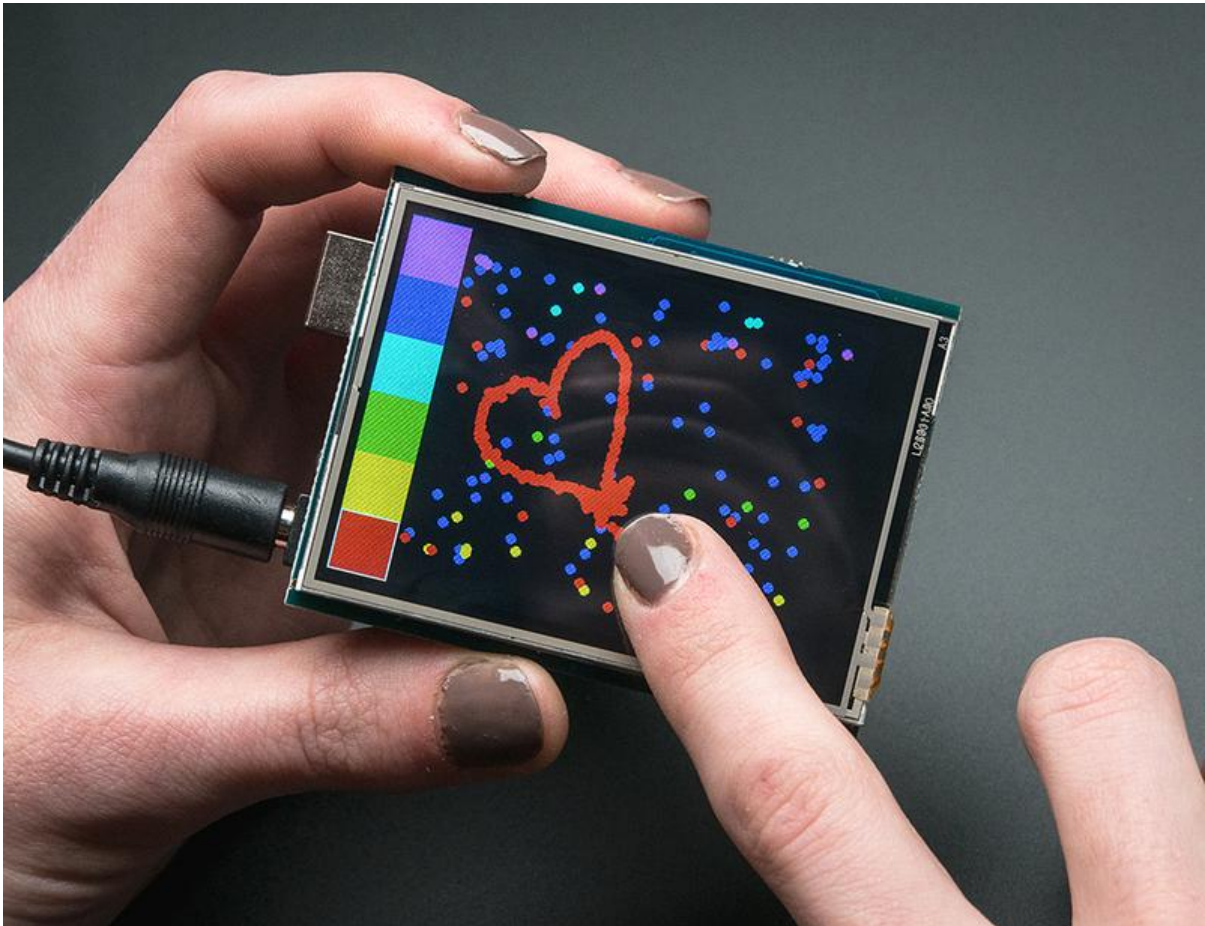




Adafruit 2.8" TFT Touch Shield v2 - Capacitive or Resistive

Created by lady ada



<https://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2>

Last updated on 2023-11-08 06:18:03 PM EST

Table of Contents

Overview	5
STMPE Resistive Pinout	8
<ul style="list-style-type: none">• Pinouts• TFT Screen Pins• Resistive Touch Controller Pins• Capacitive Touch Pins• MicroSD card Pins• Using with an Uno R3, Leonardo, Mega, etc• Using Capacitive Touch Version w/Older Arduino	
TSC2007 Resistive Pinout	12
<ul style="list-style-type: none">• STEMMA QT Connector• Default SPI Pins• ICSP Jumpers• TFT Backlight Jumper• microSD Card Slot• Touchscreen Interrupt Jumper• TSC2007 Address Jumpers• Reset Button• VddIO Jumper	
Arduino Graphics Test	15
<ul style="list-style-type: none">• Install Libraries• Install Adafruit ILI9341 TFT Library	
Adafruit GFX library	18
Drawing Bitmaps	18
STMPE Resistive Touchscreen Paint Demo	21
Arduino - Resistive TSC2007	24
<ul style="list-style-type: none">• Hardware Setup• Library Installation• Example Code	
Arduino Docs	29
Capacitive Touchscreen Paint Demo	29
<ul style="list-style-type: none">• Download the FT6206 Library• FT6206 Library Reference	
FT6206 Library Reference	31
CircuitPython Displayio Quickstart - Rev B	31
<ul style="list-style-type: none">• Preparing the Shield• Required CircuitPython Libraries• Code Example Additional Libraries• CircuitPython Code Example• Using Touch	

- [Where to go from here](#)

[CircuitPython - Resistive TSC2007](#) 40

- [Hardware Setup](#)
- [CircuitPython Usage](#)
- [Example Code](#)

[CircuitPython Docs](#) 44

[Backlight & Touch IRQ](#) 44

- [Controlling the Backlight](#)
- [Touchscreen Interrupt pin](#)

[Downloads](#) 45

- [Datasheets & Files](#)
- [Schematic and Fab Print](#)

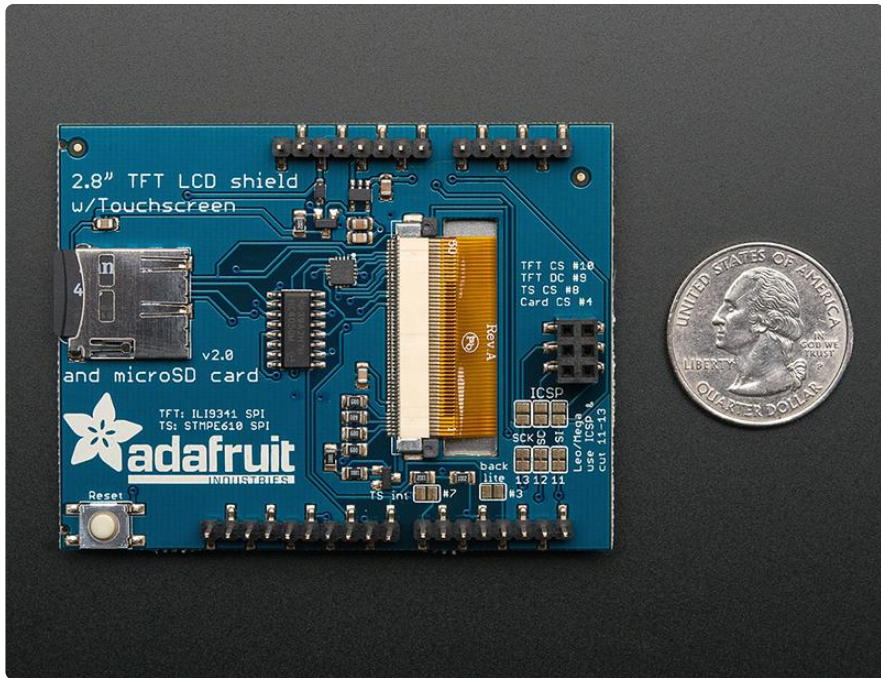
Overview



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 has a resistive touchscreen attached to it already, so you can detect finger presses anywhere on the screen. ([We also have a capacitive-touch version of this shield here \(http://adafru.it/1947\)](http://adafru.it/1947)).



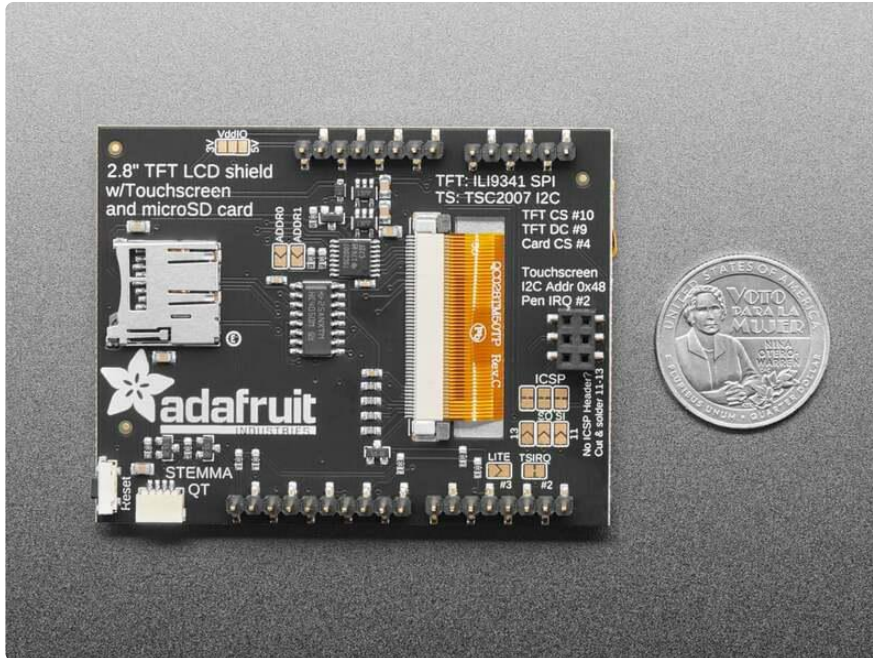
This shield uses the SPI port for sending data to the ILI9341 chipset display, and I2C for reading the TSC2007 resistive touch screen driver. That makes it work great with just about any board that is 'Arduino or Mega shaped' as long as they have the I2C SDA and SCL pins after pin D13, and the SPI 2x3 port. If you want to access the remaining pins, [we recommend a WingShield \(\)](#).



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 Adafruit Metro and classic Arduino (UNO, Duemilanove, Diecimila). Solder three jumpers and you can use it at full speed on an Arduino 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 only a few pins: five SPI pins for the display (SPI + DC and CS), I2C and IRQ pin for the touch screen, and an optional backlight pin. One more pin can be used for the micro SD card if you want to read images off of it.

