

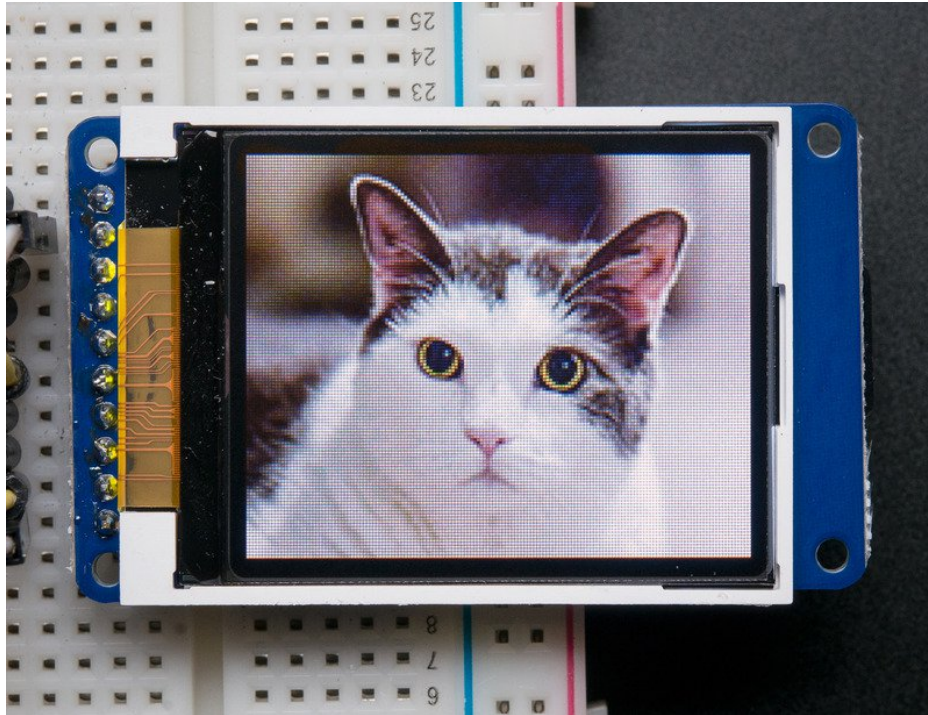
1.8" TFT Display Breakout and Shield

Created by lady ada



Last updated on 2019-08-15 08:42:04 PM UTC

Overview



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.

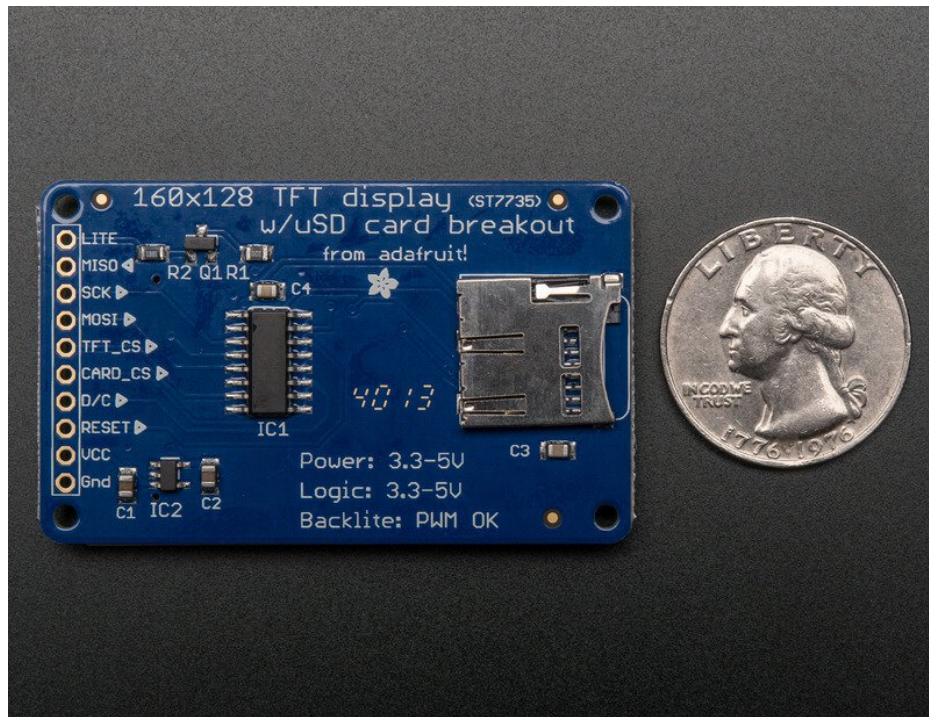
Both boards have the TFT soldered on (it uses a delicate flex-circuit connector) as well as a ultra-low-dropout 3.3V regulator and a 3/5V level shifter so you can use it with 3.3V or 5V power and logic. These also include a microSD card holder so you can easily load full color bitmaps from a FAT16/FAT32 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 \(http://adafru.it/358\)](http://adafru.it/358)

[1.8" 18-bit Color TFT Shield \(http://adafru.it/802\)](http://adafru.it/802)

1.8" TFT Breakout

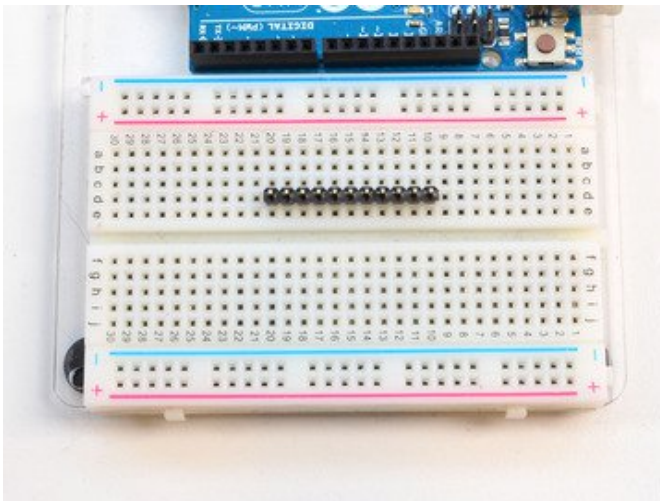
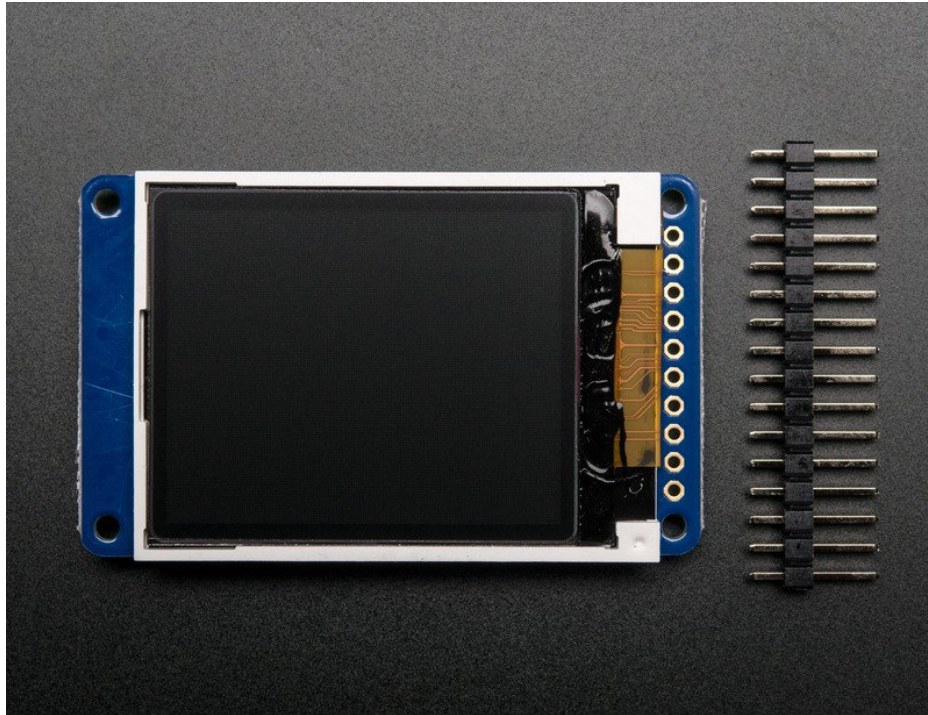


This color display uses SPI to receive image data. That means you need at least 4 pins - clock, data in, tft cs and d/c. If you'd like to have SD card usage too, add another 2 pins - data out and card cs. However, there's a couple other pins you may want to use, lets go thru them all!

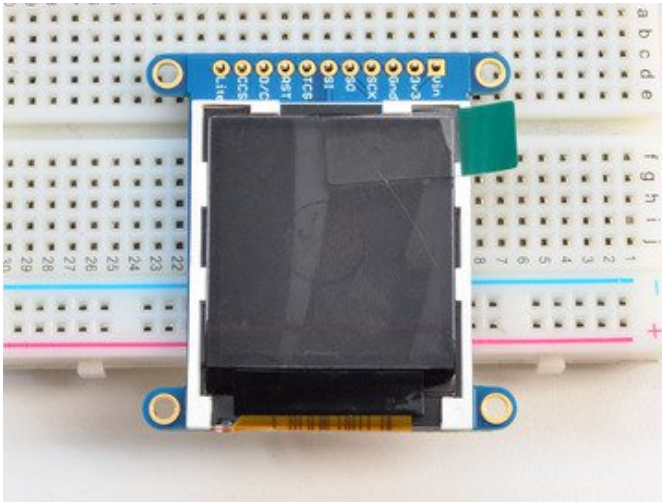
- **Lite** - this is the PWM input for the backlight control. Connect to 3-5VDC to turn on the backlight. Connect to ground to turn it off. Or, you can PWM at any frequency.
- **MISO** - this is the SPI Master In Slave Out pin, its used for the SD card. It isn't used for the TFT display which is write-only
- **SCLK** - this is the SPI clock input pin
- **MOSI** - this is the SPI 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
- **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) (<https://adafru.it/Boj>) which has a typical propagation delay of ~10ns

