Adafruit 3.5" 320x480 Color TFT Touchscreen Breakout

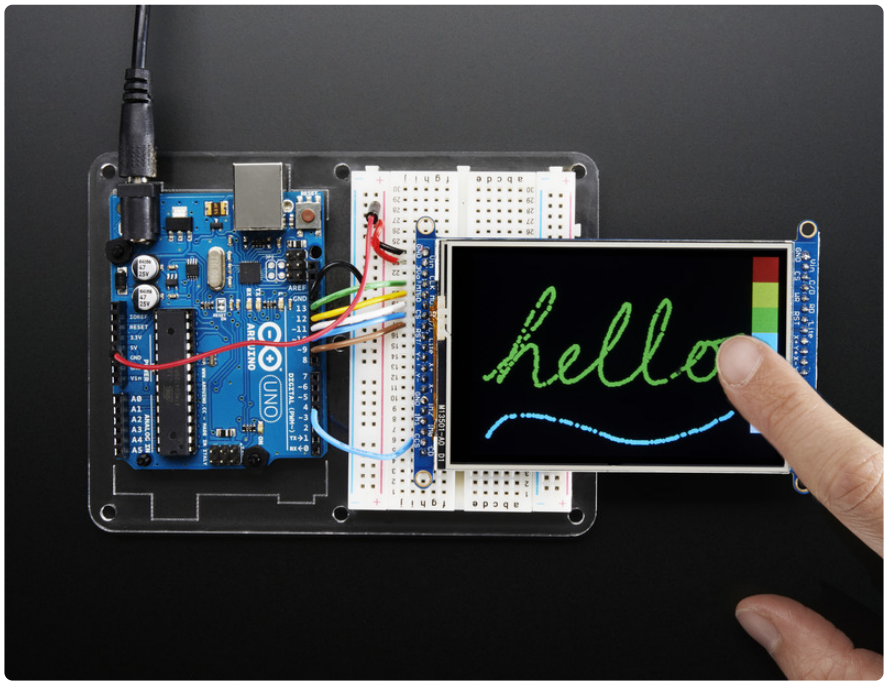
Created by lady ada

https://learn.adafruit.com/adafruit-3-5-color-320x480-tft-touchscreen-breakout

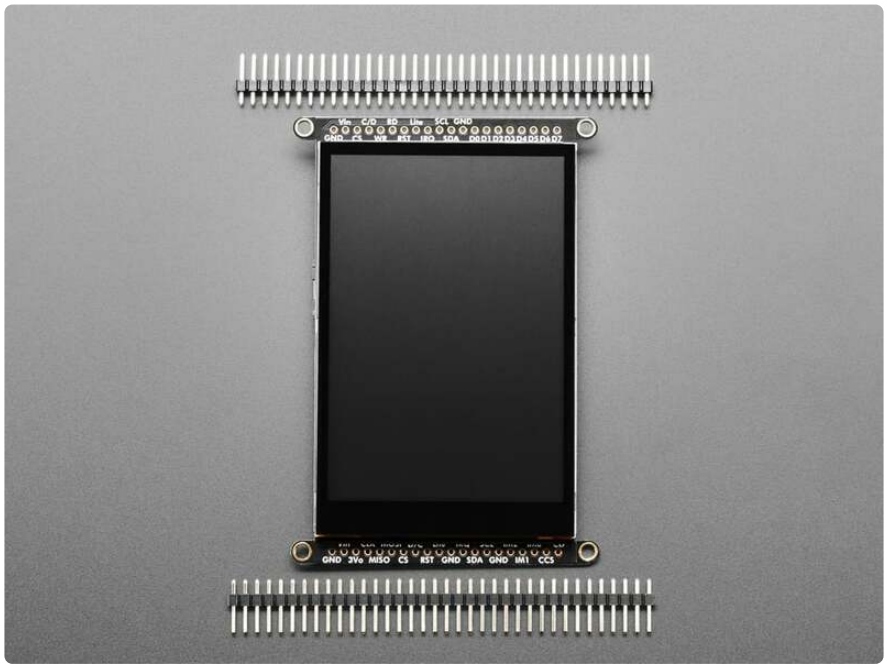
Last updated on 2025-02-10 04:22:54 PM EST

Table of Contents

Overview

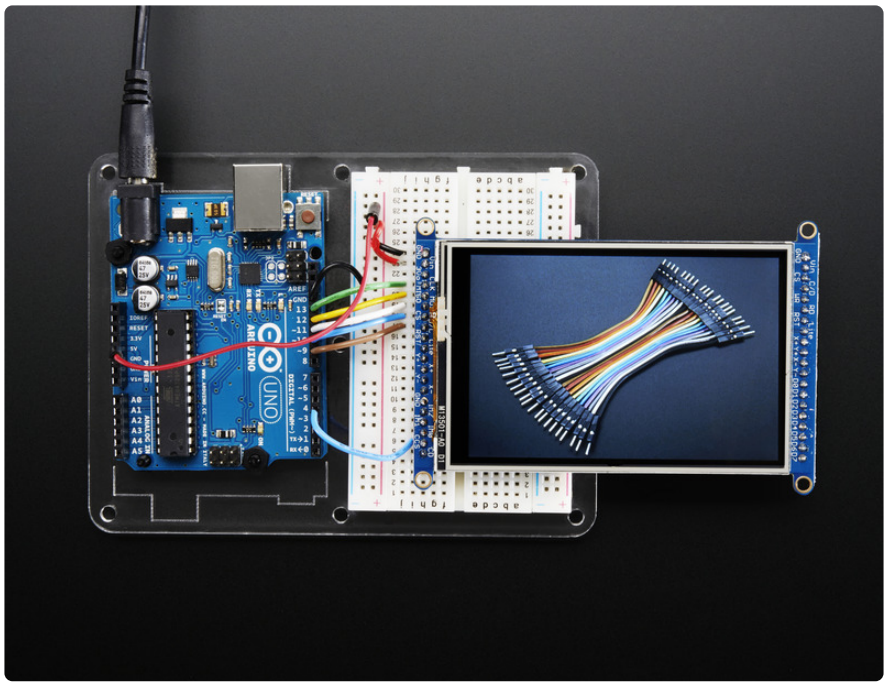


Add some jazz & pizazz to your project with a color touchscreen LCD. This TFT display is big (3.5" diagonal) bright (6 white-LED backlight) and colorful! 480x320 pixels with individual RGB pixel control, this has way more resolution than a black and white 128x64 display, and double our 2.8" TFT. This display is available with a resistive touch driver or a capacitive touch driver.

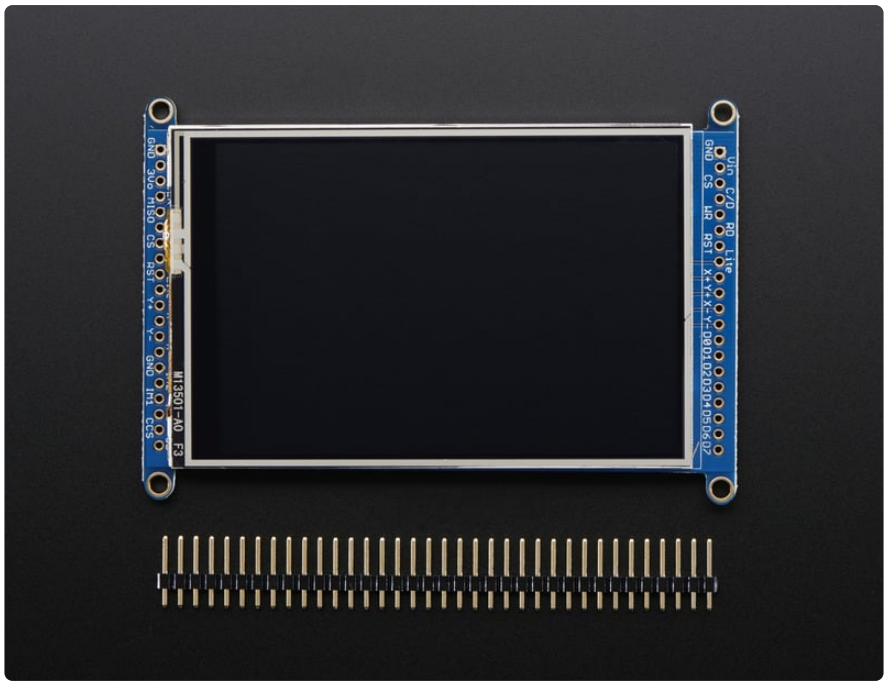


This display has a controller built into it with RAM buffering, so that almost no work is done by the microcontroller. **The display can be used in two modes: 8-bit or SPI.** For 8-bit mode, you'll need 8 digital data lines and 4 or 5 digital control lines to read and

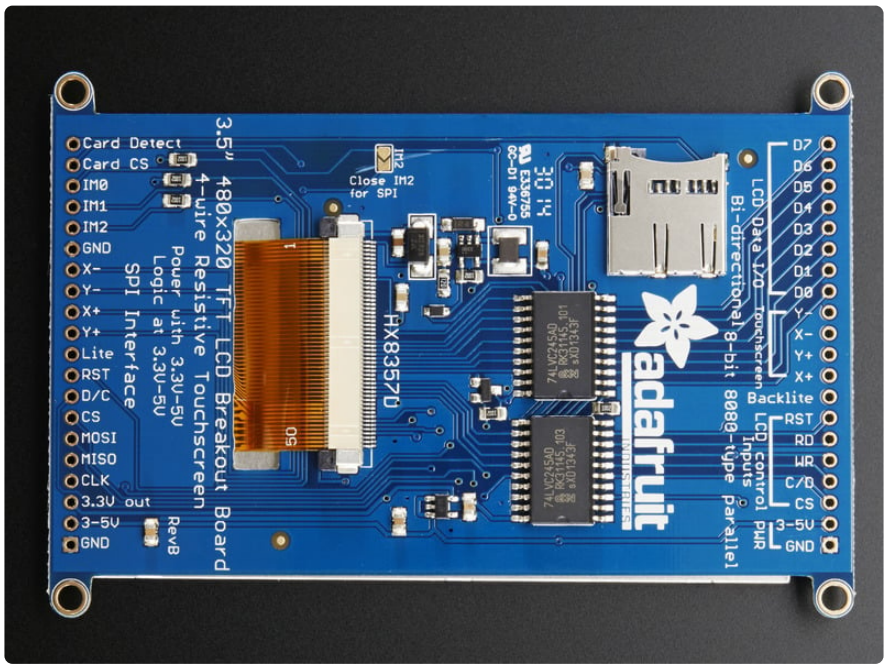
write to the display (12 lines total). SPI mode requires only 5 pins total (SPI data in, data out, clock, select, and d/c) but is slower than 8-bit mode. In addition, 4 pins are required for the touch screen (2 digital, 2 analog) or you can purchase and use our resistive touchscreen controller (not included) to use I2C or SPI (http://adafru.it/1571).



Of course, we wouldn't just leave you with a datasheet and a "good luck!". For 8-bit interface fans we've written a full open source graphics library that can draw pixels, lines, rectangles, circles, text, and more (https://adafru.it/aHk). For SPI users, we have a library as well (https://adafru.it/dQW), its separate from the 8-bit library since both versions are heavily optimized. We also have a resistive touch screen library that detects x, y and z (pressure) (https://adafru.it/aT1) and a capacitive touch screen library that detects up to 5 multi-touch points (https://adafru.it/19bw) and example code to demonstrate all of it.



Pinouts - Resistive Touch



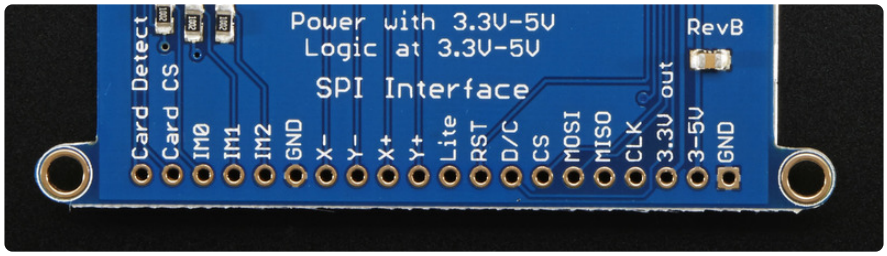
The 3.5" TFT display on this breakout supports many different modes - so many that the display itself has 50 pins. However, we think most people really only use 2 different modes, either "SPI" mode or 8-bit mode. Each 'side' of the display has all the pins required for that mode. You can switch between modes, by rewiring the display, but it cannot be used in two modes at the same time!

All logic pins, both 8-bit and SPI sides, are 3-5V logic level compatible, the 74LVX245

chips on the back perform fast level shifting so you can use either kind of logic levels. If there's data output, the levels are at at 3.3V

SPI Mode

This is what we think will be a popular mode when speed is not of the utmost importance. It doesn't use as many pins (only 4 to draw on the TFT if you skip the MISO pin), is fairly flexible, and easy to port to various microcontrollers. It also allows using a microSD card socket on the same SPI bus. However, its slower than parallel 8-bit mode because you have to send each bit at a time instead of 8-bits at a time. Tradeoffs!



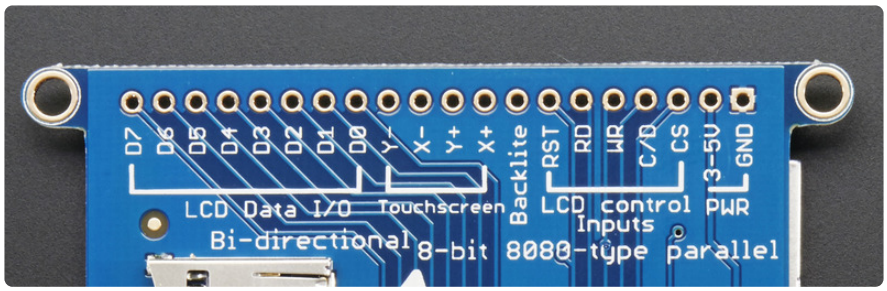
- **GND** - this is the power and signal ground pin
- **3-5V / Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3.3Vout** - this is the 3.3V output from the onboard regulator
- **CLK** - this is the SPI clock input pin
- **MISO** - this is the SPI Microcontroller In Serial Out pin, its used for the SD card mostly, and for debugging the TFT display. It isn't necessary for using the TFT display which is write-only
- **MOSI** - this is the SPI Microcontroller Out Serial In pin, it is used to send data from the microcontroller to the SD card and/or TFT
- **CS** - this is the TFT SPI chip select pin
- **D/C** - this is the TFT SPI data or command selector pin
- **RST** - this is the TFT reset pin. There's auto-reset circuitry on the breakout so this pin is not required but it can be helpful sometimes to reset the TFT if your setup is not always resetting cleanly. Connect to ground to reset the TFT
- **Lite** - this is the PWM input for the backlight control. It is by default pulled high (backlight on) you can PWM at any frequency or pull down to turn the backlight off
- **Y+ X+ Y- X-** these are the 4 resistive touch screen pads, which can be read with analog pins to determine touch points. They are completely separated from the TFT electrically (the overlay is glued on top)
- **IM2 IM1 IM0** - these are interface control set pins. In general these breakouts aren't used, and instead the onboard jumpers are used to fix the interface to SPI

or 8-bit. However, we break these out for advanced use and also for our test procedures

- **Card CS / CCS** - this is the SD card chip select, used if you want to read from the SD card.
- **Card Detect / CD** - this is the SD card detect pin, it floats when a card is inserted, and tied to ground when the card is not inserted. We don't use this in our code but you can use this as a switch to detect if an SD card is in place without trying to electrically query it. Don't forget to use a pullup on this pin if so!

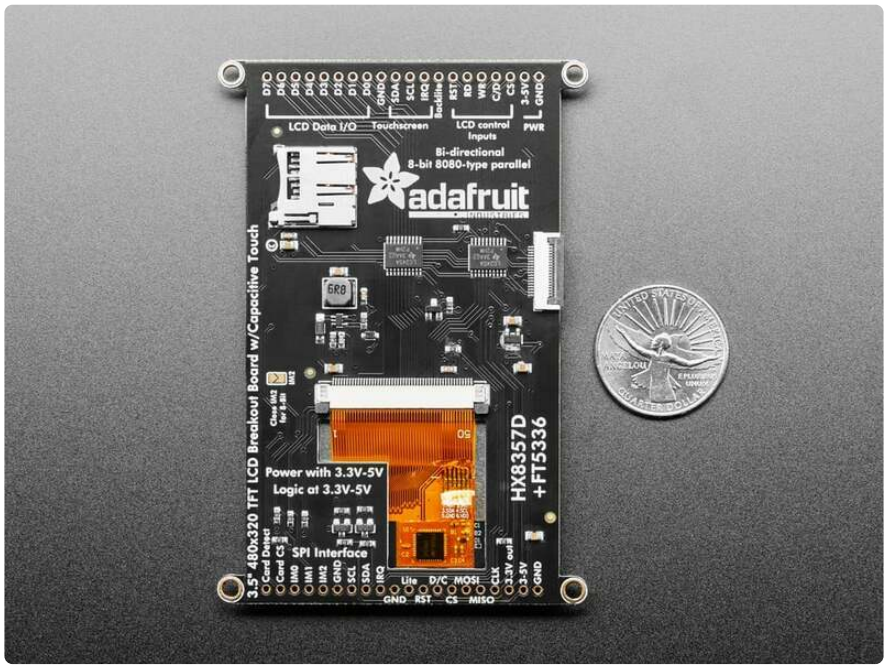
8-Bit Mode

This mode is for when you have lots of pins and want more speed. In this mode we send 8 bits at a time, so it needs way more pins, 12 or so (8 bits plus 4 control)!



- **GND** - this is the power and signal ground pin
- **3-5V (Vin)**- this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **CS** - this is the TFT 8-bit chip select pin (it is also tied to the SPI mode **CS** pin)
- **C/D** - this is the TFT 8-bit data or command selector pin (it is also tied to the SPI mode **C/D** pin)
- **WR** - this is the TFT 8-bit write strobe pin. It is also connected to the **SPI CLK** pin
- **RD** - this is the TFT 8-bit read strobe pin. You may not need this pin if you don't want to read data from the display. If it is not in use, tie it to VCC.
- **RST** - this is the TFT reset pin. There's auto-reset circuitry on the breakout so this pin is not required but it can be helpful sometimes to reset the TFT if your setup is not always resetting cleanly. Connect to ground to reset the TFT
- **Backlite** - this is the PWM input for the backlight control. It is by default pulled high (backlight on) you can PWM at any frequency or pull down to turn the backlight off
- **Y+ X+ Y- X-** these are the 4 resistive touch screen pads, which can be read with analog pins to determine touch points. They are completely separated from the TFT electrically (the overlay is glued on top)
- **D0** thru **D7** - these are the 8 bits of parallel data sent to the TFT in 8-bit mode. **D0** is the least-significant-bit and **D7** is the MSB

Pinouts - Capacitive Touch



The default I2C address for the FT5336 capacitive touch driver is **0x38**.

The 3.5" TFT display on this breakout supports many different modes - so many that the display itself has 50 pins. However, we think most people will really only use 2 different modes: either "SPI" mode or 8-bit mode. Each 'side' of the display has all the pins required for that mode. You can switch between modes by rewiring the display, but it cannot be used in two modes at the same time!

All logic pins, both 8-bit and SPI sides, are 3-5V logic level compatible, the 74LVX245 chips on the back perform fast level shifting so you can use either kind of logic levels. If there's data output, the levels are at 3.3V.

SPI Mode

This is what we think will be a popular mode when speed is not of the utmost importance. It doesn't use as many pins (only 4 to draw on the TFT if you skip the MISO pin), it is fairly flexible, and easy to port to various microcontrollers. It also allows using a microSD card socket on the same SPI bus. However, it's slower than parallel 8-bit mode because you have to send each bit at a time instead of 8-bits at a time. Tradeoffs!

- **GND** - this is the power and signal ground pin.
- **3-5V (Vin)** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3.3V (Vout)** - this is the 3.3V output from the onboard regulator.

- **CLK** - this is the SPI clock input pin.
- **MISO** - this is the SPI Microcontroller In Serial Out pin, it's used for the SD card mostly, and for debugging the TFT display. It isn't necessary for using the TFT display which is write-only.
- **MOSI** - this is the SPI Microcontroller Out Serial In pin, it is used to send data from the microcontroller to the SD card and/or TFT.
- **CS** - this is the TFT SPI chip select pin.
- **D/C** - this is the TFT SPI data or command selector pin.
- **RST** - this is the TFT reset pin. Connect to ground to reset the TFT.
- **Lite** - this is the PWM input for the backlight control. It is by default pulled high (backlight on) you can PWM at any frequency or pull down to turn the backlight off.
- **IM2 IM1 IM0** - these are interface control set pins. In general these pins aren't used, and instead the onboard jumper is used to change the interface to SPI or 8-bit. However, we break these out for advanced use.
- **Card CS / CCS** - this is the SD card chip select, used if you want to read from the SD card.
- **Card Detect / CD** - this is the SD card detect pin, it floats when a card is inserted, and is tied to ground when the card is not inserted. We don't use this in our code but you can use this as a switch to detect if an SD card is in place without trying to electrically query it. Don't forget to use a pull-up on this pin if so!

EYESPI

This display comes with an EYESPI connector, which is an 18pin 0.5mm pitch connector that allows you to use a flex cable to connect your display to your microcontroller in SPI mode. For more details, visit the EYESPI page (https://adafru.it/19bx).

8-Bit Mode

This mode is for when you have lots of pins and want more speed. In this mode we send 8 bits at a time, so it needs way more pins, 12 or so (8 bits plus 4 control)!

- **GND** - this is the power and signal ground pin.
- **3-5V (Vin)**- this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!

LCD Control Inputs

- **CS** - this is the TFT 8-bit chip select pin (it is also tied to the SPI mode **CS** pin).

- **C/D** - this is the TFT 8-bit data or command selector pin (it is also tied to the SPI mode **C/D** pin).
- **WR** - this is the TFT 8-bit write strobe pin. It is also connected to the **SPI CLK** pin.
- **RD** - this is the TFT 8-bit read strobe pin. You may not need this pin if you don't want to read data from the display. If it is not in use, tie it to VCC.
- **RST** - this is the TFT reset pin. Connect to ground to reset the TFT
- **Backlite** - this is the PWM input for the backlight control. It is by default pulled high (backlight on) you can PWM at any frequency or pull down to turn the backlight off

LCD Data I/O

- **D0** thru **D7** - these are the 8 bits of parallel data sent to the TFT in 8-bit mode. **D0** is the least-significant-bit and **D7** is the most significant bit (MSB).

8-Bit Mode Jumper

- **IM2** - In the top center of the board, above the TFT connector, is the IM2 jumper. Soldering this jumper closed enables the 8-Bit display mode. When this jumper is open (default), SPI mode is enabled.

