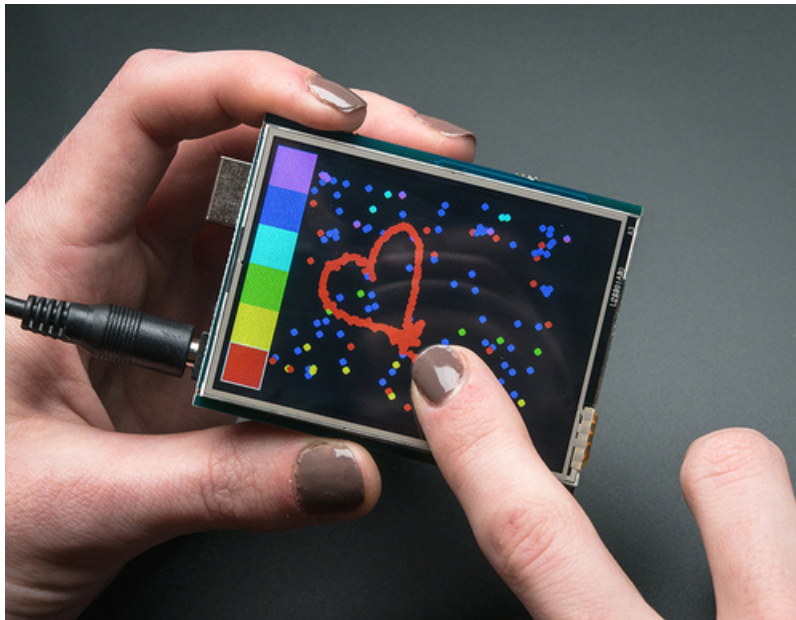# Adafruit 2.8" TFT Touch Shield v2 - Capacitive or Resistive
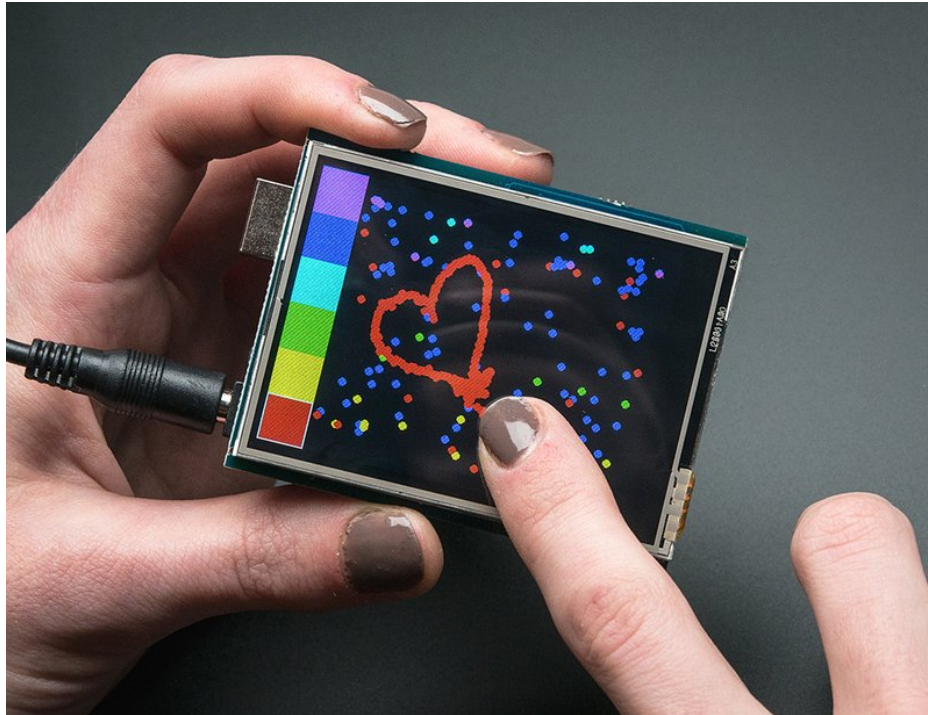
Created by lady ada