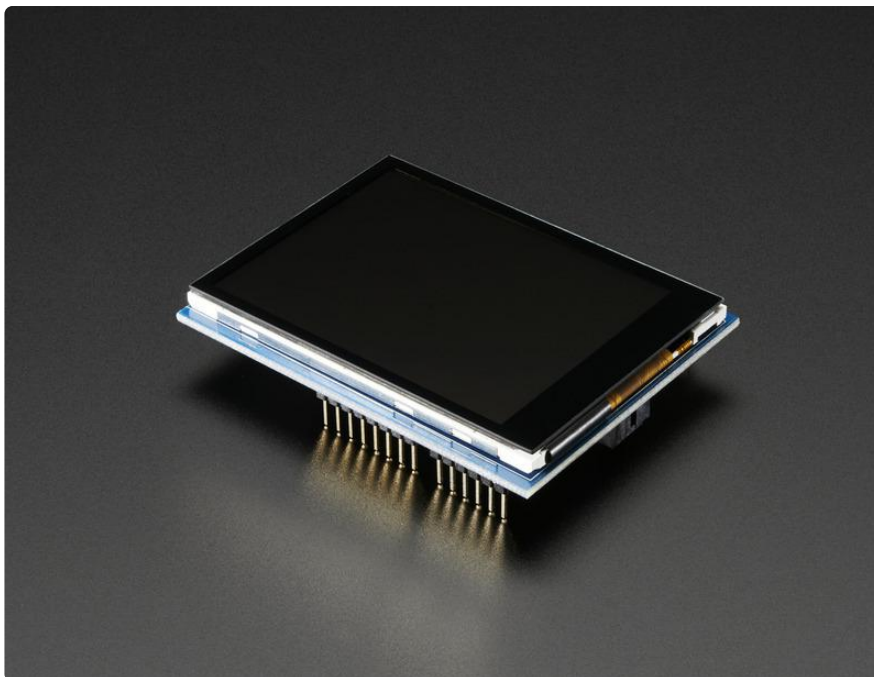
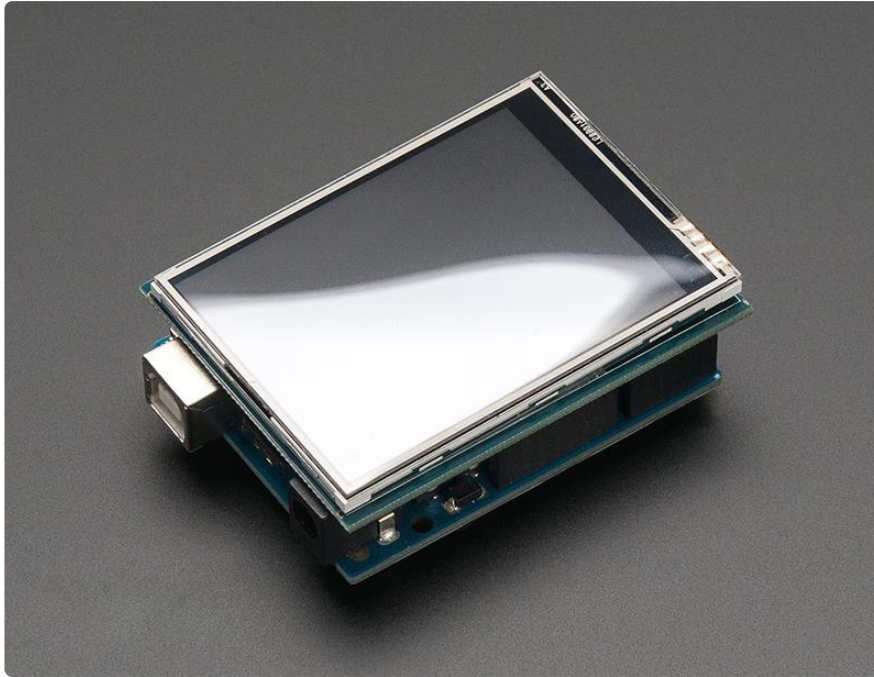


We recently did a big update to the two designs to add:

- Resistive touch screen TSC2007 Touch screen driver chip instead of STMPE811 (which was discontinued)
- Selectable VddIO for use with 3.3V or 5V chips - by default we use IOref but if your board does not have the IOref pin, you can solder the jumper closed for whatever voltage you like
- STEMMA QT port for quick connecting sensors and other I2C devices
- Right Angle Reset button is easier to press now

The shields are fully assembled, tested and ready to go. No wiring, no soldering! Simply plug it in and load up our libraries - you'll have it running in under 10 minutes! Works best with any classic Arduino or Mega -shaped board.

STMPE Resistive Pinout



For the v2 version of the resistive shield, see the next page TSC2007 Resistive Pinout

Pinouts

There are three versions of the shield. One has a capacitive touch, the other two are the older and newer versions with resistive touch. 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

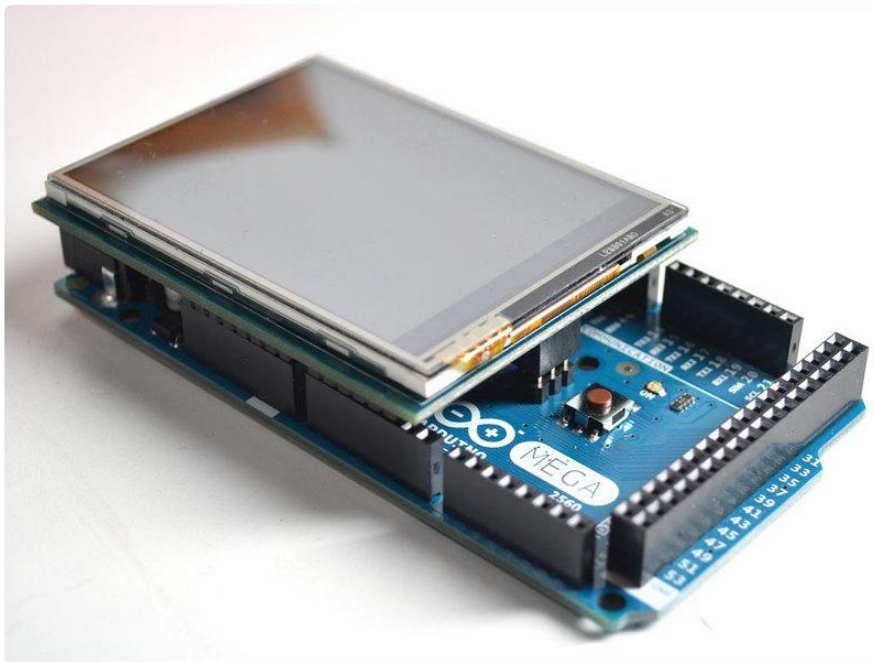
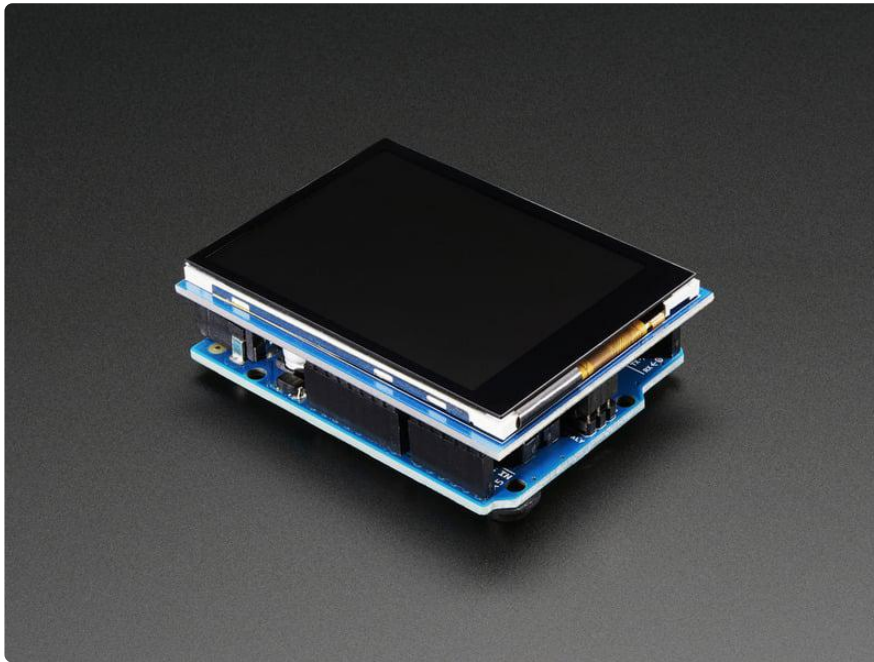
- Digital #13 or ICSP SCLK - This is the hardware SPI clock pin. By default its on the 2x3 header. By cutting a jumper and soldering another on the back, you can move this line from ICSP to digital #13. This pin is used for the TFT, microSD and resistive touch screen data clock
- Digital #12 or ICSP 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

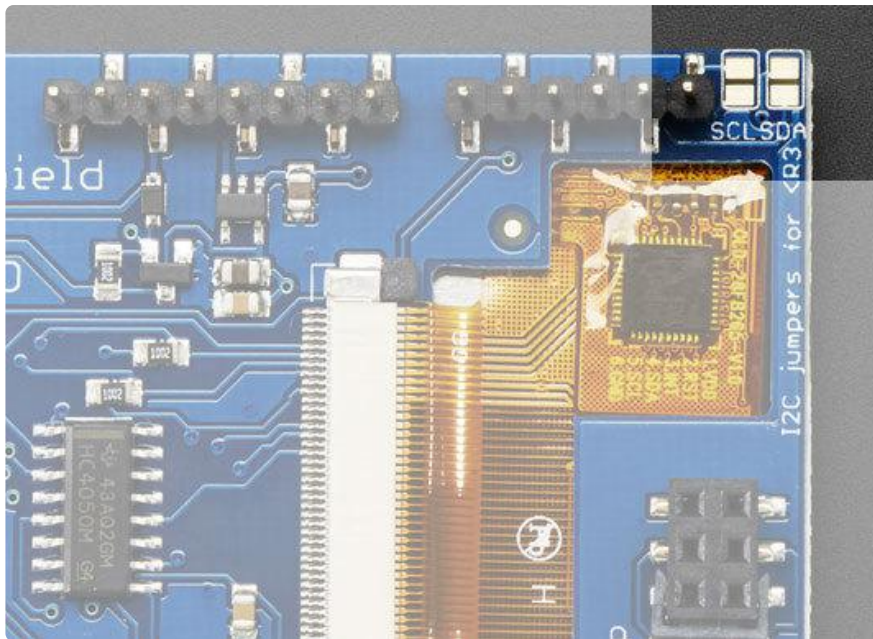
Because the TFT is about the same size as an Arduino, we pre-assemble the shield in the factory. For the Capacitive Touch, to use, simply place it onto your Arduino Uno/

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.