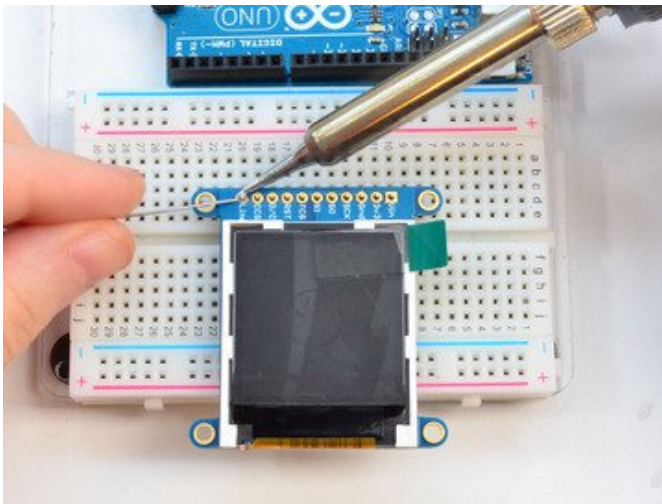
Breakout Assembly



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**

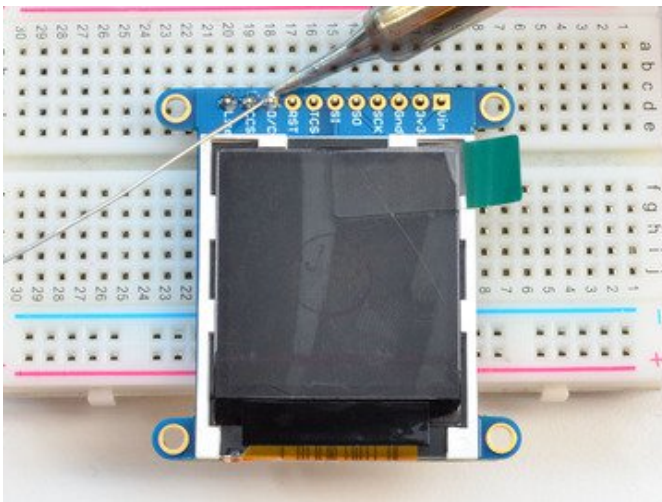


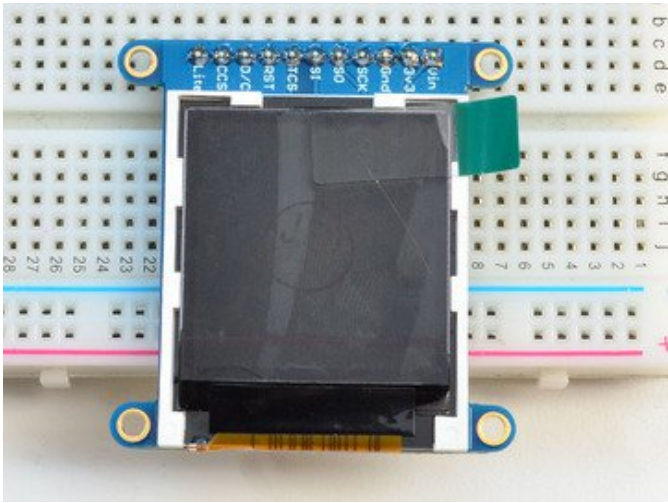
Add the breakout board:
Place the breakout board over the pins so that the short pins poke through the breakout pads



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>)).

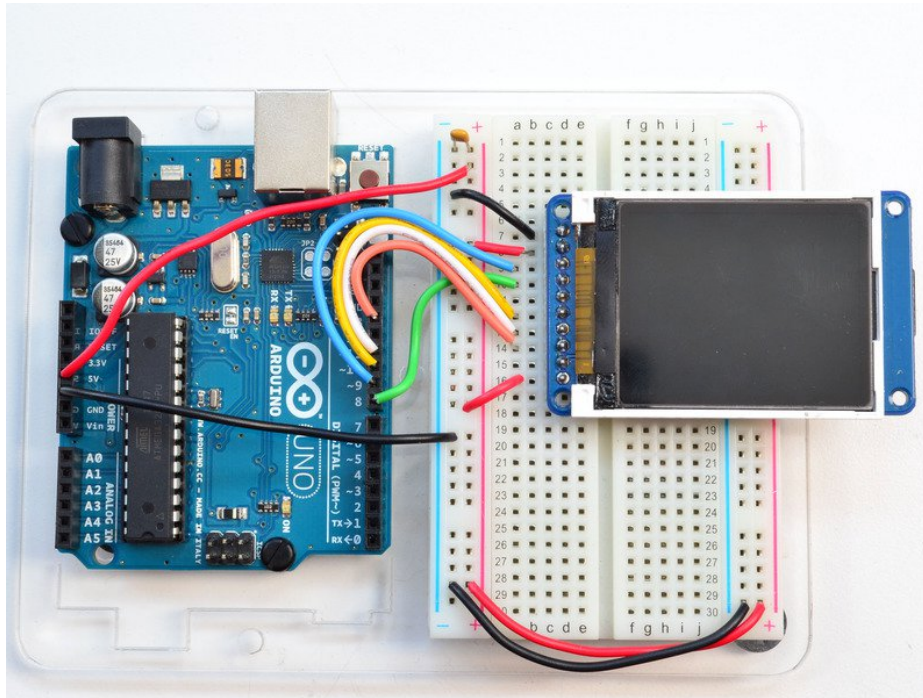




You're done! Check your solder joints visually and continue onto the next steps

Breakout Wiring & Test

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'



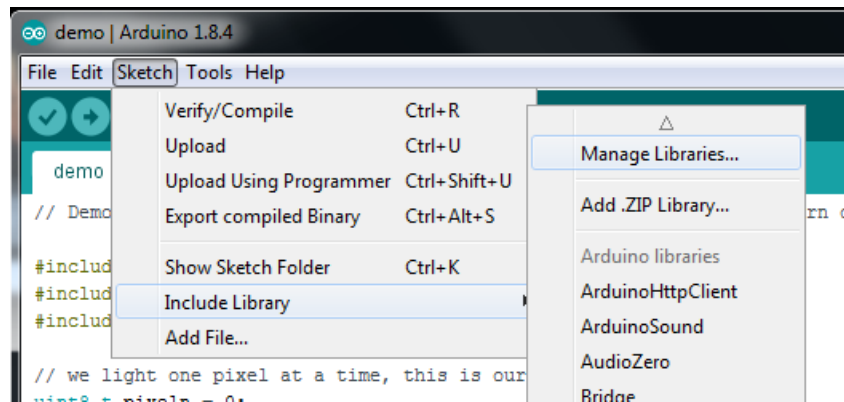
Wiring up the display in SPI mode is pretty easy as there's not that many pins! We'll be using hardware SPI, but you can also use software SPI (any pins) later. Start by connecting the power pins

- **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, that's **Digital 13**. On Mega's, it's **Digital 52** and on Leonardo/Due it's **ICSP-3** (See [SPI Connections for more details \(https://adafru.it/d5h\)](https://adafru.it/d5h)) - this is the orange wire
- **MOSI** connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, that's **Digital 11**. On Mega's, it's **Digital 51** and on Leonardo/Due it's **ICSP-4** (See [SPI Connections for more details \(https://adafru.it/d5h\)](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

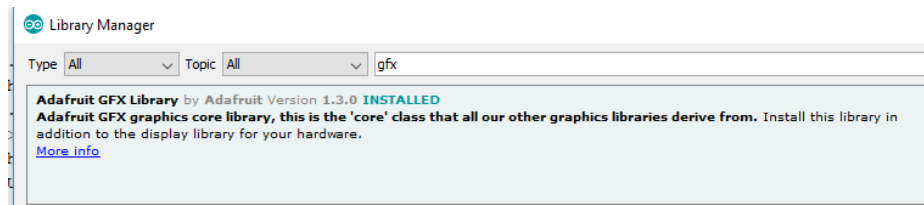
Install Adafruit ST7735 TFT Library

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.

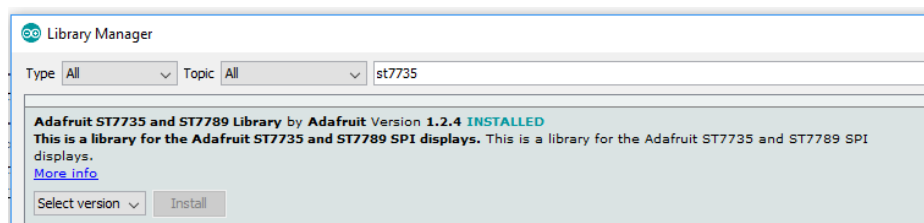
Three libraries need to be installed using the **Arduino Library Manager**...this is the preferred and modern way. From the Arduino “Sketch” menu, select “Include Library” then “Manage Libraries...”



Search for and install the Adafruit GFX library:



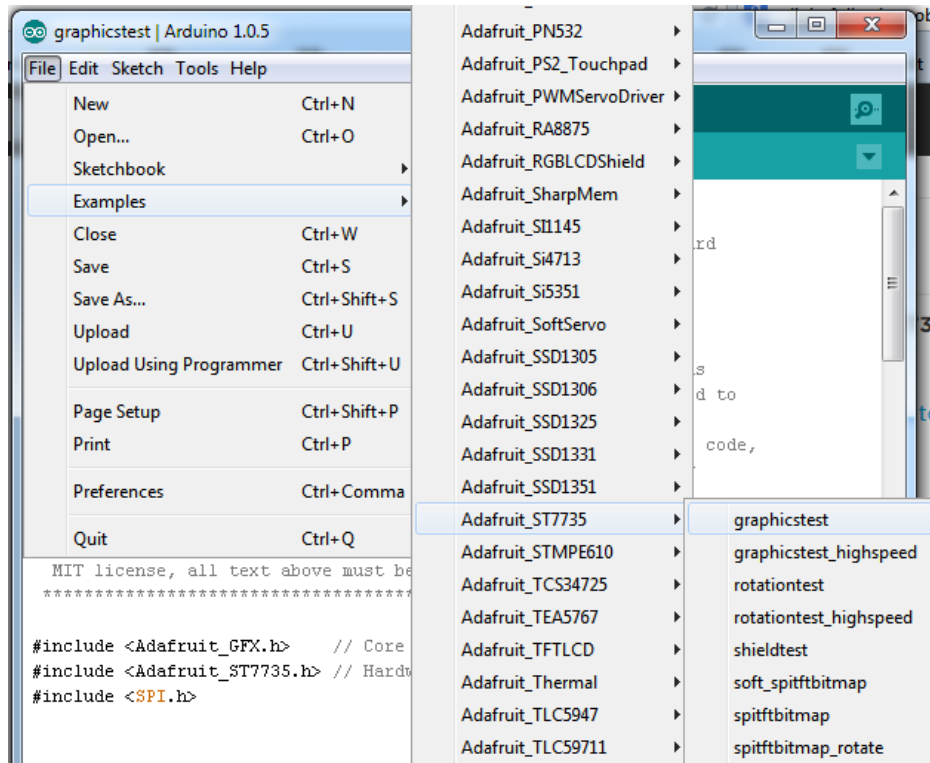
And the Adafruit ST7735 library:



Repeat the search and install steps one more time, looking for **Adafruit_ZeroDMA**. That’s the third and final library.