Capacitive Touch Pins

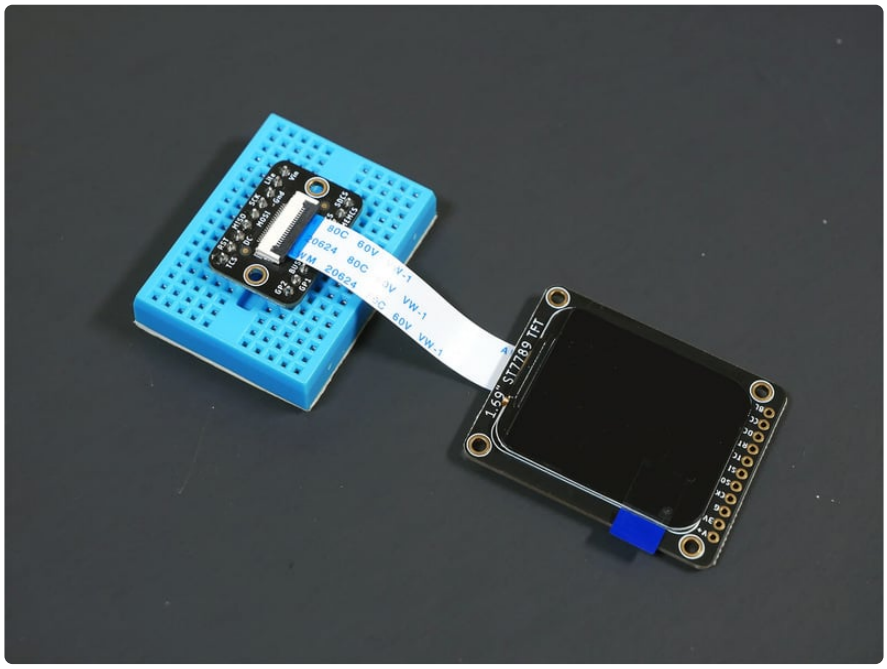
The FT5336 capacitive touch driver is connected over I2C. Its address is **0x38**.

- **SCL** - I2C clock pin, connect to your microcontroller I2C clock line.
- **SDA** - I2C data pin, connect to your microcontroller I2C data line.
- **IRQ** - The capacitive touch interrupt pin. It will drop low when a touch is detected. You can use this to reduce the SPI polling.

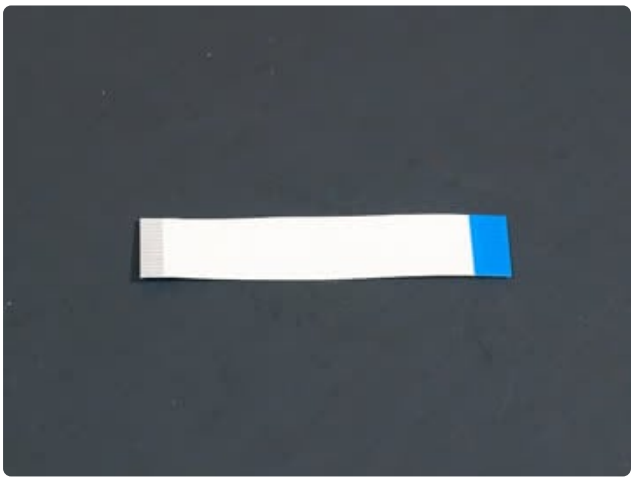
microSD Card Slot

- Located to the left of the Adafruit logo is the microSD card slot. You can use any microSD card that supports SPI mode with one CS pin.

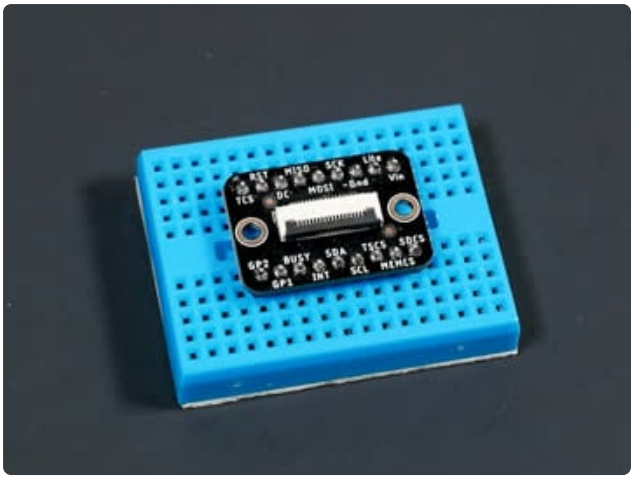
Plugging in an EYESPI Cable



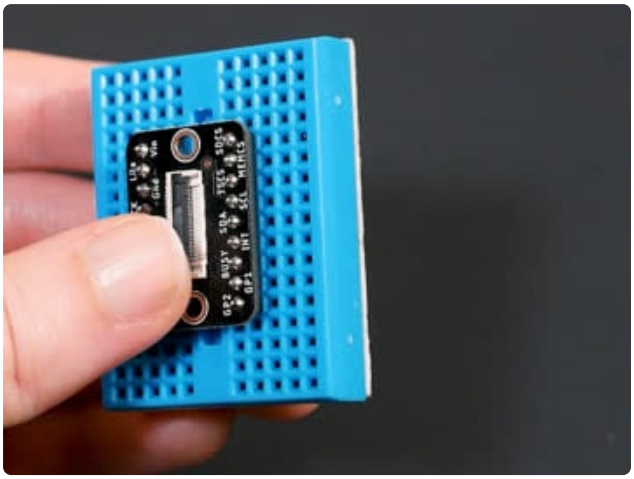
You can connect an EYESPI compatible display to the EYESPI breakout board using an EYESPI cable. An EYESPI cable is an 18 pin flexible PCB (FPC). The FPC can only be connected properly in one orientation, so be sure to follow the steps below to ensure that your display and breakout are plugged in properly.



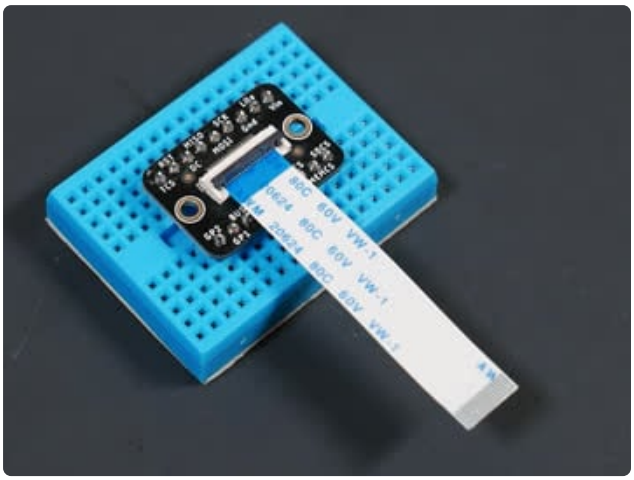
Each EYESPI cable has **blue stripes** on either end. On the other side of the cable, underneath the blue stripe, are the connector pins that make contact with the FPC connector pins on the display or breakout.



To begin inserting an EYESPI cable to an FPC connector, gently lift the FPC connector black latch up.



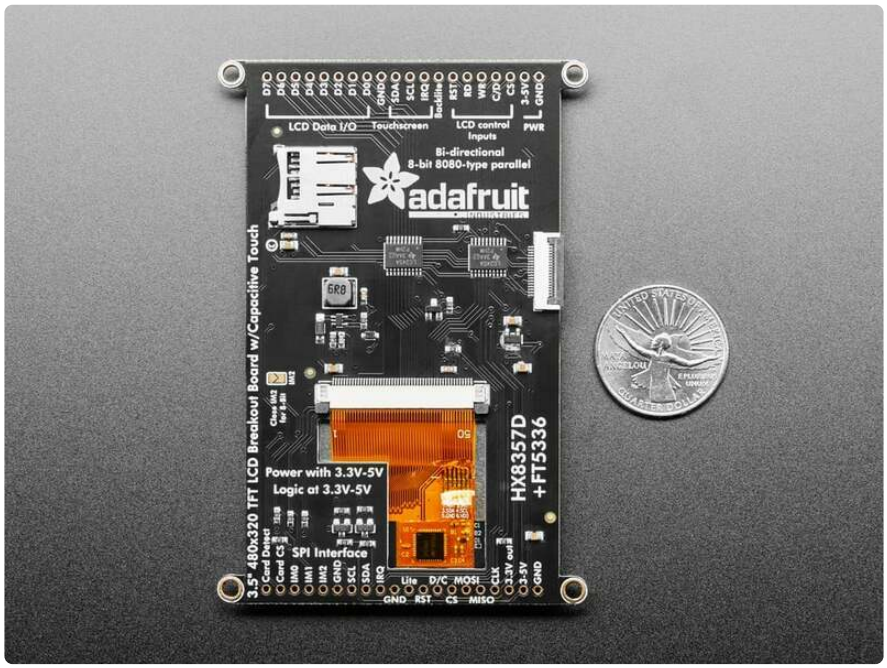
Then, insert the EYESPI cable into the open FPC connector by sliding the cable into the connector. You want to **see the blue stripe facing up towards you**. This inserts the cable pins into the FPC connector.



To secure the cable, lower the FPC connector latch onto the EYESPI cable.

Repeat this process for the FPC connector on your display. Again, ensure that the **blue stripe** on either end of the cable is facing up.

EYESPI

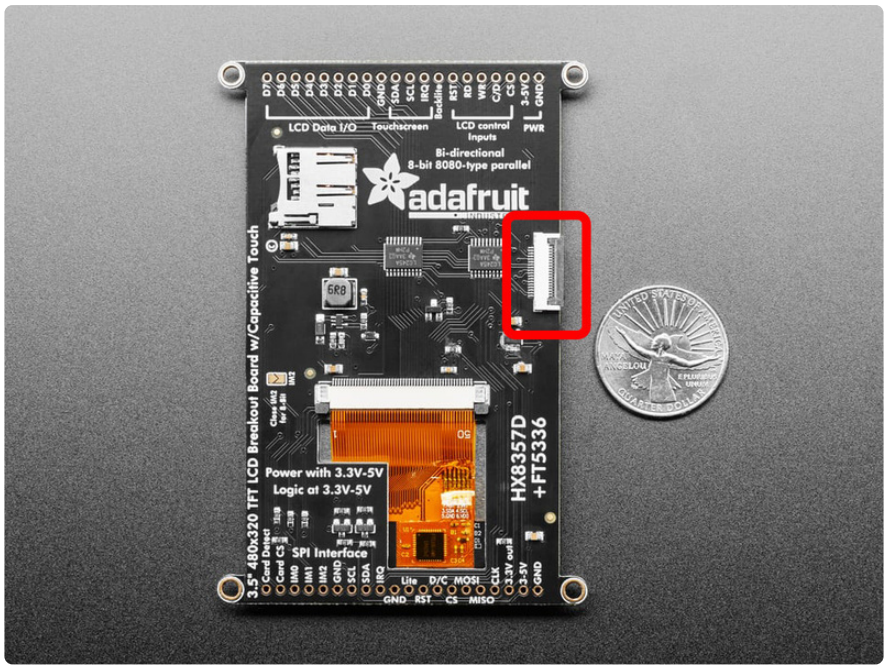


This display now comes with an **EYESPI connector**. This connector allows you to connect your display without soldering. There are EYESPI cables (https://adafru.it/18eT) available in multiple lengths, which means you can find one to fit any project. This is especially useful if your project requires the display to be freestanding, and not tied directly into a breadboard. Inspired by the popularity of STEMMA QT, it provides plug-n-play for displays!

The EYESPI Connector and Cables

The EYESPI connector is an 18 pin 0.5mm pitch FPC connector with a flip-top tab for locking in the associated flex cable. It is designed to allow you to connect a display, without needing to solder headers or wires to the display.

The EYESPI connector location on this display is indicated below.

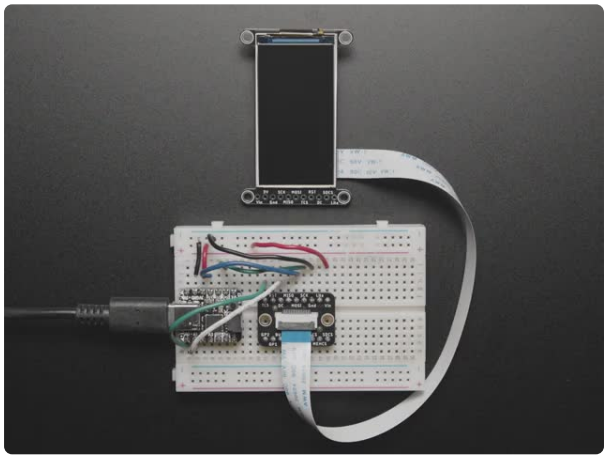


The EYESPI cables are 18 pin 0.5mm pitch flex cables. They are ~9.6mm wide, and designed to fit perfectly into the EYESPI connector. Adafruit currently offers EYESPI cables in three different lengths: 50mm (http://adafru.it/5462), 100mm (http://adafru.it/5239), and 200mm (http://adafru.it/5240).

The EYESPI connector is designed to work with 18-pin 0.5mm pitch flex cables. Other flex cables, such as Raspberry Pi camera flex cables, will not work!

Wiring Your EYESPI Display

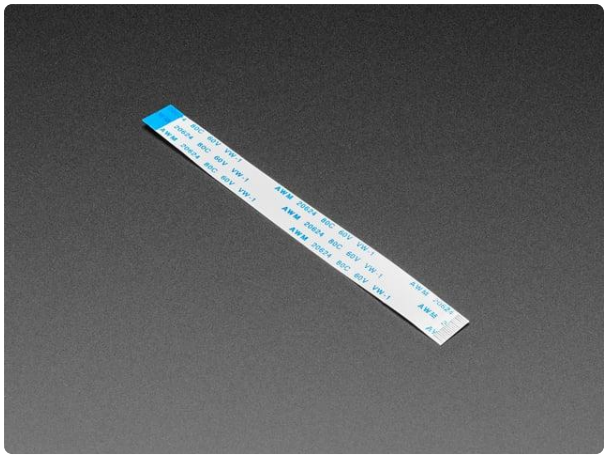
Wiring your EYESPI display to a microcontroller via the EYESPI connector requires the EYESPI breakout board and an EYESPI cable.



Adafruit EYESPI Breakout Board - 18 Pin FPC Connector

Our most recent display breakouts have come with a new feature: an 18-pin "EYE SPI" standard FPC...

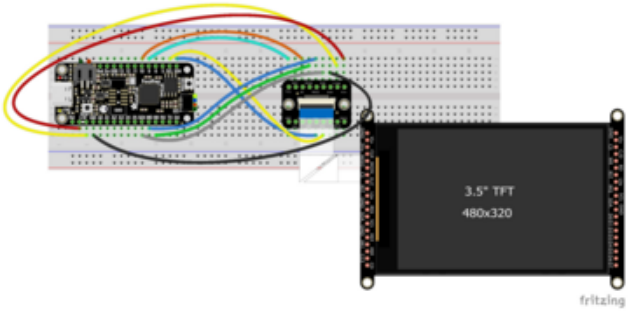
https://www.adafruit.com/product/5613



EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...
https://www.adafruit.com/product/5239

The following example shows how to connect the 3.5" TFT with Capacitive Touch Display Breakout to a Feather RP2040 using the EYESPI breakout board.

Connect the following Feather pins to the associated EYESPI breakout pins:



breakout Vin to **Feather 3.3V (red wire)**
breakout Lite to **Feather 3.3V (yellow wire)**
breakout Gnd to **Feather GND (black wire)**
breakout SCK to **Feather SCK (grey wire)**
breakout MISO to **Feather MI (green wire)**
breakout MOSI to **Feather MO (blue wire)**
breakout TCS to **Feather D9 (aqua wire)**
breakout DC to **Feather D10 (orange wire)**
breakout SCL to **Feather SCL (yellow wire)**
breakout SDA to **Feather SDA (blue wire)**

Finally, connect your **display EYESPI connector** to the **breakout EYESPI connector** using an **EYESPI cable**. For details on using the EYESPI connector properly, visit Plugging in an EYESPI Cable (https://adafru.it/18eU).

EYESPI Pins

Though there are 18 pins available on the EYESPI connector, many displays do not use all available pins. This display requires the following pins:

- **Vin** - This is the power pin. To power the board (and thus your display), connect to the same power as the logic level of your microcontroller, e.g. for a 3V micro like a Feather, use 3V, and for a 5V micro like an Arduino, use 5V.

- **Lite** - This is the PWM input for the backlight control. It is by default pulled high (backlight on), however, you can PWM at any frequency or pull down to turn the backlight off.
- **Gnd** - This is common ground for power and logic.
- **MISO** - This is the SPI MISO (**M**icrocontroller **I**n / **S**erial **O**ut) pin. It's used for the SD card. It isn't used for the display because it's write-only. It is 3.3V logic out (but can be read by 5V logic).
- **MOSI** - This is the SPI MOSI (**M**icrocontroller **O**ut / **S**erial **I**n) pin. It is used to send data from the microcontroller to the SD card and/or display.
- **SCK** - This is the SPI clock input pin.
- **TCS** - This is the TFT or eInk SPI chip select pin.
- **RST** - This is the display reset pin. Connecting to ground resets the display! It's best to have this pin controlled by the library so the display is reset cleanly, but you can also connect it to the microcontroller's Reset pin, which works for most cases. Often, there is an automatic-reset chip on the display which will reset it on power-up, making this connection unnecessary in that case.
- **DC** - This is the display SPI data/command selector pin.
- **INT** - This is the capacitive touch interrupt pin. When a touch is detected, this pin goes low.
- **SDA** - This is the I2C serial data pin. Connect to the desired I2C data pin on your microcontroller.
- **SCL** - This is the I2C serial clock pin. Connect to the desired I2C clock pin on your microcontroller.
- **TSCS** - This is the Touch Screen Chip Select pin.
- **MEMCS** - This is the Memory Chip Select. This pin is required for communicating with the RAM chip onboard the connected display.
- **SDCS** - This is the SD card chip select pin. This pin is required for communicating with the SD card holder onboard the connected display.

Wiring & Test

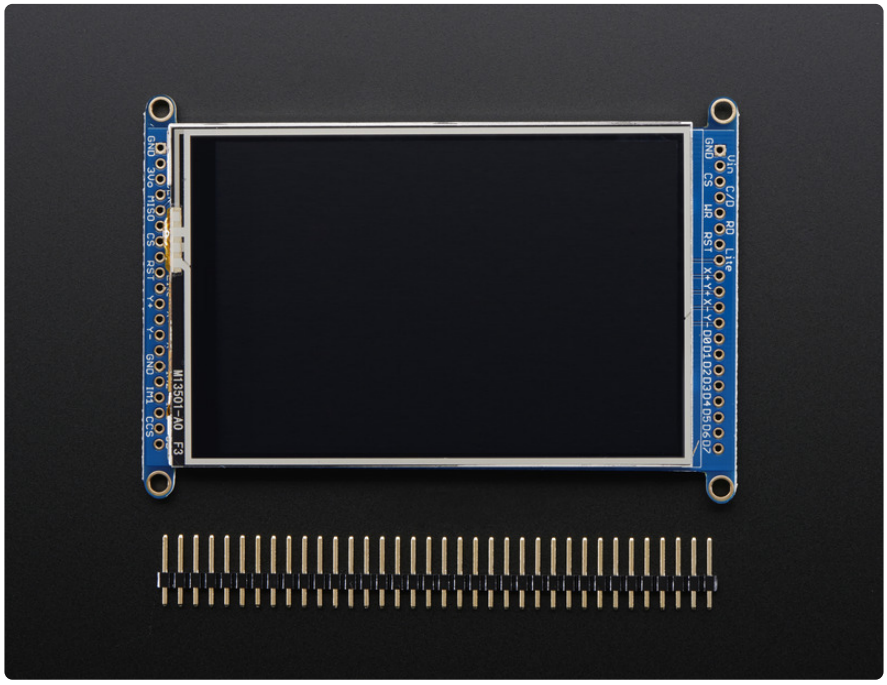
We tried to make this TFT breakout useful for both high-pin microcontrollers that can handle 8-bit data transfer modes as well as low-pincount micros like the Arduino UNO and Leonardo that are OK with SPI.

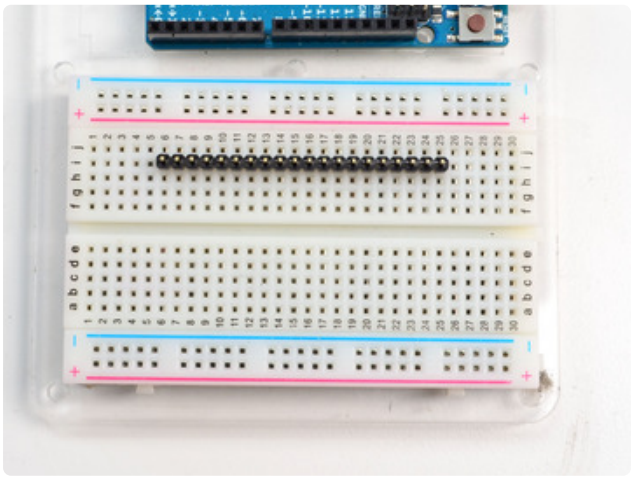
Essentially, the tradeoff is pins for speed. SPI is about 2-4 times slower than 8-bit mode, but that may not matter for basic graphics!

In addition, SPI mode has the benefit of being able to use the onboard microSD card socket for reading images. We don't have support for this in 8-bit mode so if you want to have an all-in-one image viewer type application, use SPI!