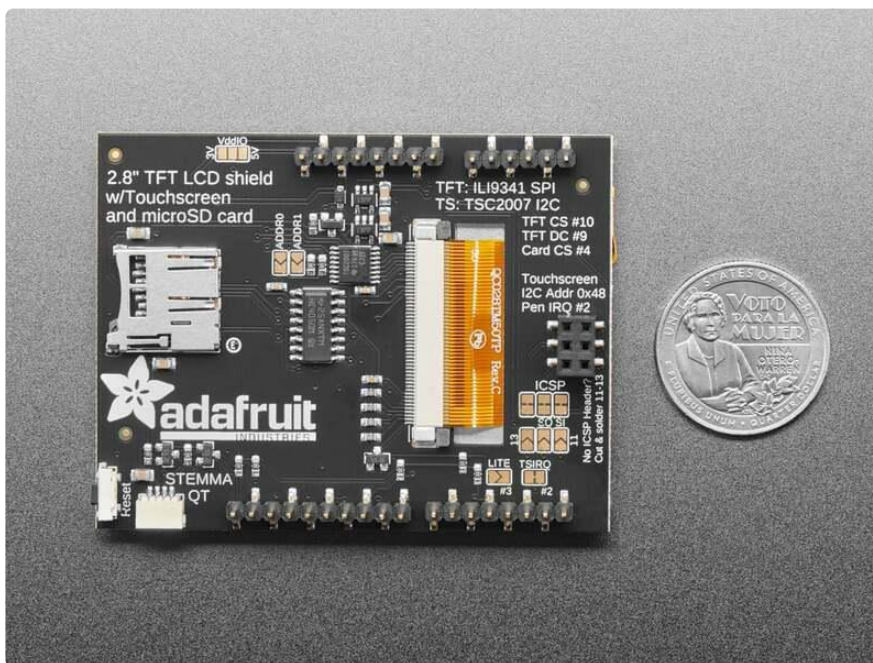


Using Capacitive Touch Version w/Older Arduino

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:



TSC2007 Resistive Pinout



The default I2C address of the TSC2007 is 0x48.

STEMMA QT Connector

- [STEMMA QT \(\)](#) - This connector, on the left side of the shield, allows you to connect to sensors and breakout boards with STEMMA QT / Qwiic connectors or to other things with [various associated accessories \(\)](#).
- SDA/SCL - The I2C pins for the STEMMA QT connector are connected to the I2C GPIO pins to the left of pin 13.

- 3.3V/GND - The power for the STEMMA QT connector is 3.3V. Ground is the common ground for power and logic.

Default SPI Pins

The TFT (connected to an ILI9341 chipset) and microSD card on the shield are controlled via SPI. The following SPI pins are connected by default with the ICSP 2x3 header:

- MOSI - This is the SPI MOSI (Microcontroller Out / Serial In) pin.
- MISO - This is the SPI MISO (Microcontroller In / Serial Out) pin.
- SCK - This is the SPI clock input pin.

The remaining SPI pins are connected to GPIO pins on the shield:

- TFT DC - This is the display SPI data/command selector pin. It is connected to pin 9.
- TFT CS - This is the TFT SPI chip select pin. It is connected to pin 10.
- SD Card CS - This is the SD card chip select pin. It is connected to pin 4.

ICSP Jumpers

If your board does not have a 2x3 ICSP header, or you want to use the GPIO SPI pins, you can cut the ICSP jumpers and solder the SPI jumpers closed.

- ICSP jumpers - These jumpers are located below the 2x3 header on the shield. They are labeled ICSP and outlined in white on the board silk. These jumpers can be cut to disconnect the SPI pins from the 2x3 header.
- SPI jumpers - These jumpers are located directly below the ICSP jumpers. They are labeled 13, SO, SI/11 and are outlined in white on the board silk. You can solder these jumpers closed to connect the SCK, MISO and MOSI pins to GPIO pins:
 - MOSI to pin 11
 - MISO to pin 12
 - SCK to pin 13

TFT Backlight Jumper

- LITE - The TFT backlight jumper is located next to the TSIRQ jumper and is labeled LITE/#3 on the board silk. You can solder this jumper closed to connect the optional backlight pin to pin 3.

microSD Card Slot

- On the back of the board, above the Adafruit logo on the board silk, is the microSD card slot. You can use any microSD card that supports SPI mode with one CS pin.

Touchscreen Interrupt Jumper

- TSIRQ - The touchscreen interrupt pin jumper is located below the SPI jumpers and is labeled TSIRQ/#2 on the board silk. You can cut this jumper to disconnect the TSC2007 interrupt pin from GPIO pin 2.

TSC2007 Address Jumpers

On the back of the board are two address jumpers, labeled ADDR0 and ADDR1, to the right of the board name label on the silk. These jumpers allow you to change the default I2C address of the TSC2007 on the shield. To do so, you solder the jumpers "closed" by connecting the two pads.

The default I2C address is 0x48. The other address options can be calculated by "adding" the ADDR0/ADDR1 to the base of 0x48.

ADDR0 sets the lowest bit with a value of 1, ADDR1 sets the next bit with a value of 2. The final address is $0x48 + A1 + A0$ which would be 0x4B.

If only ADDR0 is soldered closed, the address is $0x48 + 1 = 0x49$

If only ADDR1 is soldered closed, the address is $0x48 + 2 = 0x4A$

The table below shows all possible addresses, and whether the pin(s) should be high (closed) or low (open).

ADDR	A0	A1
0x48	L	L
0x49	H	L
0x4A	L	H
0x4B	H	H

Reset Button

- Reset - The reset button, located to the left of the STEMMA QT connector, is connected to the reset pin. It is mounted at a right angle so that it is easier to press.

VddIO Jumper

- VddIO - The VddIO jumper is located at the top of the shield above the shield name label on the board silk. The shield utilizes the IOREF pin on the attached board to supply voltage to all of the components. If your board does not have an IOREF pin, you can manually select the voltage by soldering the jumper closed.
 - For 3.3V, you'll solder the 3V pad on the left to the center pad.
 - For 5V, you'll solder the 5V pad on the right to the center pad.
- You'll want to make sure that your selected voltage on the VddIO jumper matches the logic level of your board, e.g. for a 3V board like an RP2040 Metro, use 3V, and for a 5V board like an ATmega328-based Arduino Uno, use 5V.

Arduino Graphics Test

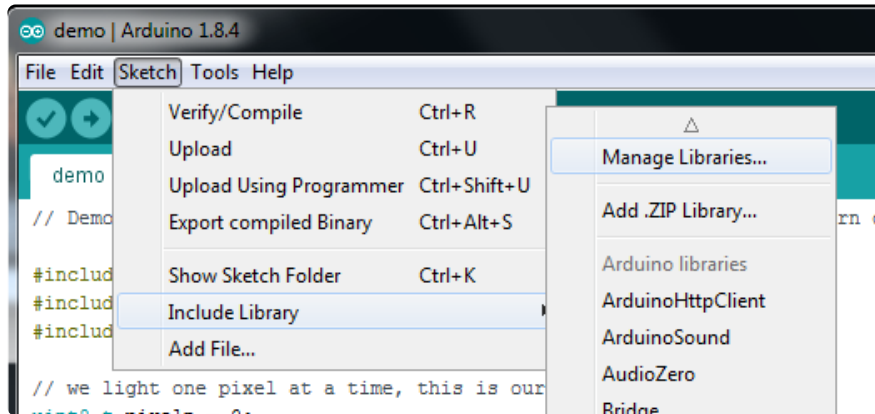
We have a library with example code ready to go for use with these TFTs. The library is not incredibly fast and optimized but its a good start and can easily be ported to other microcontrollers. However, we'll assume you're using an Arduino.

[Our github repository \(\)](#) 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...

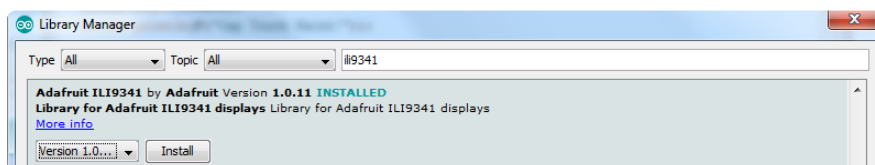


Install Adafruit ILI9341 TFT Library

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 \(\)](#) (this contains the low-level code specific to this device), and second is the [Adafruit GFX Library \(\)](#) (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

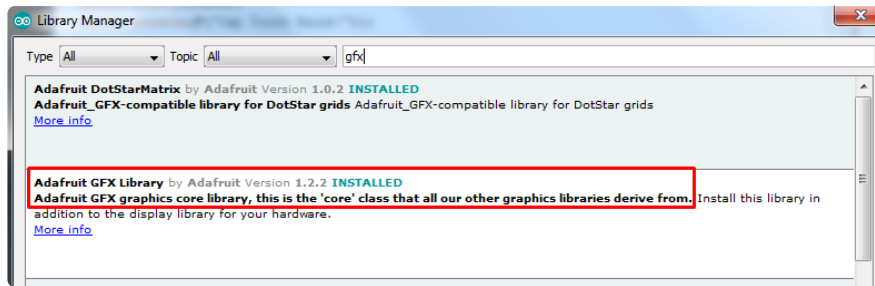
Search for ILI9341 and install the Adafruit ILI9341 library that pops up!



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 \(\)](http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use)

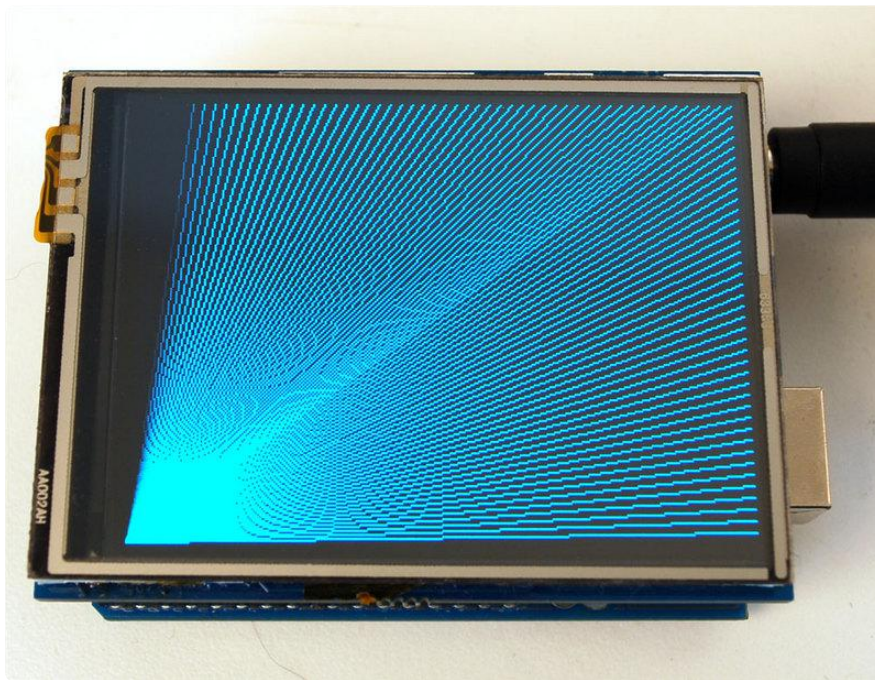
Next up, search for Adafruit GFX and locate the core library. A lot of libraries may pop up because we reference it in the description so just make sure you see Adafruit GFX Library in bold at the top.