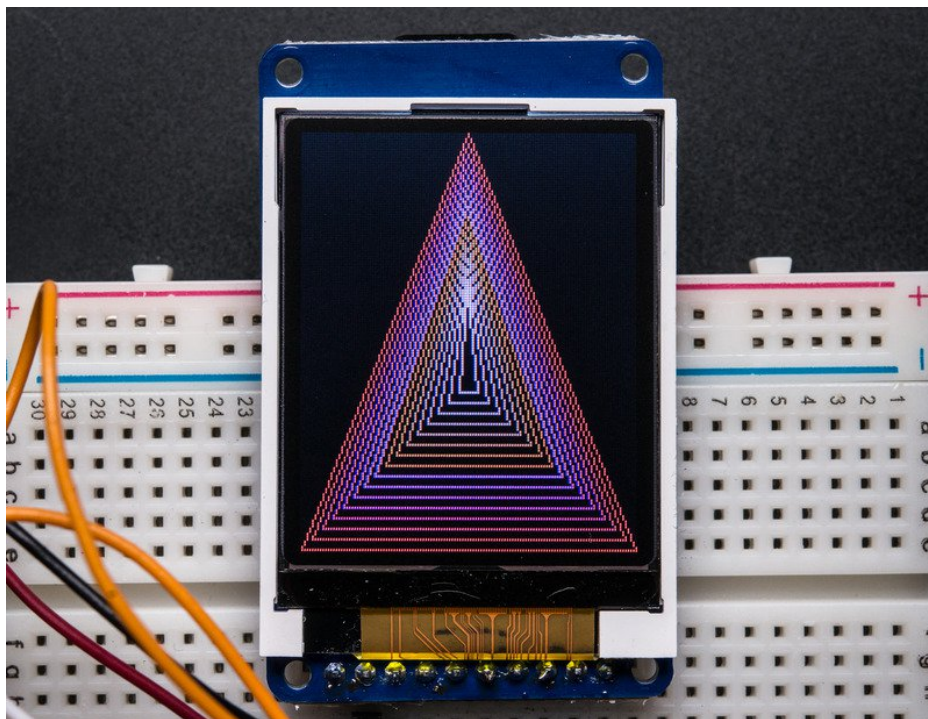
If this is all unfamiliar, we have a [tutorial introducing Arduino library concepts and installation \(https://adafru.it/aYM\)](https://adafru.it/aYM).

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
```

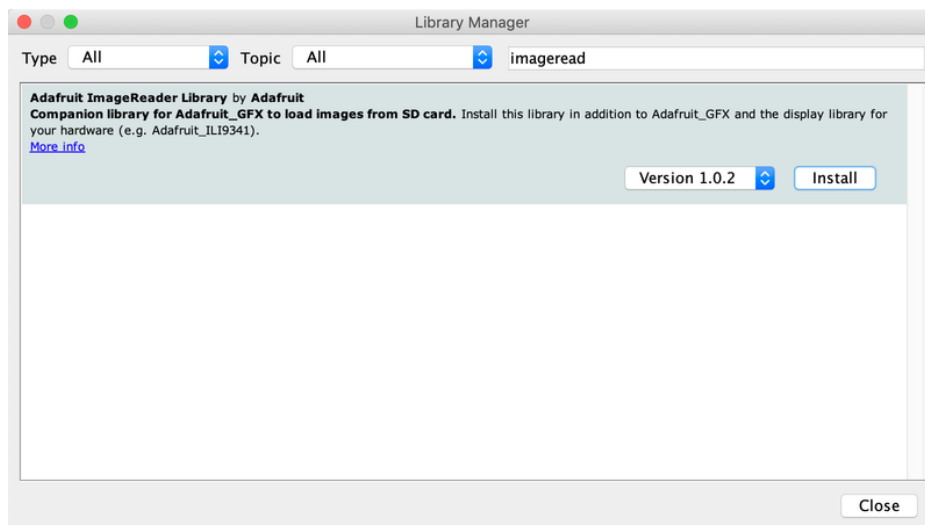
and connecting the RST line to the Arduino Reset pin. That way the Arduino will auto-reset the TFT as well.



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 \(http://adafru.it/102\)](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:



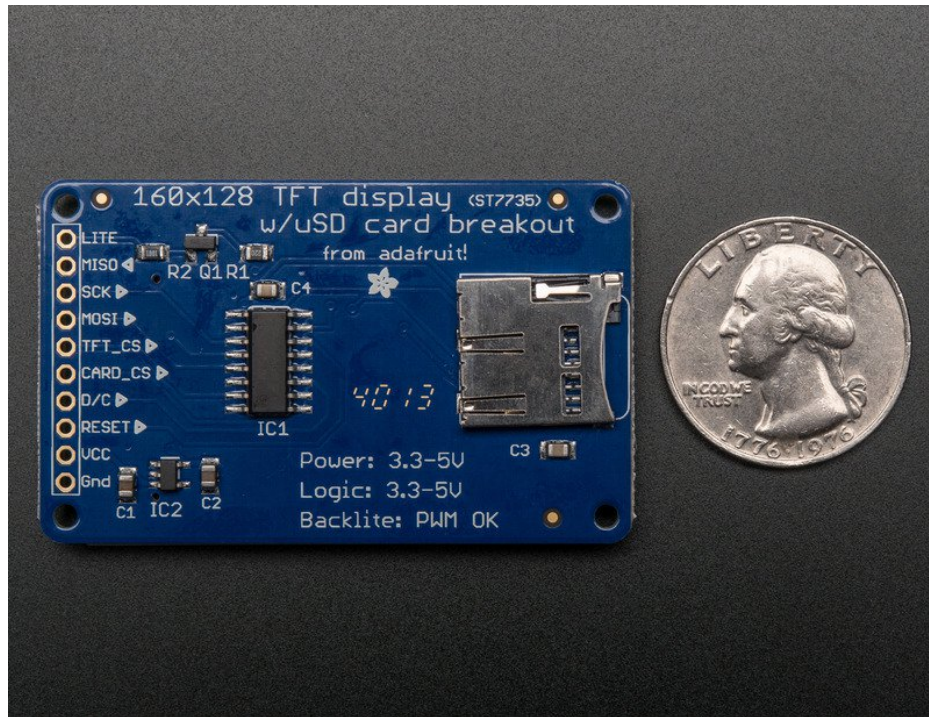
You'll also need an image. We suggest starting with this bitmap of a parrot.

<https://adafru.it/cmn>

<https://adafru.it/cmn>

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!

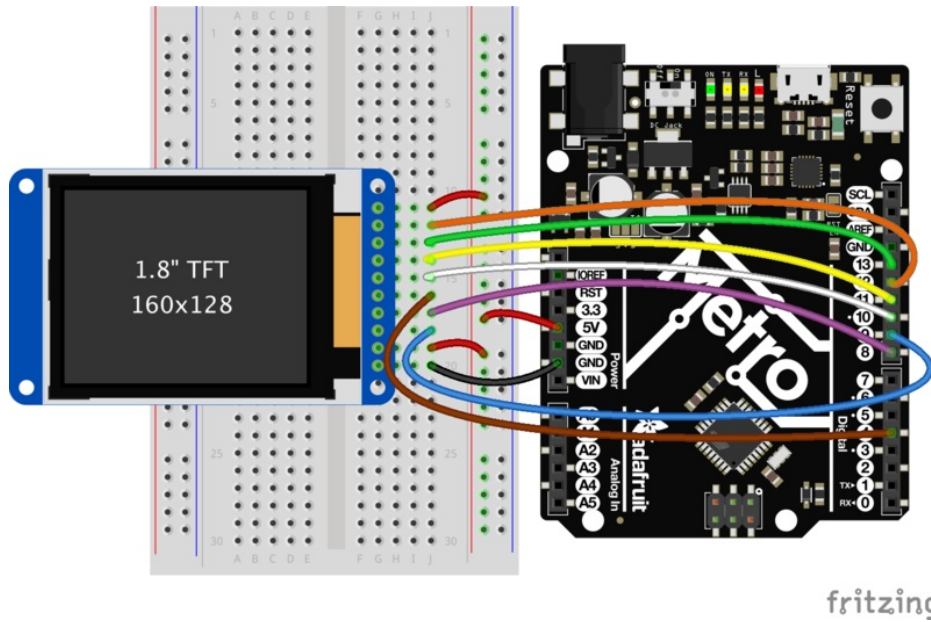
Copy the **parrot.bmp** to the microSD card and insert it into the micro SD card holder on your shield or breakout board.



Breakout Wiring

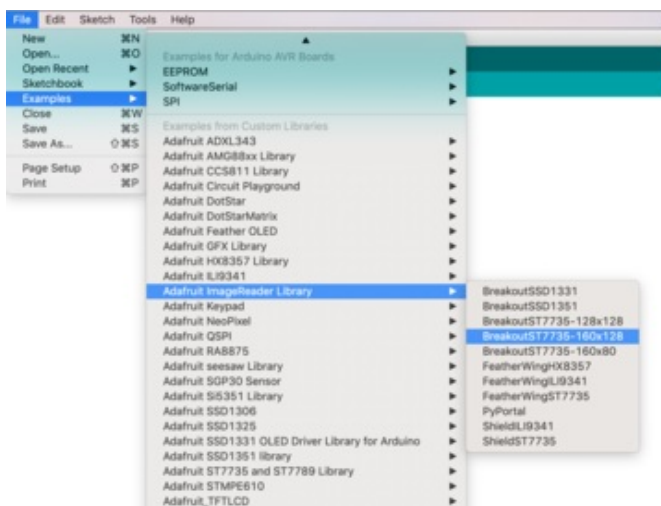
Shield users can skip directly to the "Example Sketch" section.

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.

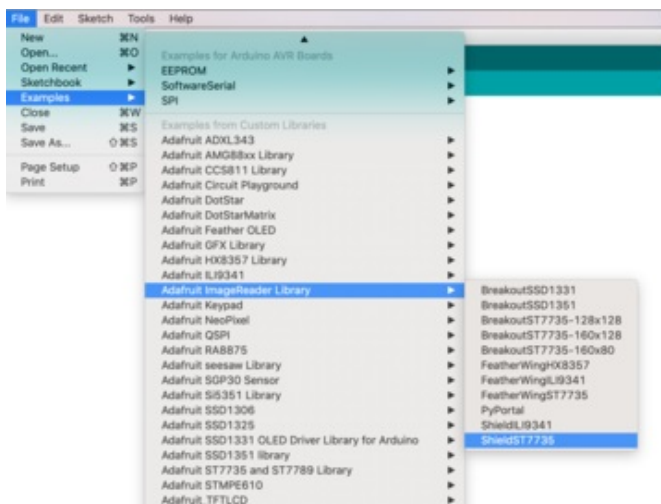


fritzing

Example Sketch



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



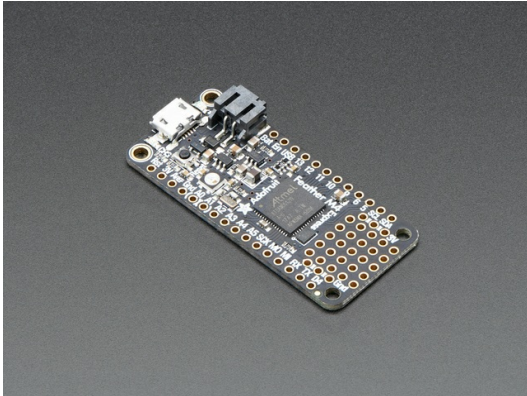
If you have the shield, open the **File**→**examples**→**Adafruit ImageReader Library**→**ShieldST7735** example.

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 \(https://adafru.it/EEem\)](https://adafru.it/EEem) guide.



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

\$22.95
IN STOCK

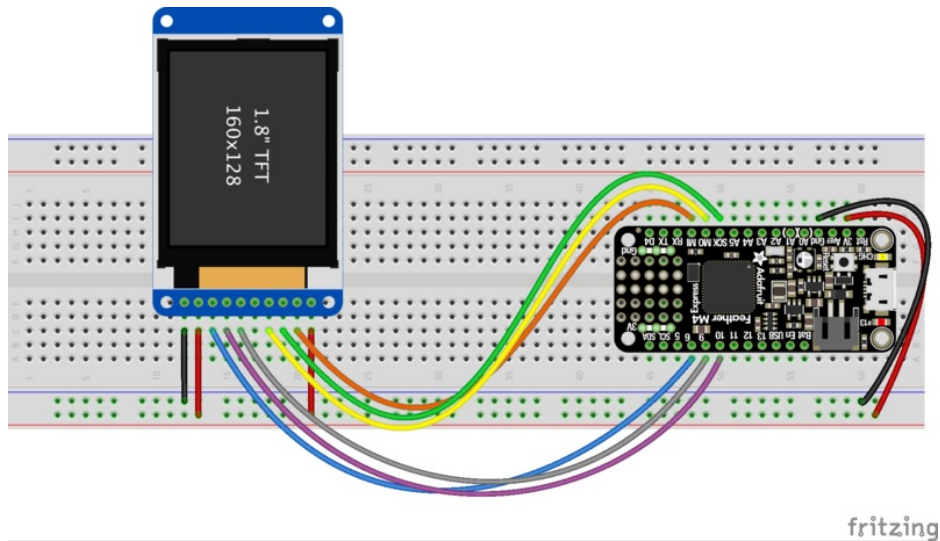
ADD TO CART

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 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 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.
- **RESET** connects to our reset pin. We'll be using **Digital 6** 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.



<https://adafru.it/FyO>
<https://adafru.it/FyO>

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) 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.

<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.

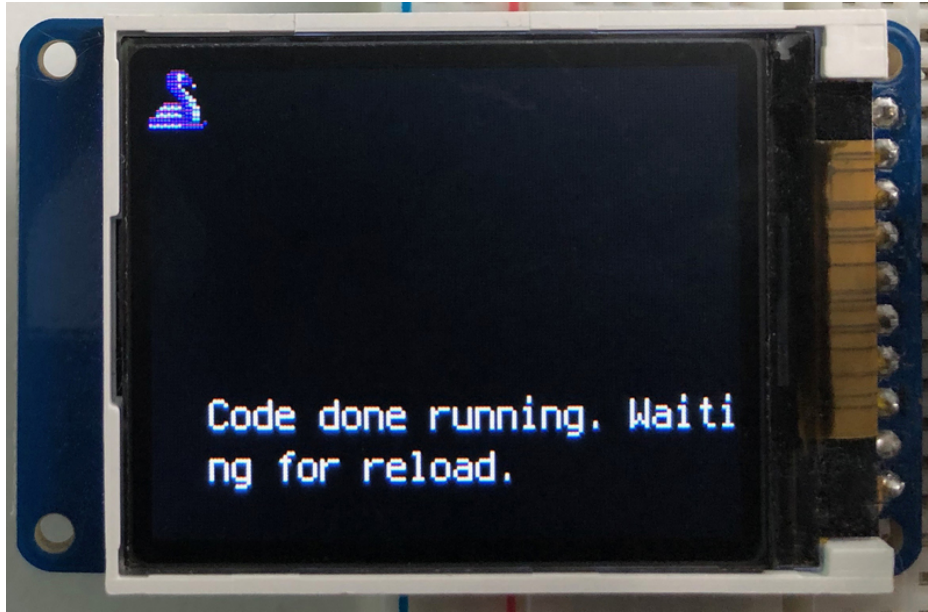
```
spi = board.SPI()
tft_cs = board.D5
tft_dc = board.D6
```

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 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 tells 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 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 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
```