Assembling Header

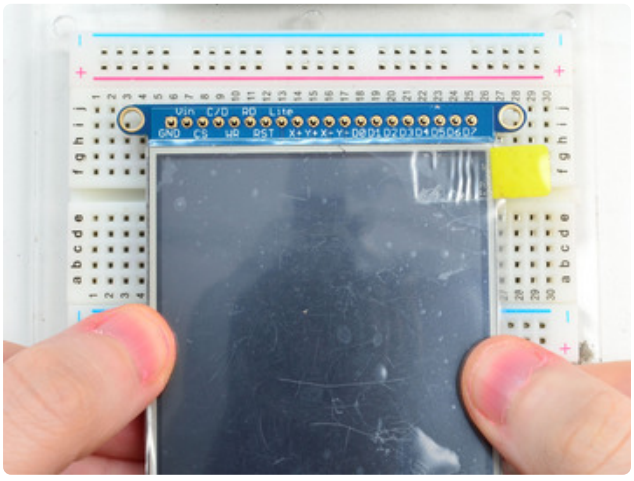
Either way, if you're using a breadboard, you'll need to solder header onto one or two of the sides. The procedure is the same for both sides





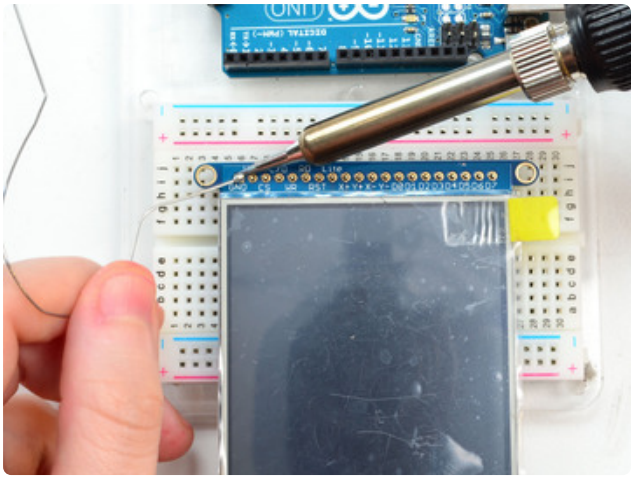
Prepare the header strip:

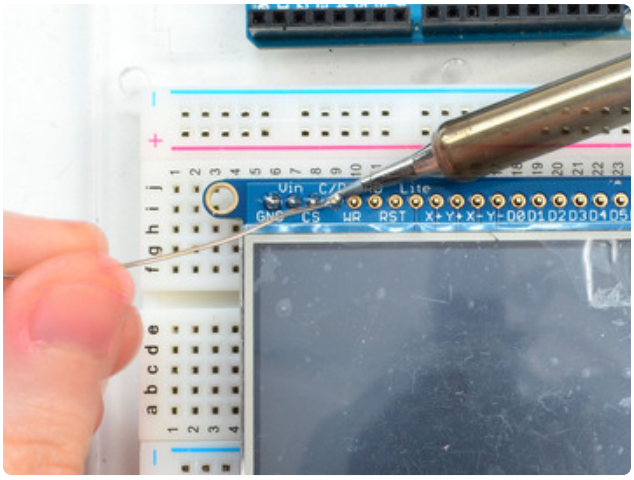
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

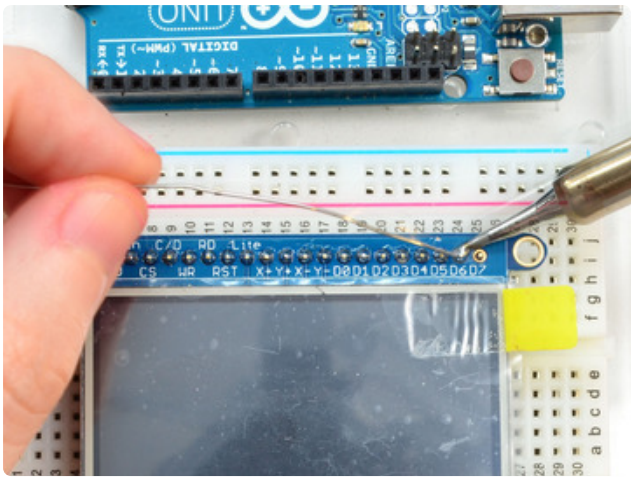


Add the breakout board:

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

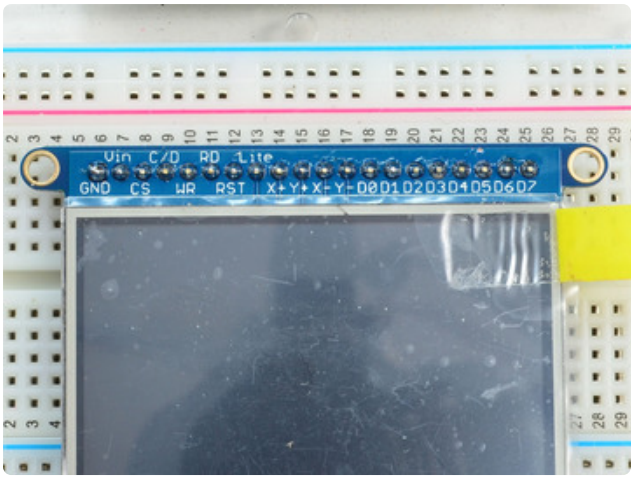






Add the breakout board:

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



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

---

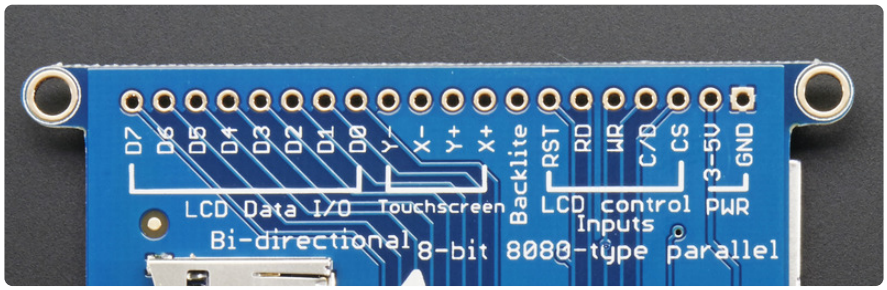
8-Bit Wiring & Test

8-Bit Wiring

Wiring up the 8-bit mode is kind of a pain, so we really only recommend doing it for UNO (which we show) and Mega (which we describe, and is pretty easy since its 8 pins in a row). Anything else, like a Leonardo or Micro, we strongly recommend going with SPI mode since we don't have an example for that. Any other kind of 'Arduino compatible' that isn't an Uno, try SPI first. The 8-bit mode is hand-tweaked in the **Adafruit_TFTLCD pin_magic.h** file. Its really only for advanced users who are totally cool with figuring out bitmasks for various ports & pins.

Really, we'll show how to do the UNO but anything else? go with SPI!

Make sure you're soldering and connecting to the 8-bit side!

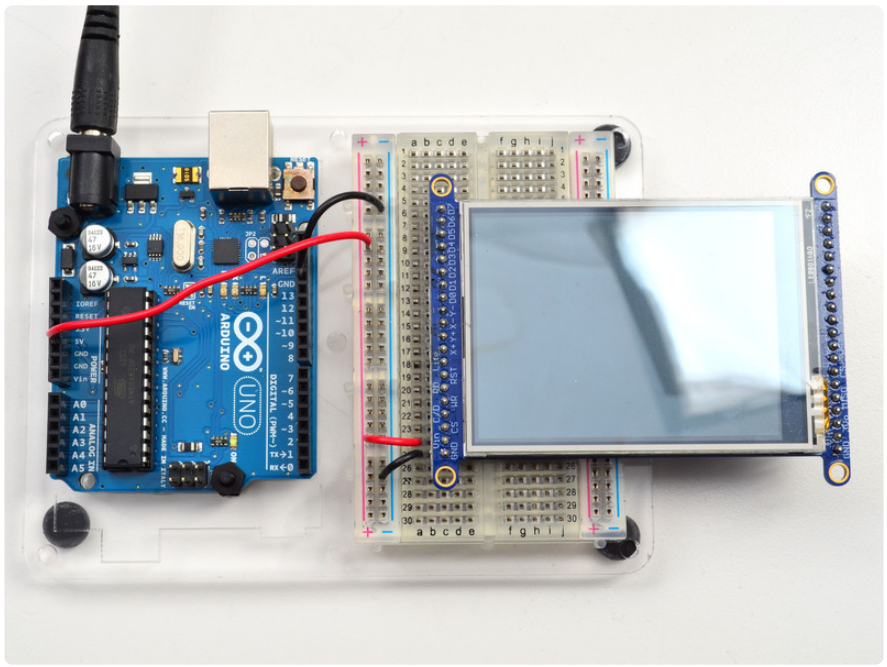


Part 1 - Power & backlight test

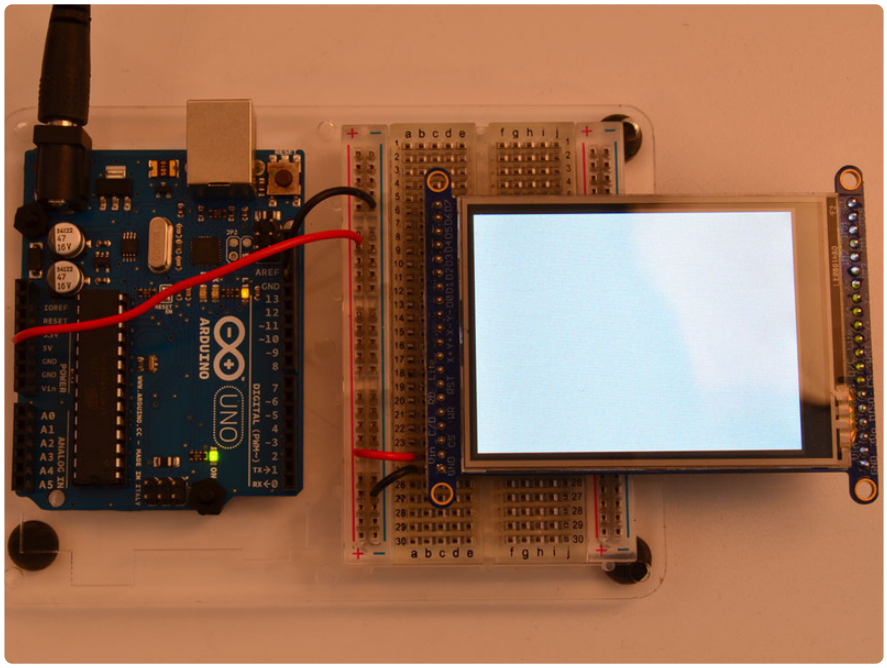
In these images we show using our 2.8" TFT but its the exact same pinout, just a tad smaller!

Begin by wiring up the **3-5VDC** and **GND** pins.

Connect the **3-5V** pin to **5V** and **GND** to **GND** on your Arduino. I'm using the breadboard rails but you can also just wire directly.



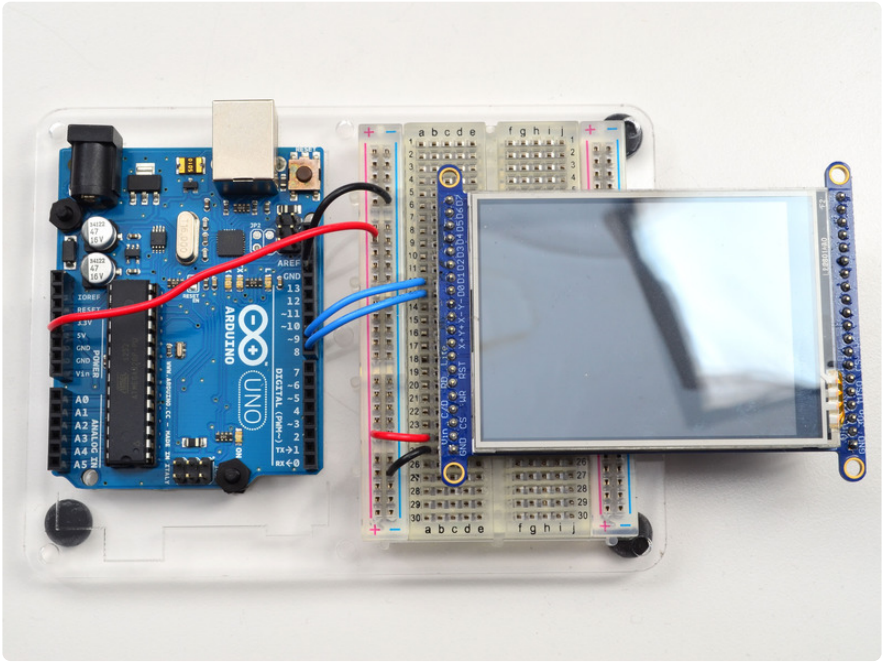
Power it up and you should see the white backlight come on



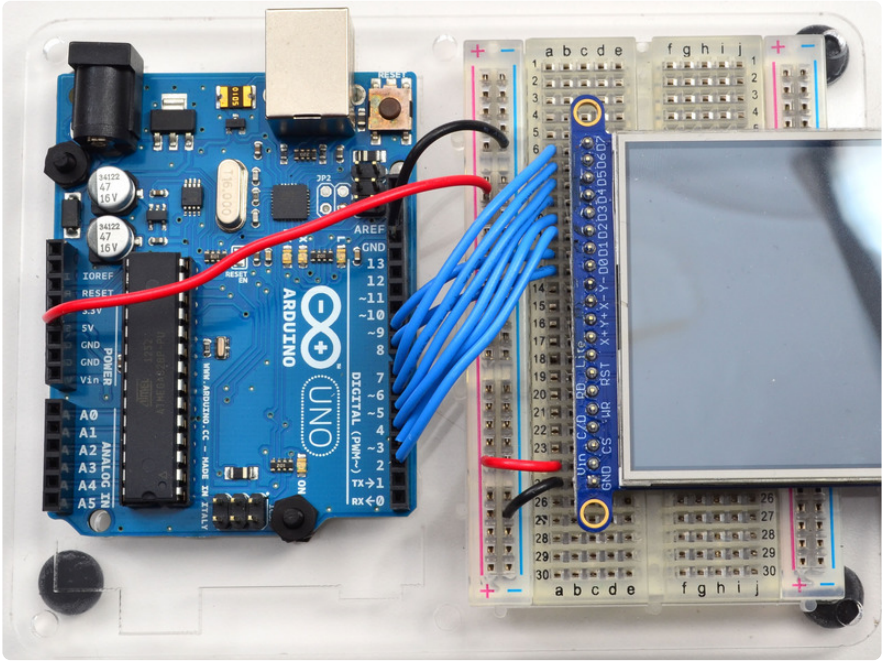
Part 2 - Data Bus Lines

Now that the backlight is working, we can get the TFT LCD working. There are many pins required, and to keep the code running fairly fast, we have 'hardcoded' Arduino digital pins **#2-#9** for the 8 data lines.

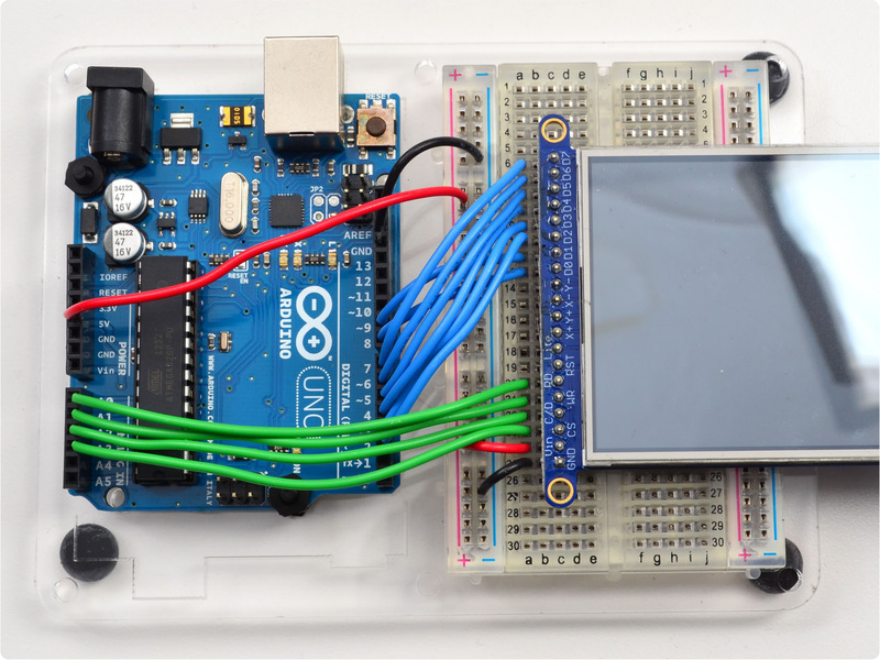
However, they are not in that order! D0 and D1 go to **digital #8** and **#9**, then D2-D7 connect to **#2** thru **#7**. This is because Arduino pins #0 and #1 are used for serial data so we can't use them



Begin by connecting **D0** and **D1** to digital **#8** and **9** respectively as seen above. If you're using a Mega, connect the TFT Data Pins **D0-D1** to Mega pins **#22-23**, in that order. Those Mega pins are on the 'double' header.

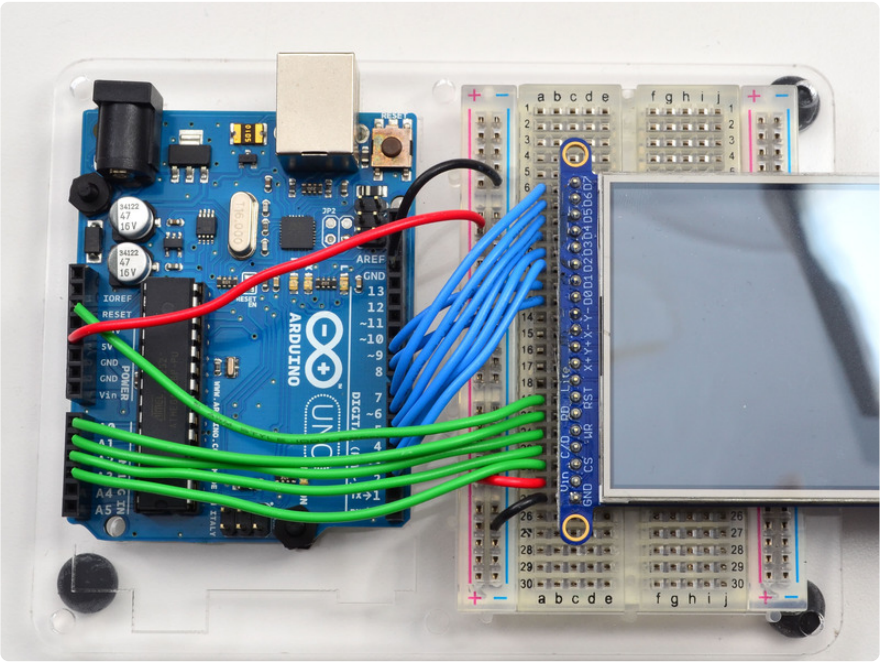


Now you can connect the remaining 6 pins over. Connect **D2-D7** on the TFT pins to digital **2** thru **7** in that order. If you're using a Mega, connect the TFT Data Pins **D2-D7** to Mega pins **#24-29**, in that order. Those Mega pins are on the 'double' header.



In addition to the 8 data lines, you'll also need 4 or 5 control lines. These can later be reassigned to any digital pins, they're just what we have in the tutorial by default.

- Connect the third pin **CS** (**Chip Select**) to Analog 3
- Connect the fourth pin **C/D** (**Command/Data**) to Analog 2
- Connect the fifth pin **WR** (**Write**) to Analog 1
- Connect the sixth pin **RD** (**Read**) to Analog 0



You can connect the seventh pin **RST** (**Reset**) to the Arduino Reset line if you'd like. This will reset the panel when the Arduino is Reset. You can also use a digital pin for the LCD reset if you want to manually reset. There's auto-reset circuitry on the board

so you probably don't need to use this pin at all and leave it disconnected

The **RD** pin is used to read the chip ID off the TFT. Later, once you get it all working, you can remove this pin and the ID test, although we suggest keeping it since its useful for debugging your wiring.

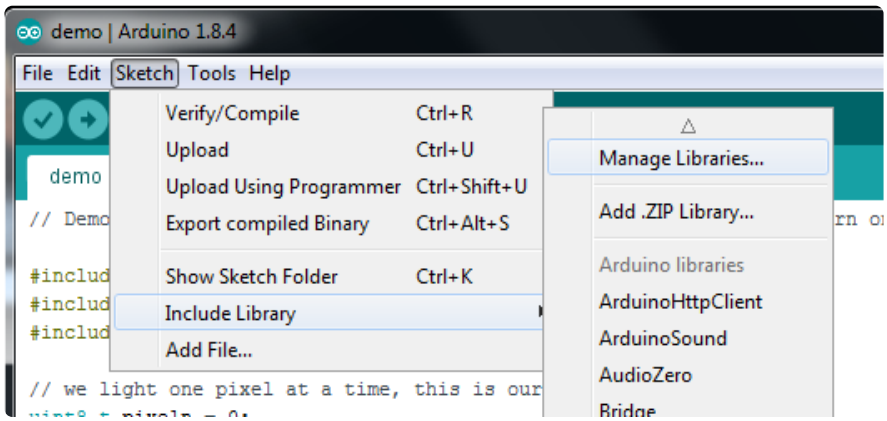
OK! Now we can run some code

8-Bit Library Install

We have example code ready to go for use with these TFTs. It's written for Arduino, which should be portable to any microcontroller by adapting the C++ source.

Two libraries need to be downloaded and installed: the TFTLCD library (https://adafru.it/aHk) and the GFX library. (https://adafru.it/aJa) You can install these libraries through the Arduino library manager.

Open up the Arduino library manager:



Search for the **Adafruit GFX** library and install it:

If using an older Arduino IDE (pre-1.8.10), also locate and install **Adafruit_BusIO** (newer versions do this automatically).

Next, search for **Adafruit TFTLCD** and install it:

We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

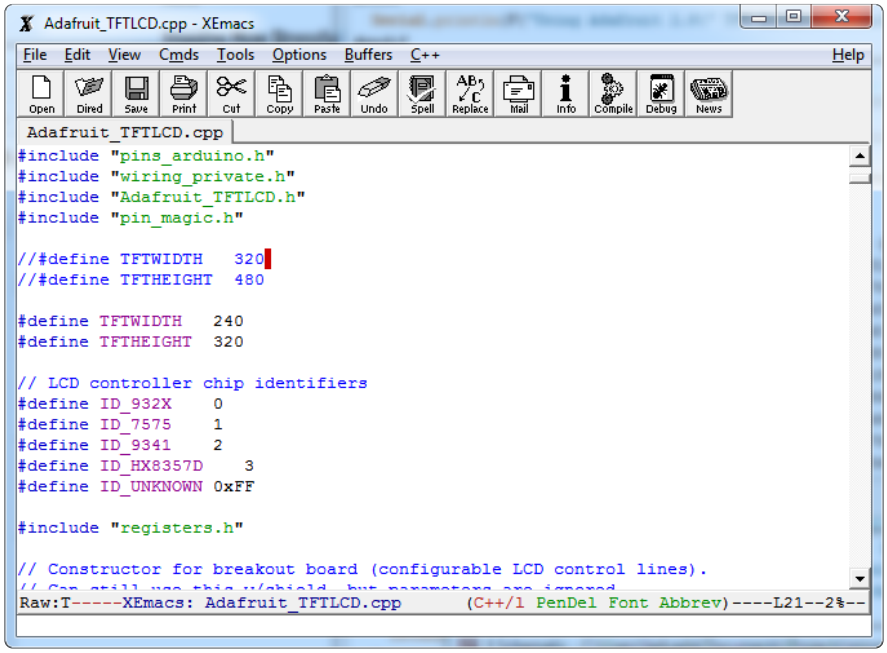
Prepare TFTLCD Library

In the **Adafruit_TFTLCD** Library folder, you may need to edit **Adafruit_TFTLCD.h**. On about line 12, you will see

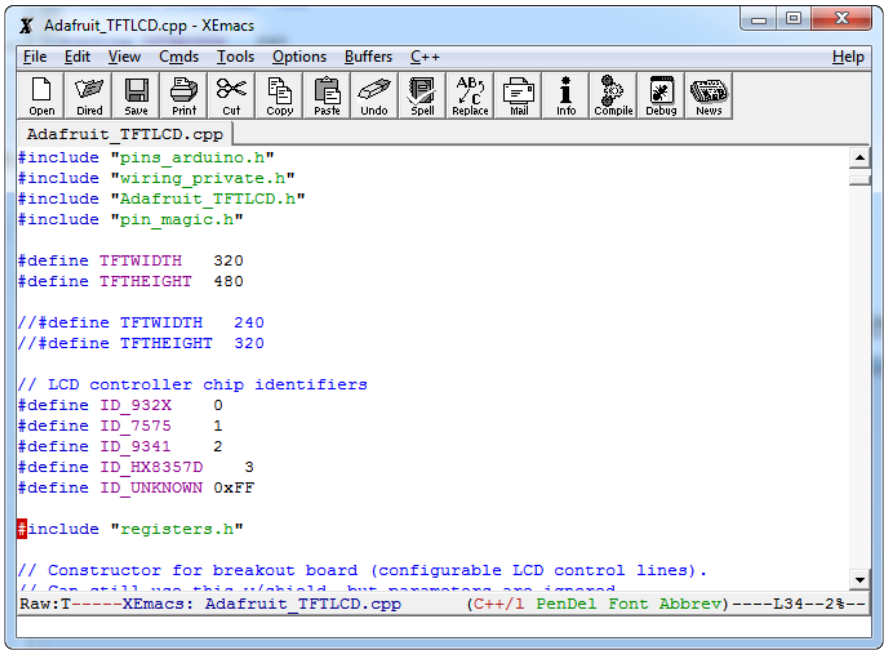
#define USE_ADAFRUIT_SHIELD_PINOUT

Make sure this line is commented out with a // in front (it should but if you're having issues, its worth checking.