Last updated on 2020-06-11 03:25:37 PM EDT

Spice up your Arduino project with a beautiful large touchscreen display shield with built in microSD card connection. This TFT display is big (2.8" diagonal) bright (4 white-LED backlight) and colorful (18-bit 262,000 different shades)! 240x320 pixels with individual pixel control. It has way more resolution than a black and white 128x64 display. As a bonus, this display comes with a **resistive or capacitive** touchscreen attached to it already, so you can detect finger presses anywhere on the screen.
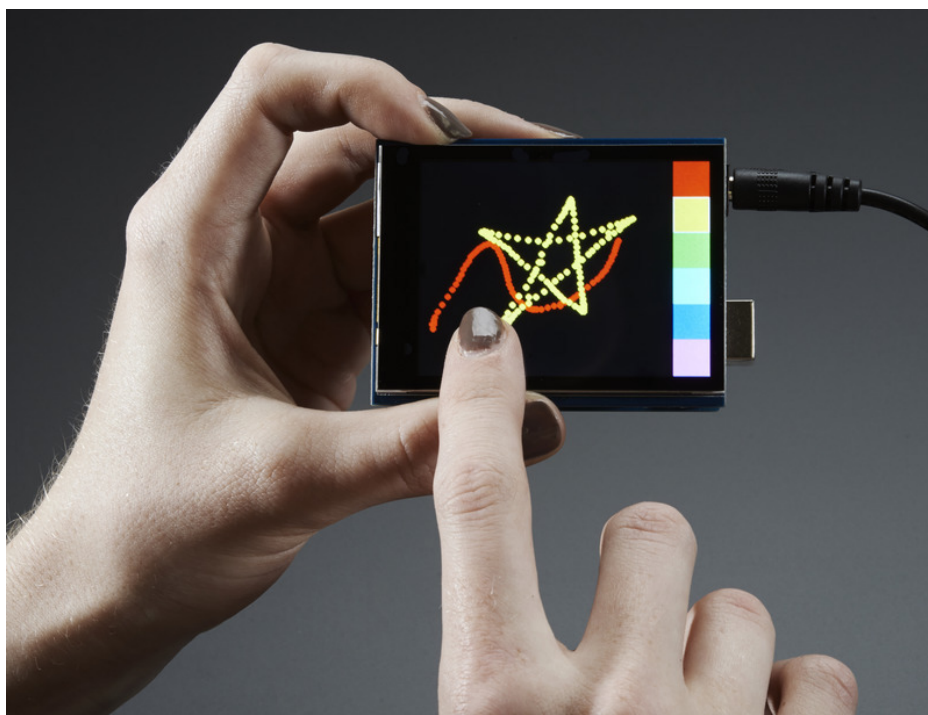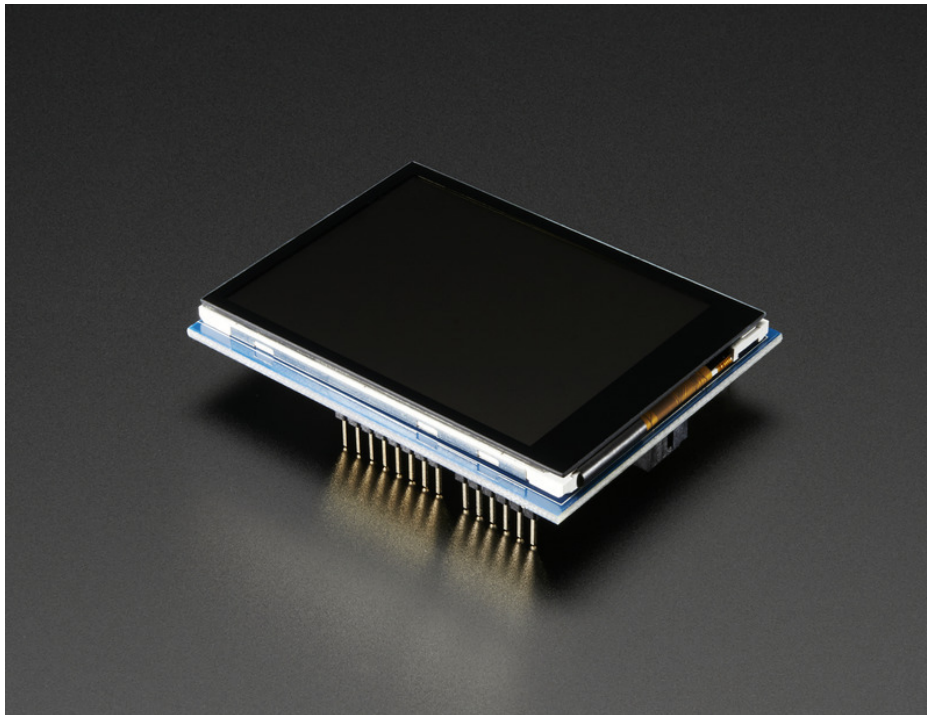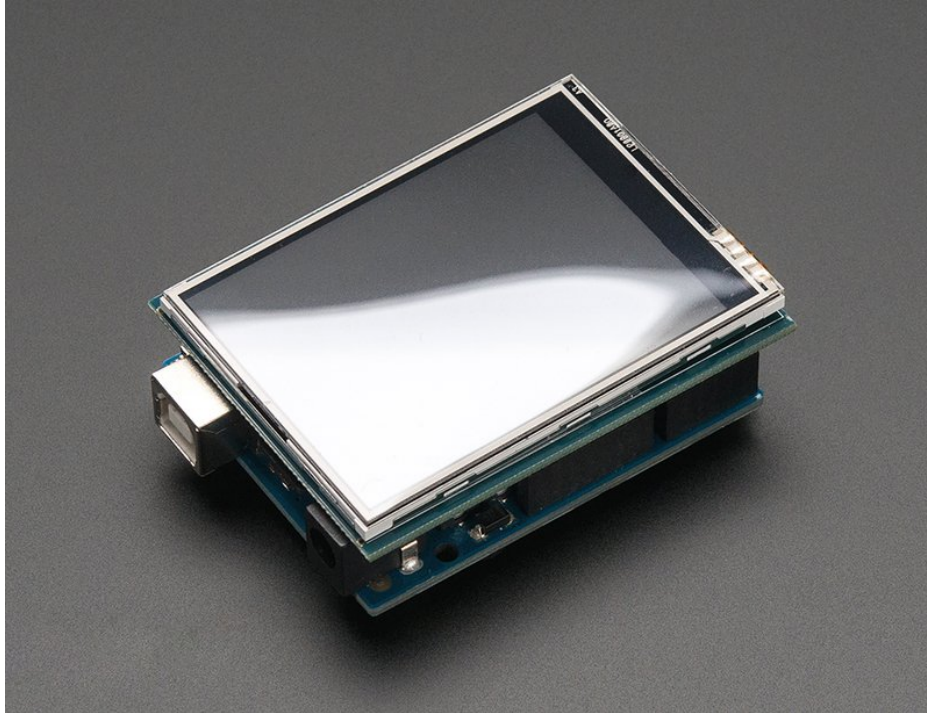
This shield uses a SPI display - its much easier to use with Mega & Leonardo than our v1 shield. We also include an SPI resistive touchscreen controller or a I2C capacitive touch screen controller so you only need one or two additional pins to add a high quality touchscreen controller. Even with all the extras, the price is lower thanks to our parts sourcing & engineering skillz!
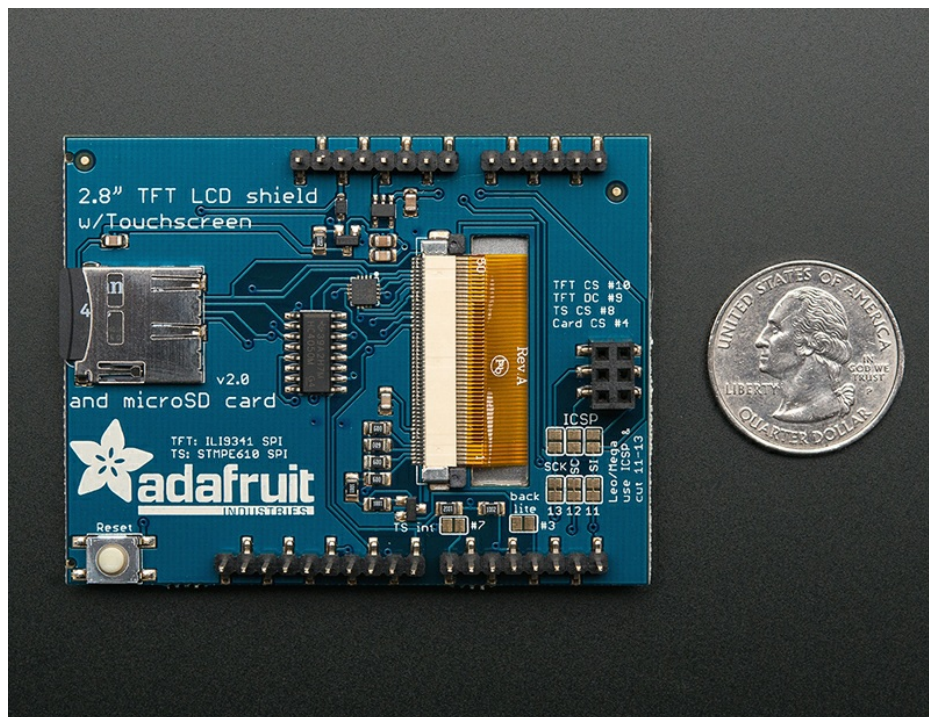
The shield is fully assembled, tested and ready to go. No wiring, no soldering! Simply plug it in and load up our library - you'll have it running in under 10 minutes! Works best with any classic Arduino (UNO/Duemilanove/Diecimila). Solder three jumpers and you can use it at full speed on a Leonardo or Mega as well.

This display shield has a controller built into it with RAM buffering, so that almost no work is done by the microcontroller. This shield needs fewer pins than our v1 shield, so you can connect more sensors, buttons and LEDs: 5 SPI pins for the display, another pin or two for the touchscreen controller and another pin for uSD card if you want to read images off of it.

Connecting

Pinouts

There's two versions of the shield. One has a resistive touch screen, one has a capacitive one. The *TFT display* and pinouts is the same for both. The microSD card is the same too. The differences come in on the touch screen controller and whether the SPI pins are connected to pins 11-13 or the ICSP header by default.

TFT Screen Pins

- **Digital #13** *or* **ICSP SCLK** - This is the hardware SPI clock pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #13. This pin is used for the TFT, microSD and resistive touch screen data clock
- **Digital #12** *or* **ICSP MISO** - This is the hardware SPI Microcontroller In Serial Out pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #12. This pin is used for the TFT, microSD and resistive touch screen data
- **Digital #11** *or* **ICSP MOSI** - This is the hardware SPI Microcontroller Out Serial In pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #11. This pin is used for the TFT, microSD and resistive touch screen data
- **Digital #10** - This is the TFT CS (chip select pin). It's used by the Arduino to tell the TFT that it wants to send/receive data from the TFT only
- **Digital #9** - This is the TFT DC (data/command select) pin. It's used by the Arduino to tell the TFT whether it wants to send data or commands

