the display (https://adafru.it/aP8)
- Datasheet for the display driver chip (https://adafru.it/aP9).
- EagleCAD PCB files for TFT shield (https://adafru.it/qxB)
- EagleCAD PCB files for TFT breakout (https://adafru.it/qxC)
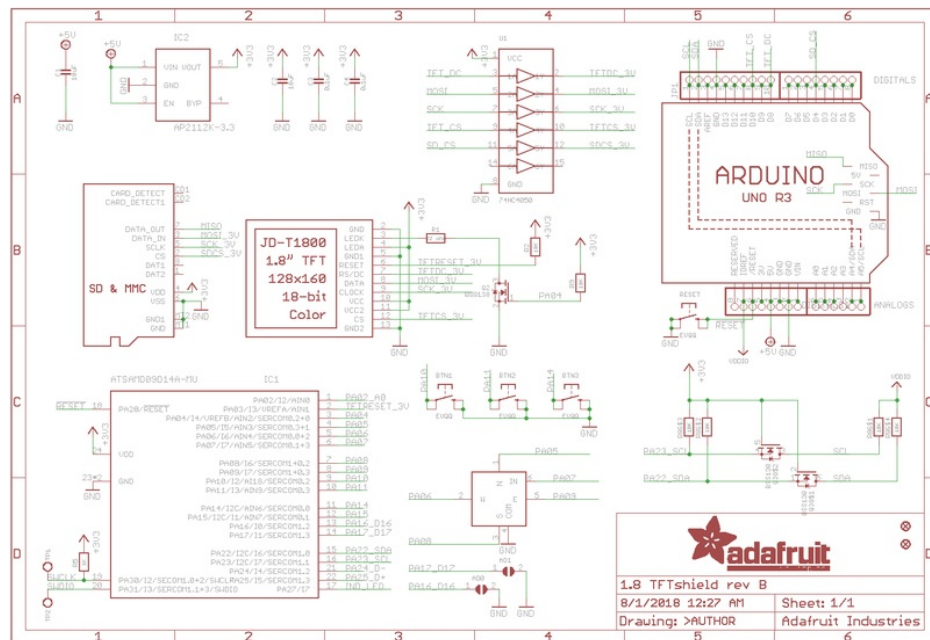
Breakout Schematic
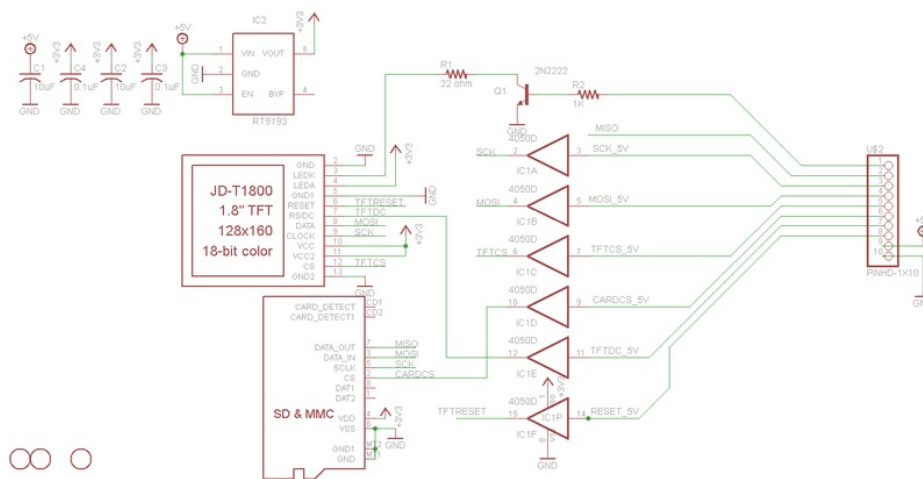
For the level shifter we use the CD74HC4050 which has a typical propagation delay of ~10ns
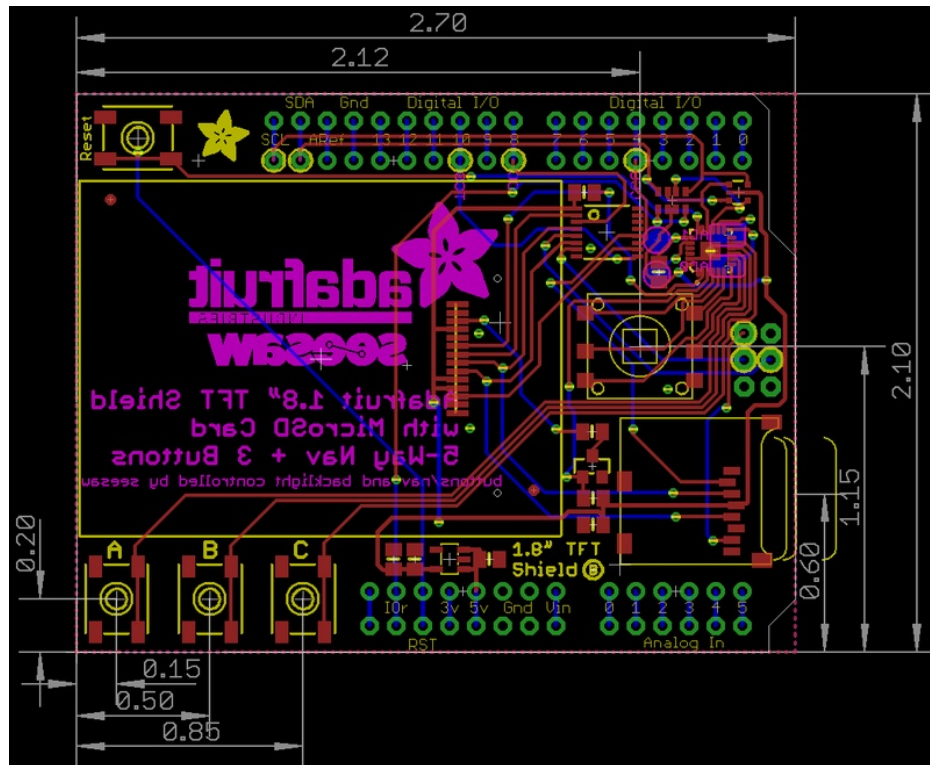
Breakout Fabrication print
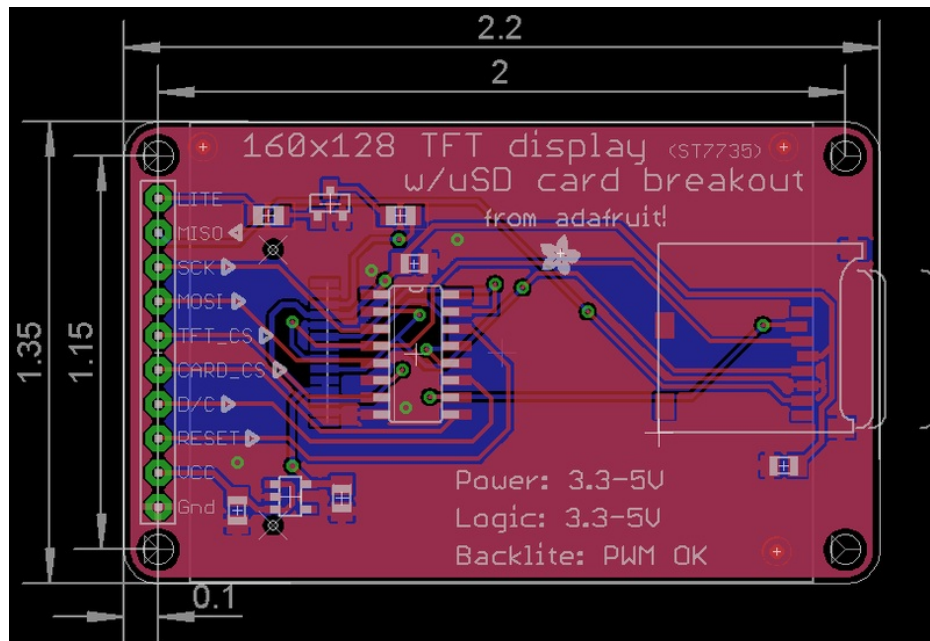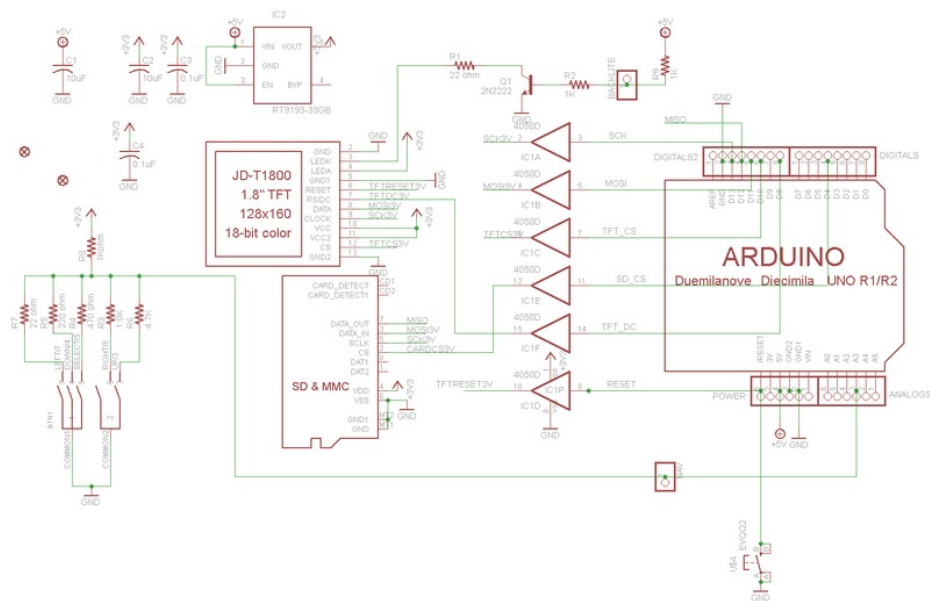
Shield v2 Schematic & Fab Print

This is the newer seesaw version

Shield V1 Schematic & Fab Print

This is the 'original' non-seesaw version

Компания «ЭлектроПласт» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:
- Оперативные поставки широкого спектра электронных компонентов отечественного и импортного производства напрямую от производителей и с крупнейших мировых складов;
- Поставка более 17-ти миллионов наименований электронных компонентов;
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- Лицензия ФСБ на осуществление работ с использованием сведений, составляющих государственную тайну;
- Поставка специализированных компонентов (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Aeroflex, Peregrine, Syfer, Eurofarad, Texas Instrument, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits,General Dynamics и др.);

Помимо этого, одним из направлений компании «ЭлектроПласт» является направление «Источники питания». Мы предлагаем Вам помощь Конструкторского отдела:
- Подбор оптимального решения, техническое обоснование при выборе компонента;
- Подбор аналогов;
- Консультации по применению компонента;
- Поставка образцов и прототипов;
- Техническая поддержка проекта;
- Защита от снятия компонента с производства.

Как с нами связаться

Телефон: 8 (812) 309 58 32 (многоканальный)
Факс: 8 (812) 320-02-42
Электронная почта: org@eplast1.ru
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, дом 2, корпус 4, литера А.