Next up, we originally designed this library for 320x240 TFTs. Since this is a 480x320, we have to adjust the size the library is expecting. Open up **Adafruit_TFTLCD.cpp** and find these lines:

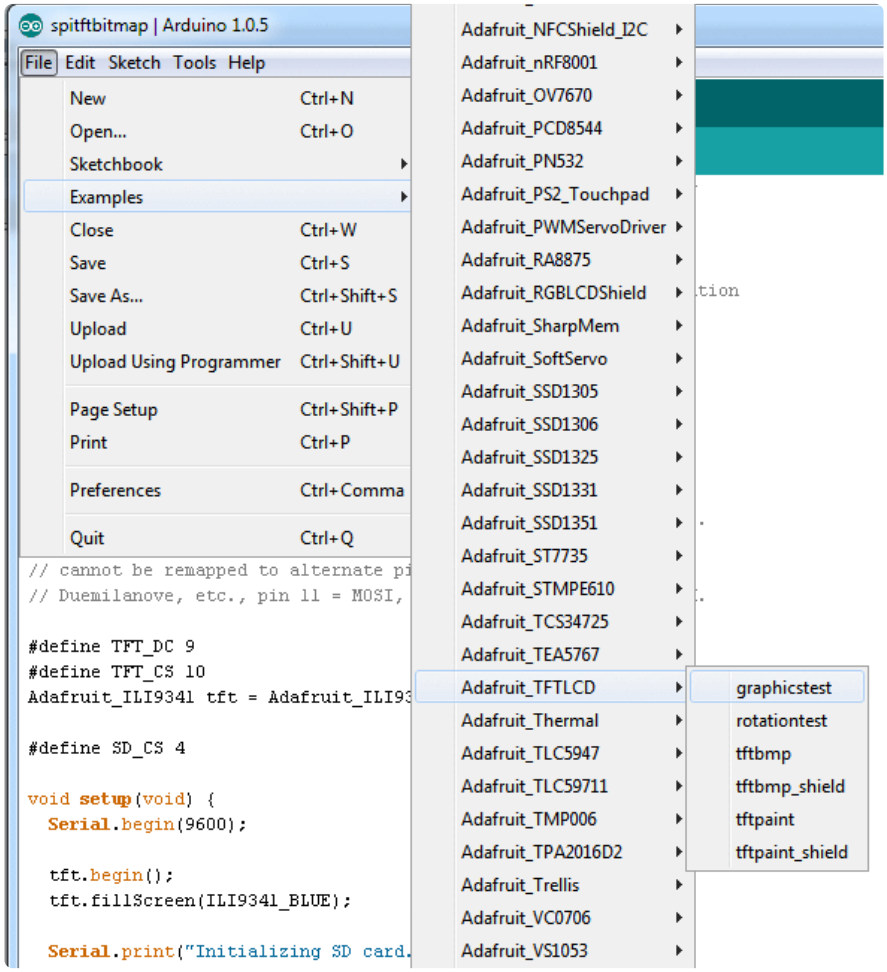


Comment out the 240 and 320 lines, and uncomment the 320 and 480 lines:



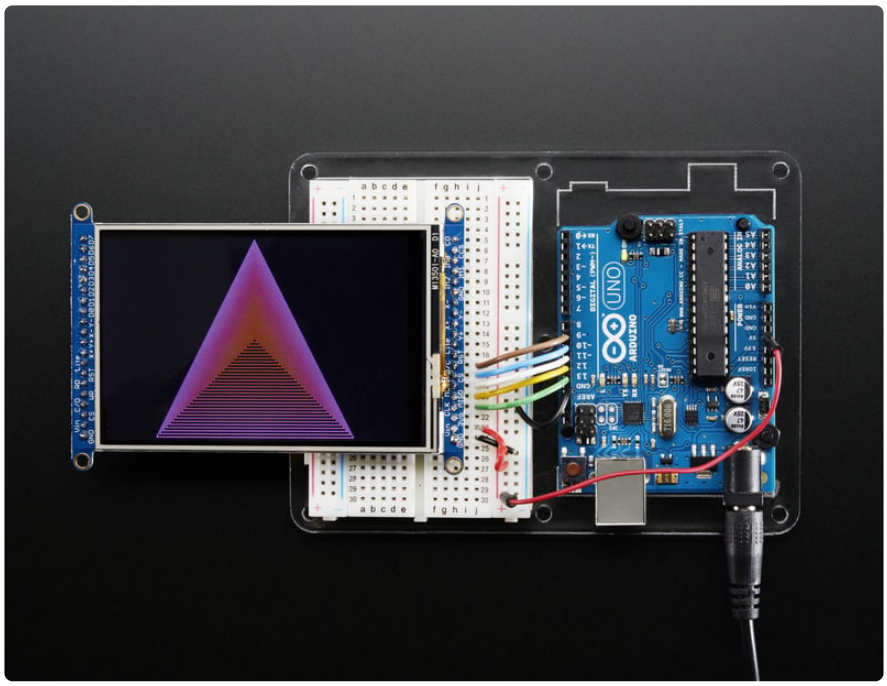
Save it, now you can upload the demo!

After restarting the Arduino software, you should see a new **example** folder called **Adafruit_TFTLCD** and inside, an example called **graphicstest**. Upload that sketch to your Arduino.

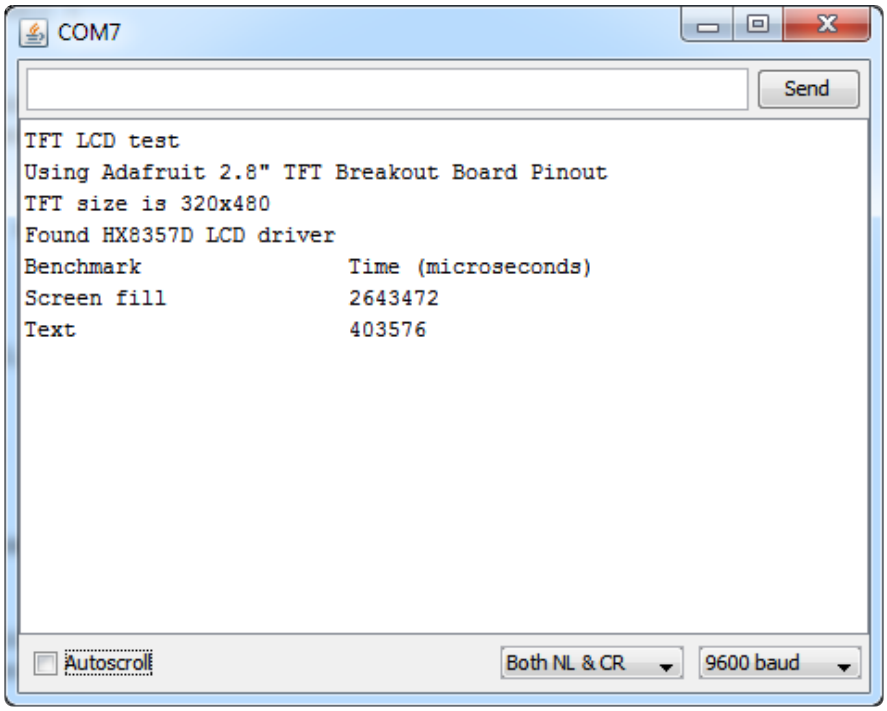


You may need to press the Reset button to reset the Arduino and TFT. You should see a collection of graphical tests draw out on the TFT.

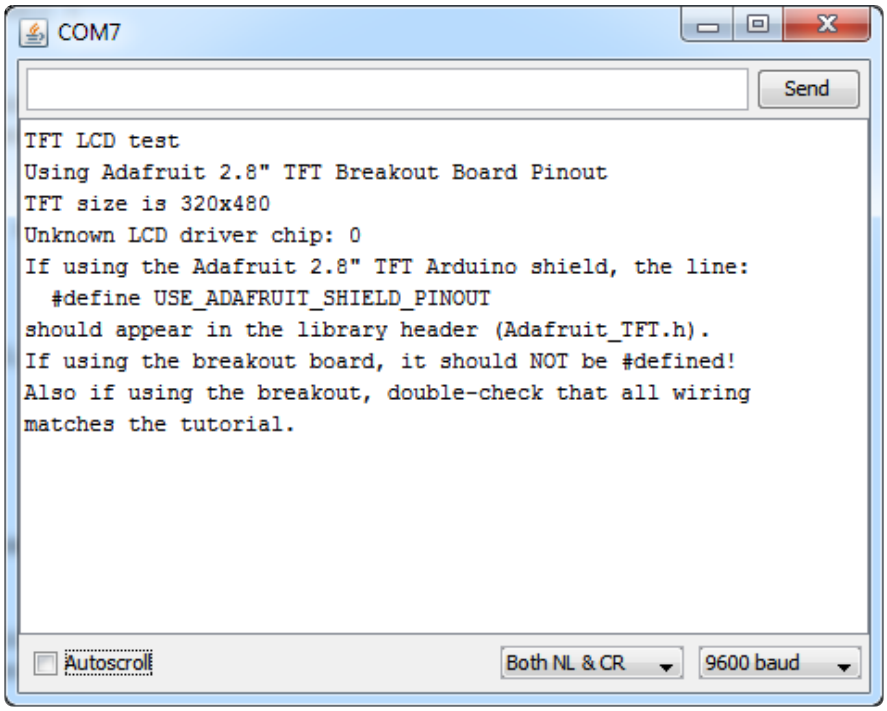
(The images below shows SPI wiring but the graphical output should be similar!)



If you're having difficulties, check the serial console.The first thing the sketch does is read the driver code from the TFT. It should be **0x8357** (for the **HX8357D** controller inside)

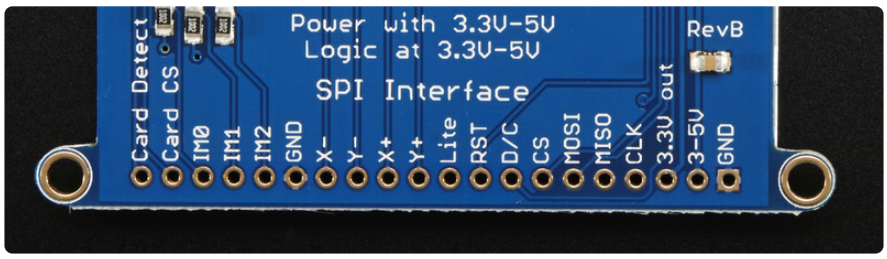


If you **Unknown Driver Chip** then it's probably something with your wiring, double check and try again!



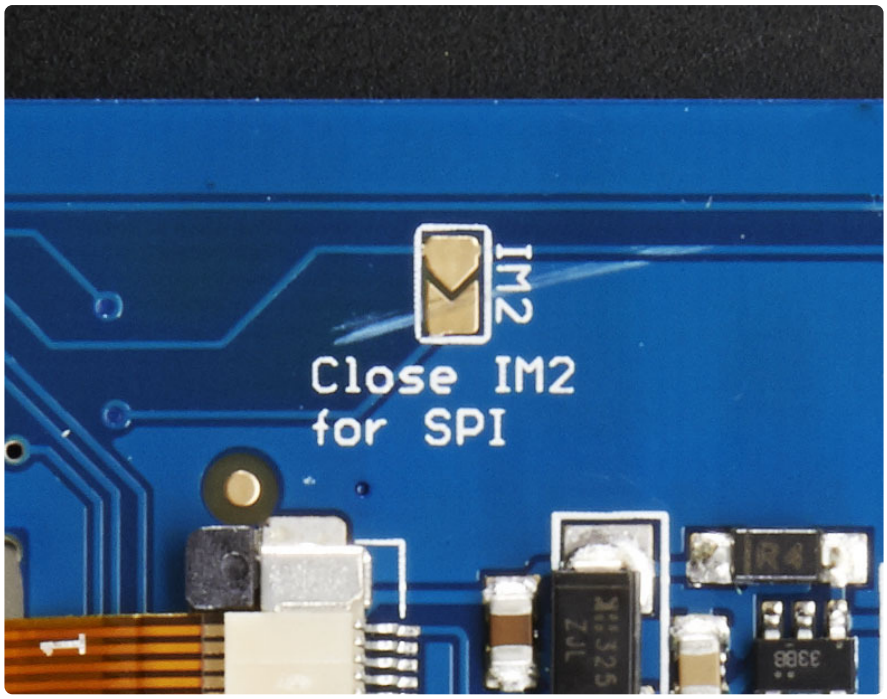
SPI Wiring & Test

Don't forget, we're using the SPI interface side of the PCB!

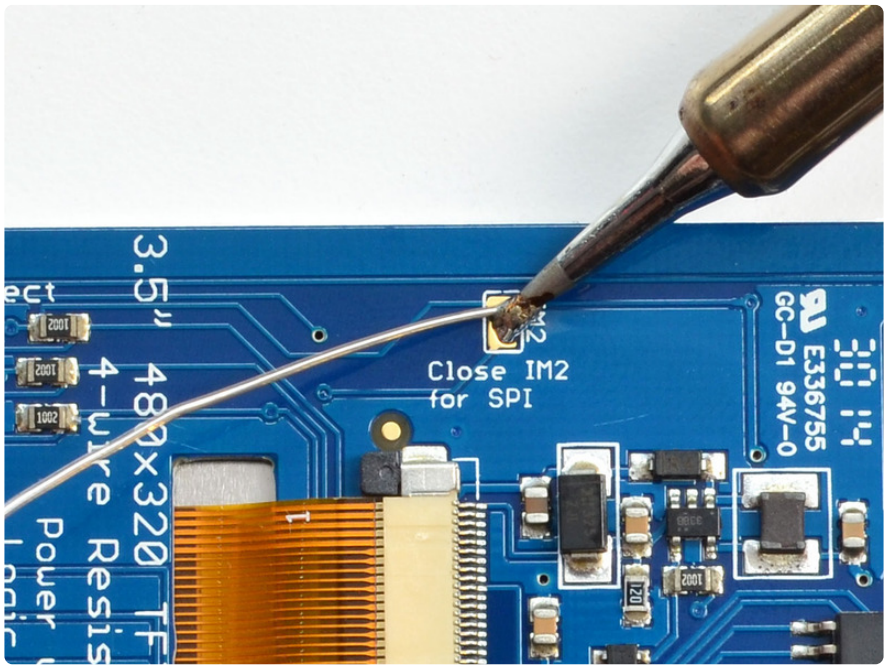


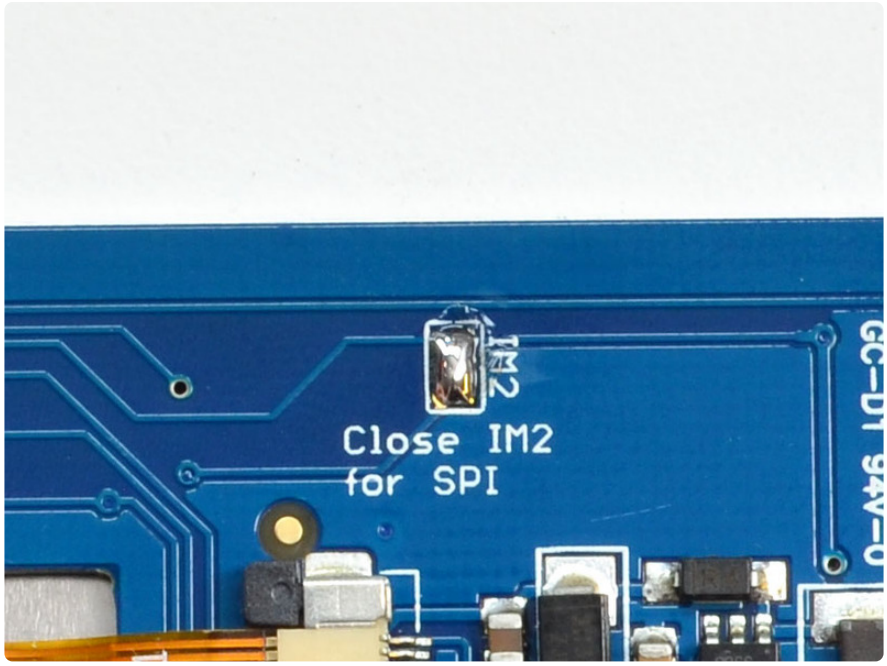
SPI Mode Jumpers

Before you start, we'll need to tell the display to put us in SPI mode so it will know which pins to listen to. To do that, we have to connect the **IM2** pin to 3.3V. The easiest way to do that is to solder closed the **IM2** jumper on the back of the PCB. Turn over the PCB and find the solder jumper:



With your soldering iron, melt solder to close the jumper indicated **IM2**





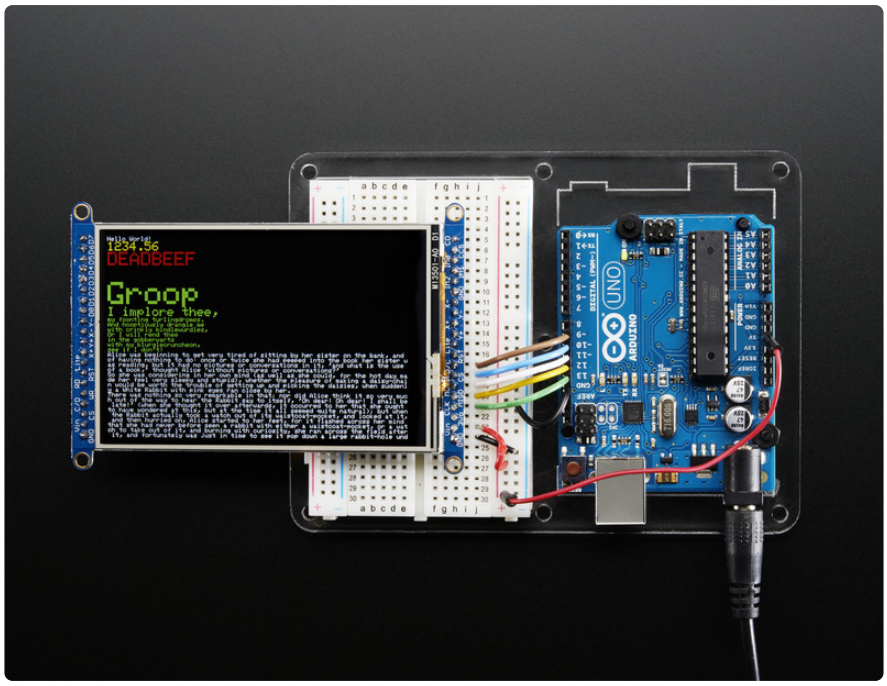
If you really don't want to solder, you can also wire the breakout pin to the **3vo** pin, just make sure you don't tie it to 5V by accident! For that reason, we suggest going with the solder-jumper route.

Wiring

Wiring up the display in SPI mode is much easier than 8-bit mode since there's way fewer wires. Start by connecting the power pins

- **3-5V Vin** connects to the Arduino **5V** pin
- **GND** connects to Arduino ground
- **CLK** connects to SPI clock. On Arduino Uno/Duemilanove/328-based, thats **Digital 13**. On Mega's, its **Digital 52** and on Leonardo/Due its **ICSP-3** (See SPI Connections for more details (https://adafru.it/d5h))
- **MISO** connects to SPI MISO. On Arduino Uno/Duemilanove/328-based, thats **Digital 12**. On Mega's, its **Digital 50** and on Leonardo/Due its **ICSP-1** (See SPI Connections for more details (https://adafru.it/d5h))
- **MOSI** connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, thats **Digital 11**. On Mega's, its **Digital 51** and on Leonardo/Due its **ICSP-4** (See SPI Connections for more details (https://adafru.it/d5h))
- **CS** connects to our SPI Chip Select pin. We'll be using **Digital 10** but you can later change this to any pin
- **D/C** connects to our SPI data/command select pin. We'll be using **Digital 9** but you can later change this pin too.

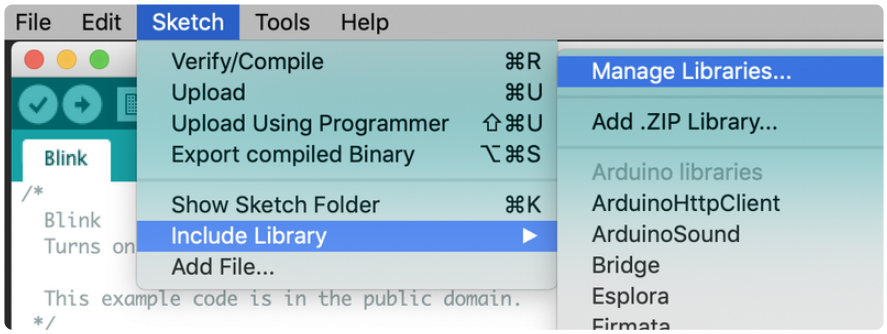
That's it! You do not need to connect the **RST** or other pins for now.



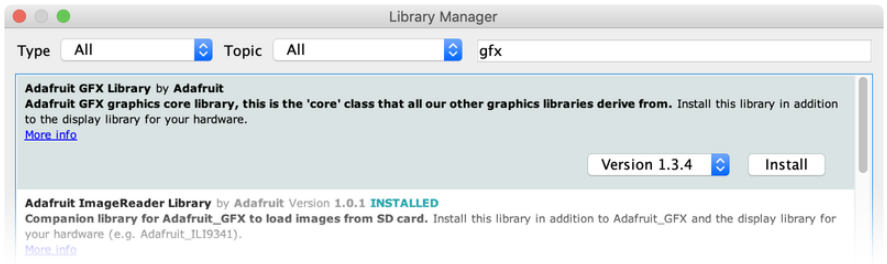
Install Arduino Libraries

We have example code ready to go for use with these TFTs. It's written for Arduino, which should be portable to any microcontroller by adapting the C++ source.