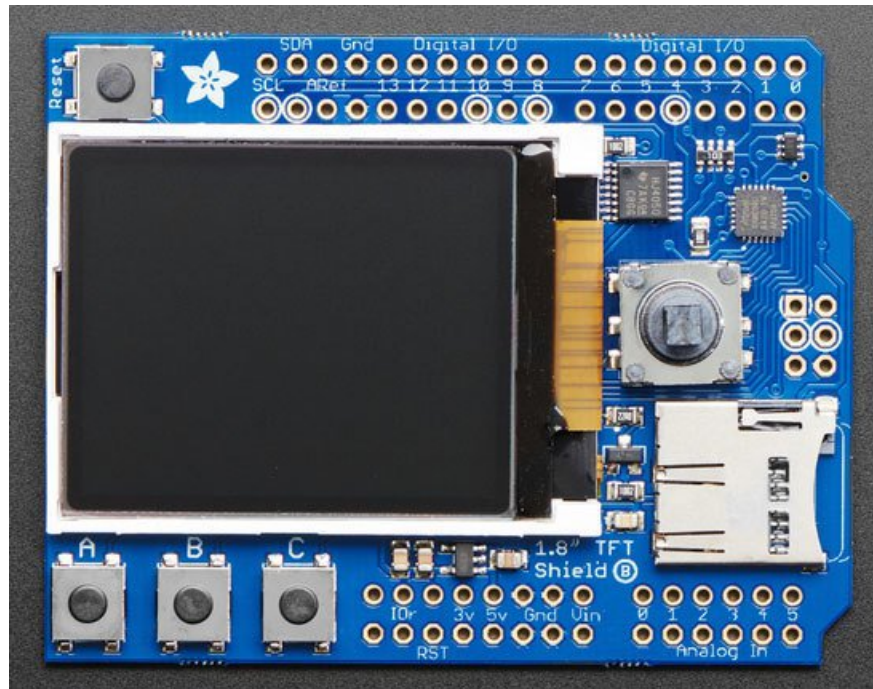


Where to go from here

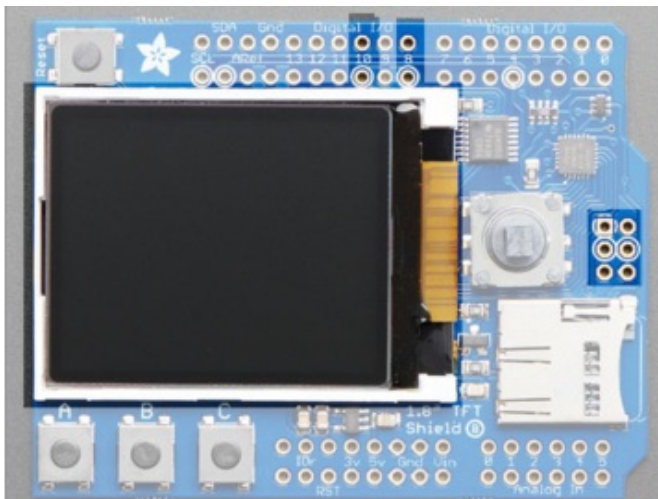
Be sure to check out this excellent [guide to CircuitPython Display Support Using displayio \(https://adafru.it/EGh\)](https://adafru.it/EGh)

1.8" TFT Shield V2

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



TFT Display

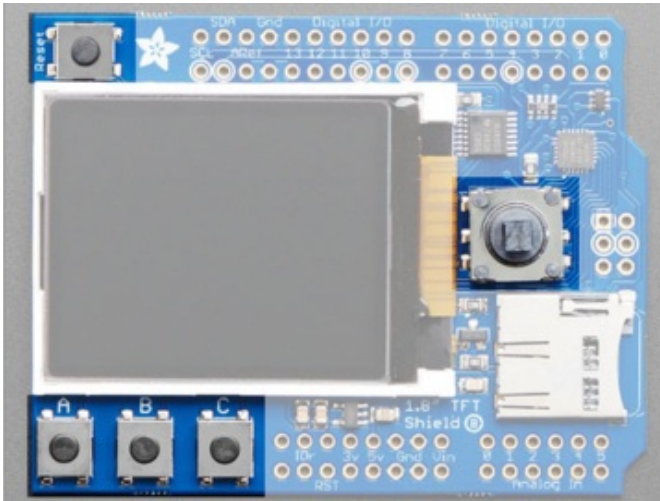


In the center is the 1.8" TFT display. This display is full color (16-bit RGB), 128x160 pixels, and has a backlight. The display receives data over SPI plus two pins:

- 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



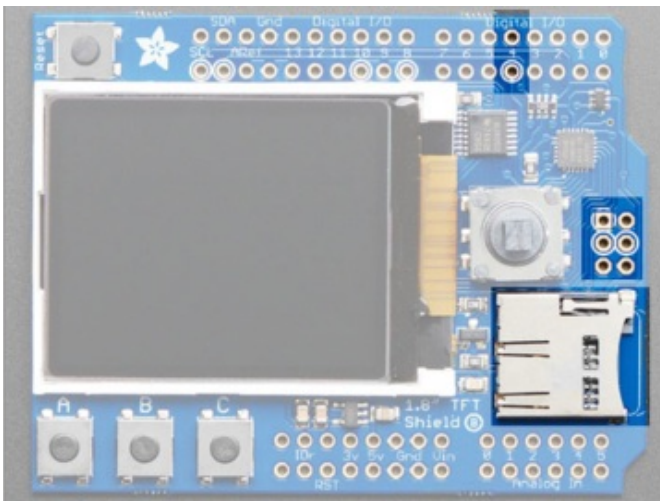
In addition of the display, you also get a bunch of user-interface buttons.