Resistive Touch Controller Pins

- **Digital #13** *or* **ICSP SCLK** - This is the hardware SPI clock pin. By default its on digital #13. By cutting a jumper and soldering another on the back, you can move this line from digital to the ICSP 2x3 header. This pin is used for the TFT, microSD and resistive touch screen data clock
- **Digital #12** *or* **ICSP MISO** - This is the hardware SPI Microcontroller In Serial Out pin. By default its on digital #12. By cutting a jumper and soldering another on the back, you can move this line from digital to the ICSP 2x3 header. This pin is used for the TFT, microSD and resistive touch screen data
- **Digital #11** *or* **ICSP MOSI** - This is the hardware SPI Microcontroller Out Serial In pin. By default its on digital #11.By cutting a jumper and soldering another on the back, you can move this line from digital to the ICSP 2x3 header. This pin is used for the TFT, microSD and resistive touch screen data
- **Digital #8** - This is the STMPE610 Resistive Touch CS (chip select pin). It's used by the Arduino to tell the Resistive controller that it wants to send/receive data from the STMPE610 only
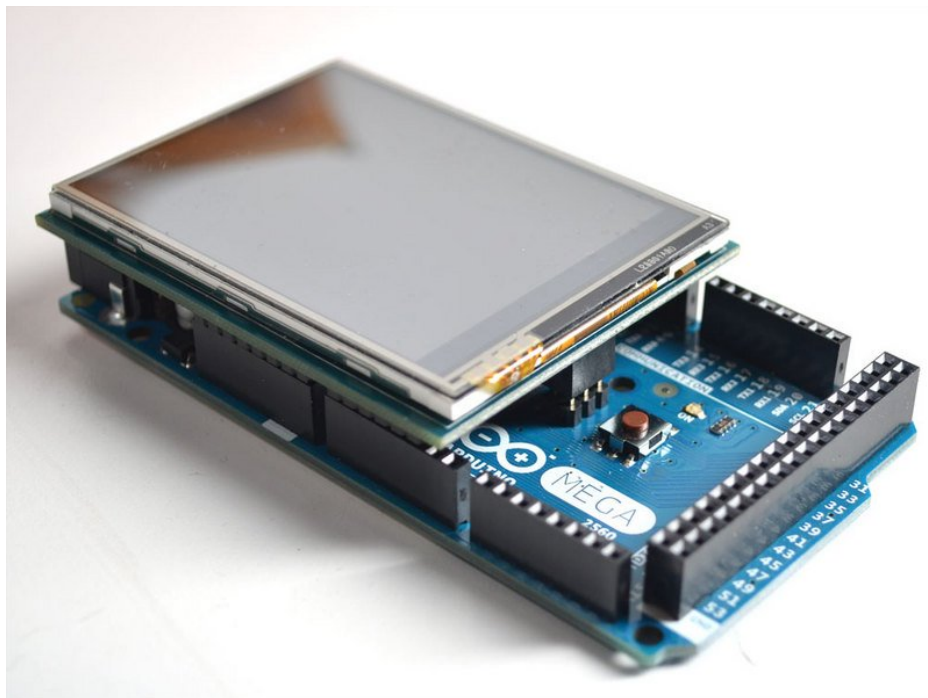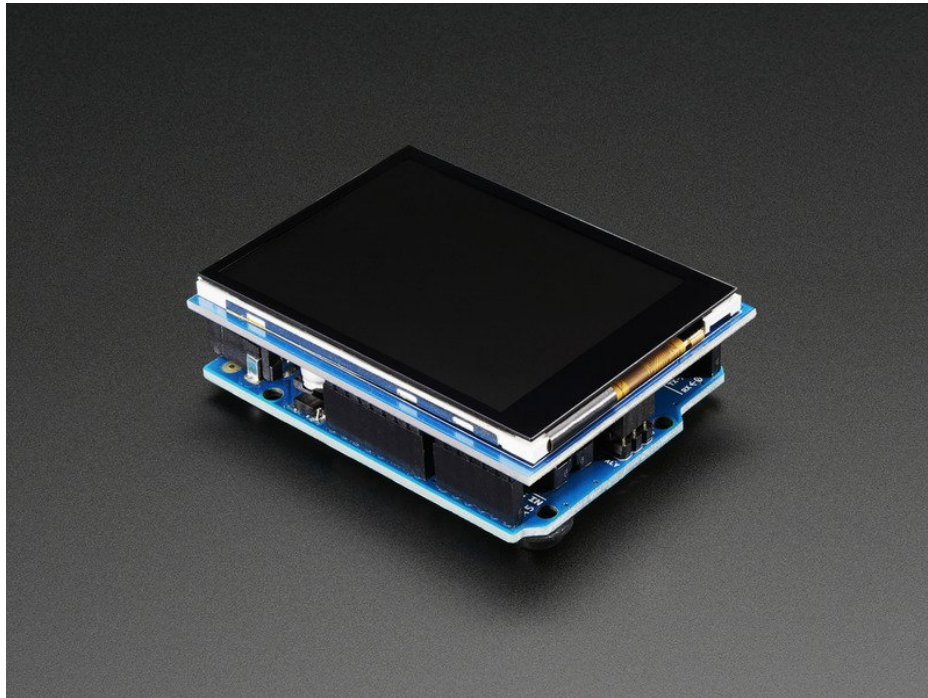
Capacitive Touch Pins

- **SDA** - This is the I2C data pin used by the FT6206 capacitive touch controller chip. It can be shared with other I2C devices. On UNO's this pin is also known as Analog 4.
- **SCL** - This is the I2C clock pin used by the FT6206 capacitive touch controller chip. It can be shared with other I2C devices. On UNO's this pin is also known as Analog 5.

MicroSD card Pins

- **Digital #13** *or* **ICSP SCLK** - This is the hardware SPI clock pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #13. This pin is used for the TFT, microSD and resistive touch screen data clock
- **Digital #12** *or* **ICSP MISO** - This is the hardware SPI Microcontroller In Serial Out pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #12. This pin is used for the TFT, microSD and resistive touch screen data
- **Digital #11** *or* **ICSP MOSI** - This is the hardware SPI Microcontroller Out Serial In pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #11. This pin is used for the TFT, microSD and resistive touch screen data
- **Digital #4** - This is the uSD CS (chip select pin). It's used by the Arduino to tell the uSD that it wants to send/receive data from the uSD only

Using with an Uno R3, Leonardo, Mega, etc

Because the TFT is about the same size as an Arduino, we pre-assemble the shield in the factory. For the Capacitive Touch, to use, simply place it onto your Arduino Uno/Duemilanove/compatible. No wiring, no soldering! **Bam!** For the Resistive touch, you will need to cut the traces between 11-13 and solder bridge the ICSP pins.

Using Capacitive Touch Version w/Older Arduino

If you have an old Arduino without the SCL/SDA pins brought out (pre UNO R3) use a soldering iron to short these two jumpers:
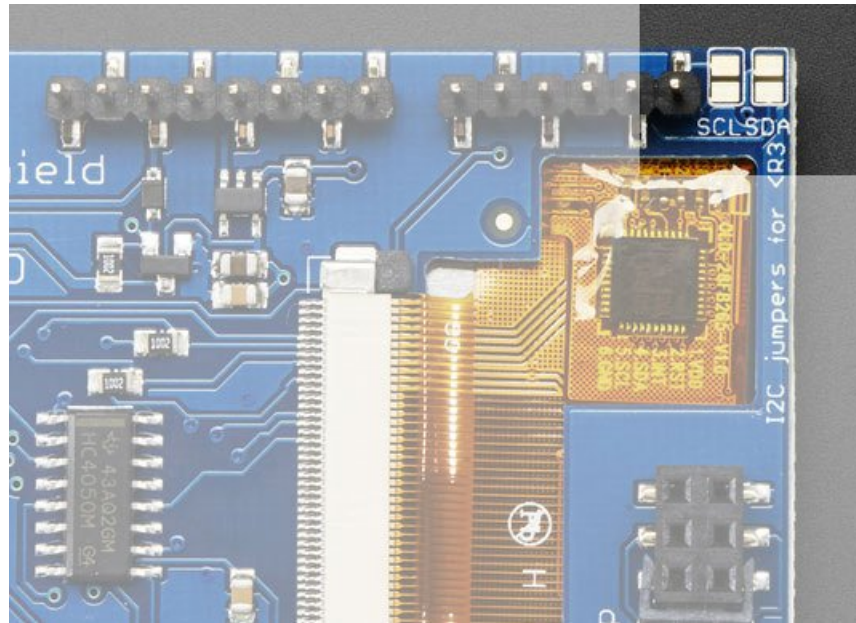
Arduino Graphics Test

We have a library with example code ready to go for use with these TFTs. The library is not incredibly fast and optimized but its a good start and can easily be ported to other micrcontrollers. However, we'll assume you're using an Arduino.

Our github repository (https://adafru.it/d4d) contains all the code and examples you'll need for driving the TFT.

Install Libraries

You'll need a few libraries to use this display

From within the Arduino IDE, open up the **Library Manager...**
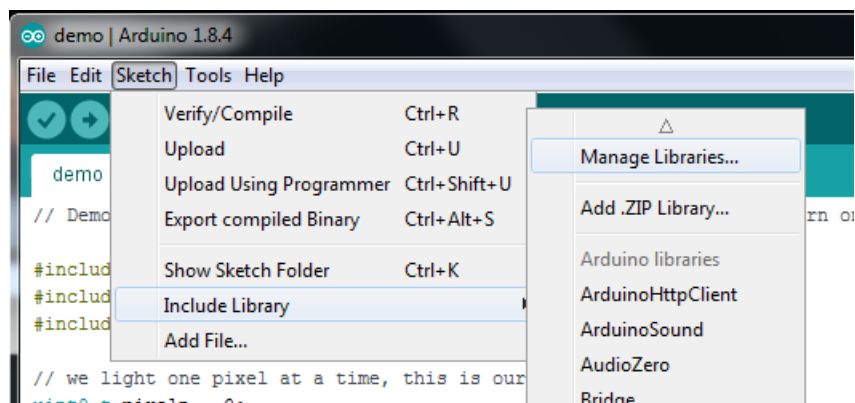
Install Adafruit ILI9341 TFT Library

We have example code ready to go for use with these TFTs.