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



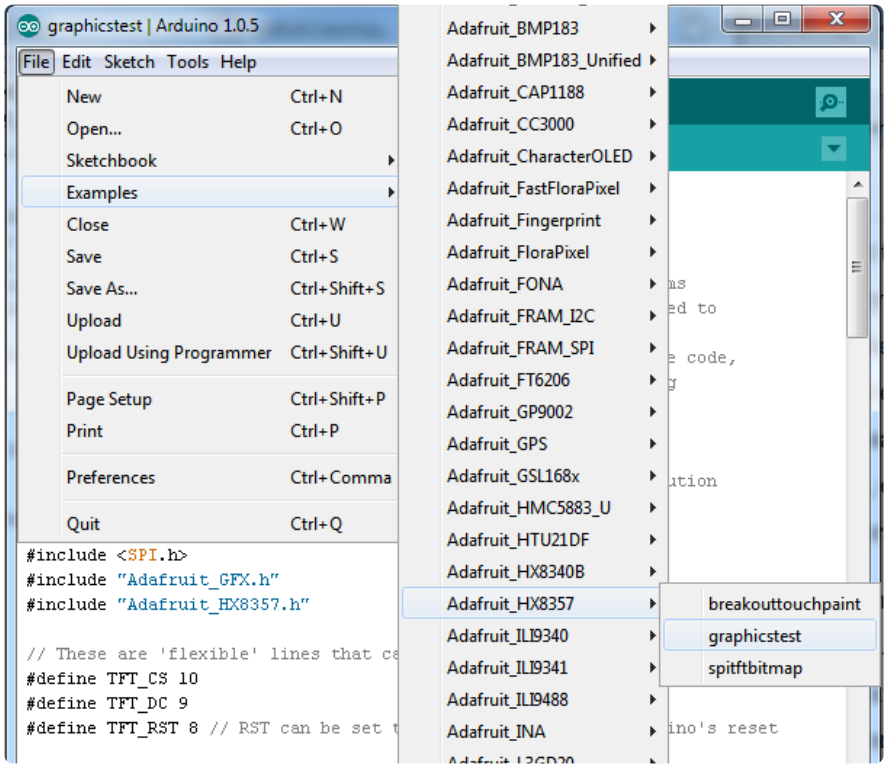
Type “gfx” in the search field to quickly find the first library — **Adafruit_GFX**:



If using an older Arduino IDE (pre-1.8.10), do the same for **Adafruit_BusIO** (newer versions do this one automatically).

Repeat the search and install steps, looking for the **Adafruit_HX8357** library.

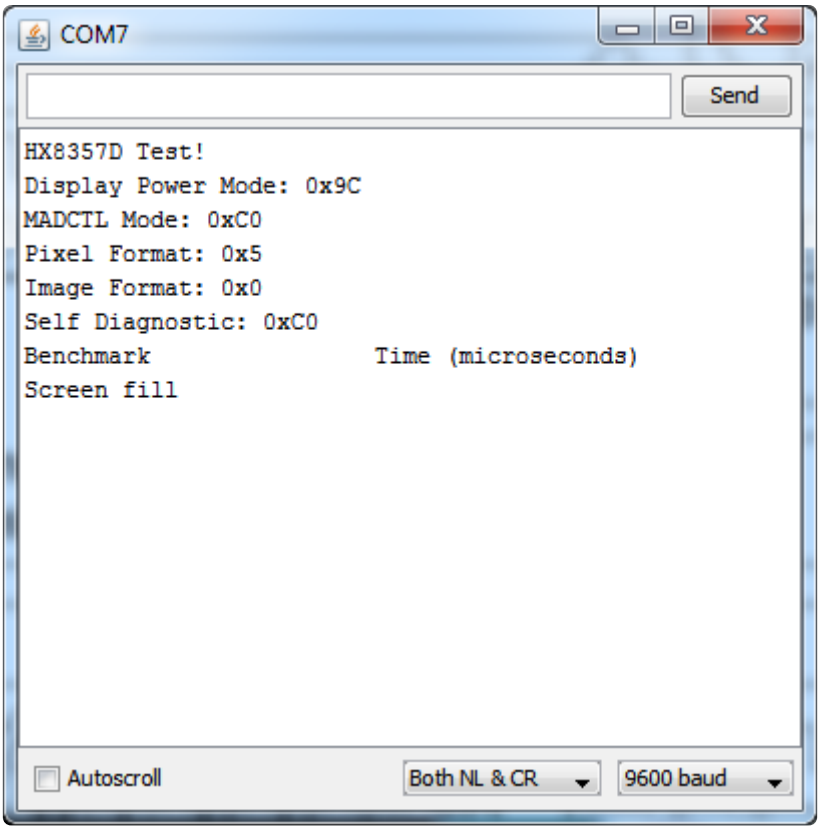
After restarting the Arduino software, you should see a new **example** folder called **Adafruit_HX8357** and inside, an example called **graphicstest**.



Upload the graphicstest sketch to your Arduino. You may need to press the Reset button to reset the Arduino and TFT. You should see a collection of graphical tests draw out on the TFT.

If you're having difficulties, check the serial console.The first thing the sketch does is read the driver configuration from the TFT, you should see the same numbers as below

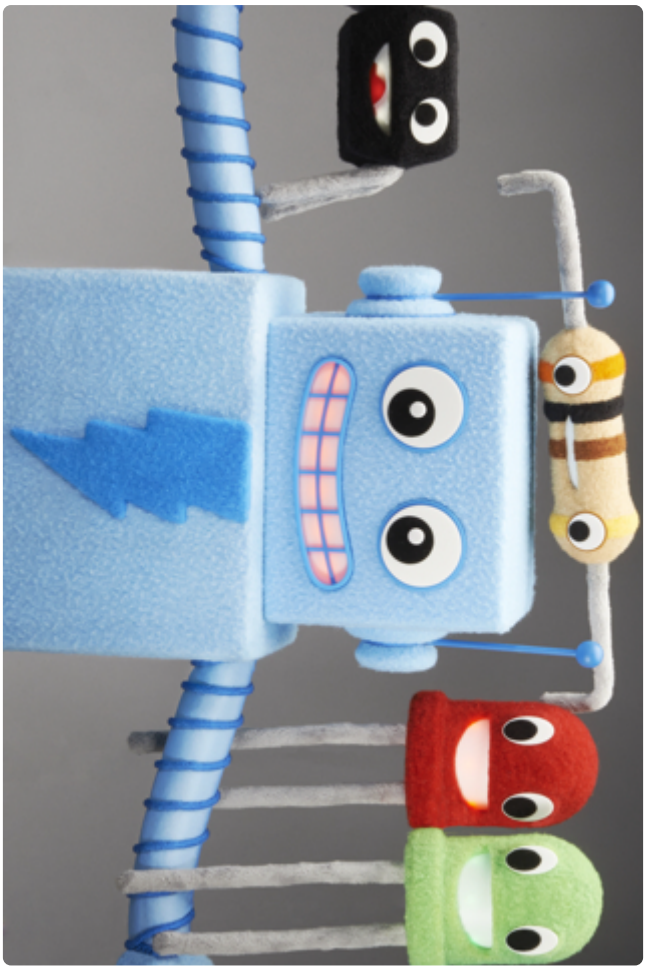
If you did not connect up the MISO line to the TFT, you wont see the read configuation bytes so please make sure you connect up the MISO line for easy debugging! Once its all working, you can remove the MISO line



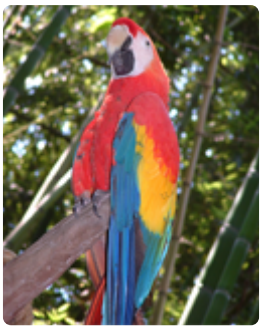
Bitmaps (SPI Mode)

There is a built-in microSD card slot on the FeatherWing, 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), and an SD card reader on whatever computer you're currently reading this with.

It's really easy to draw bitmaps. Lets start by downloading this image of Adabot and friends:



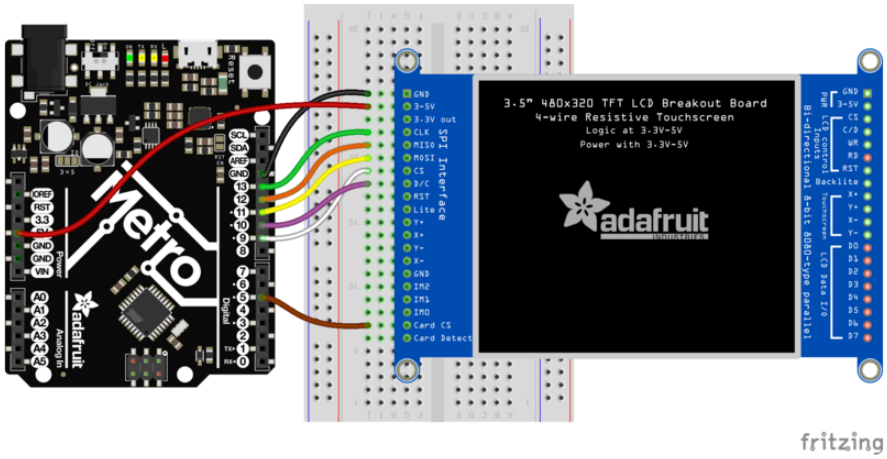
Download these two smaller images as well:



The files should be renamed (if needed) to “**adabot.bmp**”, “**parrot.bmp**” and “**wales.bmp**”, respectively, and copied to the base directory of the microSD card (not inside a folder).

(If it’s easier, you can also find these images in the “images” folder within the Adafruit_ImageReader library folder.)

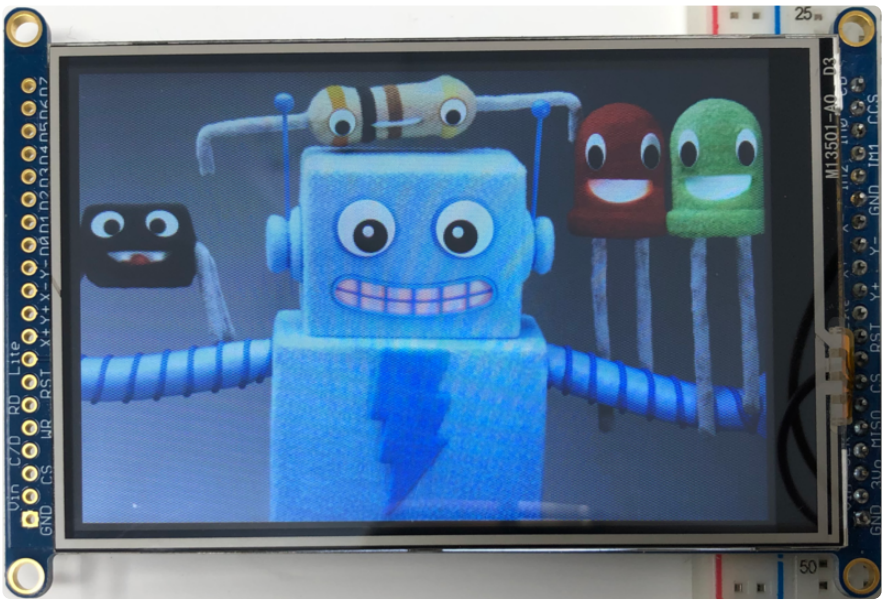
You'll need to connect up the **CCS** pin to **Digital 5** on your Arduino and swap the **D/C** and **CS** pins as well. See the Fritzing diagram below.



3.5"_TFT_Breakout.fzz

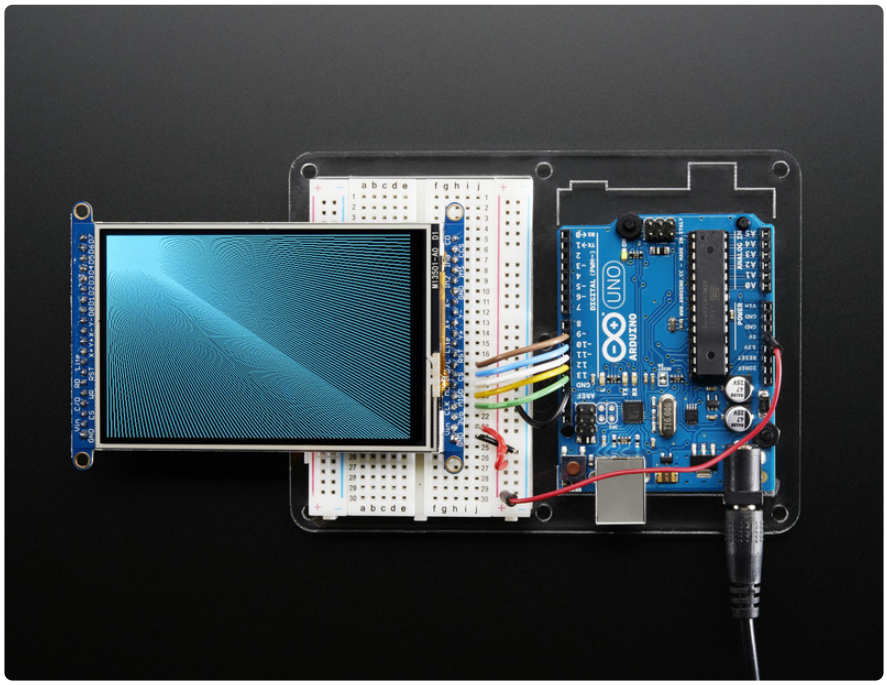
https://adafru.it/19by

Insert the microSD card into the socket in the shield. Now select the sketch **file→examples→Adafruit_ImageReader→FeatherWingHX8357** and upload this example to your Feather + Wing. You will see the your electronic friends appear! (Plus parrots...and if you're using one of the more powerful Feather boards, a whole lot of dragons.)



The Adafruit_ImageReader library, which is being used here to display .BMP images, is fully explained in its own page of the Adafruit_GFX guide (https://adafru.it/DpM).

Adafruit GFX library



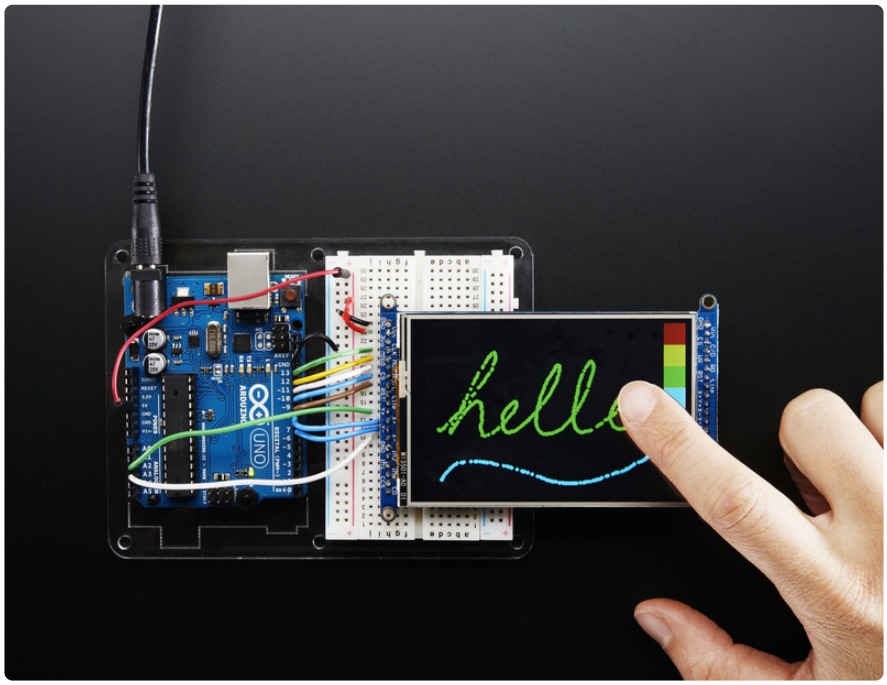
The Adafruit_GFX library for Arduino provides a common syntax and set of graphics functions for all of our TFT, LCD and OLED displays. This allows Arduino sketches to easily be adapted between display types with minimal fuss...and any new features, performance improvements and bug fixes will immediately apply across our complete offering of color displays.

The GFX library is what lets you draw points, lines, rectangles, round-rects, triangles, text, etc.

Check out our detailed tutorial here http://learn.adafruit.com/adafruit-gfx-graphics-library (https://adafru.it/aPx)

It covers the latest and greatest of the GFX library. The GFX library is used in both 8-bit and SPI modes so the underlying commands (drawLine() for example) are identical!

Touchscreen - Resistive Touch



The LCD has a 3.5" 4-wire resistive touch screen glued onto it. You can use this for detecting finger-presses, stylus', etc. You'll need 4 pins to talk to the touch panel, and at least 2 must be analog inputs. The touch screen is a completely separate part from the TFT, so be aware if you rotate the display or have the TFT off or reset, the touch screen doesn't "know" about it - its just a couple resistors!

We have a demo for the touchscreen + TFT that lets you 'paint' simple graphics. There's versions for both SPI and 8-bit mode and are included in the libraries. Just make sure you have gone thru the TFT test procedure already since this builds on that.

Remember, if you rotate the screen drawing with setRotation() you'll have to use map() or similar to flip around the X/Y coordinates for the touchscreen as well! It doesn't know about drawing rotation

Download Library

Begin by grabbing our analog/resistive touchscreen library from github (https://adafru.it/aT1) (or just click the download button)

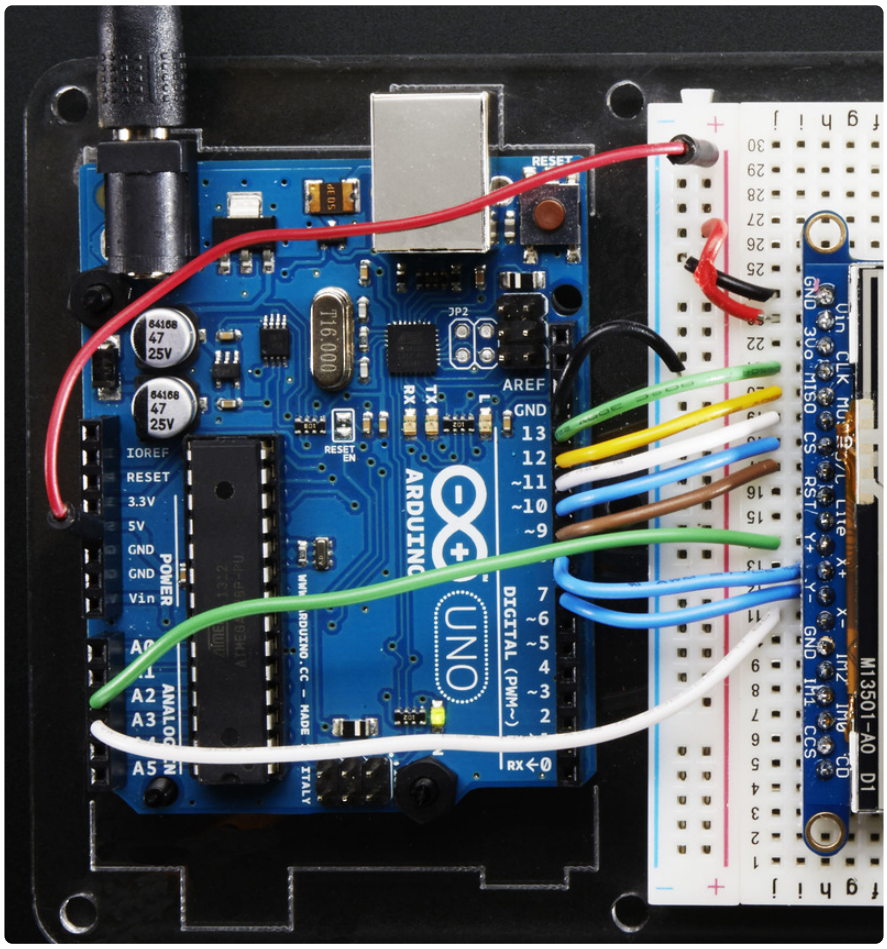
Download Adafruit Touchscreen Library

Touchscreen Paint (SPI mode)

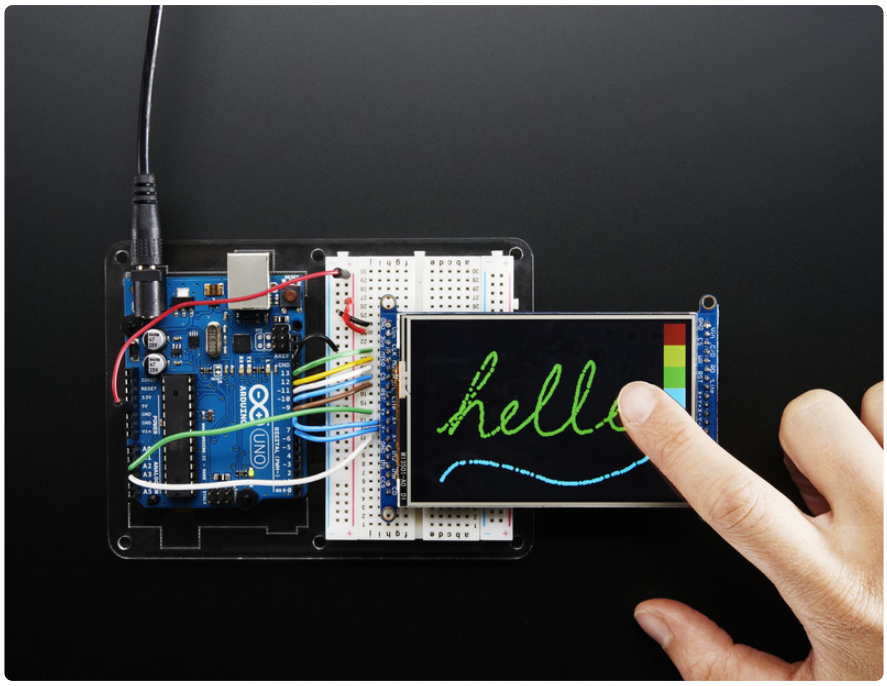
An additional 4 pins are required for the touchscreen. For the two analog pins, we'll use **A2** and **A3**. For the other two connections, you can pin any two digital pins but we'll be using **D8** and **D7** since they are available.

Wire the additional 4 pins as follows:

- Y+ to Arduino **A2**
- X+ to Arduino **D8**
- Y- to Arduino **D7**
- X- to Arduino **A3**



Load up the **breakoutTouchPaint** example from the **Adafruit_HX8357** library and try drawing with your fingernail! You can select colors by touching the 'pallette' of colors on the right



Touchscreen Paint (8-Bit mode)

Another 4 pins seems like a lot since already 12 are taken up with the TFT **but** you can **reuse** some of the pins for the TFT LCD! This is because the resistance of the panel is high enough that it doesn't interfere with the digital input/output and we can query the panel in between TFT accesses, when the pins are not being used.