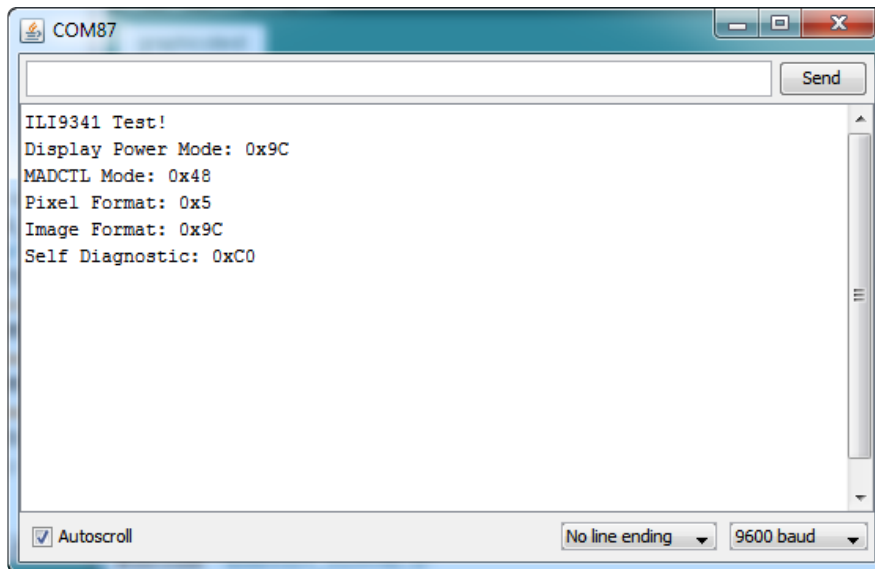
Install it!



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

Restart the Arduino software. You should see a new example folder called Adafuit_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.





Adafruit GFX library

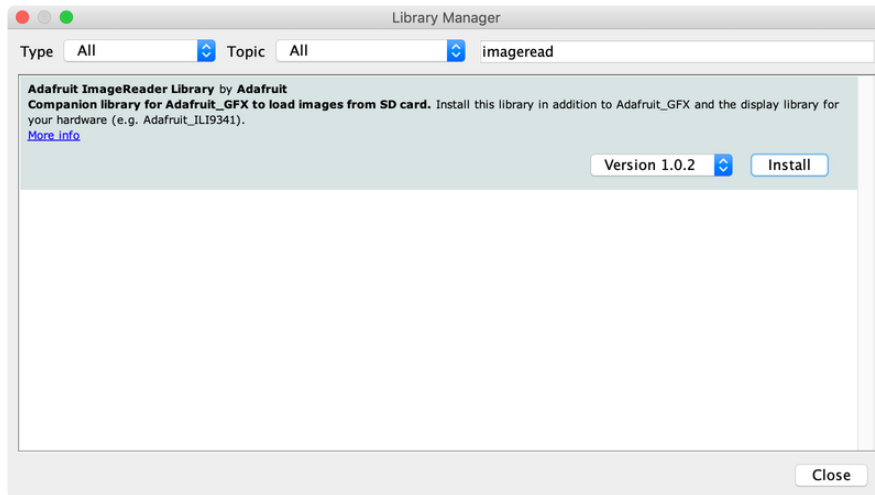
The TFT LCD library is based off of the Adafruit GFX graphics core library. GFX has many ready to go functions that should help you start out with your project. Its not exhaustive and we'll try to update it if we find a really useful function. Right now it supports pixels, lines, rectangles, circles, 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 \(\)!](#)

Drawing Bitmaps

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→ShieldILI9341 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.

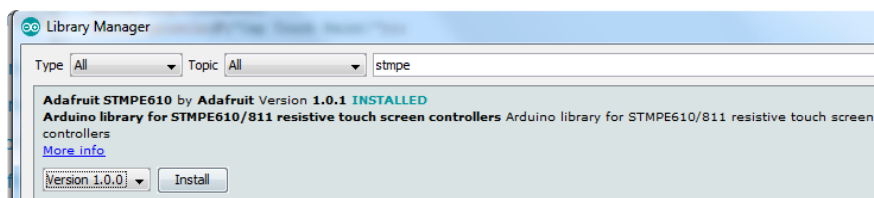
[Image loading is explained in greater depth in the Adafruit_GFX library guide. \(\)](#)

STMPE Resistive Touchscreen Paint Demo

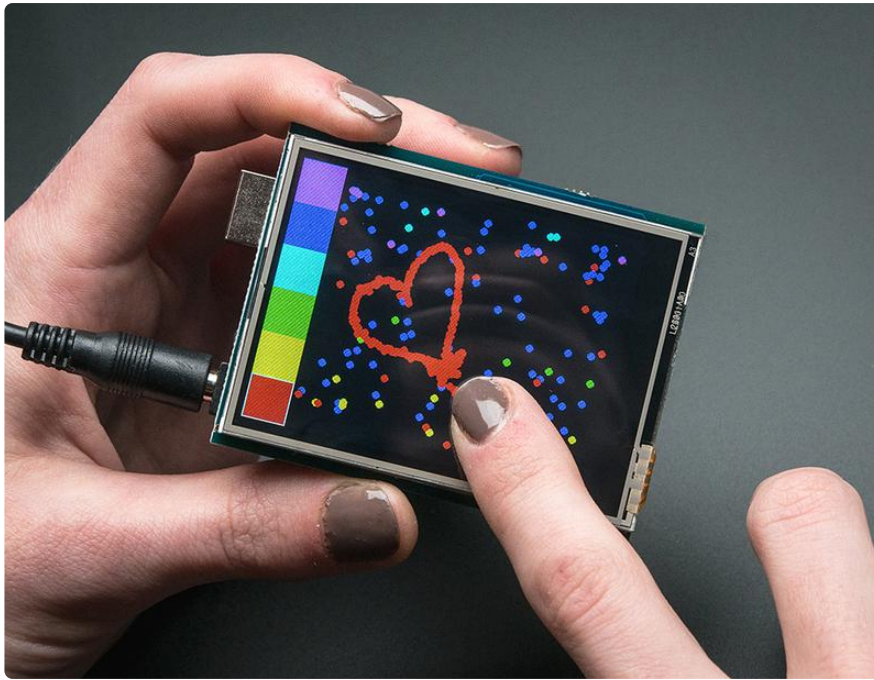
This page is for the Resistive Touch Screen version of the Shield with STMPE chipset

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 \(\)](#) - 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 will certainly work. Do not press harder and harder until the screen cracks!

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.

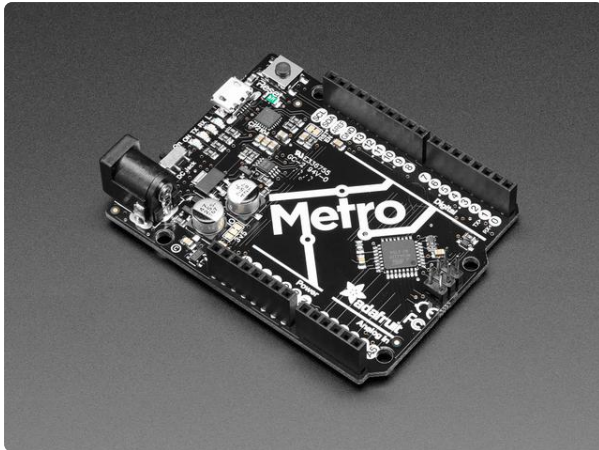
One last point (pun intended!) since the touchscreen driver stores points in a buffer, you may want to ask the driver "is the touchscreen being pressed RIGHT NOW?" You can do that with

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

Arduino - Resistive TSC2007

Using the 2.8" TFT Touch Shield v2 with Arduino involves plugging the shield into your classic Arduino or Mega-shaped board. Then, you'll install the necessary libraries and upload the example code to the board.

This page uses the Metro 328 for demonstrating Arduino usage. You can use the same concepts to get going with any classic Arduino or Mega-shaped board.

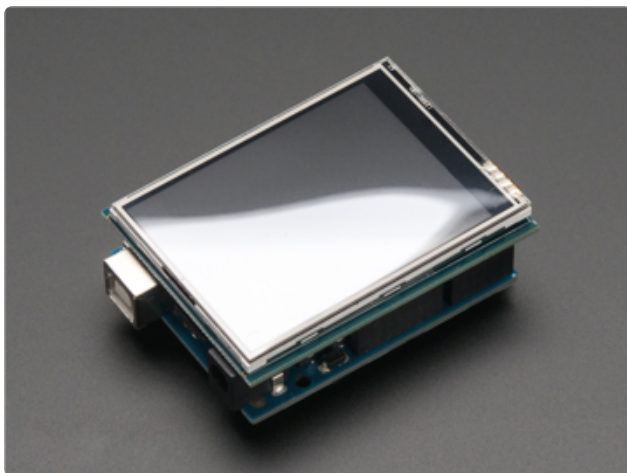


[Adafruit METRO 328 - Arduino Compatible - with Headers](https://www.adafruit.com/product/2488)

This is the Adafruit METRO Arduino-Compatible - with headers. It's a fully assembled and tested microcontroller and physical computing board with...

<https://www.adafruit.com/product/2488>

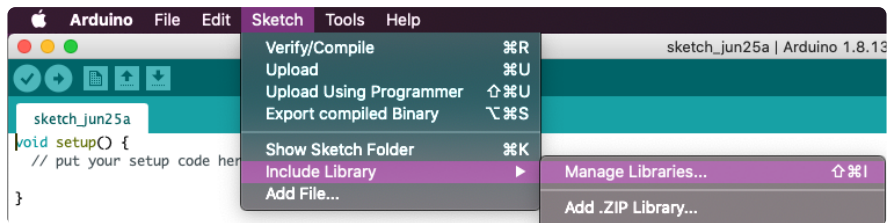
Hardware Setup



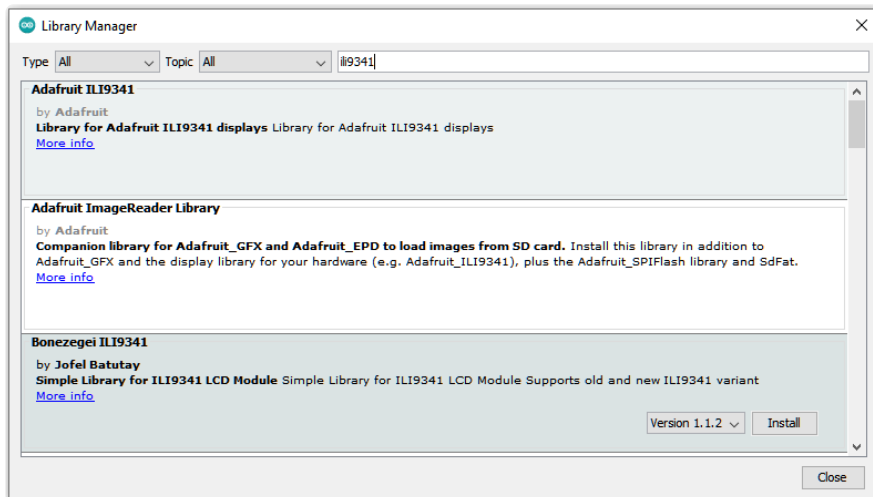
Plug the shield into your Arduino or Mega-shaped board. The GPIO pins are keyed and should only plug in one way.