Two libraries need to be downloaded and installed: first is the Adafruit ILI9341 library (https://adafru.it/d4d) (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). If you have **Adafruit_GFX** already, make sure its the most recent version since we've made updates for better performance
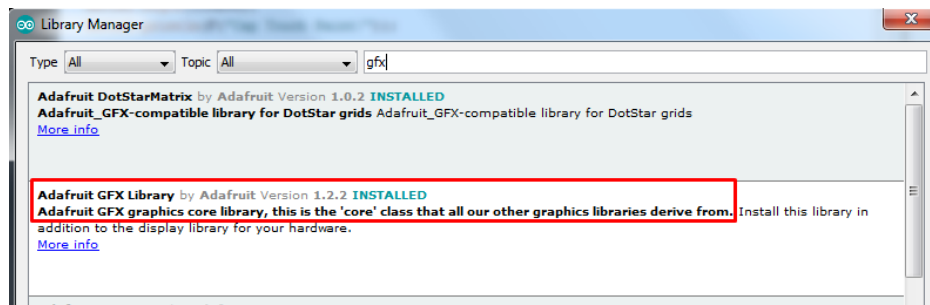
Search for **ILI9341** and install the **Adafruit ILI9341** library that pops up!
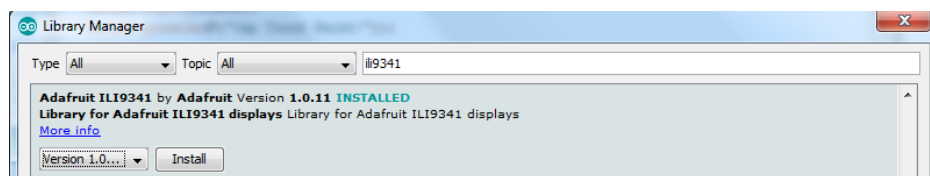
For more details, especially for first-time library installers, check out our great tutorial at http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)
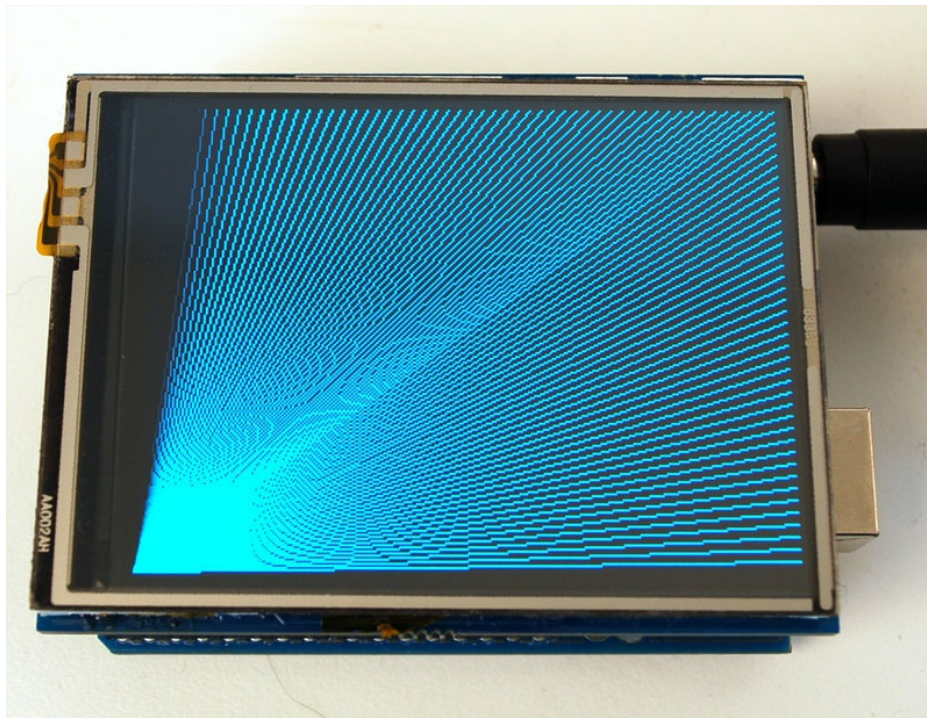
Next up, search for **Adafruit GFX** and locate the core library. A lot of libraries may pop up because we reference it in the description so just make sure you see **Adafruit GFX Library** in bold at the top.
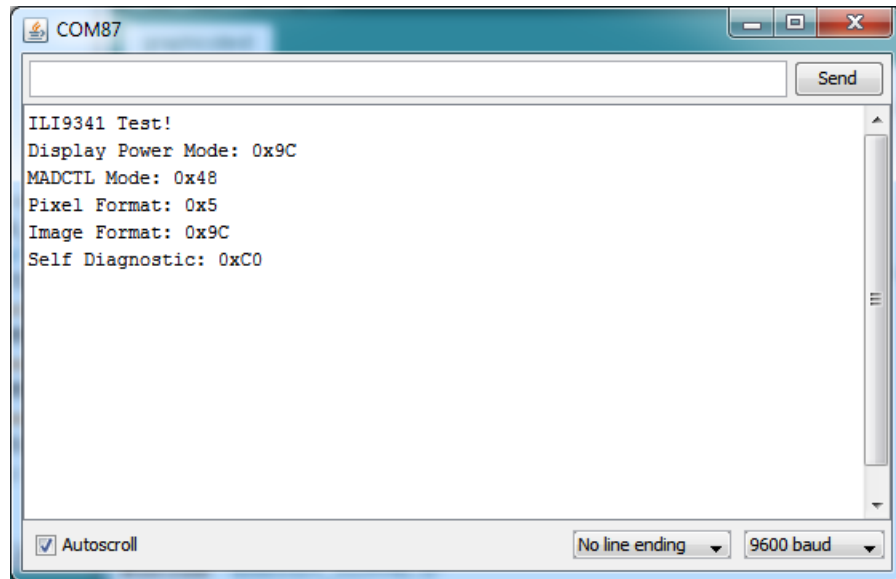
Install it!

If using an older version of the Arduino IDE (pre-1.8.10), repeat this process one more time, looking for the **Adafruit_BusIO** library. Install that one too.

Restart the Arduino software. You should see a new **example** folder called **Adafruit_ILI9341** and inside, an example called **graphicstest**. Upload that sketch to your Arduino! You should see a collection of graphical tests draw out on the TFT.

```
ILI9341 Test!
Display Power Mode: 0x9C
MADCTL Mode: 0x48
Pixel Format: 0x5
Image Format: 0x9C
Self Diagnostic: 0xC0
```

Adafruit GFX library

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.

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

Resistive Touchscreen Paint Demo

> 🚫 This page is for the Resistive Touch Screen version of the Shield!

The LCD has a 2.8" 4-wire resistive touch screen glued onto it. You can use this for detecting finger-presses, stylus', etc. Normally, you'll need 4 pins to talk to the touch panel **but** we decided to go all snazzy and put a dedicated touch screen driver onto the shield. The driver shares the SPI pins with the TFT and SD card, so only one extra pin is needed (digital #8) This allows you to query the controller when you're ready to read touchscreen data, and saves 3 pins.

To control the touchscreen you'll need one more library (https://adafru.it/d4f) - the STMPE610 controller library which does all the low level chatting with the STMPE610 driver chip.
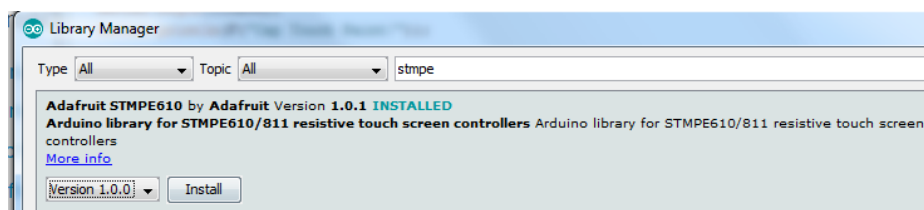
Once you have the library installed, restart the IDE. Now from the **examples->Adafruit_ILI9341** menu select **touchpaint** and upload it to your Arduino.
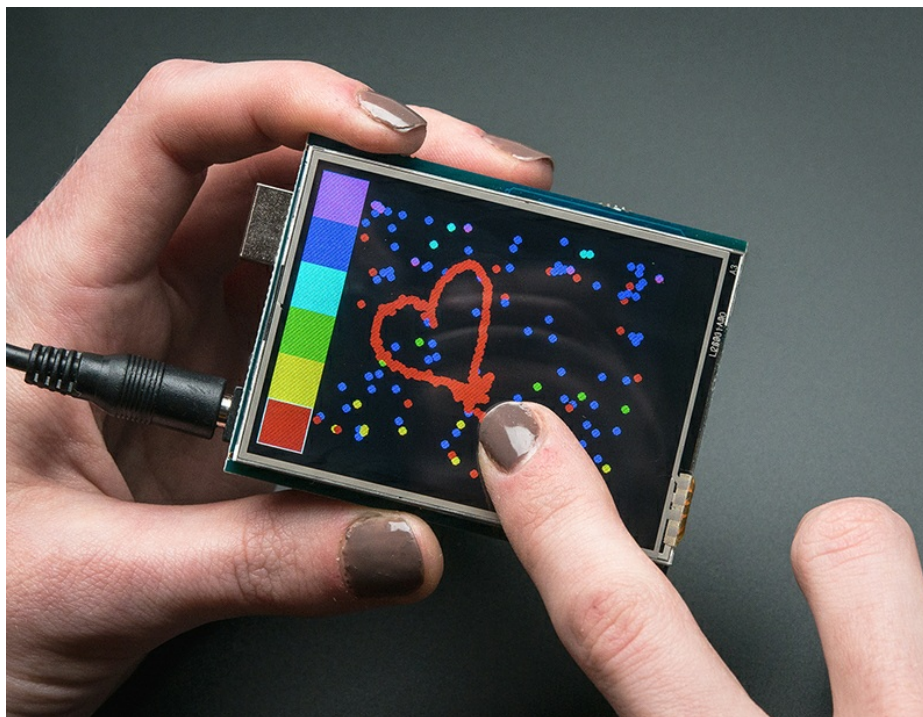
> 🚫 The touch screen is made of a thin glass sheet, and its very fragile - a small crack or break will make the entire touch screen unusable. Don't drop or roughly handle the TFT and be especially careful of the corners and edges. When pressing on the touchscreen, sometimes people can use the tip of their fingers, or a fingernail. If you don't find the touchscreen responds well to your fingers, you can use a rounded stylus which