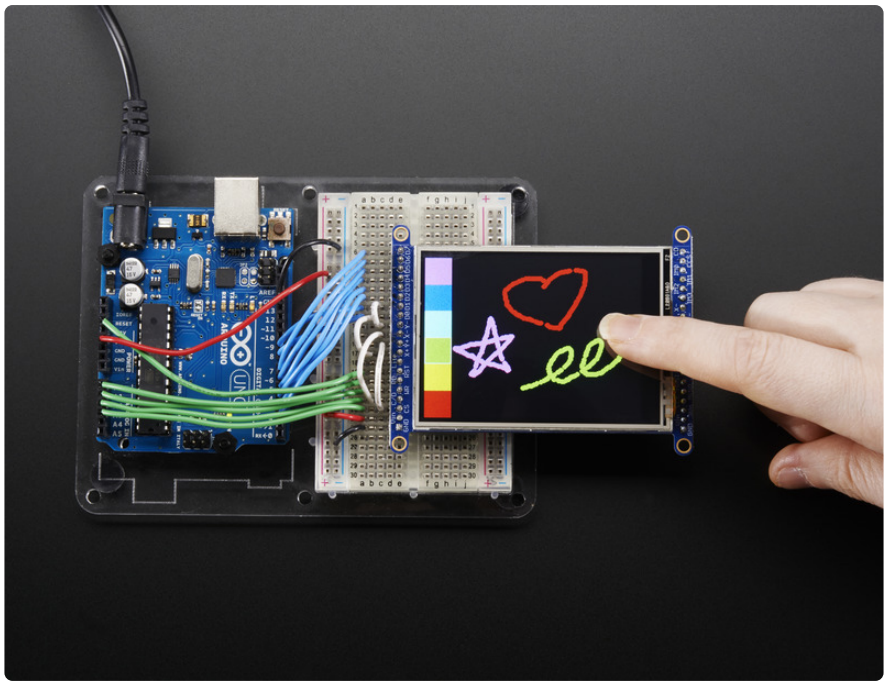
We'll be building on the wiring used in the previous drawing test for UNO

You can wire up the 4 touchscreen pins as follows. Starting from the top

- **Y-** connects to digital **#9** (also **D1**)
- The next one down (**X-**) connects to **Analog 2** (also **C/D**)
- The next one over (**Y+**) connects to **Analog 3** (also **CS**)
- The last one (**X+**) connects to digital **8**. (also **D0**)

The X- and Y+ pins pretty much have to connect to those analog pins (or to analog 4/5) but Y-/X+ can connect to any digital or analog pins.

The image below shows the wiring, its for the 2.8" TFT but its the same wiring setup



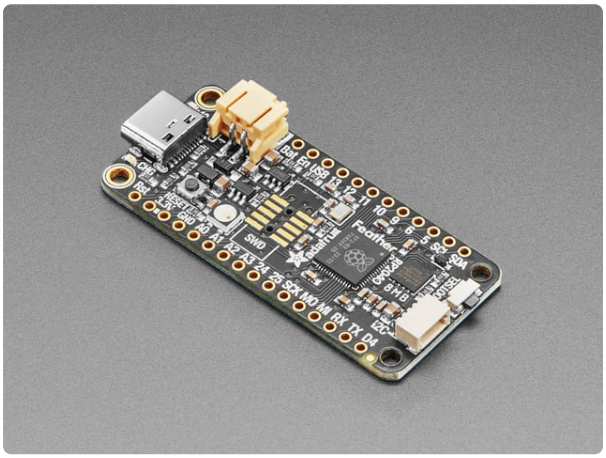
Load up the **tftpaint** example from the **Adafruit_TFTLCD** library and try drawing with your fingernail! You can select colors by touching the 'pallette' of colors on the right

---

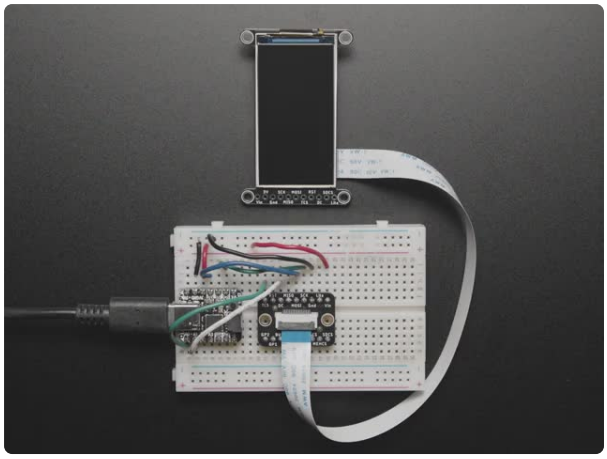
Touchscreen - Capacitive Touch

Using the 3.5" TFT Breakout with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller, plugging in your EYESPI compatible screen via the EYESPI cable, installing the library for your display type and running the provided example code.

This page uses the Feather RP2040 for demonstrating Arduino usage. You can use the same concepts to get going with any board.



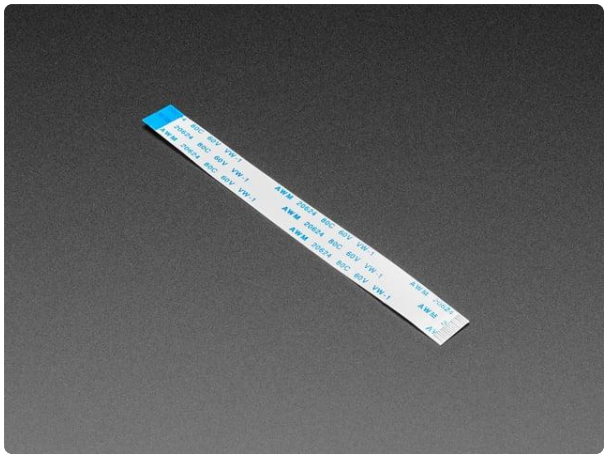
Adafruit Feather RP2040
A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the Feather...
https://www.adafruit.com/product/4884



[Adafruit EYESPI Breakout Board - 18 Pin FPC Connector](https://www.adafruit.com/product/5613)

Our most recent display breakouts have come with a new feature: an 18-pin "EYE SPI" standard FPC...

https://www.adafruit.com/product/5613



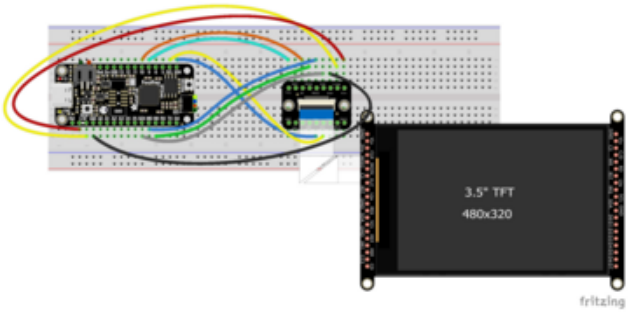
[EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type](https://www.adafruit.com/product/5239)

Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...

https://www.adafruit.com/product/5239

Wiring

Here is an Adafruit Feather RP2040 wired up to the EYESPI breakout with the display:

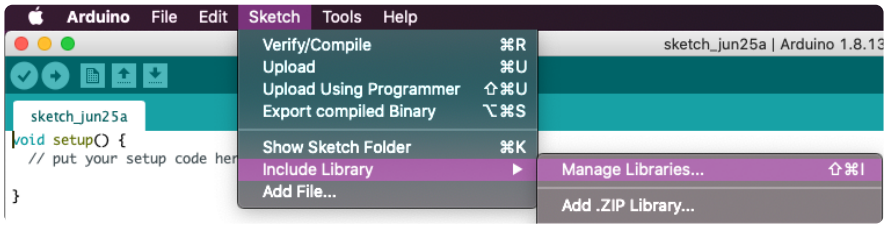


Board 3.3V to **breakout VIN (red wire)**
Board 3.3V to **breakout Lite (yellow wire)**
Board GND to **breakout GND (black wire)**
Board pin SCK to **breakout SCK (blue wire)**
Board pin MI to **breakout MISO (green wire)**
Board pin MO to **breakout MOSI (yellow wire)**
Board pin 10 to **breakout DC (orange wire)**
Board pin 9 to **breakout TCS (aqua wire)**
Board pin SCL to **breakout SCL (yellow wire)**
Board pin SDA to **breakout SDA (blue wire)**

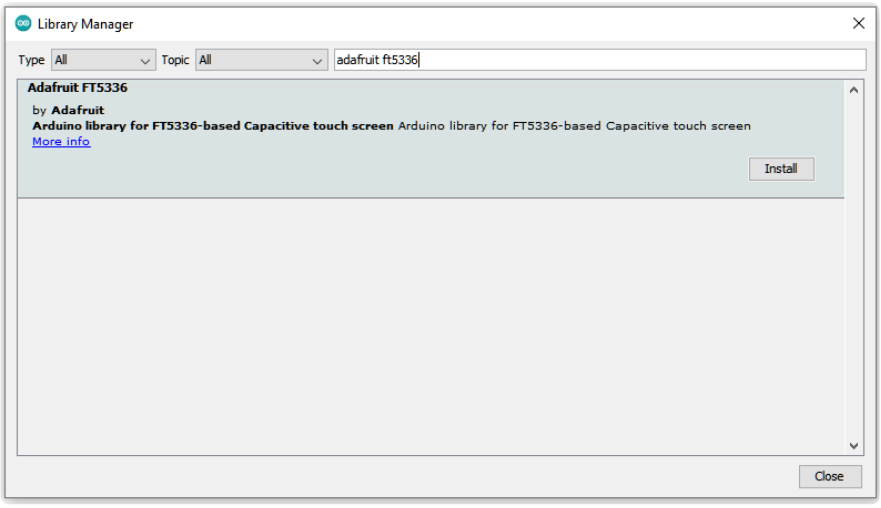
Attach the TFT screen to the **EYESPI breakout** with an **EYESPI cable** as described on the Plugging in an EYESPI Cable (https://adafru.it/19bz) page.

Library Installation

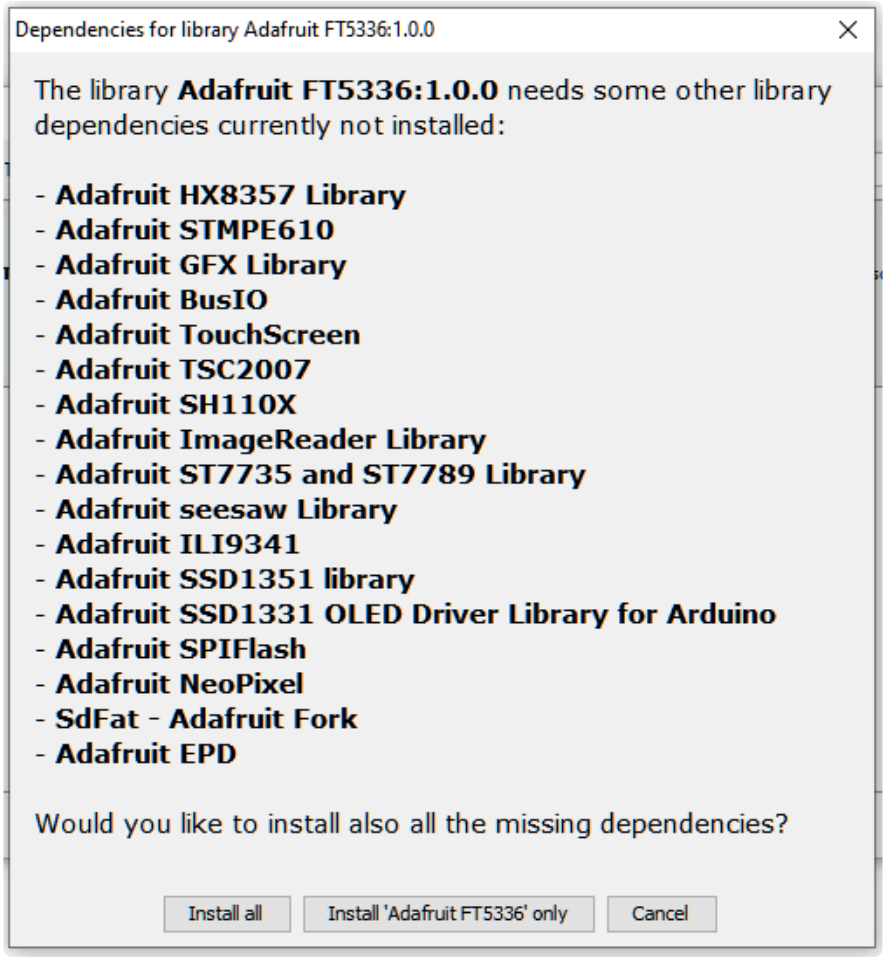
You can install the **Adafruit FT5336** library for Arduino using the Library Manager in the Arduino IDE.



Click the **Manage Libraries ...** menu item, search for **Adafruit FT5336**, and select the **Adafruit_FT5336** library:



If asked about dependencies, click "Install all".



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

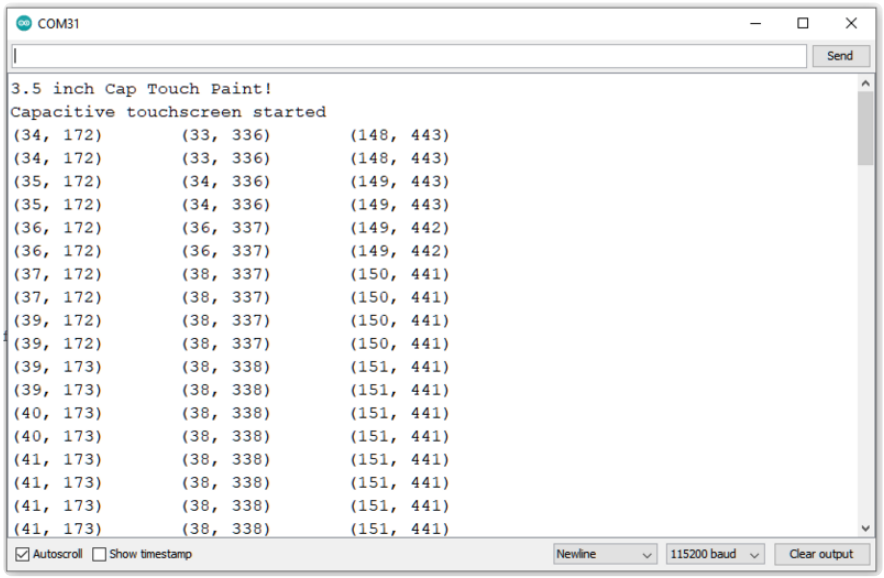
If the dependencies are already installed, you must make sure you update them through the Arduino Library Manager before loading the example!

Example Code

```
/***************************************************
  This is our touchscreen painting example for the Adafruit HX8357
  with FT5336 captouch breakout
  ----> http://www.adafruit.com/products/5846

  Check out the links above for our tutorials and wiring diagrams

  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_HX8357.h>
#include <Adafruit_FT5336.h>

// The FT5336 uses hardware I2C (SCL/SDA)
Adafruit_FT5336 ctp = Adafruit_FT5336();
#define FT5336_MAXTOUCHES  5

// The display also uses hardware SPI, plus #9 & #10
#define TFT_CS         9
#define TFT_RST        -1
#define TFT_DC         10
Adafruit_HX8357 tft = Adafruit_HX8357(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);     // pause the serial port

  Serial.println("3.5 inch Cap Touch Paint!");

  tft.begin();

  if (! ctp.begin(FT53XX_DEFAULT_ADDR, &Wire)) {  // pass in 'sensitivity'
coefficient and I2C bus
    Serial.println("Couldn't start FT5336 touchscreen controller");
    while (1) delay(10);
  }

  Serial.println("Capacitive touchscreen started");

  tft.fillScreen(HX8357_BLACK);

  // make the color selection boxes
  tft.fillRect(0, 0, BOXSIZE, BOXSIZE, HX8357_RED);
  tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, HX8357_YELLOW);
  tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, HX8357_GREEN);
```

```
  tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, HX8357_CYAN);
  tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, HX8357_BLUE);
  tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, HX8357_MAGENTA);
  tft.fillRect(BOXSIZE*6, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
  tft.fillRect(BOXSIZE*7, 0, BOXSIZE, BOXSIZE, HX8357_BLACK);

  // select the current color 'red'
  tft.drawRect(0, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
  currentcolor = HX8357_RED;
}

void loop() {
  uint8_t touches = ctp.touched();
  // Wait for a touch
  if (! touches) {
    return;
  }

  // Retrieve the points, up to 5!
  TS_Point ps[FT5336_MAXTOUCHES];
  ctp.getPoints(ps, FT5336_MAXTOUCHES);

  for (int i=0; i<FT5336_MAXTOUCHES; i++) {
    // Check if z (pressure) is zero, skip if so
    if (ps[i].z == 0) continue;
    /*
    ps[i].x = map(ps[i].x, 0, 320, 0, 320);
    ps[i].y = map(ps[i].y, 0, 480, 0, 480);
    */
    // Print out the remapped/rotated coordinates
    Serial.print("("); Serial.print(ps[i].x);
    Serial.print(", "); Serial.print(ps[i].y);
    Serial.print(")\t");
  }
  Serial.println();

  if (ps[0].y < BOXSIZE) {
     oldcolor = currentcolor;

     if (ps[0].x < BOXSIZE) {
       currentcolor = HX8357_RED;
       tft.drawRect(0, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
     } else if (ps[0].x < BOXSIZE*2) {
       currentcolor = HX8357_YELLOW;
       tft.drawRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
     } else if (ps[0].x < BOXSIZE*3) {
       currentcolor = HX8357_GREEN;
       tft.drawRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
     } else if (ps[0].x < BOXSIZE*4) {
       currentcolor = HX8357_CYAN;
       tft.drawRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
     } else if (ps[0].x < BOXSIZE*5) {
       currentcolor = HX8357_BLUE;
       tft.drawRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
     } else if (ps[0].x <= BOXSIZE*6) {
       currentcolor = HX8357_MAGENTA;
       tft.drawRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
     } else if (ps[0].x <= BOXSIZE*7) {
       currentcolor = HX8357_WHITE;
       tft.drawRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, HX8357_RED);
     } else {
        // erase
        tft.fillScreen(HX8357_BLACK);

        // make the color selection boxes
        tft.fillRect(0, 0, BOXSIZE, BOXSIZE, HX8357_RED);
        tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, HX8357_YELLOW);
        tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, HX8357_GREEN);
        tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, HX8357_CYAN);
```

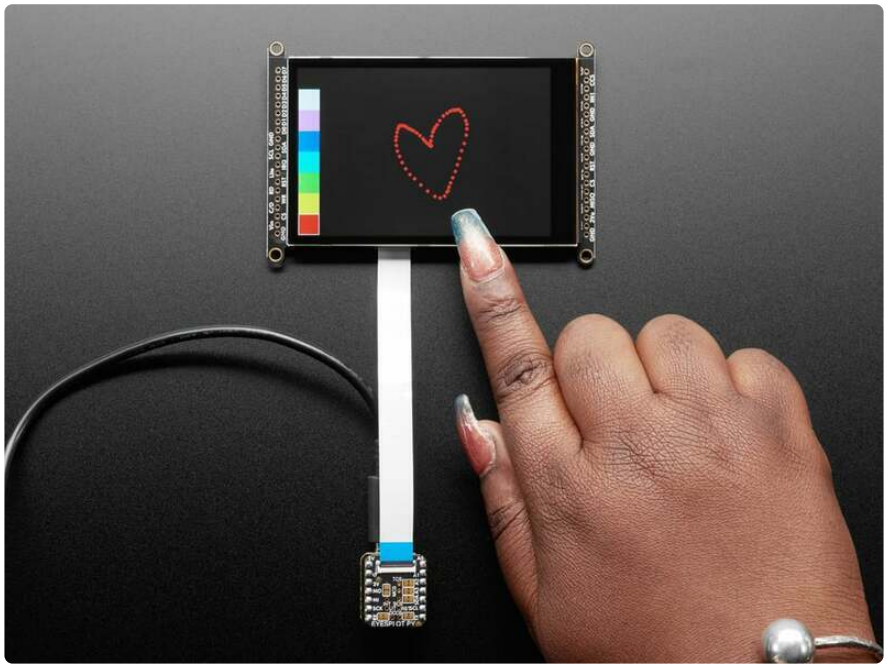
```
        tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, HX8357_BLUE);
        tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, HX8357_MAGENTA);
        tft.fillRect(BOXSIZE*6, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
      }

      if (oldcolor != currentcolor) {
        if (oldcolor == HX8357_RED)
          tft.fillRect(0, 0, BOXSIZE, BOXSIZE, HX8357_RED);
        if (oldcolor == HX8357_YELLOW)
          tft.fillRect(BOXSIZE, 0, BOXSIZE, BOXSIZE, HX8357_YELLOW);
        if (oldcolor == HX8357_GREEN)
          tft.fillRect(BOXSIZE*2, 0, BOXSIZE, BOXSIZE, HX8357_GREEN);
        if (oldcolor == HX8357_CYAN)
          tft.fillRect(BOXSIZE*3, 0, BOXSIZE, BOXSIZE, HX8357_CYAN);
        if (oldcolor == HX8357_BLUE)
          tft.fillRect(BOXSIZE*4, 0, BOXSIZE, BOXSIZE, HX8357_BLUE);
        if (oldcolor == HX8357_MAGENTA)
          tft.fillRect(BOXSIZE*5, 0, BOXSIZE, BOXSIZE, HX8357_MAGENTA);
        if (oldcolor == HX8357_WHITE)
          tft.fillRect(BOXSIZE*6, 0, BOXSIZE, BOXSIZE, HX8357_WHITE);
      }
    }
    for (int i=0; i<FT5336_MAXTOUCHES; i++) {
      // Check if z (pressure) is zero, skip if so
      if (ps[i].z == 0) continue;

      if (((ps[i].y-PENRADIUS) > BOXSIZE) && ((ps[i].y+PENRADIUS) < tft.height())) {
        tft.fillCircle(ps[i].x, ps[i].y, PENRADIUS, currentcolor);
      }
    }
  }
```



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. If you press with more than one point, you'll see multiple sets of coordinates.

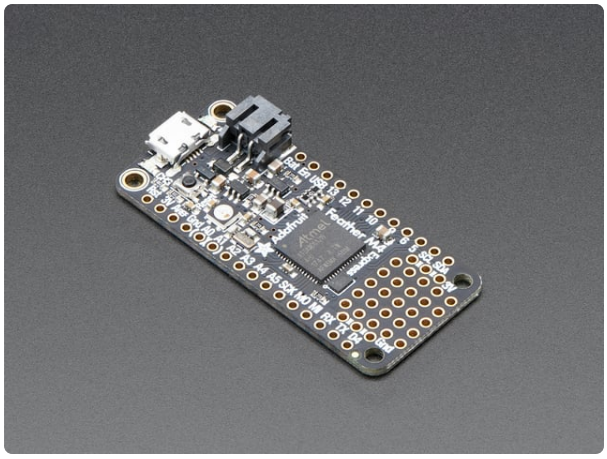
The first number is the X coordinate, the second number is the Y coordinate and the last two numbers are Z "pressure" coordinates that can tell you how hard the touch pad is being pressed.

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



CircuitPython Displayio Quickstart - Resistive Touch

You will need a board capable of running CircuitPython such as the Metro M0 Express or the Metro M4 Express. You can also use boards such as the Feather M0 Express or the Feather M4 Express. We recommend either the Metro M4 or the Feather M4 Express because it's much faster and works better for driving a display. For this guide, we will be using a Feather M4 Express. The steps should be about the same for the Feather M0 Express or either of the Metros. If you haven't already, be sure to check out our Feather M4 Express (https://adafru.it/EEm) guide.