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

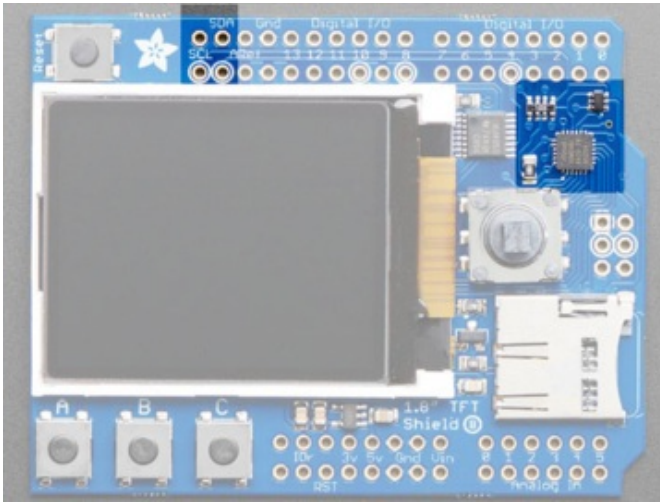
SD Card



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 doesn't conflict

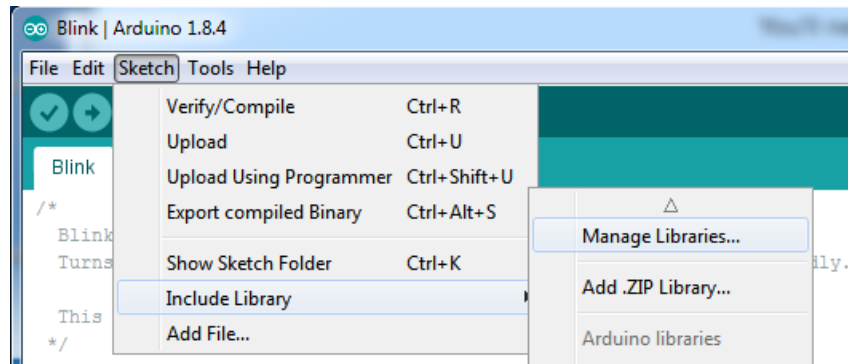
Don't forget! Since the seesaw chip is used for the TFT backlight and reset, **you need to activate it even if you are not reading the buttons or joystick.**

Testing the Shield

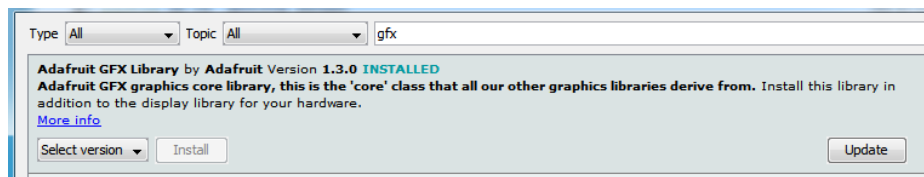
You can test your assembled shield using the example code from the library.

Start by installing a bunch of libraries!

Open the Arduino Library manager



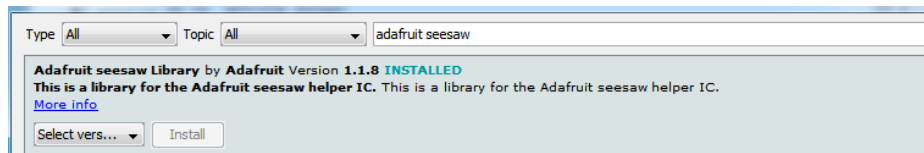
Install the **Adafruit GFX Library**



Adafruit ST7735 Library



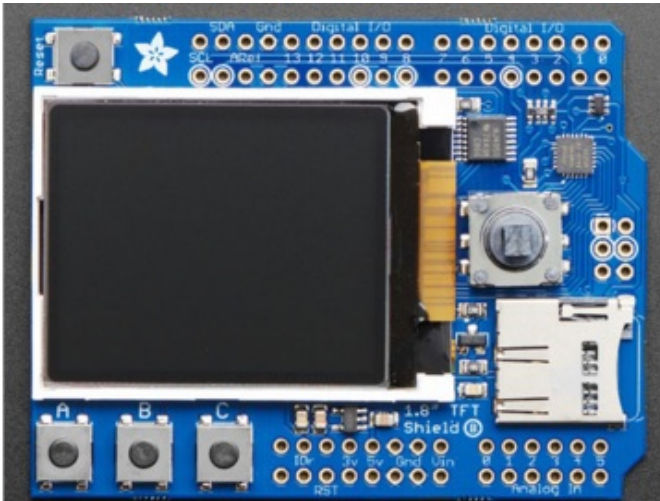
Adafruit seesaw Library



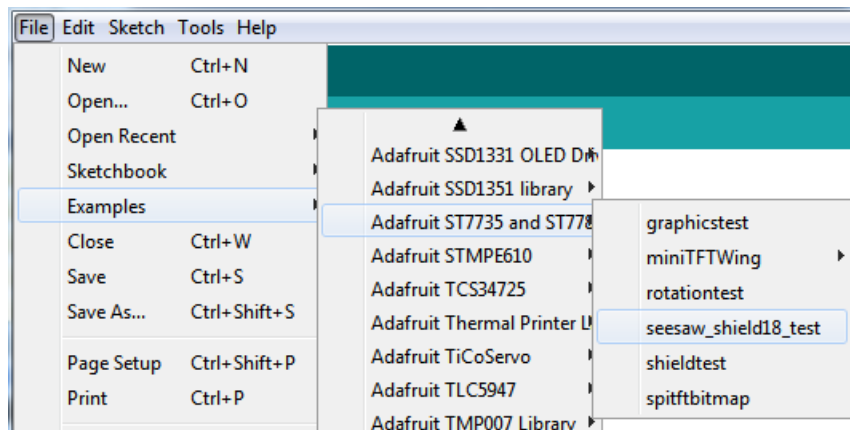
You can read more about installing libraries in our tutorial(<https://adafru.it/aYG>).

Restart the Arduino IDE.

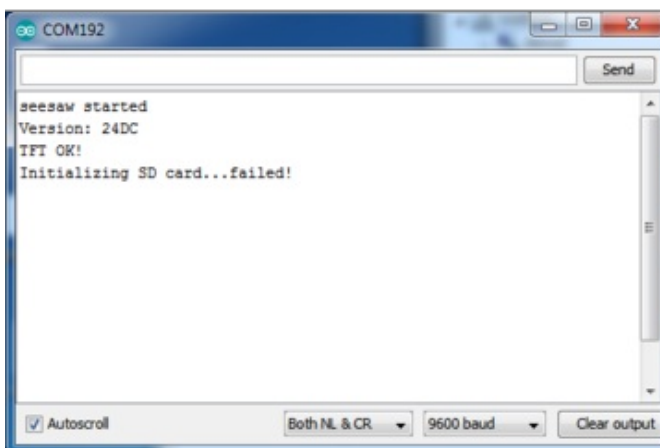
1.8" Shield with seesaw



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

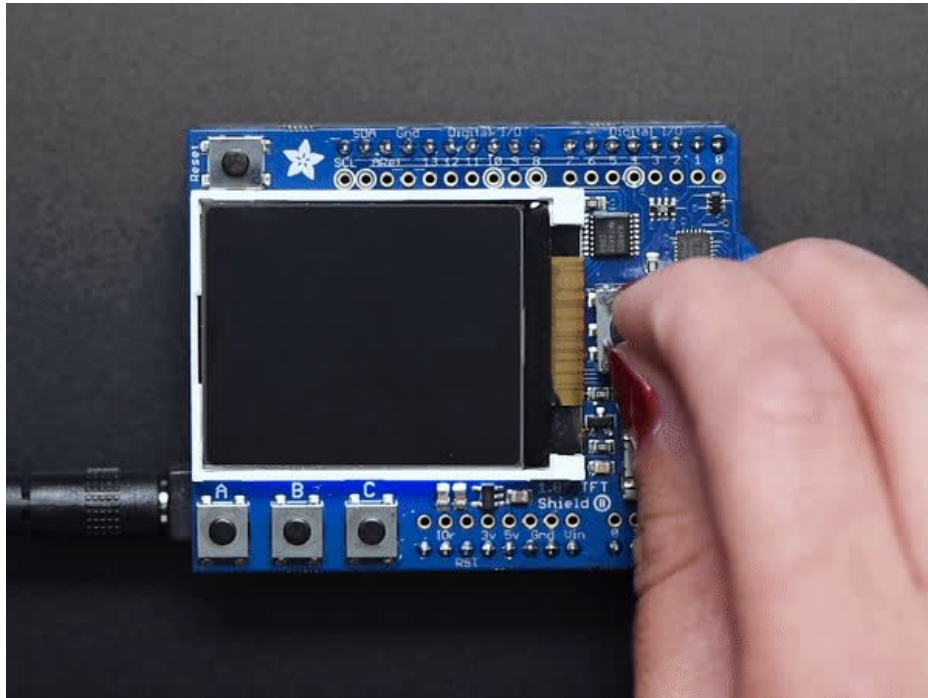
Check that the seesaw chip is detected, you should see text display on the TFT after a quick draw test.

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.



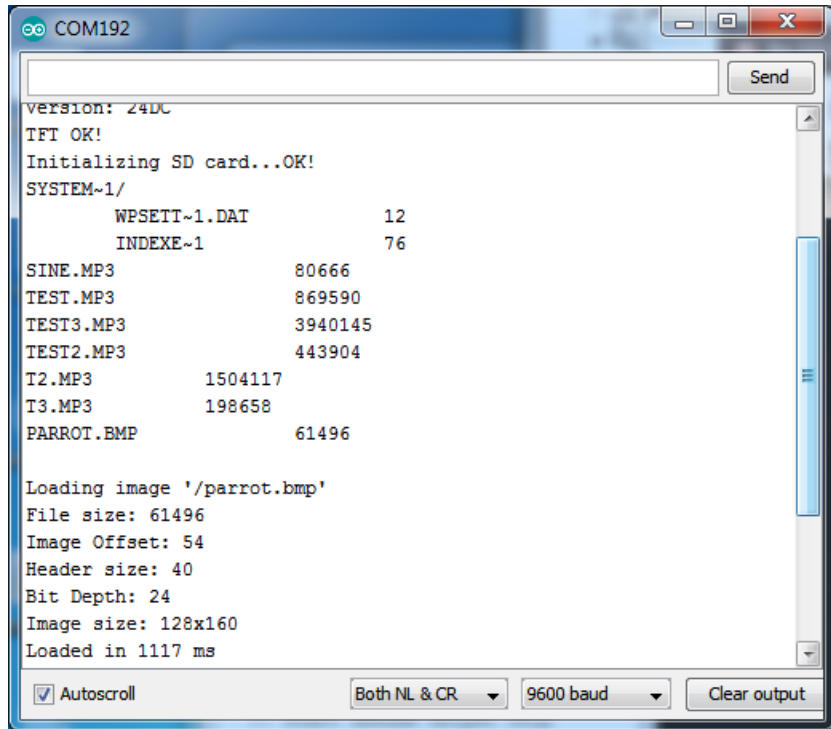
The graphics don't look identical to the below but you should still press all the buttons as shown!



For more details about seesaw, check out our guide (<https://adafruit.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

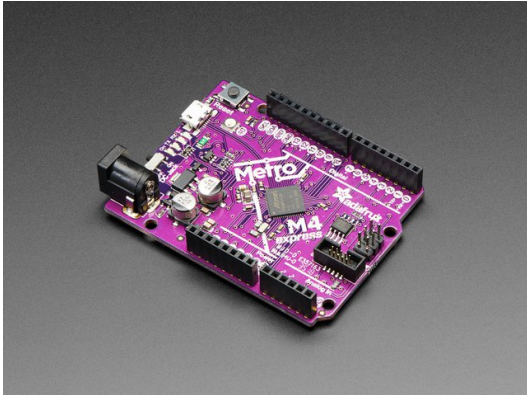
Displaying a Bitmap

If you have [parrot.bmp](https://adafruit.it/cm1) (<https://adafruit.it/cm1>) stored on the SD card you will get a nice parrot display once the buttons have all been pressed



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 \(https://adafru.it/Fkt\)](https://adafru.it/Fkt) guide.