Getting data from the touchscreen is fairly straight forward. Start by creating the touchscreen object with

> **Adafruit_STMPE610 ts = Adafruit_STMPE610(STMPE_CS);**

We're using hardware SPI so the clock, mosi and miso pins are not defined here. For the shield, CS is #8 always. Then you can start the touchscreen with

> **ts.begin()**

Check to make sure this returns a True value, which means the driver was found. If it wasn't, make sure you have the hardware SPI jumpers set up right: for Leonardo/Mega the ICSP jumpers get closed.

Now you can call

> **if (! ts.bufferEmpty()**)

to check if there's any data in the buffer. The touchscreen driver will store touchpoints at all times. When you're ready to get the data, just check if there's any data in the buffer. If there is, you can call

> **TS_Point p = ts.getPoint();**

To get the oldest point from the buffer. TS_Point has **.x .y** and **.z** data points. The **x** and **y** points range from 0 to 4095. The STMPE610 does not store any calibration data in it and it doesn't know about rotation. **So if you want to rotate the screen you'll need to manually rotate the x/y points!** The z point is 'pressure' and ranges from 0 to 255, we don't use it here but you can experiment with it on your own, the harder you press, the lower the number.

Since data from the STMPE610 comes in 0-4095 but our screen is 320 pixels by 240 pixels, we can use **map** to convert 0-4095 to 0-320 or 0-240. Something like

> **p.x = map(p.x, 0, 4095, 0, tft.width());**
> **p.y = map(p.y, 0, 4095, 0, tft.height());**

However, the touchscreen is a bit bigger than the screen, so we actually need to ignore presses beyond the touchscreen itself. We found that these numbers reflected the true range that overlaps the screen

> **#define TS_MINX 150**
> **#define TS_MINY 130**
> **#define TS_MAXX 3800**
> **#define TS_MAXY 4000**

So we use

> **p.x = map(p.x, TS_MINX, TS_MAXX, 0, tft.width());**
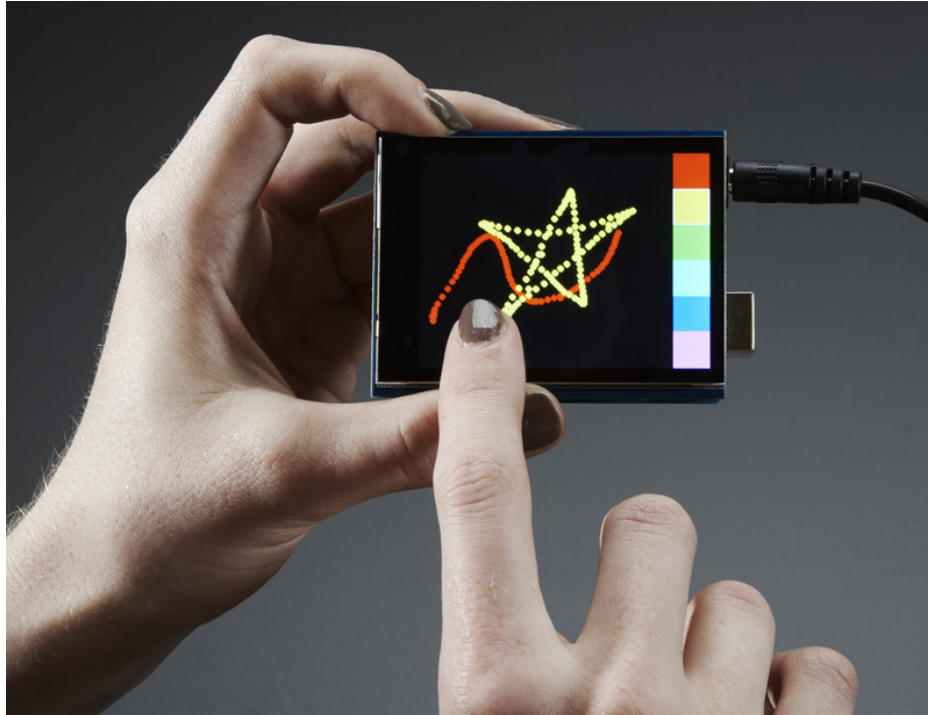> **p.y = map(p.y, TS_MINY, TS_MAXY, 0, tft.height());**

instead.

One last point (pun intended!) since the touchscreen driver stores points in a buffer, you may want to ask the driver "is the touchscreen being pressed RIGHT NOW?" You can do that with

```
if (ts.touched())
```

> This page is for the Capacitive Touch Screen version of the Shield!

We now have a super-fancy capacitive touch screen version of this shield. Instead of a resistive controller that needs calibration and pressing down, the capacitive has a hard glass cover and can be used with a gentle fingertip. It is a single-touch capacitive screen only!

The capacitive touch screen controller communicates over I2C, which uses two hardwire pins. However, you can share these pins with other sensors and displays as long as they don't conflict with I2C address 0x38.

Download the FT6206 Library

To control the touchscreen you'll need one more library (https://adafru.it/dGG) - the FT6206 controller library which does all the low level chatting with the FT6206 driver chip. Use the library manager and search for **FT6206** and select the Adafruit FT6206 library:
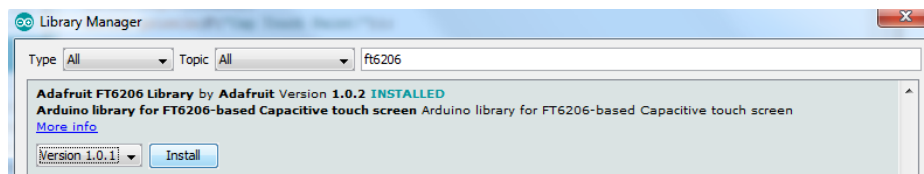
Once you have the library installed, restart the IDE. Now from the **examples->Adafruit_FT6206** menu select **CapTouchPaint** and upload it to your Arduino.

The touch screen is made of a thin glass sheet, and its very fragile - a small crack or break will make the entire touch screen unusable. Don't drop or roughly handle the TFT and be especially careful of the corners and edges. When pressing on the touchscreen, remember you cannot use a fingernail, it must be a fingerpad. Do not press harder and harder until the screen cracks!