Library Installation

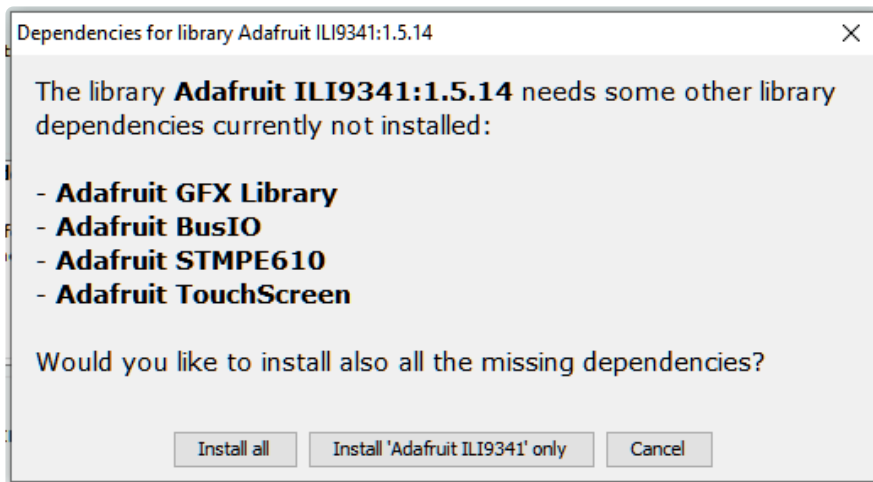
You can install the Adafruit ILI9341 library for Arduino using the Library Manager in the Arduino IDE.



Click the Manage Libraries... menu item, search for Adafruit ILI9341, and select the Adafuit_ILI9341 library:

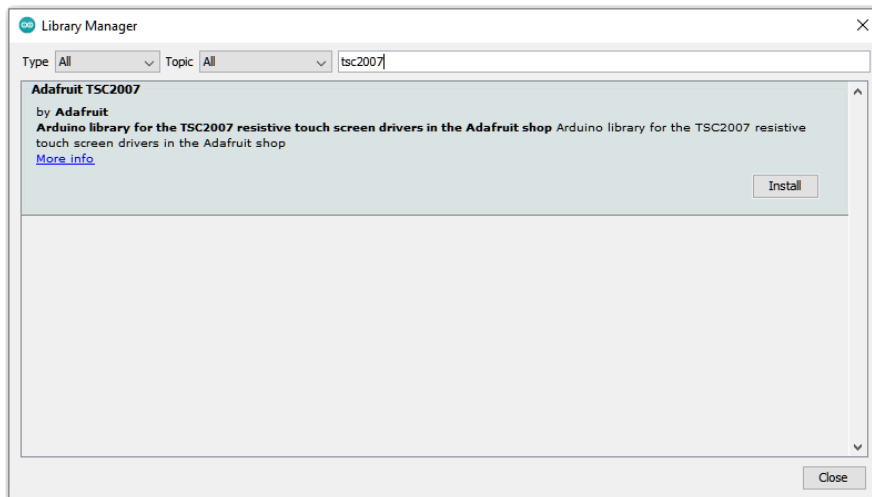


If asked about dependencies, click "Install all".

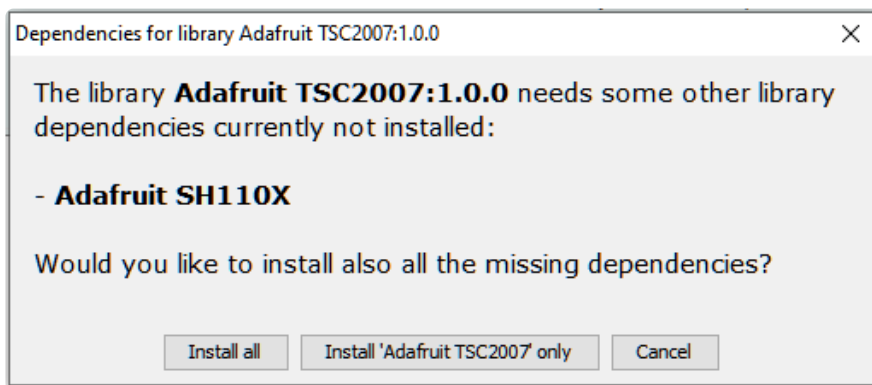


If the "Dependencies" window does not come up, then you already have the dependencies installed.

Then you'll install the Adafruit TSC2007 library for the touch screen. Click the Manage Libraries... menu item, search for Adafruit TSC2007, and select the Adafruit TSC2007 library:



If asked about dependencies, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

If the dependencies are already installed, you must make sure you update them through the Arduino Library Manager before loading the example!

Example Code

```
// SPDX-FileCopyrightText: 2023 Limor Fried/Ladyada for Adafruit Industries
// SPDX-License-Identifier: MIT

/*****
This is our touchscreen painting example for the updated Adafruit
ILI9341 Shield with TSC2007
----> http://www.adafruit.com/products/1651

Check out the links above for our tutorials and wiring diagrams
These displays use SPI to communicate, 4 or 5 pins are required to
interface (RST is optional)
Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.
MIT license, all text above must be included in any redistribution
*****/
```

```

*****/

#include <Adafruit_GFX.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_TSC2007.h>

// This is calibration data for the raw touch data to the screen coordinates
#define TS_MINX 150
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000
#define TS_MIN_PRESSURE 200

Adafruit_TSC2007 ts;

// The display also uses hardware SPI, plus #9 & #10
#define TFT_CS 10
#define TFT_DC 9
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

// Size of the color selection boxes and the paintbrush size
#define BOXSIZE 40
#define PENRADIUS 3
int oldcolor, currentcolor;

void setup(void) {

  Serial.begin(115200);
  // while (!Serial) delay(10);

  tft.begin();

  if (!ts.begin()) {
    Serial.println("Couldn't start touchscreen controller");
    while (1);
  }
  Serial.println("Touchscreen started");

  tft.fillScreen(ILI9341_BLACK);

  // make the color selection boxes
  tft.fillRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_RED);
  tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_YELLOW);
  tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_GREEN);
  tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_CYAN);
  tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_BLUE);
  tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_MAGENTA);

  // select the current color 'red'
  tft.drawRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  currentcolor = ILI9341_RED;
}

void loop(){
  uint16_t x, y, z1, z2;
  if (ts.read_touch(&x, &y, &z1, &z2) && (z1 > TS_MIN_PRESSURE)) {

    Serial.print("Touch point: (");
    Serial.print(x); Serial.print(", ");
    Serial.print(y); Serial.print(", ");
    Serial.print(z1); Serial.print(" / ");
    Serial.print(z2); Serial.println(")");

    // Scale from ~0->4000 to tft.width using the calibration #'s
    x = map(x, TS_MINX, TS_MAXX, 0, tft.width());
    y = map(y, TS_MINY, TS_MAXY, 0, tft.height());
  }
}

```

```

if (y < BOXSIZE) {
  oldcolor = currentcolor;

  if (x < BOXSIZE) {
    currentcolor = ILI9341_RED;
    tft.drawRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  } else if (x < BOXSIZE*2) {
    currentcolor = ILI9341_YELLOW;
    tft.drawRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  } else if (x < BOXSIZE*3) {
    currentcolor = ILI9341_GREEN;
    tft.drawRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  } else if (x < BOXSIZE*4) {
    currentcolor = ILI9341_CYAN;
    tft.drawRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  } else if (x < BOXSIZE*5) {
    currentcolor = ILI9341_BLUE;
    tft.drawRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  } else if (x < BOXSIZE*6) {
    currentcolor = ILI9341_MAGENTA;
    tft.drawRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_WHITE);
  }

  if (oldcolor != currentcolor) {
    if (oldcolor == ILI9341_RED)
      tft.fillRect(0, 0, BOXSIZE, BOXSIZE, ILI9341_RED);
    if (oldcolor == ILI9341_YELLOW)
      tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, ILI9341_YELLOW);
    if (oldcolor == ILI9341_GREEN)
      tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, ILI9341_GREEN);
    if (oldcolor == ILI9341_CYAN)
      tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, ILI9341_CYAN);
    if (oldcolor == ILI9341_BLUE)
      tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, ILI9341_BLUE);
    if (oldcolor == ILI9341_MAGENTA)
      tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, ILI9341_MAGENTA);
  }
}
if (((y-PENRADIUS) > BOXSIZE) && ((y+PENRADIUS) < tft.height())) {
  tft.fillCircle(x, y, PENRADIUS, currentcolor);
}
}
}
}

```

```

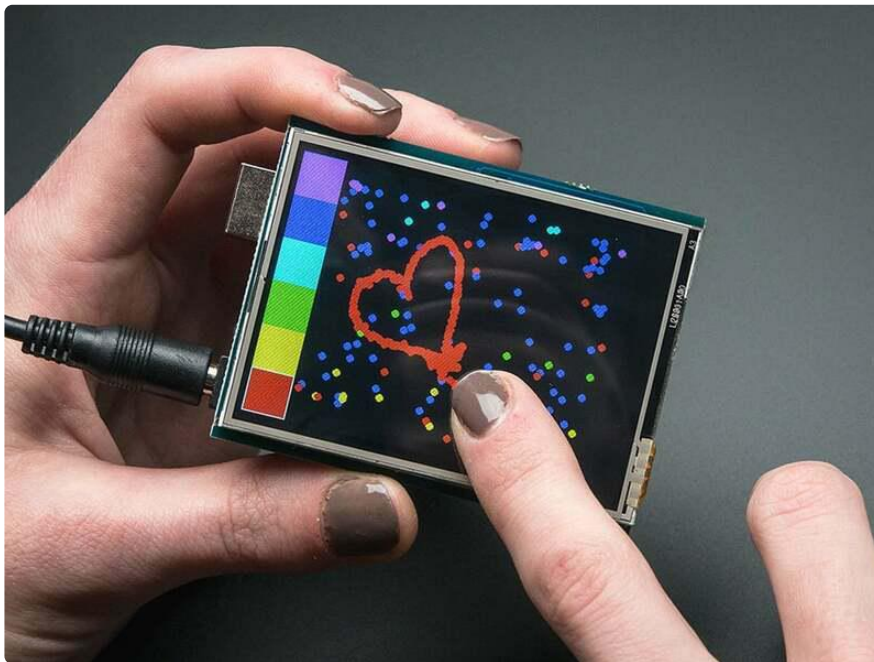
COM20
Touchscreen started
Touch point: (803, 1705, 213 / 2824)
Touch point: (812, 1697, 224 / 2747)
Touch point: (818, 1693, 222 / 2810)
Touch point: (825, 1692, 220 / 2800)
Touch point: (926, 1615, 221 / 2933)
Touch point: (938, 1618, 240 / 2787)
Touch point: (943, 1609, 213 / 2921)
Touch point: (1050, 1524, 228 / 2843)
Touch point: (1052, 1521, 254 / 2812)
Touch point: (1208, 1477, 281 / 2809)
Touch point: (1219, 1481, 318 / 2723)
Touch point: (1272, 1483, 307 / 2772)
Touch point: (1343, 1433, 335 / 2720)
Touch point: (1359, 1425, 370 / 2606)
Touch point: (1365, 1419, 379 / 2554)
Touch point: (1372, 1411, 377 / 2593)
Touch point: (1381, 1398, 374 / 2602)
Touch point: (1430, 1368, 368 / 2608)
Touch point: (1497, 1347, 369 / 2648)
Touch point: (1502, 1348, 398 / 2566)
Touch point: (1502, 1337, 408 / 2555)

```

Upload the sketch to your board and open up the Serial Monitor (Tools -> Serial Monitor) at 115200 baud. In the Serial Monitor, you should see the values from the touch screen being printed out.