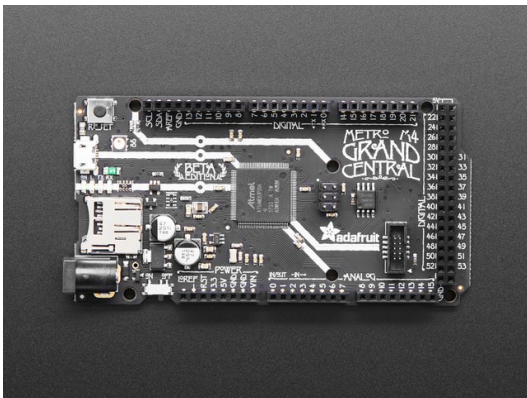


[Adafruit Metro M4 feat. Microchip ATSAMD51](#)

\$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 \(https://adafru.it/DK7\)](https://adafru.it/DK7) guide.

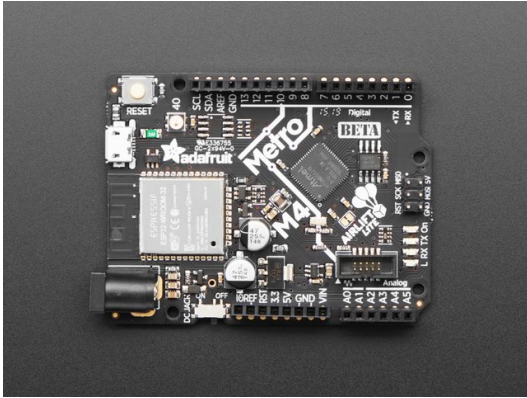


[Adafruit Grand Central M4 Express featuring the SAMD51](#)

\$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 \(https://adafru.it/EZh\)](https://adafru.it/EZh) guide.



Adafruit Metro M4 Express AirLift (WiFi) - Lite

OUT OF STOCK

OUT OF STOCK

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 \(https://adafru.it/drl\)](https://adafru.it/drl). 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 \(https://adafru.it/Amd\)](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`
- `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

ss = TFTShield18()

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

displayio.release_displays()
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(.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
```

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
```

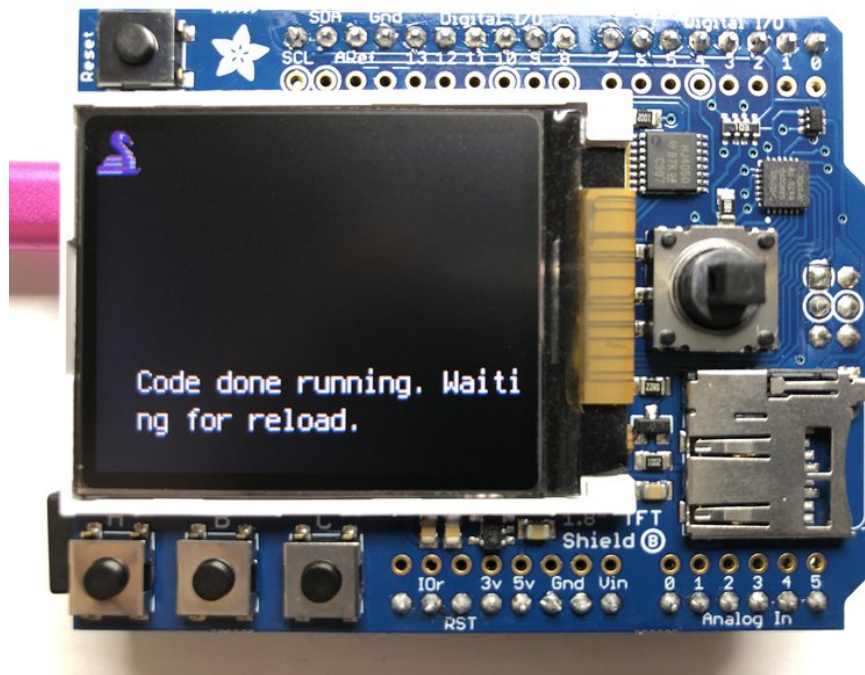
In the next two lines, we release the displays. This is important because if the Metro is reset, the display pins are not automatically released and this makes them available for use again. We set the display bus to FourWire which makes use of the SPI bus.

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

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.

```
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!")

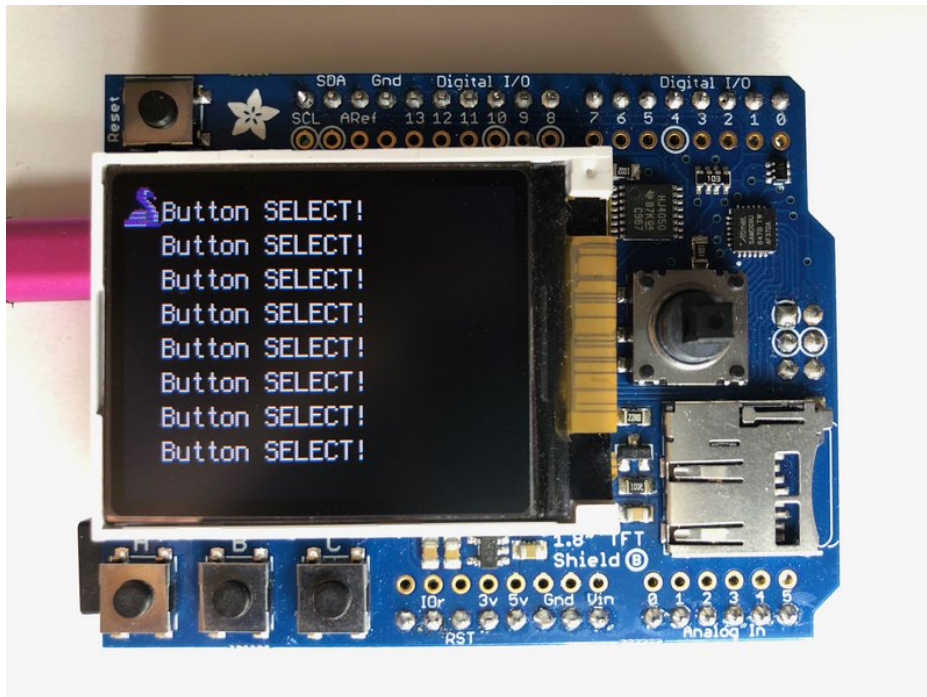
    if buttons.a:
        print("Button A!")

    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 \(https://adafruit.it/EGh\)](https://adafruit.it/EGh)

Original V1 Shield

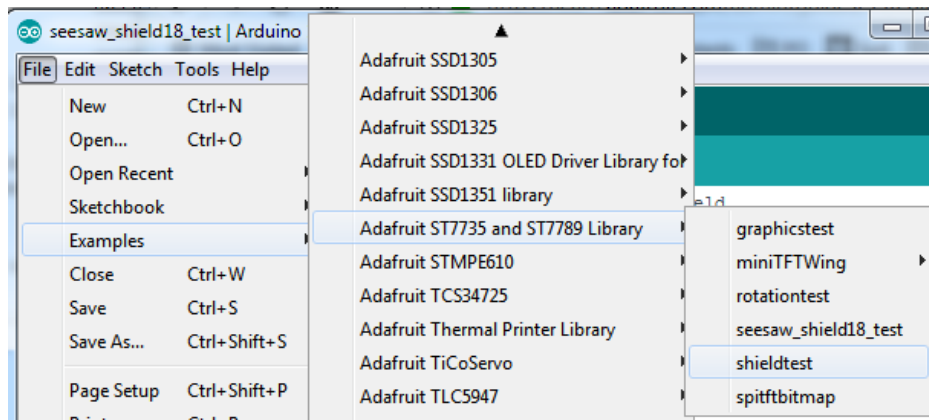
Original V1.0 Shield



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);
```

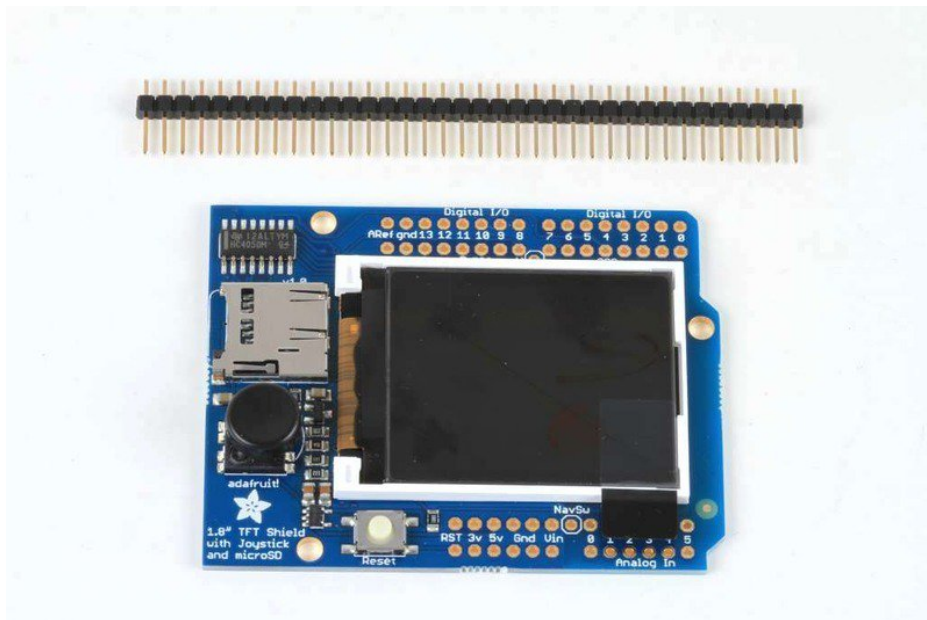
Mega and **Leonardo** users should use the "Option 2" version of the constructor for compatibility:

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

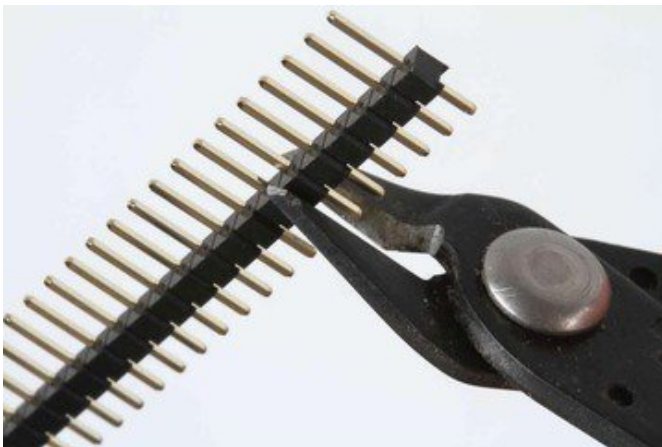
Be sure to select only one option and comment out the other with a pair of //s.