FT6206 Library Reference

Getting data from the touchscreen is fairly straight forward. Start by creating the touchscreen object with

> **Adafruit_FT6206 ts** = **Adafruit_FT6206();**

We're using hardware I2C which is fixed in hardware so no pins are defined.
Then you can start the touchscreen with

> **ts.begin()**

Check to make sure this returns a True value, which means the driver was found. You can also call **begin(threshvalue)** wish a number from 0-255 to set the touch threshhold. The default works pretty well but if you're having too much sensitivity (or not enought) you can try tweaking it

Now you can call

> **if (ts.touched())**

to check if the display is being touched, if so call:

> **TS_Point p** = **ts.getPoint();**

To get the touch point from the controller. TS_Point has **.x** and **.y** data points. The **x** and **y** points range from 0 to 240 and 0 to 320 respectively. This corresponds to each pixel on the display. The FT6206 does not need to be 'calibrated' but it also doesn't know about rotation. **So if you want to rotate the screen you'll need to manually rotate the x/y points!**

FT6206 Library
Reference

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

Its 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:
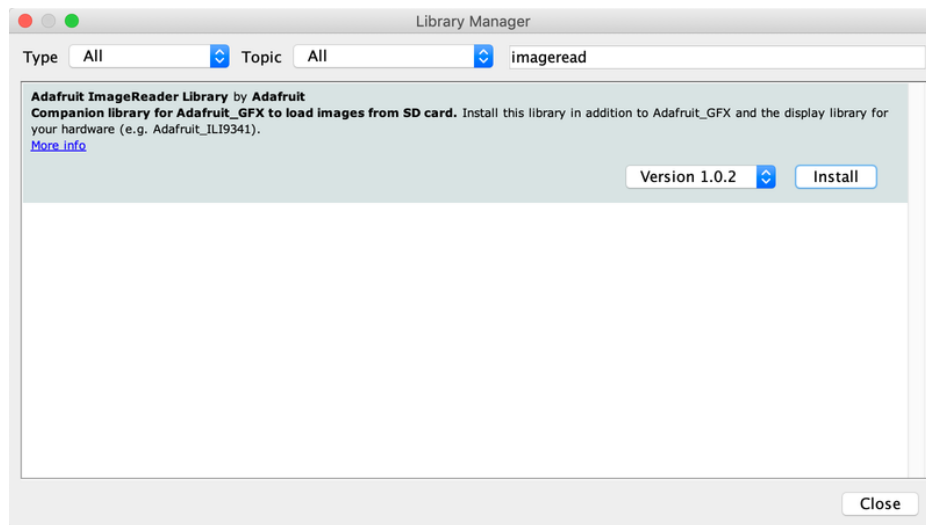
With the library installed, let's proceed by downloading this image of pretty flowers (pix by johngineer):

Copy **purple.bmp** into the base directory of a microSD card and insert it into the microSD socket in the shield. Now upload the **File→Examples→Adafruit_ImageReader→ShieldlLI9341** example sketch to your Arduino + shield. You will see the flowers appear!

To make new bitmaps, make sure they are less than 240 by 320 pixels and save them in **24-bit BMP format**! They must be in 24-bit format, even if they are not 24-bit color as that is the easiest format for the Arduino to decode. You can rotate images using the **setRotation()** procedure.

The ShieldILI9341 example sketch shows everything you need to work with BMP images. Here's just the vital bits broken out...

Several header files are included at the top of the sketch. All of these are required...they let us access the SD card and the display, and provide the image-reading functions:

```
#include <SPI.h>
#include <SD.h>
#include <Adafruit_GFX.h>        // Core graphics library
#include <Adafruit_ILI9341.h>    // Hardware-specific library
#include <Adafruit_ImageReader.h> // Image-reading functions
```

Several #defines relate to hardware pin numbers, all fixed values when using the shield.

Then we declare the tft screen object, and the image-reader object like so:

```
#define SD_CS   4 // SD card select pin
#define TFT_CS 10 // TFT select pin
#define TFT_DC  9 // TFT display/command pin

Adafruit_ILI9341     tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
Adafruit_ImageReader reader; // Class w/image-reading functions
```

After the SD and TFT's `begin()` functions have been called (see the example sketch again, in the `setup()` function), you can then call `reader.drawBMP()` to load an image from the card to the screen:

```
ImageReturnCode stat;
stat = reader.drawBMP("/purple.bmp", tft, 0, 0);
```

You can draw as many images as you want — though remember the names must be less than 8 characters long. Call like so:

```
reader.drawBMP(filename, tft, x, y);
```

'x' and 'y' are pixel coordinates where top-left corner of the image will be placed. Images can be placed anywhere on screen...even partially off screen, the library will clip the section to load.

Image loading is explained in greater depth in the Adafruit_GFX library guide. (https://adafru.it/DpM)

Both the resistive and capacitive versions of this shield have the ability to dim the backlight and get an interrupt from the resistive or capacitive touch controller chip on-board.
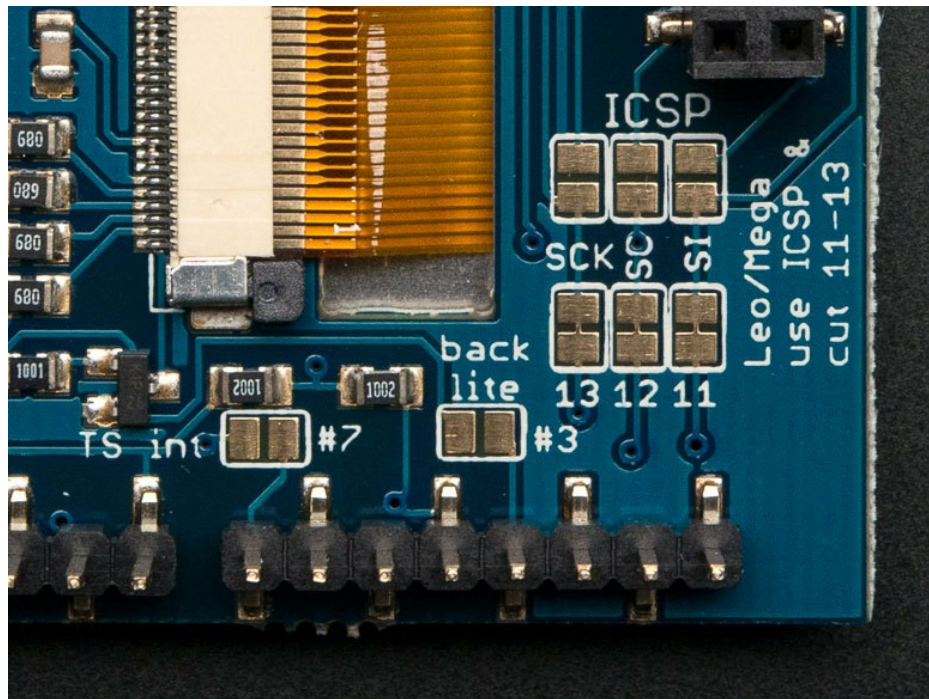
Controlling the Backlight

By default, we assume you'll want the backlight on all the time. However, you may want to PWM control or otherwise turn off the LED backlight to save power. You can do this with a simple hack. On the back, look for the **backlight** jumper.

On the resistive TFT touch shield
Solder the jumper labeled **Pin 3**. Then you can use Digital 3 to control the backlight.

On the capacitive TFT touch shield
Solder the jumper labeled **Pin 5**. Then you can use Digital 5 to control the backlight.

Touchscreen Interrupt pin

Advanced users may want to get an interrupt on a pin (or even, just test a pin rather than do a full SPI query) when the touchscreen is pressed. You can do that by jumpering the #7 solder jumper labeled **TS int**. We didn't want it to connect to #2 or #3 since those are the Leonardo I2C pins. You can use pin change interrupts to get an interrupt callback on #7. (https://adafru.it/d4h)Or, with a little blue wire, advanced users can connect a wire from the TS interrupt pad to any pin they choose. We find that querying/polling the chip is fast enough for most beginner Arduino projects!

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.
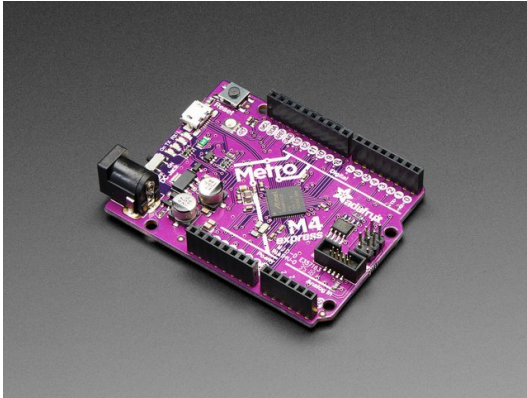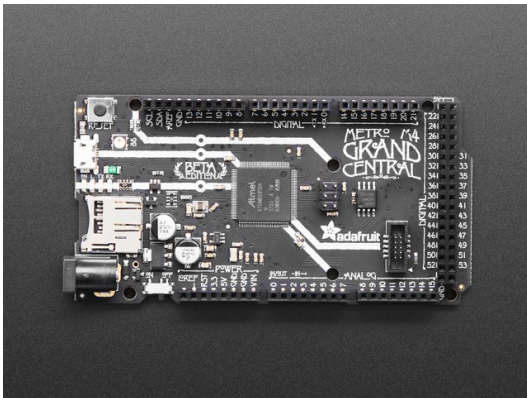
Adafruit Metro M4 feat. Microchip ATSAMD51

$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.
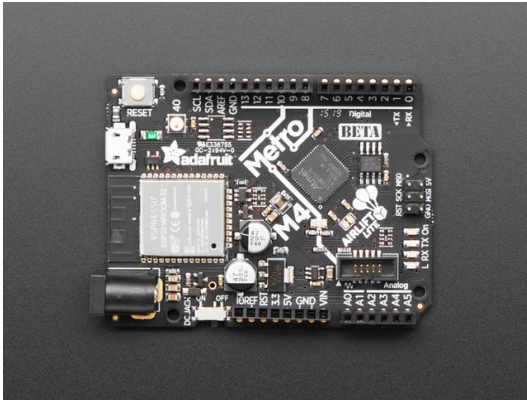
Adafruit Grand Central M4 Express featuring the SAMD51

$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

### 2.8" Capacitive TFT Touch Shield

This shield should be ready to go. By default, the SPI connections should be using the ICSP header. If you have modified it on the underside to use Pins 11-13, this will need to be changed back in order to work with displayio. After that the shield should be ready to go.