The first number is the X coordinate, the second number is the Y coordinate and the last two numbers are Z "pressure" coordinates that can tell you how hard the touch pad is being pressed.

On the TFT, you'll be able to doodle with the different colors on the left side of the screen.

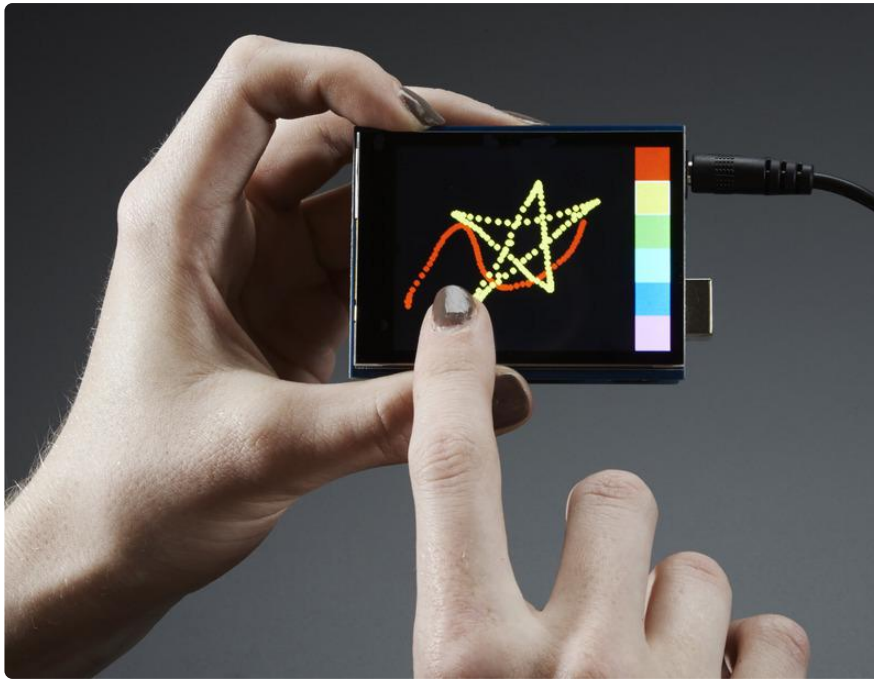


Arduino Docs

[Arduino Docs \(\)](#)

Capacitive Touchscreen Paint Demo

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

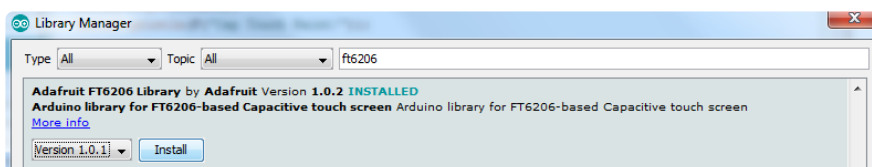


We now have a super-fancy capacitive touch screen version of this shield. Instead of a resistive controller that needs calibration and pressing down, the capacitive has a hard glass cover and can be used with a gentle fingertip. It is a single-touch capacitive screen only!

The capacitive touch screen controller communicates over I2C, which uses two hardware 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 \(\)](#) - 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)` with a number from 0-255 to set the touch threshold. The default works pretty well but if you're having too much sensitivity (or not enough) 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!

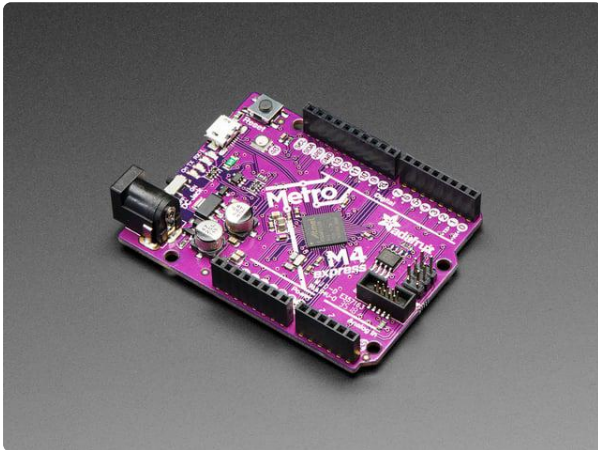
FT6206 Library Reference

[FT6206 Library Reference \(\)](#)

CircuitPython Displayio Quickstart - Rev B

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 ATSAM51 \(\)](#) guide.

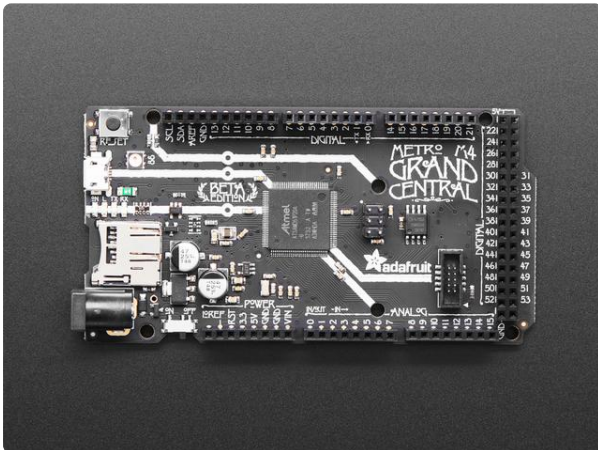


Adafruit Metro M4 feat. Microchip ATSAM51

Are you ready? Really ready? Cause here comes the fastest, most powerful Metro ever. The Adafruit Metro M4 featuring the Microchip ATSAM51. This...

<https://www.adafruit.com/product/3382>

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 \(\)](#) guide.



Adafruit Grand Central M4 Express featuring the SAM51

Are you ready? Really ready? Cause here comes the Adafruit Grand Central featuring the Microchip ATSAM51. This dev board is so big, it's not...

<https://www.adafruit.com/product/4064>

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 \(\)](#) guide.



Adafruit Metro M4 Express AirLift (WiFi) - Lite

Give your next project a lift with AirLift - our witty name for the ESP32 co-processor that graces this Metro M4. You already know about the Adafruit Metro...

<https://www.adafruit.com/product/4000>

Preparing the Shield

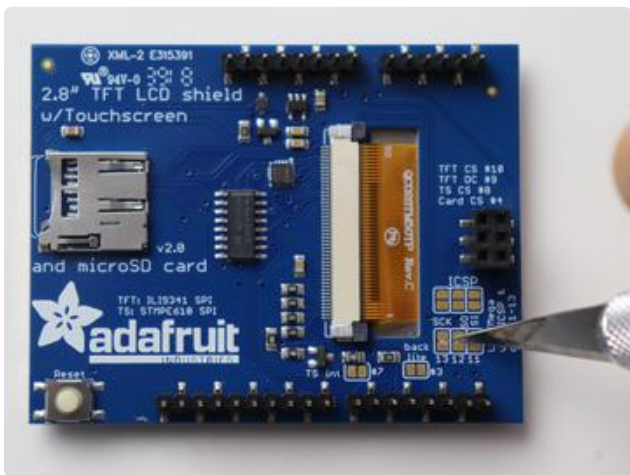
2.8" Capacitive TFT Touch Shield

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.

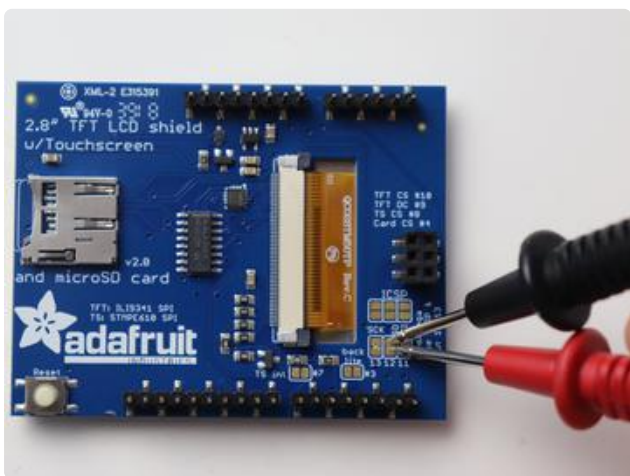
The following steps are not necessary with the newer v2 version of the shield!

2.8" Resistive TFT Touch Shield v1

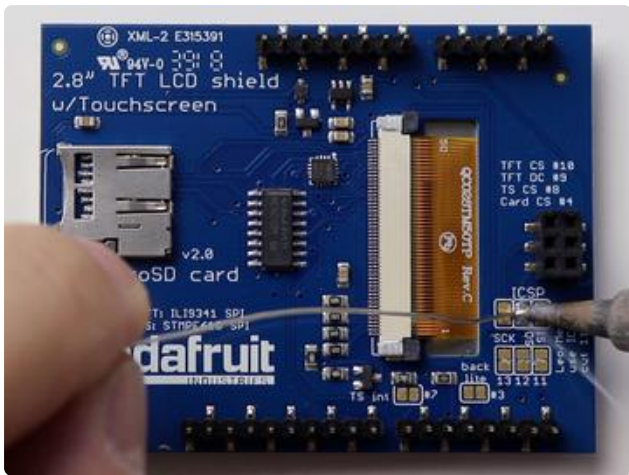
The earlier v1 version of the shield will need a slight modification to use with our boards capable of running CircuitPython. This is because the SPI interface is only available through the ICSP header.