Now upload the sketch to see the graphical display!

Assembling the Shield



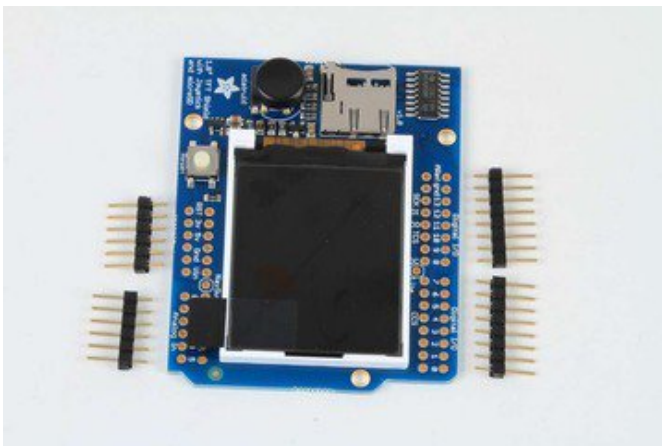
The shield comes with all surface mount parts pre-soldered. All that remains is to install the headers!

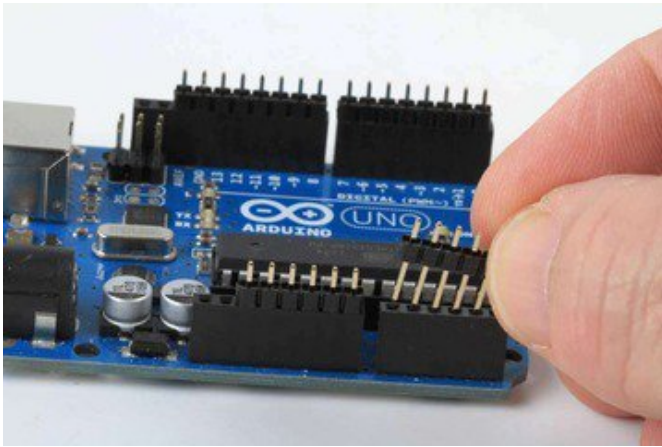


Cut the Header Sections

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.

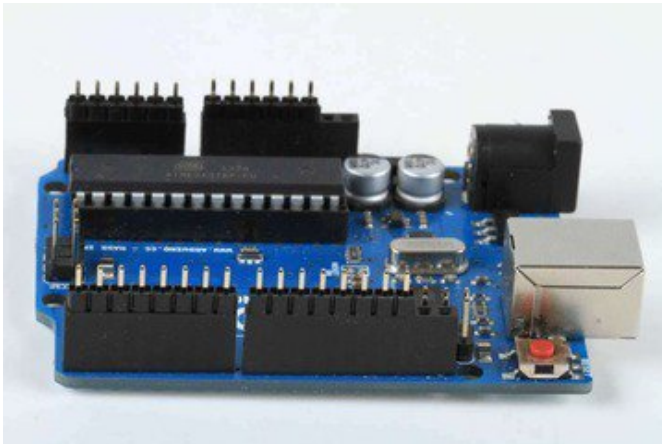
You can use wire-cutters as shown or pliers to snap them apart between pins.

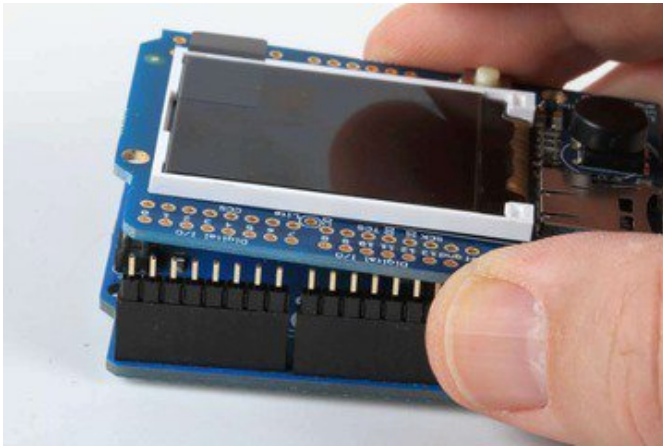




Insert the Headers into an Arduino
To align the header strips for soldering, insert them (long pins down) into the headers of an Arduino.

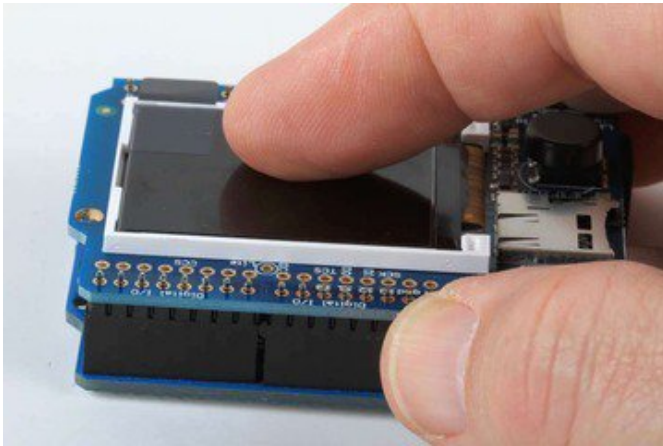
Note that for R3 and later Arduinos, there will be an extra 2 unused pins on the end closest the USB and DC power jacks.

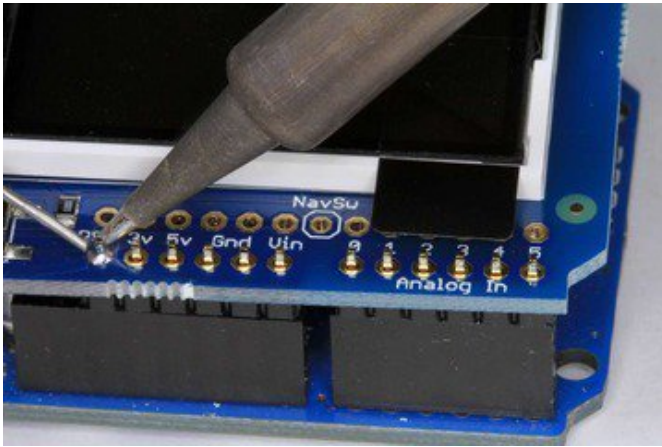




Add the Shield

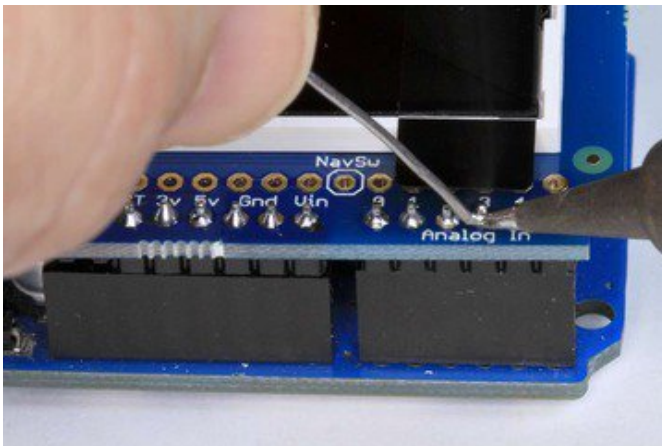
Place the shield over the header strips so that the short pins stick up through the holes.



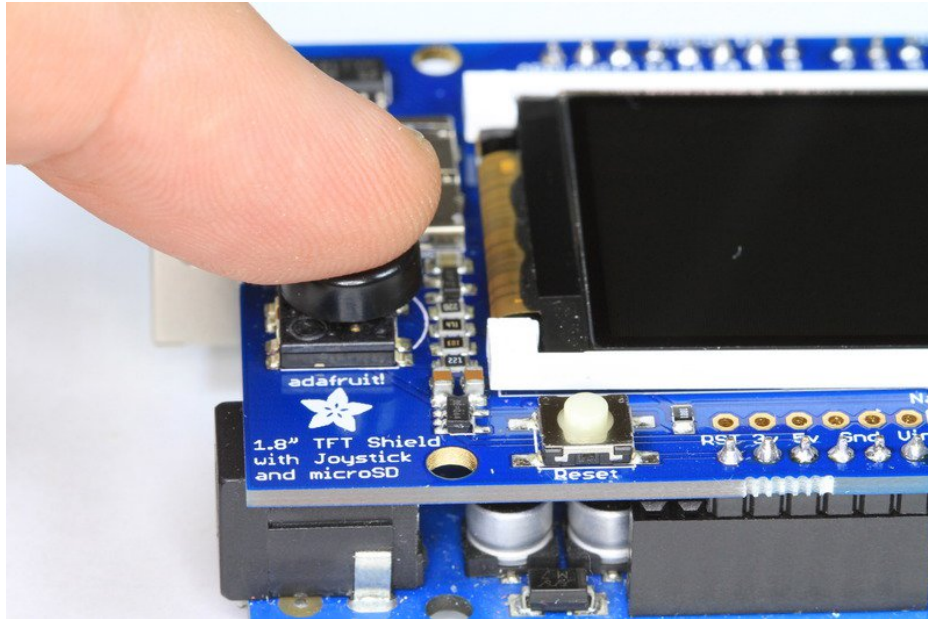


And Solder!
Solder each pin to assure good electrical contact.

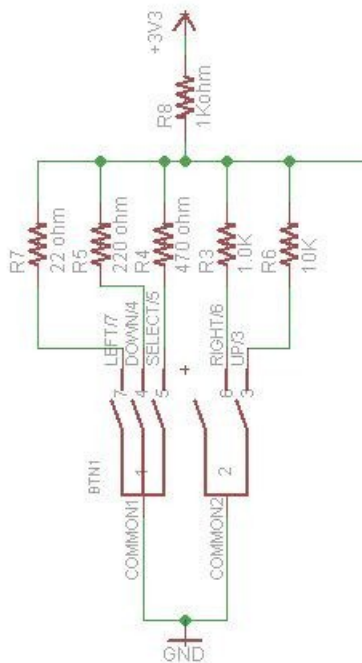
For tips on soldering see the [Adafruit Guide to Excellent Soldering \(https://adafru.it/aTk\)](https://adafru.it/aTk).