2.8" Resistive TFT Touch Shield

This shield will need a slight modification to use with our boards capable of running CircuitPython. This is because the SPI interface is only available through the ICSP header.
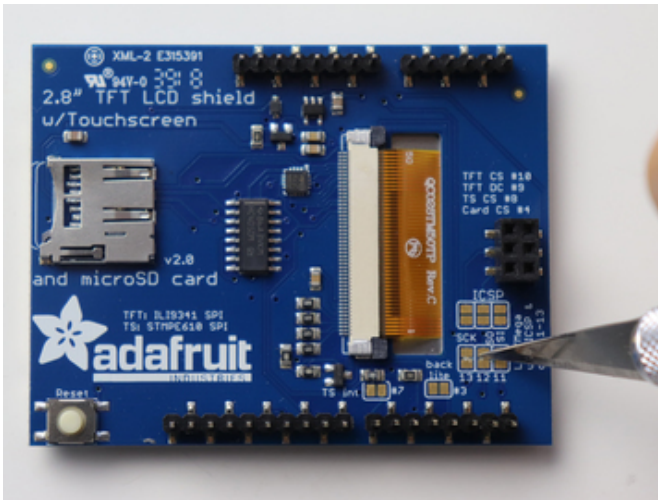
First start by taking a sharp knife and cutting the traces that connect to pins 11-13.
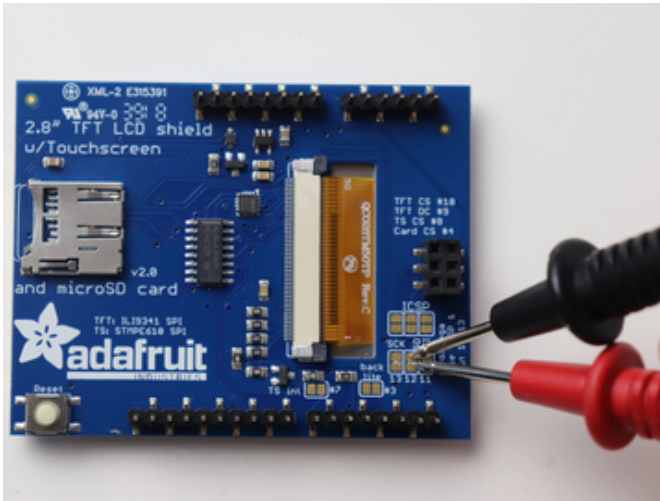
Next, take a multimeter set to the continuity setting and test that the traces were successfully separated.
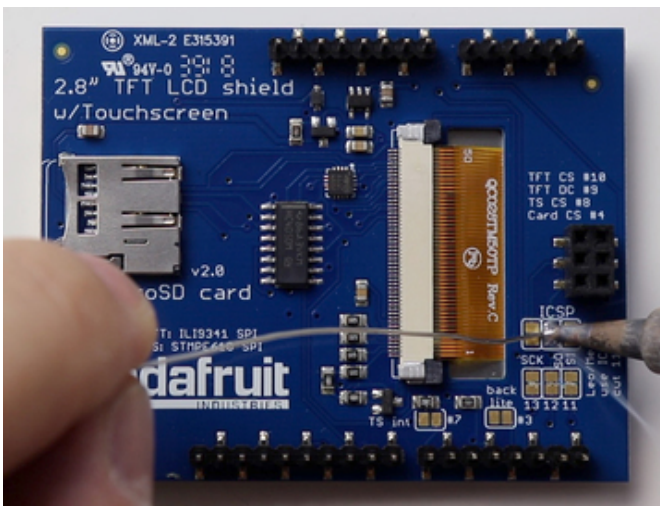
Next, apply a little solder to create a solder bridge on all three of the ICSP pads.
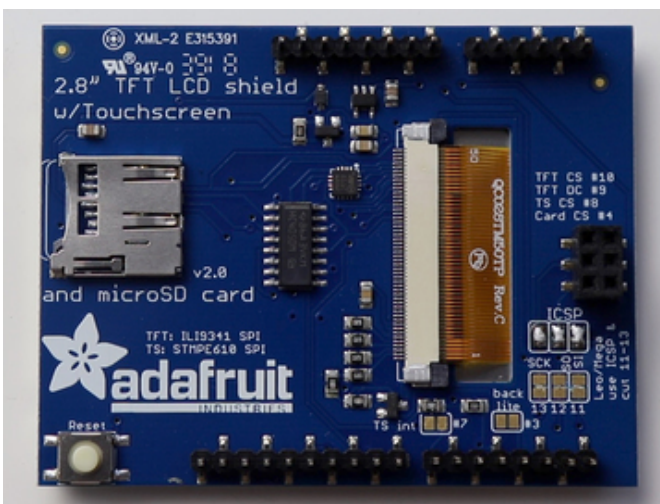
Once all three are solder bridged, then the shield should be all ready to go.

Required CircuitPython Libraries

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

First, make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next, you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Remember for non-express boards, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_ili9341**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_ili9341** 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.

Pinouts are for the 2.8" TFT Shield
"""
import board
import terminalio
import displayio
from adafruit_display_text import label
import adafruit_ili9341

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

# Use Hardware SPI
spi = board.SPI()

# Use Software SPI if you have a shield with pins 11-13 jumpered
# import busio
# spi = busio.SPI(board.D11, board.D13)

tft_cs = board.D10
tft_dc = board.D9

display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs)
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)

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

# Draw a green background
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00  # Bright Green

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

splash.append(bg_sprite)

# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088  # Purple
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=20, y=20)
splash.append(inner_sprite)

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

while True:
    pass
```

Let's take a look at the sections of code one by one. We start by importing the board so that we can initialize `SPI`, `displayio`, `terminalio` for the font, a `label`, and the `adafruit_ili9341` driver.

```
import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_ili9341
```

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()
```

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.D9
```

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 initialize the driver with a width of 320 and a height of 240. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update.

```
display = adafruit_ili9341.ILI9341(display_bus, width=320, height=240)
```
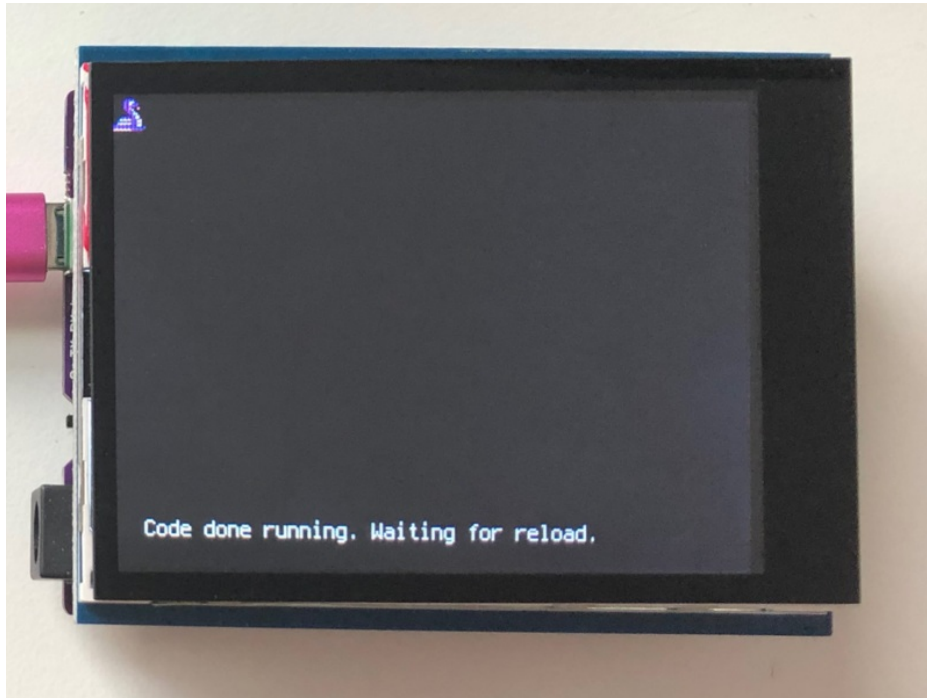
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

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

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