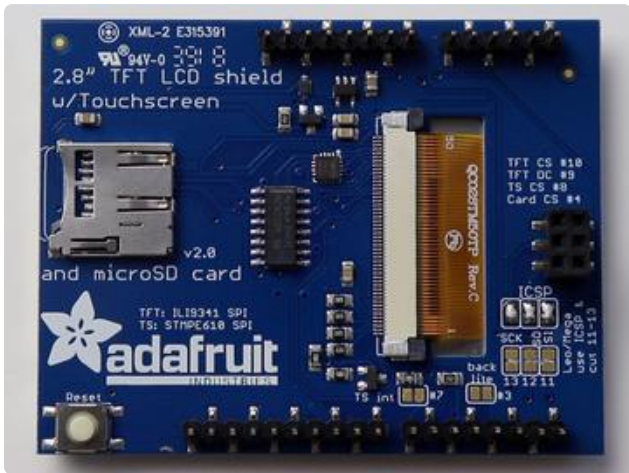
First start by taking a sharp knife and cutting the traces that connect to pins 11-13.



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.

`Adafruit_CircuitPython_ILI9341`

First, make sure you are running the [latest version of Adafruit CircuitPython \(\)](#) 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 \(\)](#). Our introduction guide has [a great page on how to install the library bundle \(\)](#) 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.

Adafruit_CircuitPython_Display_Text

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

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
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
import fourwire
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 = fourwire.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()
display.root_group = 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(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

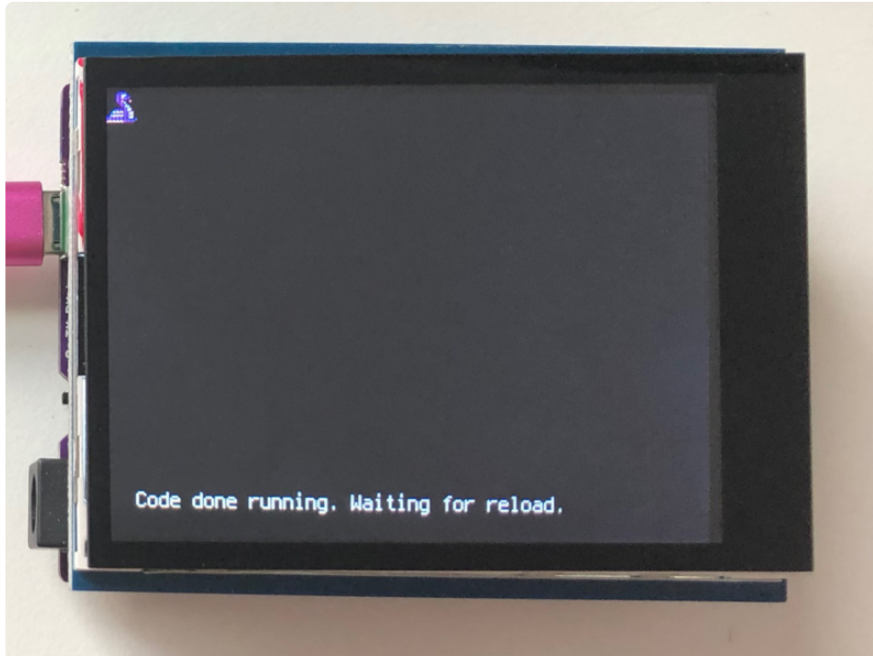
```

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.

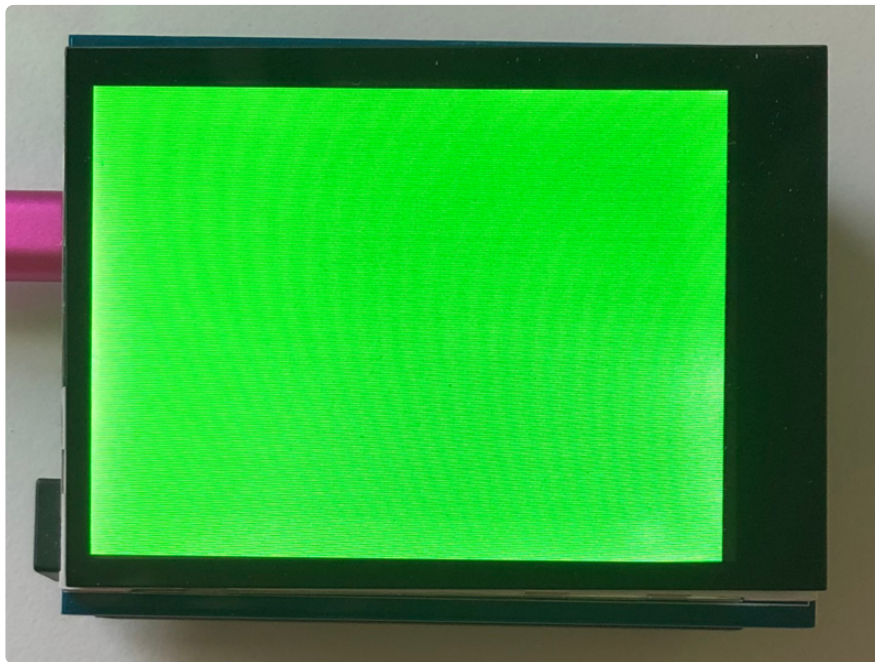
```
splash = displayio.Group(max_size=10)
display.show(splash)
```

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 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 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 \(\)](#) library.
- For enabling resistive touch use the [Adafruit_CircuitPython_STMPE610 \(\)](#) library.

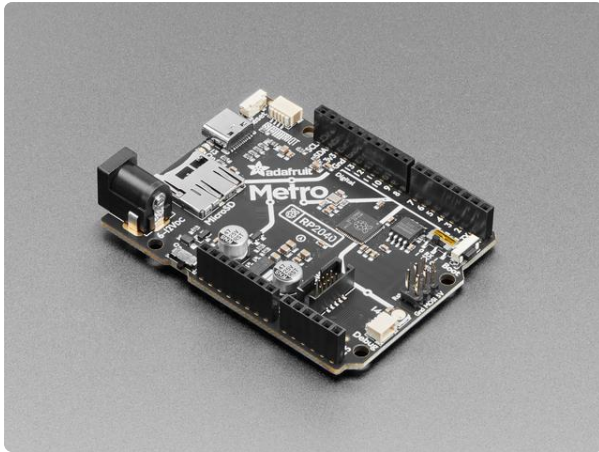
Where to go from here

Be sure to check out this excellent [guide to CircuitPython Display Support Using displayio \(\)](#)

CircuitPython - Resistive TSC2007

Using the 2.8" TFT Touch Shield v2 with CircuitPython involves plugging the shield into your classic Arduino or Mega-shaped board. Then, you load the code and necessary libraries onto your board to run the example.

This page uses the Metro RP2040 for demonstrating CircuitPython usage. You can use the same concepts to get going with any classic Arduino or Mega-shaped board.

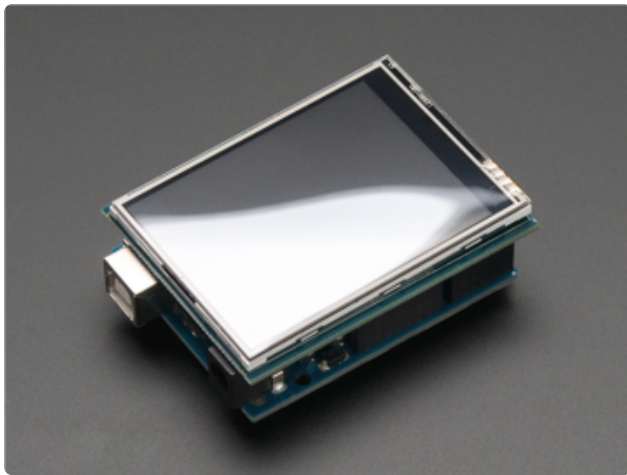


Adafruit Metro RP2040

Choo! Choo! This is the RP2040 Metro Line, making all station stops at "Dual Cortex M0+ mountain", "264K RAM round-about" and "16 Megabytes of Flash..."

<https://www.adafruit.com/product/5786>

Hardware Setup



Plug the shield into your Arduino or Mega-shaped board. The GPIO pins are keyed and should only plug in one way.

CircuitPython Usage

To use with CircuitPython, you need to first install the necessary libraries, and their dependencies, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

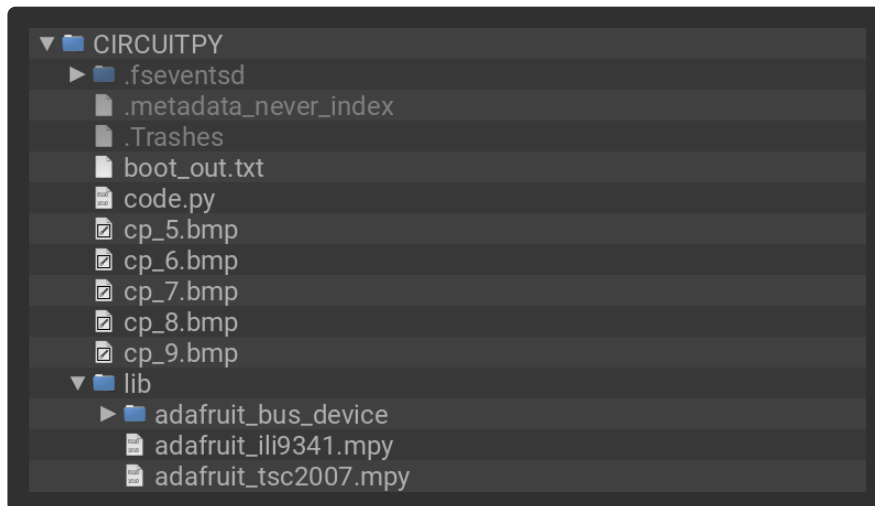
Thankfully, we can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file.

Connect the microcontroller to your computer via a known-good USB power+data cable. The board shows up as a thumb drive named CIRCUITPY. Copy the entire lib folder, the bitmap image files, and the code.py file to your CIRCUITPY drive.

Your CIRCUITPY/lib folder should contain the following folder and files:

- /adafruit_bus_device
- adafruit_ili9341.mpy
- adafruit_tsc2007.mpy

Once you have copied over the necessary folders and files, your CIRCUITPY drive should resemble the following:



Example Code

```
# SPDX-FileCopyrightText: 2023 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
This test will initialize the display using displayio and display
a bitmap image. The image advances when the touch screen is touched.