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;
}

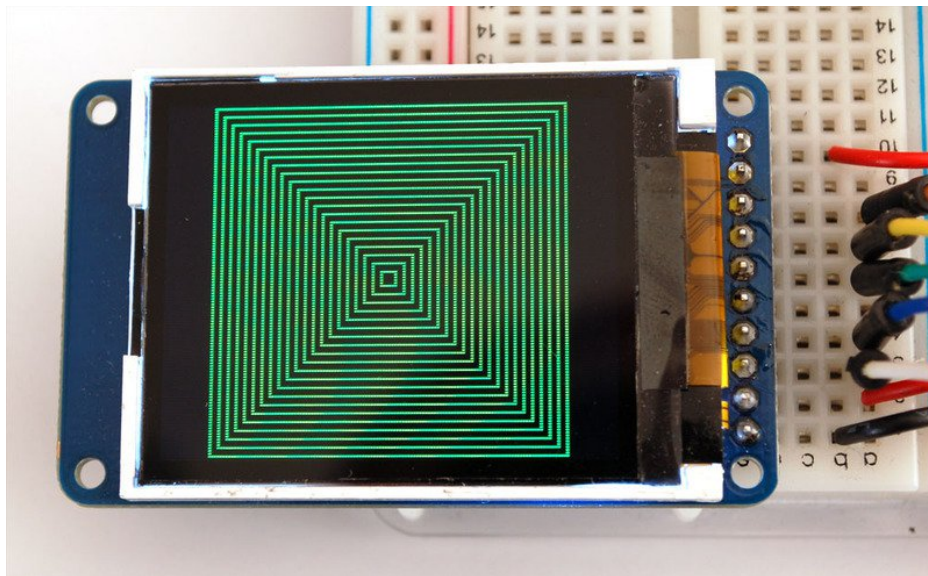
void loop()
{
  int joy = CheckJoystick();
  switch (joy)
  {
    case Left:
      Serial.println("Left");
      break;
    case Right:
      Serial.println("Right");
      break;
    case Up:
      Serial.println("Up");
      break;
    case Down:
      Serial.println("Down");
      break;
    case 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 rewriting 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. Its 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 \(https://adafru.it/aHm\)](https://adafru.it/aHm) (this contains the low-level code specific to this device), and second is the [Adafruit GFX Library \(https://adafru.it/aJa\)](https://adafru.it/aJa) (which handles graphics operations common to many displays we carry). Download both ZIP files, uncompress and rename the folders to 'Adafruit_ST7735' and 'Adafruit_GFX' respectively, place them inside your Arduino libraries folder and restart the Arduino IDE. If this is all unfamiliar, we have a [tutorial introducing Arduino library concepts and installation \(https://adafru.it/aYG\)](https://adafru.it/aYG).



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

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.

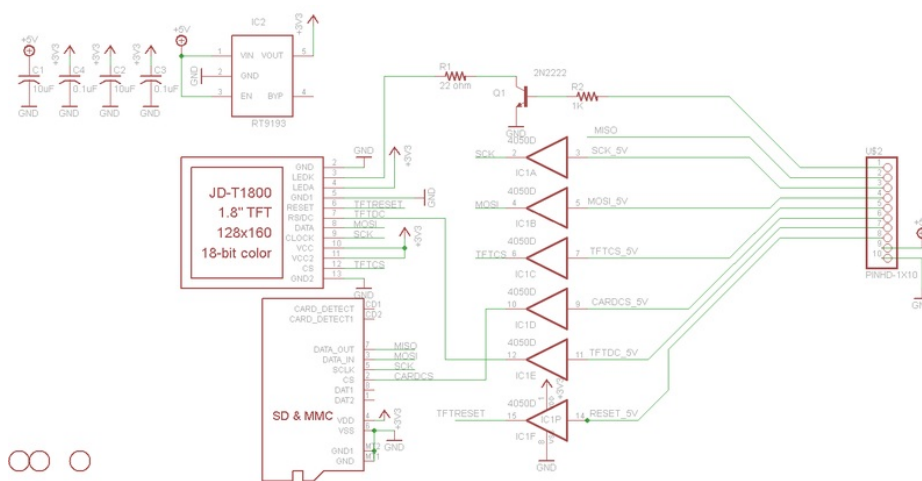
In Arduino, use `delay()` to add a few milliseconds before calling `tft.begin()`. Adjust the amount of delay as needed to see how little you can get away with for your specific setup.

Downloads

Files & Datasheets

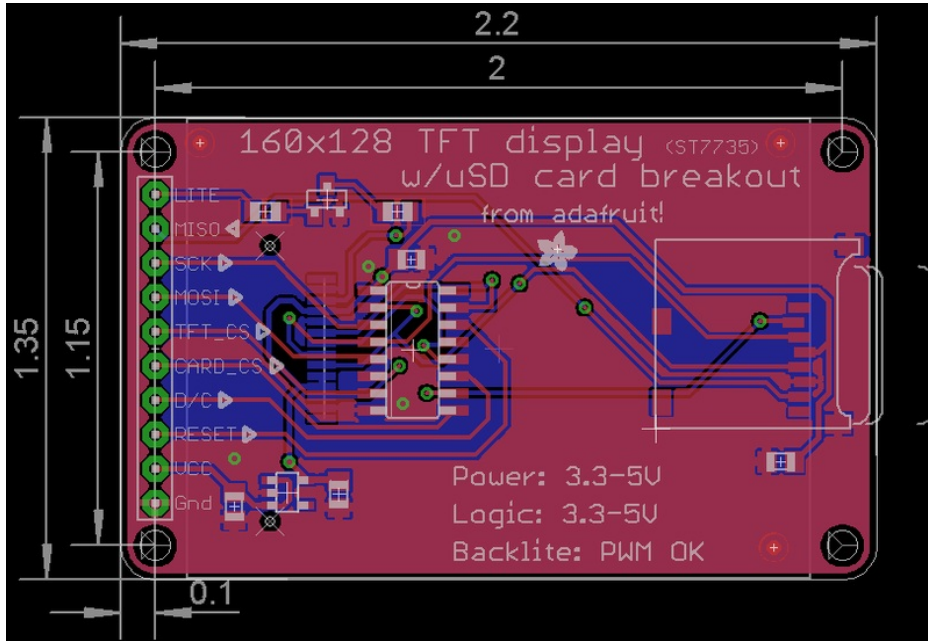
- [Adafruit GFX library \(https://adafru.it/aJa\)](https://adafru.it/aJa)
- [Adafruit ST7735 library \(https://adafru.it/aHm\)](https://adafru.it/aHm)(See our detailed tutorial for installation assistance (<https://adafru.it/aYG>))
- [Fritzing object in the Adafruit library \(https://adafru.it/aP3\)](https://adafru.it/aP3)
- [Datasheet for the display \(https://adafru.it/aP8\)](https://adafru.it/aP8)
- [Datasheet for the display driver chip \(https://adafru.it/aP9\)](https://adafru.it/aP9).
- [EagleCAD PCB files for TFT shield \(https://adafru.it/qxB\)](https://adafru.it/qxB)
- [EagleCAD PCB files for TFT breakout \(https://adafru.it/qxC\)](https://adafru.it/qxC)

Breakout Schematic



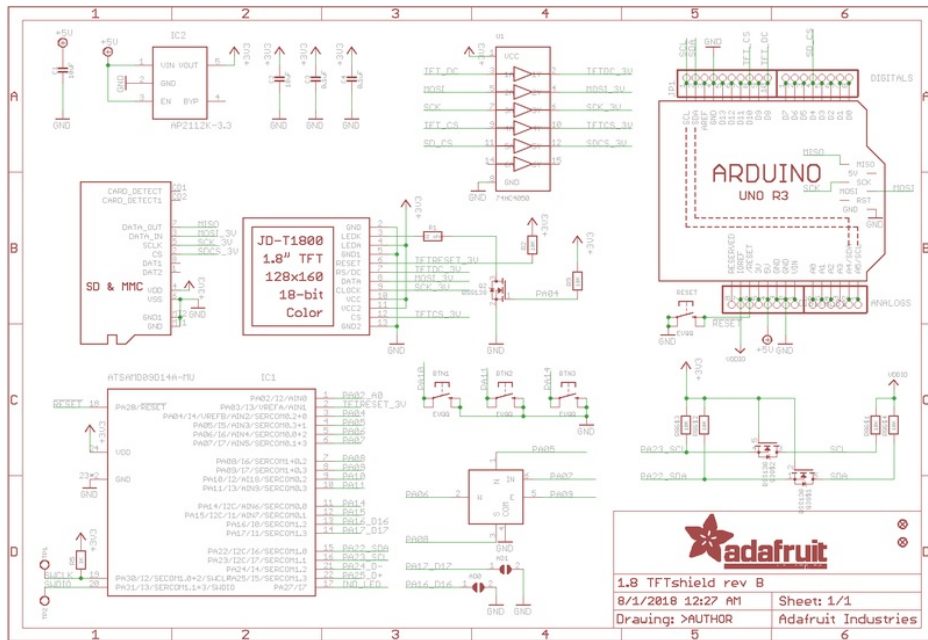
For the level shifter we use the CD74HC4050 which has a typical propagation delay of ~10ns

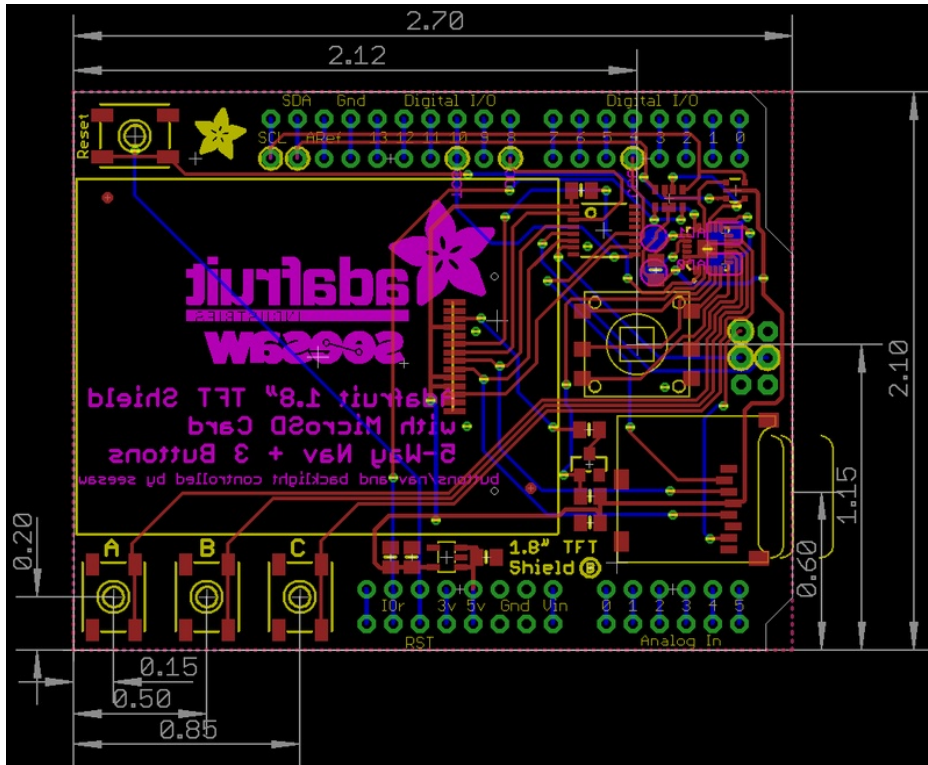
Breakout Fabrication print



Shield v2 Schematic & Fab Print

This is the newer seesaw version





Shield V1 Schematic & Fab Print

This is the 'original' non-seesaw version