```
color_bitmap = displayio.Bitmap(320, 240, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x00FF00 # Bright Green
```

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

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

Next we will create a smaller purple rectangle. The easiest way to do this is the create a new bitmap that is a little smaller than the full screen with a single color and place it in a specific location. In this case we will create a bitmap that is 20 pixels smaller on each side. The screen is 320x240, so we'll want to subtract 40 from each of those numbers.

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

```
# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(280, 200, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0xAA0088 # Purple
inner_sprite = displayio.TileGrid(inner_bitmap,
                                  pixel_shader=inner_palette,
                                  x=20, y=20)
splash.append(inner_sprite)
```

Since we are adding this after the first rectangle, it's automatically drawn on top. Here's what it looks like now.
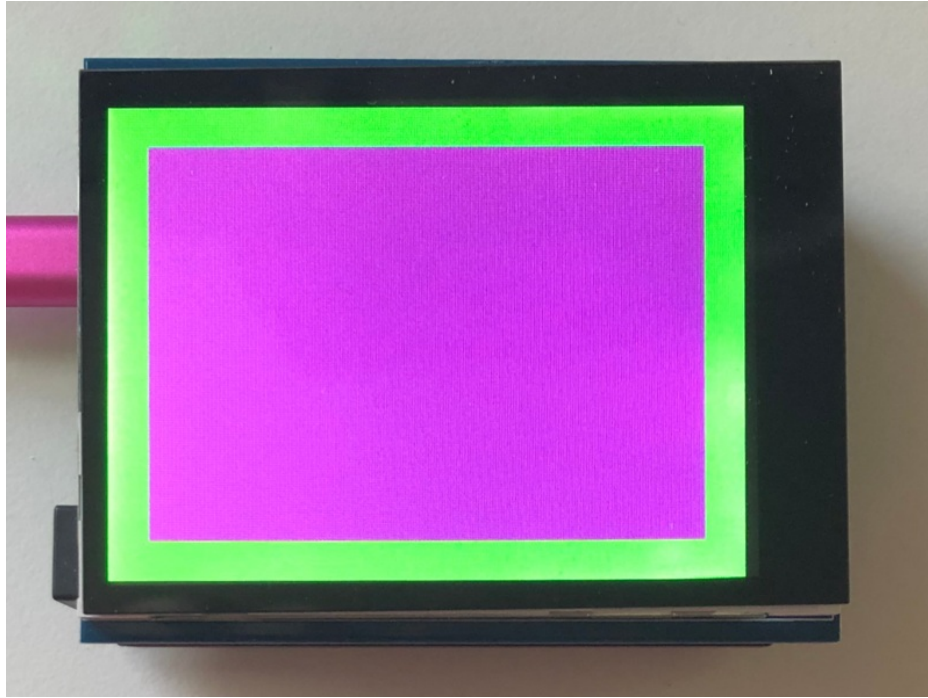
Next let's add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font and scale it up by a factor of three. To scale the label only, we will make use of a subgroup, which we will then add to the main group.

Labels are centered vertically, so we'll place it at 120 for the Y coordinate, and around 57 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some yellow text, so we'll pass it a value of `0xFFFF00`.

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

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
```

Using Touch

We won't be covering how to use the touchscreen on the shield with CircuitPython in this guide, but the libraries required to use it are:

- For enabling capacitive touch use the Adafruit_CircuitPython_FocalTouch (https://adafru.it/Fsy) library.
- For enabling resistive touch use the Adafruit_CircuitPython_STMPE610 (https://adafru.it/Fsz) library.

Where to go from here

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

Downloads

Datasheets & Files

- STMPE610 (https://adafru.it/d4k)
- ILI9341 (TFT controller) (https://adafru.it/d4l)
- Raw 2.8" Resistive TFT datasheet (https://adafru.it/sEt)
- Raw 2.8" Capacitive TFT datasheet (https://adafru.it/rwA)
- FT6206 Datasheet (https://adafru.it/sEu) & App note (https://adafru.it/dRn) (capacitive chip)
- PCB CAD files for both resistive and capacitive versions on GitHub (https://adafru.it/pQb)

Schematic

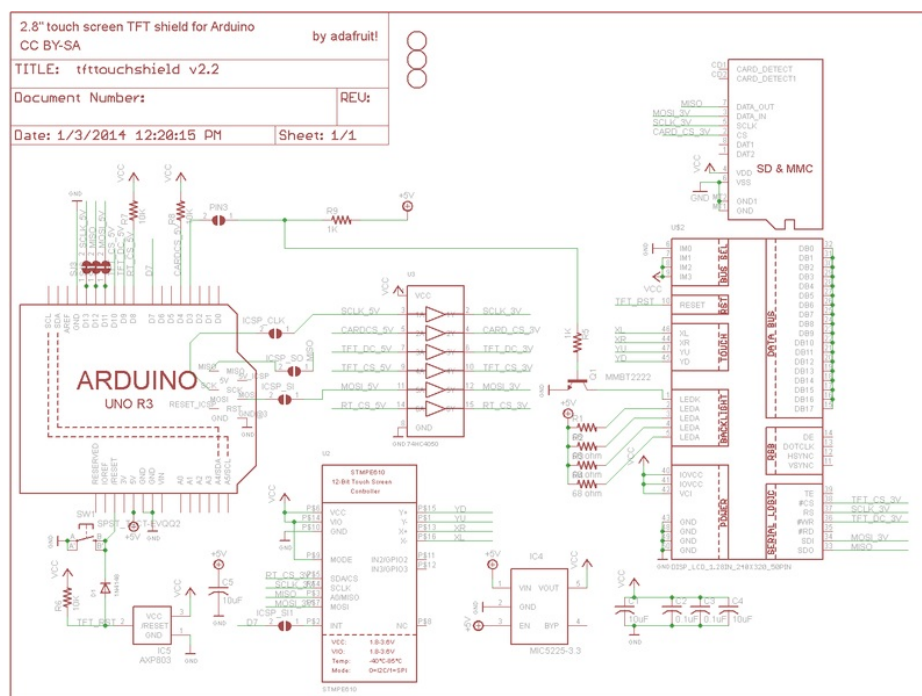Schematic of the v2 Resistive touchshield

Diagram showing the TFT (yellow outline) underlying Arduino mounting holes (thin white line), PCP outline (rectangular thin white line) and 'visible portion' of the TFT (dashed inner line)

Schematic of the v2 Capacitive touchscreen

MOSCHIP.RU
ВМЕСТЕ МЫ СОЗДАЕМ БУДУЩЕЕ

ПОСТАВКА
ЭЛЕКТРОННЫХ КОМПОНЕНТОВ

многоканальный
+7 495 668 12 70
info@moschip.ru

Данный компонент на территории Российской Федерации

Вы можете приобрести в компании MosChip.

Для оперативного оформления запроса Вам необходимо перейти по данной ссылке:

http://moschip.ru/get-element

Вы можете разместить у нас заказ для любого Вашего проекта, будь то серийное производство или разработка единичного прибора.

В нашем ассортименте представлены ведущие мировые производители активных и пассивных электронных компонентов.

Нашей специализацией является поставка электронной компонентной базы двойного назначения, продукции таких производителей как XILINX, Intel (ex.ALTERA), Vicor, Microchip, Texas Instruments, Analog Devices, Mini-Circuits, Amphenol, Glenair.

Сотрудничество с глобальными дистрибьюторами электронных компонентов, предоставляет возможность заказывать и получать с международных складов практически любой перечень компонентов в оптимальные для Вас сроки.

На всех этапах разработки и производства наши партнеры могут получить квалифицированную поддержку опытных инженеров.

Система менеджмента качества компании отвечает требованиям в соответствии с ГОСТ Р ИСО 9001, ГОСТ РВ 0015-002 и ЭС РД 009

Офис по работе с юридическими лицами:

105318, г.Москва, ул.Щербаковская д.3, офис 1107, 1118, ДЦ «Щербаковский»

Телефон: +7 495 668-12-70 (многоканальный)

Факс: +7 495 668-12-70 (доб.304)

E-mail: info@moschip.ru

Skype отдела продаж:
moschip.ru   moschip.ru_6
moschip.ru_4   moschip.ru_9