Pinouts are for the 2.8" TFT Shield
"""
import os
import board
import displayio
import adafruit_ili9341
import adafruit_tsc2007

# 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_width = 320
```

```

display_height = 240

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

i2c = board.I2C()

irq_dio = None
tsc = adafruit_tsc2007.TSC2007(i2c, irq=irq_dio)

groups = []
images = []
for filename in os.listdir('/'):
    if filename.lower().endswith('.bmp') and not filename.startswith('.'):
        images.append("/"+filename)
print(images)

for i in range(len(images)):
    splash = displayio.Group()
    bitmap = displayio.OnDiskBitmap(images[i])
    tile_grid = displayio.TileGrid(bitmap, pixel_shader=bitmap.pixel_shader)
    splash.append(tile_grid)
    groups.append(splash)

index = 0
touch_state = False

display.show(groups[index])
while True:
    if tsc.touched and not touch_state:
        point = tsc.touch
        touch_state = True
        if point["pressure"] < 200: # ignore touches with no 'pressure' as false
            continue
        print("Touchpoint: (%d, %d, %d)" % (point["x"], point["y"],
point["pressure"]))
        # left side of the screen
        if point["y"] < 2000:
            index = (index - 1) % len(images)
            display.show(groups[index])
        # right side of the screen
        else:
            index = (index + 1) % len(images)
            display.show(groups[index])
    if not tsc.touched and touch_state:
        touch_state = False

```

Once everything is saved to the CIRCUITPY drive, [connect to the serial console \(\)](#) to see the data printed out!

```

CircuitPython REPL
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
['/cp_9.bmp', '/cp_5.bmp', '/cp_6.bmp', '/cp_7.bmp', '/cp_8.bmp']
Touchpoint: (1994, 1119, 427)
Touchpoint: (1284, 3229, 293)
Touchpoint: (1933, 1804, 622)
Touchpoint: (1827, 1425, 418)
Touchpoint: (1785, 1339, 361)
Touchpoint: (1790, 1624, 471)
Touchpoint: (1622, 1808, 399)
Touchpoint: (1677, 1856, 544)
Touchpoint: (2113, 1997, 590)

```

The code will open all of the bitmap files that are on the CIRCUITPY drive and add them to the `images` array. The first image will then be displayed on the TFT. In the

loop, if you touch the screen you'll see the coordinates and pressure print to the serial console.

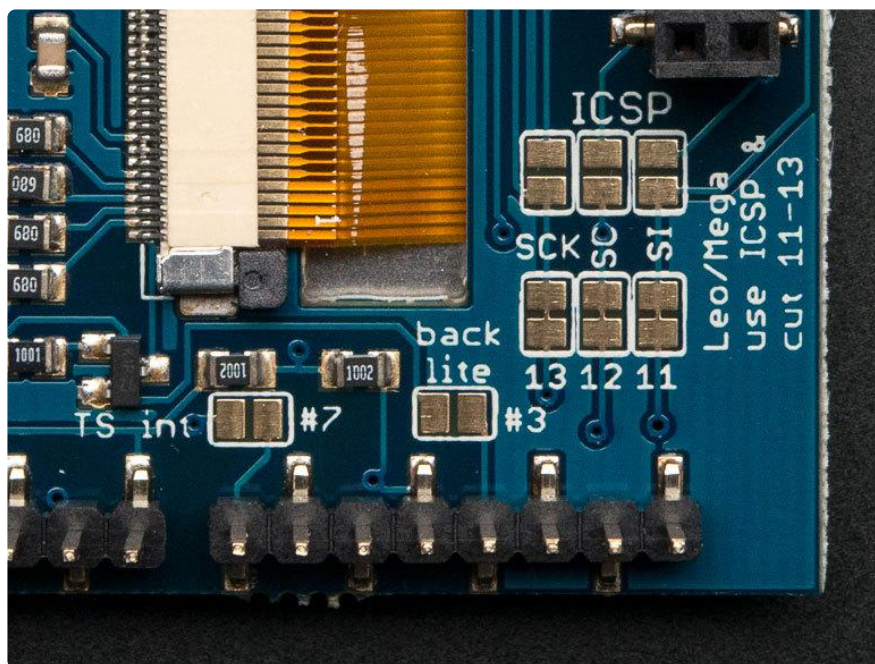
If you press on the left side of the screen, the display will show the previous bitmap in the array. If you press on the right side of the screen, the display will show the next bitmap in the array.

CircuitPython Docs

[CircuitPython Docs \(\)](#)

Backlight & Touch IRQ

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. \(\)](#) 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!

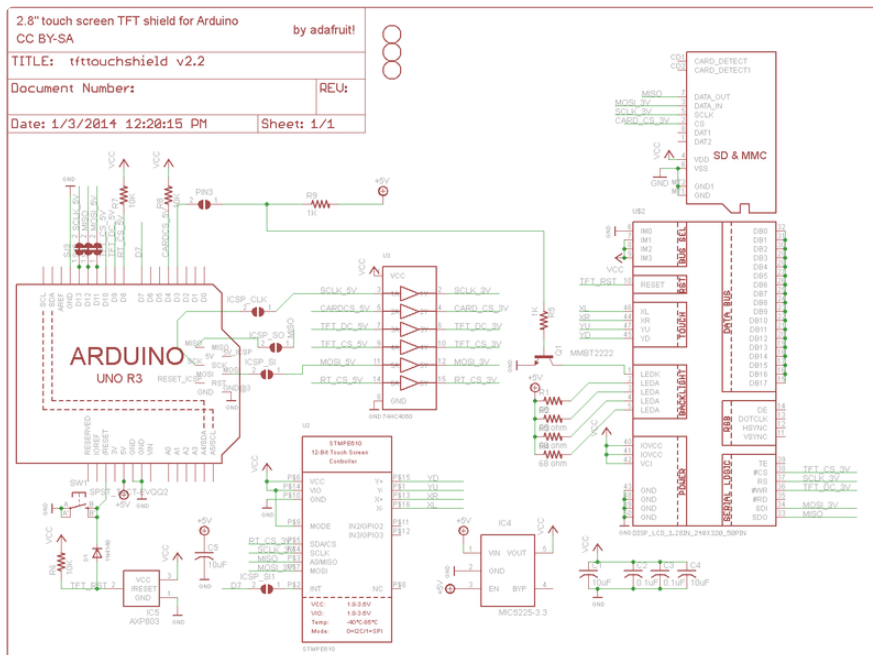
Downloads

Datasheets & Files

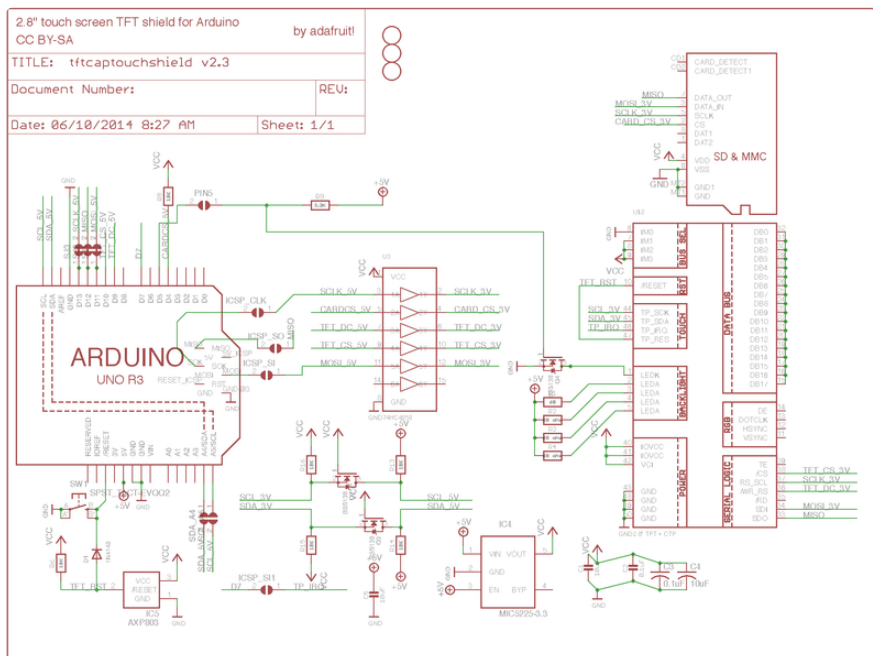
- [STMPE610 Datasheet \(\)](#)
- [TSC2007 Datasheet \(\)](#)
- [ILI9341 \(TFT controller\) \(\)](#)
- [Raw 2.8" Resistive TFT datasheet \(\)](#)
- [Raw 2.8" Capacitive TFT datasheet \(\)](#)
- [FT6206 Datasheet \(\)](#) & [App note \(\)](#) (capacitive chip)
- [PCB CAD files for both resistive and capacitive versions on GitHub \(\)](#)
- [Fritzing file for the Capacitive TFT Touch Shield v2 \(\)](#)
- [Fritzing file for the Resistive TFT Touch Shield v2 \(\)](#)
- [Fritzing file for the Resistive TFT Touch Shield v2 Rev E \(\)](#)

Schematic and Fab Print

Rev B Resistive Touch Shield

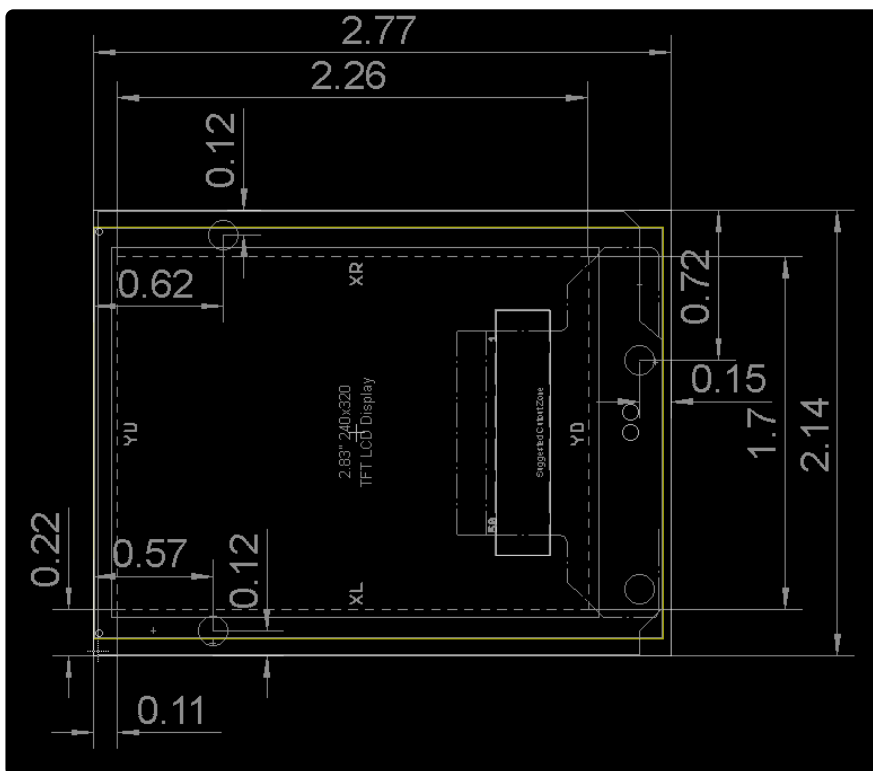


Rev B Capacitive Touch Shield



Rev B Capacitive and Resistive Fab Print

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)



Rev E Schematic and Fab Print - Resistive