Adafruit Feather M4 Express - Featuring ATSAMD51

It's what you've been waiting for, the Feather M4 Express featuring ATSAMD51. This Feather is fast like a swift, smart like an owl, strong like a ox-bird (it's half ox,...

https://www.adafruit.com/product/3857

For this guide, we'll assume you have a Feather M4 Express. The steps should be about the same for the Feather M0 Express. To start, if you haven't already done so,

follow the assembly instructions for the Feather M4 Express in our Feather M4 Express guide (https://adafru.it/EEm).

Preparing the Breakout

Before using the TFT Breakout, you will need to solder the headers or some wires to it. Be sure to check out the Adafruit Guide To Excellent Soldering (https://adafru.it/drl). Also, follow the SPI Wiring & Test (https://adafru.it/FxS) page of this guide to be sure your display is setup for SPI. After that, the breakout should be ready to go.

Required CircuitPython Libraries

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

Adafruit_CircuitPython_HX8357

https://adafru.it/EGf

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_hx8357**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_hx8357** 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

https://adafru.it/FiA

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

CircuitPython Code Example

```
# 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.
"""

import board
import terminalio
import displayio
from adafruit_display_text import label
from adafruit_hx8357 import HX8357

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

spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10

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

display = HX8357(display_bus, width=480, height=320)

# Make the display context
splash = displayio.Group()
display.root_group = splash

color_bitmap = displayio.Bitmap(480, 320, 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(440, 280, 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=137, y=160)
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_hx8357` driver.

```
import board
import displayio
import terminalio
```

```
from adafruit_display_text import label
from adafruit_hx8357 import HX8357
```

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

```
displayio.release_displays()
```

Next, we set the SPI object to the board's SPI with the easy shortcut function `board.SPI()`. By using this function, it finds the SPI module and initializes using the default SPI parameters. Next we set the Chip Select and Data/Command pins that will be used.

```
spi = board.SPI()
tft_cs = board.D9
tft_dc = board.D10
```

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 480 and a height of 320. 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 = HX8357(display_bus, width=480, height=320)
```

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()
display.root_group = splash
```

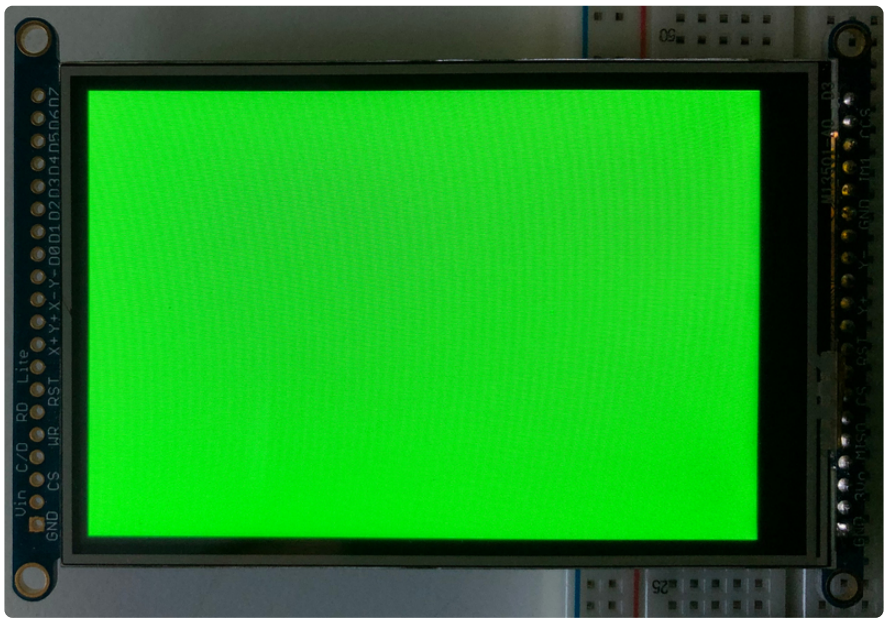
After that we create a Bitmap which is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. The Bitmaps can currently handle up to 256 different colors. We create a Palette with one color and set that color to 0x00FF00 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(480, 320, 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)
```

This creates a solid green background which we will draw on top of.



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 480x320, 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(440, 280, 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 160 for the Y coordinate, and around 137 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=137, y=160)
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

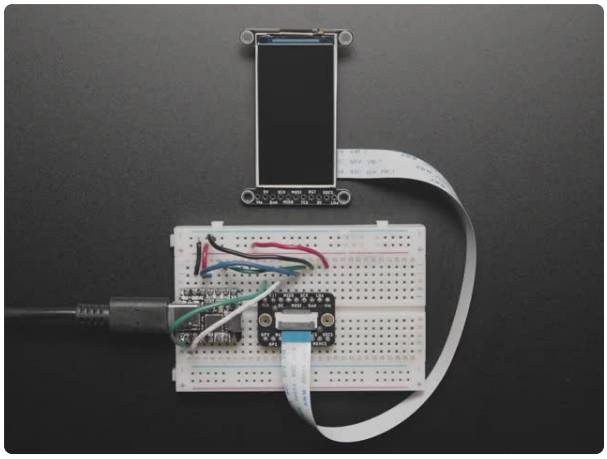
We won't be covering how to use the touchscreen with CircuitPython in this guide, but the library required for enabling resistive touch is the Adafruit_CircuitPython_STMPE610 (https://adafru.it/Fsz) library.

Where to go from here

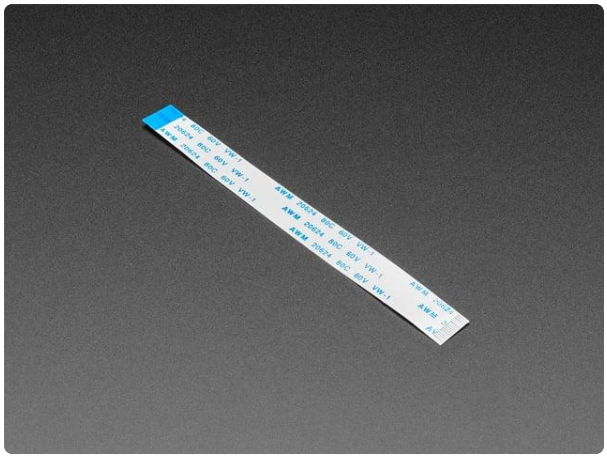
Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafru.it/EGh)

CircuitPython - Capacitive Touch

Using the 3.5" 320x480 Color TFT Touchscreen breakout with CircuitPython involves wiring up the TFT to your CircuitPython-compatible microcontroller and plugging in your EYESPI-compatible display via an EYESPI cable. Then, you load the code and necessary libraries onto your microcontroller.



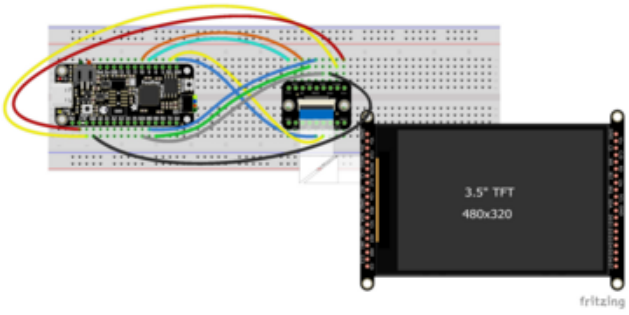
[Adafruit EYESPI Breakout Board - 18 Pin FPC Connector]
Our most recent display breakouts have come with a new feature: an 18-pin "EYE SPI" standard FPC...
https://www.adafruit.com/product/5613



[EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type]
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...
https://www.adafruit.com/product/5239

Wiring

First, wire the EYESPI breakout to your CircuitPython-compatible microcontroller. The diagram below shows wiring up to a Feather RP2040. Then, connect the TFT via an EYESPI cable to the breakout.



breakout Vin to **Feather 3.3V (red wire)**
breakout Lite to **Feather 3.3V (yellow wire)**
breakout Gnd to **Feather GND (black wire)**
breakout SCK to **Feather SCK (grey wire)**
breakout MISO to **Feather MI (green wire)**
breakout MOSI to **Feather MO (blue wire)**
breakout TCS to **Feather D9 (aqua wire)**
breakout DC to **Feather D10 (orange wire)**
breakout SCL to **Feather SCL (yellow wire)**
breakout SDA to **Feather SDA (blue wire)**

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.

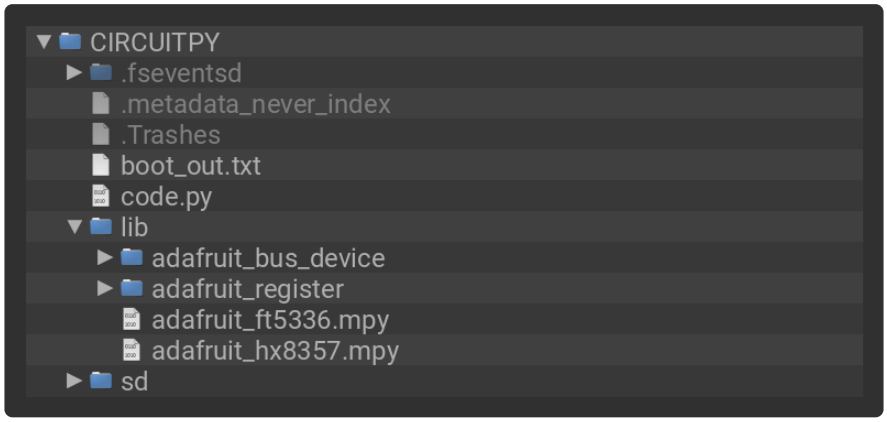
Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file.

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** and the **code.py** file to your **CIRCUITPY** drive.

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

- **/adafruit_bus_device**
- **/adafruit_register**
- **adafruit_ft5336.mpy**
- **adafruit_hx8357.mpy**

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



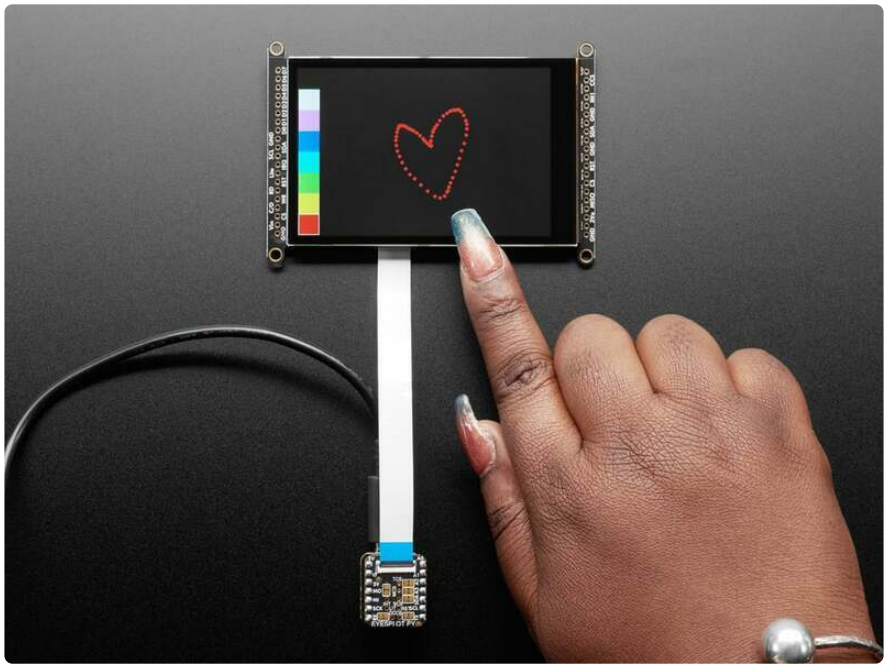
Example Code

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

"""
Touch paint example for HX83570 + FT5336 TFT Breakout
"""

import board
import displayio
from adafruit_hx8357 import HX8357
import adafruit_ft5336
```

```
displayio.release_displays()

spi = board.SPI()
# for eyespi bff
# tft_cs = board.TX
# tft_dc = board.RX
# else:
tft_cs = board.D9
tft_dc = board.D10

display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs)
# display is rotated to align x, y with touch screen x, y
display = HX8357(display_bus, width=320, height=480, rotation=90)

i2c = board.I2C()  # uses board.SCL and board.SDA
touch = adafruit_ft5336.Adafruit_FT5336(i2c)

pixel_size = 10
palette_width = 45
palette_height = 320 // 8

splash = displayio.Group()
display.root_group = splash

bitmap = displayio.Bitmap(320, 480, 8)
color_palette = displayio.Palette(8)
color_palette[0] = 0x000000
color_palette[1] = 0xFF0000
color_palette[2] = 0xFFFF00
color_palette[3] = 0x00FF00
color_palette[4] = 0x00FFFF
color_palette[5] = 0x0000FF
color_palette[6] = 0xFF00FF
color_palette[7] = 0xFFFFFF

tile_grid = displayio.TileGrid(bitmap, pixel_shader=color_palette)
# tilegrid is flipped to align x, y with touch screen x, y
tile_grid.flip_y = True
tile_grid.flip_x = True

splash.append(tile_grid)

display.root_group = splash

current_color = 7

for i in range(palette_width):
    for j in range(palette_height):
        bitmap[j + palette_height, i] = 1
        bitmap[j + palette_height * 2, i] = 2
        bitmap[j + palette_height * 3, i] = 3
        bitmap[j + palette_height * 4, i] = 4
        bitmap[j + palette_height * 5, i] = 5
        bitmap[j + palette_height * 6, i] = 6
        bitmap[j + palette_height * 7, i] = 0

while True:
    if touch.touched:
        try:
            for t in touch.points:
                x = t[0]
                y = t[1]
                print(x, y)
                if not 0 <= x < display.width or not 0 <= y < display.height:
                    continue  # Skip out of bounds touches
                if y < palette_width:
                    current_color = bitmap[x, y]
                else:
                    for i in range(pixel_size):
```

```
                    for j in range(pixel_size):
                        x_pixel = x - (pixel_size // 2) + i
                        y_pixel = y - (pixel_size // 2) + j
                        if (
                            0 <= x_pixel < display.width
                            and 0 <= y_pixel < display.height
                        ):
                            bitmap[x_pixel, y_pixel] = current_color
        except RuntimeError:
            pass
```

Once everything is copied over, on your display, you should the touch paint example begin to run. You can touch the different colors on the side of the TFT and doodle on the screen.



Python Example Wiring

It's easy to use display breakouts with Python and the Adafruit CircuitPython RGB Display (https://adafru.it/u1C) module. This module allows you to easily write Python code to control the display.

We'll cover how to wire the display to your Raspberry Pi. First assemble your display.

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Connect the display as shown below to your Raspberry Pi.

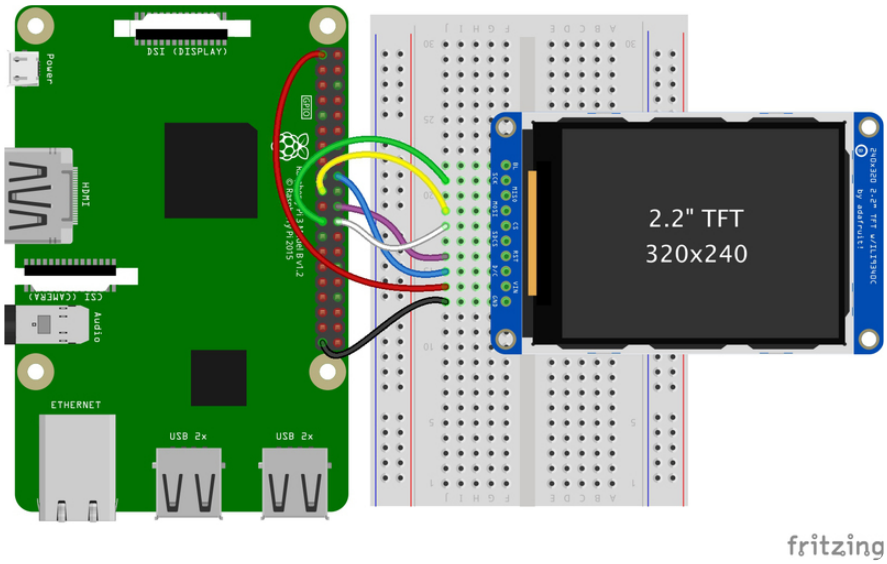
Note this is not a kernel driver that will let you have the console appear on the TFT. However, this is handy when you can't install an fbtft driver, and want to use the TFT purely from 'user Python' code!

You can only use this technique with Linux/computer devices that have hardware SPI support, and not all single board computers have an SPI device so check before continuing

ILI9341 and HX-8357-based Displays

2.2" Display

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



Download the Fritzing Diagram

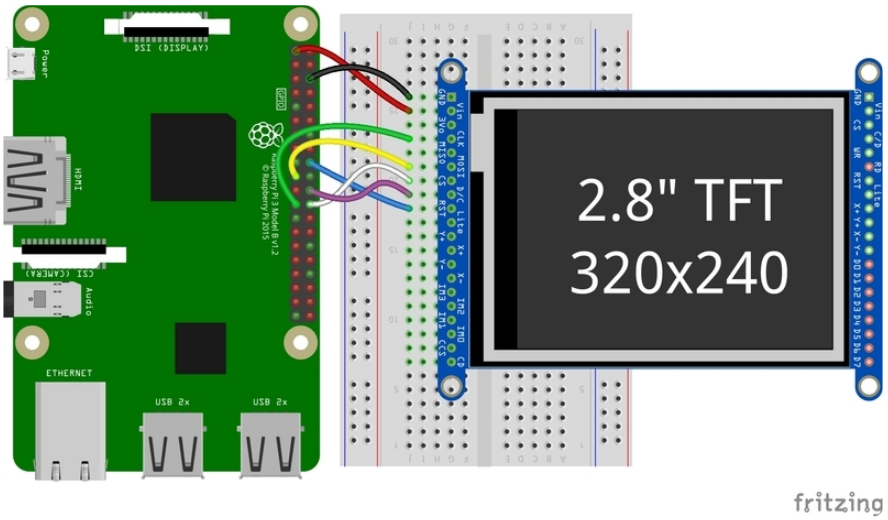
https://adafru.it/H6C

2.4", 2.8", 3.2", and 3.5" Displays

These displays are set up to use the 8-bit data lines by default. We want to use them for SPI. To do that, you'll need to either solder bridge some pads on the back or connect the appropriate IM lines to 3.3V with jumper wires. Check the back of your display for the correct solder pads or IM lines to put it in SPI mode.

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

These larger displays are set to use 8-bit data lines by default and may need to be modified to use SPI.



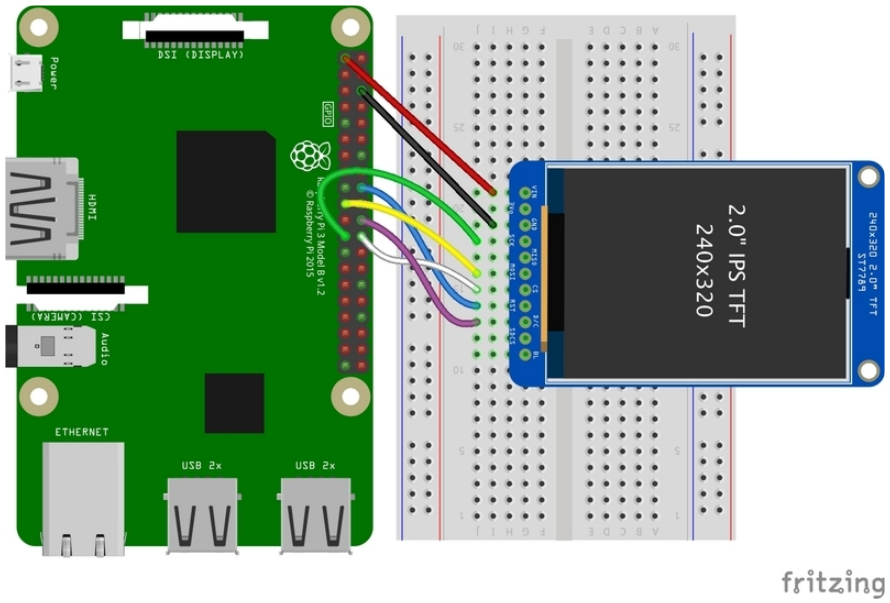
Download the Fritzing Diagram

https://adafru.it/H7a

ST7789 and ST7735-based Displays
1.3", 1.54", and 2.0" IPS TFT Display

- **Vin** connects to the Raspberry Pi's **3V** pin

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

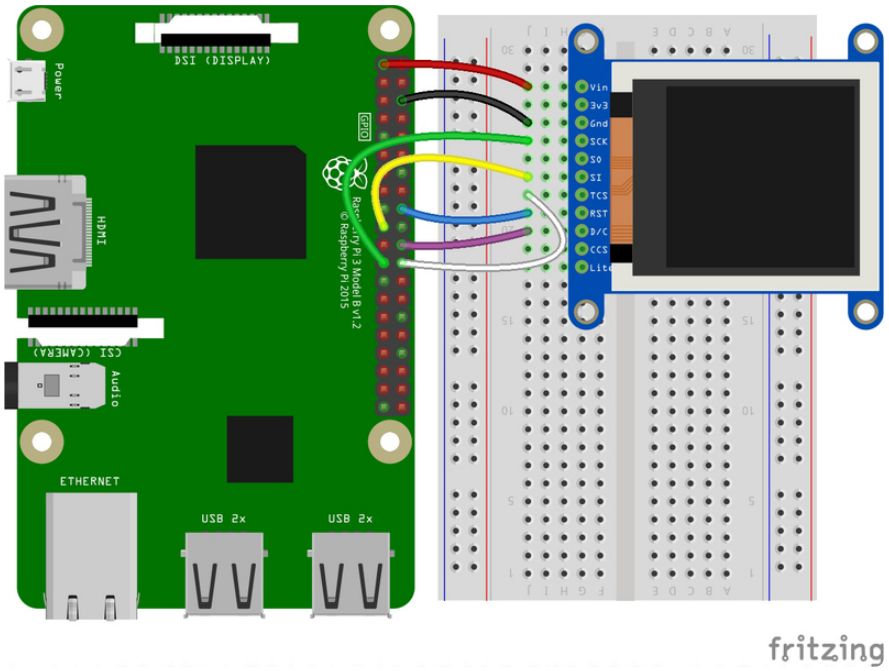


Download the Fritzing Diagram

https://adafru.it/H7A

0.96", 1.14", and 1.44" Displays

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

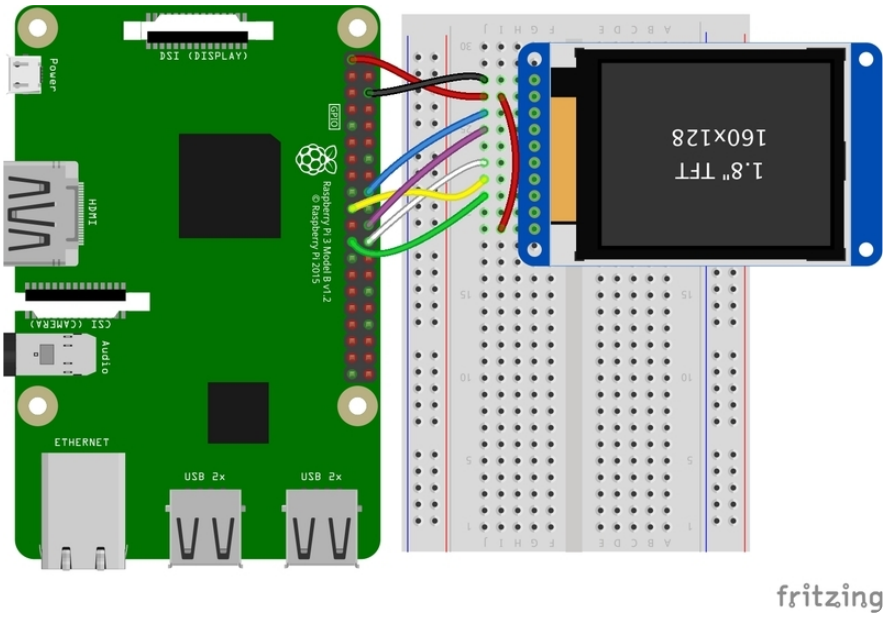


Download the Fritzing Diagram

https://adafru.it/H7B

1.8" Display

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

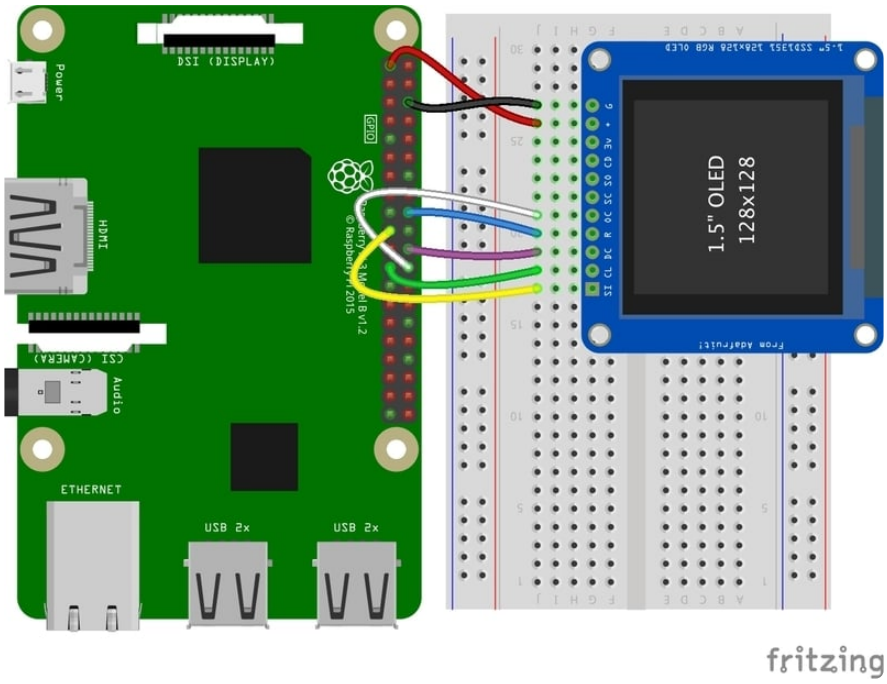


Download the Fritzing Diagram

https://adafru.it/H8a

SSD1351-based Displays
1.27" and 1.5" OLED Displays

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

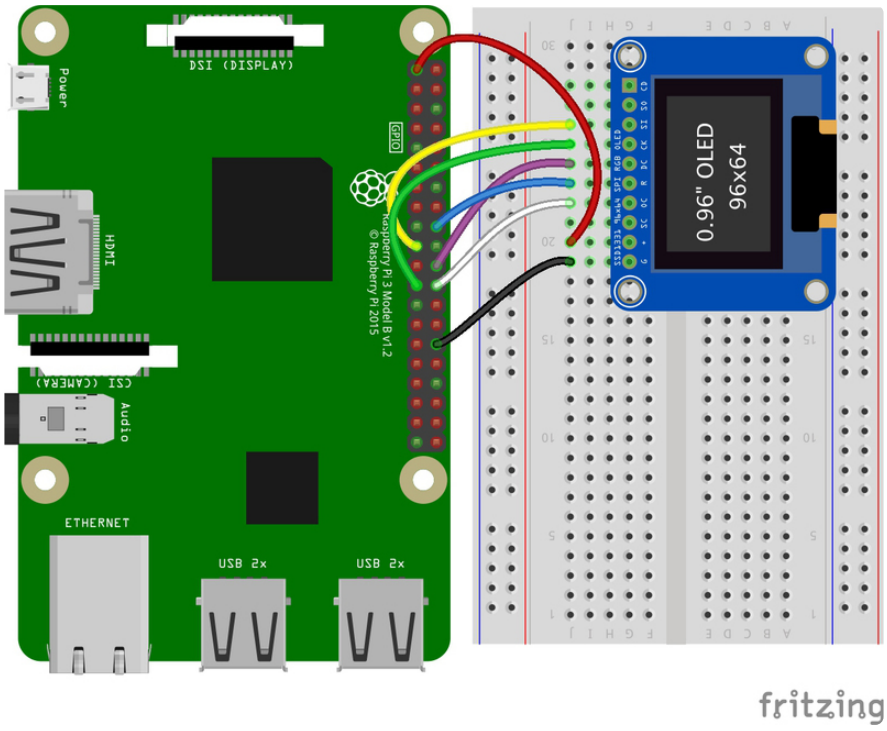


Download the Fritzing Diagram

https://adafru.it/H8A

SSD1331-based Display
0.96" OLED Display

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



Download the Fritzing Diagram

https://adafru.it/OaF

Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling SPI on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

If you have previously installed the Kernel Driver with the PiTFT Easy Setup, you will need to remove it first in order to run this example.

Python Installation of RGB Display Library

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-rgb-display`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

If that complains about pip3 not being installed, then run this first to install it:

- `sudo apt-get install python3-pip`

DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

- `sudo apt-get install fonts-dejavu`

This package was previously calls **ttf-dejavu**, so if you are running an older version of Raspberry Pi OS, it may be called that.

Pillow Library

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

- `sudo apt-get install python3-pil`

If you installed the PIL through PIP, you may need to install some additional libraries:

- `sudo apt-get install libopenjp2-7 libtiff5 libatlas-base-dev`

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

Python Usage

> If you have previously installed the Kernel Driver with the PiTFT Easy Setup, you will need to remove it first in order to run this example.

Now that you have everything setup, we're going to look over three different examples. For the first, we'll take a look at automatically scaling and cropping an image and then centering it on the display.

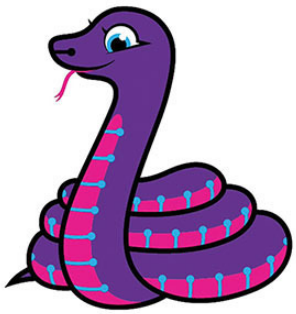
Turning on the Backlight

On some displays, the backlight is controlled by a separate pin such as the 1.3" TFT Bonnet with Joystick. On such displays, running the below code will likely result in the display remaining black. To turn on the backlight, you will need to add a small snippet of code. If your backlight pin number differs, be sure to change it in the code:

```
# Turn on the Backlight
backlight = DigitalInOut(board.D26)
backlight.switch_to_output()
backlight.value = True
```

Displaying an Image

Here's the full code to the example. We will go through it section by section to help you better understand what is going on. Let's start by downloading an image of Blinka. This image has enough border to allow resizing and cropping with a variety of display sizes and rations to still look good.



Make sure you save it as **blinka.jpg** and place it in the same folder as your script. Here's the code we'll be loading onto the Raspberry Pi. We'll go over the interesting parts.

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

"""
Be sure to check the learn guides for more usage information.

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries
"""

import digitalio
import board
```

```
from PIL import Image, ImageDraw
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

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

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

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                           # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                           # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,       # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,    # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                         # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height
image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)
```

```
image = Image.open("blinka.jpg")

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

# Crop and center the image
x = scaled_width // 2 - width // 2
y = scaled_height // 2 - height // 2
image = image.crop((x, y, x + width, y + height))

# Display image.
disp.image(image)
```

So we start with our usual imports including a couple of Pillow modules and the display drivers. That is followed by defining a few pins here. The reason we chose these is because they allow you to use the same code with the PiTFT if you chose to do so.

```
import digitalio
import board
from PIL import Image, ImageDraw
import adafruit_rgb_display.ili9341 as ili9341
import adafruit_rgb_display.st7789 as st7789
import adafruit_rgb_display.hx8357 as hx8357
import adafruit_rgb_display.st7735 as st7735
import adafruit_rgb_display.ssd1351 as ssd1351
import adafruit_rgb_display.ssd1331 as ssd1331

# Configuration for CS and DC pins
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)
```

Next we'll set the baud rate from the default 24 MHz so that it works on a variety of displays. The exception to this is the SSD1351 driver, which will automatically limit it to 16MHz even if you pass 24MHz. We'll set up out SPI bus and then initialize the display.

We wanted to make these examples work on as many displays as possible with very few changes. The ILI9341 display is selected by default. For other displays, go ahead and comment out these lines:

```
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
```

and uncomment the line appropriate for your display and possibly the line below in the case of longer initialization sequences. The displays have a rotation property so that it can be set in just one place.

```
#disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
#disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
#disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
#disp = hx8357.HX8357(spi, rotation=180,                           # 3.5" HX8357
#disp = st7735.ST7735R(spi, rotation=90,                           # 1.8" ST7735R
#disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
#disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,       # 0.96" MiniTFT
Rev A ST7735R
#disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,    # 0.96" MiniTFT
Rev B ST7735R
#x_offset=26, y_offset=1,#disp = ssd1351.SSD1351(spi,
rotation=180,                          # 1.5" SSD1351
#disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
#disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE
)
```

Next we read the current rotation setting of the display and if it is 90 or 270 degrees, we need to swap the width and height for our calculations, otherwise we just grab the width and height. We will create an `image` with our dimensions and use that to create a `draw` object. The `draw` object will have all of our drawing functions.

```
# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width   # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width   # we swap height/width to rotate it to landscape!
    height = disp.height
image = Image.new('RGB', (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)
```

Next we clear whatever is on the screen by drawing a black rectangle. This isn't strictly necessary since it will be overwritten by the image, but it kind of sets the stage.

```
# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)
```

Next we open the Blinka image, which we've named **blinka.jpg**, which assumes it is in the same directory that you are running the script from. Feel free to change it if it doesn't match your configuration.

```
image = Image.open("blinka.jpg")
```

Here's where it starts to get interesting. We want to scale the image so that it matches either the width or height of the display, depending on which is smaller, so that we have some of the image to chop off when we crop it. So we start by calculating the width to height ration of both the display and the image. If the height is the closer of the dimensions, we want to match the image height to the display height and let it be a bit wider than the display. Otherwise, we want to do the opposite.

Once we've figured out how we're going to scale it, we pass in the new dimensions and using a **Bicubic** rescaling method, we reassign the newly rescaled image back to `image`. Pillow has quite a few different methods to choose from, but Bicubic does a great job and is reasonably fast.

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

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

```
# Crop and center the image
x = scaled_width // 2 - width // 2
y = scaled_height // 2 - height // 2
image = image.crop((x, y, x + width, y + height))
```

Finally, we take our image and display it. At this point, the image should have the exact same dimensions at the display and fill it completely.

```
disp.image(image)
```

Drawing Shapes and Text

In the next example, we'll take a look at drawing shapes and text. This is very similar to the displayio example, but it uses Pillow instead. Here's the code for that.

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

"""
This demo will draw a few rectangles onto the screen along with some text
on top of that.

This example is for use on (Linux) computers that are using CPython with
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!

Author(s): Melissa LeBlanc-Williams for Adafruit Industries
"""

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

# First define some constants to allow easy resizing of shapes.
BORDER = 20
FONTSIZE = 24

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

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

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

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                           # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                           # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,       # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,    # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                         # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height

image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a green filled box as the background
draw.rectangle((0, 0, width, height), fill=(0, 255, 0))
disp.image(image)

# Draw a smaller inner purple rectangle
draw.rectangle(
    (BORDER, BORDER, width - BORDER - 1, height - BORDER - 1), fill=(170, 0, 136)
)

# Load a TTF Font
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
FONTSIZE)

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text(
    (width // 2 - font_width // 2, height // 2 - font_height // 2),
    text,
```

```
    font=font,
    fill=(255, 255, 0),
)

# Display image.
disp.image(image)
```

Just like in the last example, we'll do our imports, but this time we're including the `ImageFont` Pillow module because we'll be drawing some text this time.

```
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
```

Next we'll define some parameters that we can tweak for various displays. The `BORDER` will be the size in pixels of the green border between the edge of the display and the inner purple rectangle. The `FONTSIZE` will be the size of the font in points so that we can adjust it easily for different displays.

```
BORDER = 20
FONTSIZE = 24
```

Next, just like in the previous example, we will set up the display, setup the rotation, and create a draw object. **If you have are using a different display than the ILI9341, go ahead and adjust your initializer as explained in the previous example.** After that, we will setup the background with a green rectangle that takes up the full screen. To get green, we pass in a tuple that has our **Red**, **Green**, and **Blue** color values in it in that order which can be any integer from `0` to `255`.

```
draw.rectangle((0, 0, width, height), fill=(0, 255, 0))
disp.image(image)
```

Next we will draw an inner purple rectangle. This is the same color value as our example in displayio quickstart, except the hexadecimal values have been converted to decimal. We use the `BORDER` parameter to calculate the size and position that we want to draw the rectangle.

```
draw.rectangle((BORDER, BORDER, width - BORDER - 1, height - BORDER - 1),
               fill=(170, 0, 136))
```

Next we'll load a TTF font. The `DejaVuSans.ttf` font should come preloaded on your Pi in the location in the code. We also make use of the `FONTSIZE` parameter that we discussed earlier.

```
# Load a TTF Font
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf',
FONTSIZE)
```

Now we draw the text Hello World onto the center of the display. You may recognize the centering calculation was the same one we used to center crop the image in the previous example. In this example though, we get the font size values using the `getsize()` function of the font object.

```
# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text((width//2 - font_width//2, height//2 - font_height//2),
          text, font=font, fill=(255, 255, 0))
```

Finally, just like before, we display the image.

```
disp.image(image)
```

Displaying System Information

In this last example we'll take a look at getting the system information and displaying it. This can be very handy for system monitoring. Here's the code for that example:

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

"""
This will show some Linux Statistics on the attached display. Be sure to adjust
to the display you have connected. Be sure to check the learn guides for more
usage information.

This example is for use on (Linux) computers that are using CPython with
```

```
Adafruit Blinka to support CircuitPython libraries. CircuitPython does
not support PIL/pillow (python imaging library)!
"""

import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_rgb_display import ili9341
from adafruit_rgb_display import st7789  # pylint: disable=unused-import
from adafruit_rgb_display import hx8357  # pylint: disable=unused-import
from adafruit_rgb_display import st7735  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1351  # pylint: disable=unused-import
from adafruit_rgb_display import ssd1331  # pylint: disable=unused-import

# Configuration for CS and DC pins (these are PiTFT defaults):
cs_pin = digitalio.DigitalInOut(board.CE0)
dc_pin = digitalio.DigitalInOut(board.D25)
reset_pin = digitalio.DigitalInOut(board.D24)

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

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

# pylint: disable=line-too-long
# Create the display:
# disp = st7789.ST7789(spi, rotation=90,                            # 2.0" ST7789
# disp = st7789.ST7789(spi, height=240, y_offset=80, rotation=180,  # 1.3", 1.54"
ST7789
# disp = st7789.ST7789(spi, rotation=90, width=135, height=240, x_offset=53,
y_offset=40, # 1.14" ST7789
# disp = st7789.ST7789(spi, rotation=90, width=172, height=320, x_offset=34, #
1.47" ST7789
# disp = st7789.ST7789(spi, rotation=270, width=170, height=320, x_offset=35, #
1.9" ST7789
# disp = hx8357.HX8357(spi, rotation=180,                           # 3.5" HX8357
# disp = st7735.ST7735R(spi, rotation=90,                           # 1.8" ST7735R
# disp = st7735.ST7735R(spi, rotation=270, height=128, x_offset=2, y_offset=3,   #
1.44" ST7735R
# disp = st7735.ST7735R(spi, rotation=90, bgr=True, width=80,       # 0.96" MiniTFT
Rev A ST7735R
# disp = st7735.ST7735R(spi, rotation=90, invert=True, width=80,    # 0.96" MiniTFT
Rev B ST7735R
# x_offset=26, y_offset=1,
# disp = ssd1351.SSD1351(spi, rotation=180,                         # 1.5" SSD1351
# disp = ssd1351.SSD1351(spi, height=96, y_offset=32, rotation=180, # 1.27" SSD1351
# disp = ssd1331.SSD1331(spi, rotation=180,                         # 0.96" SSD1331
disp = ili9341.ILI9341(
    spi,
    rotation=90,  # 2.2", 2.4", 2.8", 3.2" ILI9341
    cs=cs_pin,
    dc=dc_pin,
    rst=reset_pin,
    baudrate=BAUDRATE,
)
# pylint: enable=line-too-long

# Create blank image for drawing.
# Make sure to create image with mode 'RGB' for full color.
if disp.rotation % 180 == 90:
    height = disp.width  # we swap height/width to rotate it to landscape!
    width = disp.height
else:
    width = disp.width  # we swap height/width to rotate it to landscape!
    height = disp.height
```

```
image = Image.new("RGB", (width, height))

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=(0, 0, 0))
disp.image(image)

# First define some constants to allow easy positioning of text.
padding = -2
x = 0

# Load a TTF font.  Make sure the .ttf font file is in the
# same directory as the python script!
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 24)

while True:
    # Draw a black filled box to clear the image.
    draw.rectangle((0, 0, width, height), outline=0, fill=0)

    # Shell scripts for system monitoring from here:
    # https://unix.stackexchange.com/questions/119126/command-to-display-memory-
usage-disk-usage-and-cpu-load
    cmd = "hostname -I | cut -d' ' -f1"
    IP = "IP: " + subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\",
$3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = 'df -h | awk \'$NF=="/"{printf "Disk: %d/%d GB  %s", $3,$2,$5}\''
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "cat /sys/class/thermal/thermal_zone0/temp |  awk '{printf \"CPU Temp: %.
1f C\", $(NF-0) / 1000}'"  # pylint: disable=line-too-long
    Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.
    y = padding
    draw.text((x, y), IP, font=font, fill="#FFFFFF")
    y += font.getsize(IP)[1]
    draw.text((x, y), CPU, font=font, fill="#FFFF00")
    y += font.getsize(CPU)[1]
    draw.text((x, y), MemUsage, font=font, fill="#00FF00")
    y += font.getsize(MemUsage)[1]
    draw.text((x, y), Disk, font=font, fill="#0000FF")
    y += font.getsize(Disk)[1]
    draw.text((x, y), Temp, font=font, fill="#FF00FF")

    # Display image.
    disp.image(image)
    time.sleep(0.1)
```

Just like the last example, we'll start by importing everything we imported, but we're adding two more imports. The first one is `time` so that we can add a small delay and the other is `subprocess` so we can gather some system information.

```
import time
import subprocess
import digitalio
import board
from PIL import Image, ImageDraw, ImageFont
import adafruit_rgb_display.ili9341 as ili9341
```

Next, just like in the first two examples, we will set up the display, setup the rotation, and create a draw object. **If you have are using a different display than the ILI9341, go ahead and adjust your initializer as explained in the previous example.**

Just like in the first example, we're going to draw a black rectangle to fill up the screen. After that, we're going to set up a couple of constants to help with positioning text. The first is the `padding` and that will be the Y-position of the top-most text and the other is `x` which is the X-Position and represents the left side of the text.

```
# First define some constants to allow easy positioning of text.
padding = -2
x = 0
```

Next, we load a font just like in the second example.

```
font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', 24)
```

Now we get to the main loop and by using `while True:`, it will loop until **Control+C** is pressed on the keyboard. The first item inside here, we clear the screen, but notice that instead of giving it a tuple like before, we can just pass `0` and it will draw black.

```
draw.rectangle((0, 0, width, height), outline=0, fill=0)
```

Next, we run a few scripts using the `subprocess` function that get called to the Operating System to get information. The in each command is passed through awk in order to be formatted better for the display. By having the OS do the work, we don't have to. These little scripts came from `https://unix.stackexchange.com/questions/119126/command-to-display-memory-usage-disk-usage-and-cpu-load`

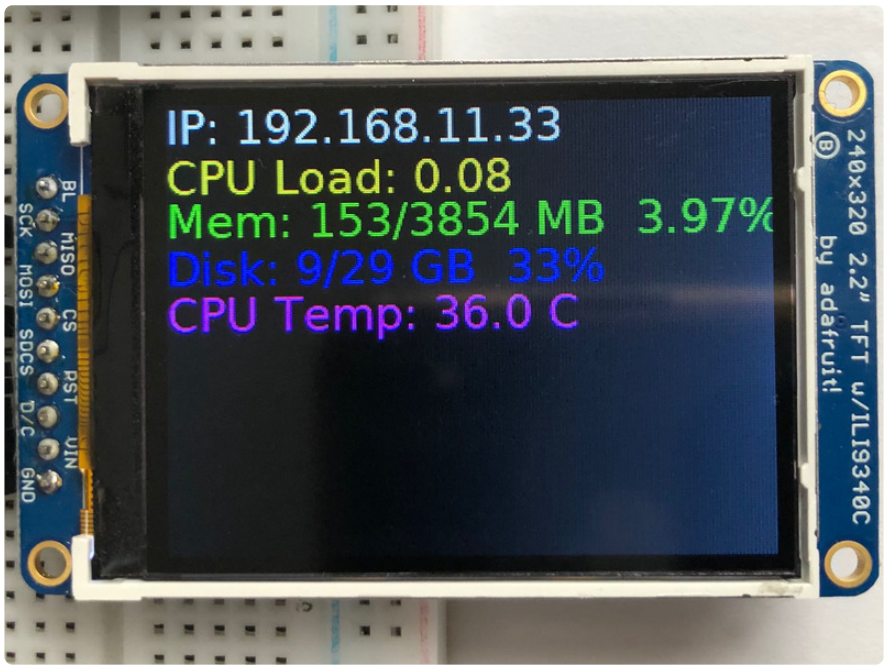
```
cmd = "hostname -I | cut -d\' \' -f1"
IP = "IP: "+subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\", $3,$2,$3*100/$2 }'"
MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "df -h | awk '$NF==\"/\"{printf \"Disk: %d/%d GB  %s\", $3,$2,$5}'"
Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")
cmd = "cat /sys/class/thermal/thermal_zone0/temp |  awk \'{printf \"CPU Temp: %.1f
C\", $(NF-0) / 1000}\'" # pylint: disable=line-too-long
Temp = subprocess.check_output(cmd, shell=True).decode("utf-8")
```

Now we display the information for the user. Here we use yet another way to pass color information. We can pass it as a color string using the pound symbol, just like we would with HTML. With each line, we take the height of the line using `getsize()` and move the pointer down by that much.

```
y = padding
draw.text((x, y), IP, font=font, fill="#FFFFFF")
y += font.getsize(IP)[1]
draw.text((x, y), CPU, font=font, fill="#FFFF00")
y += font.getsize(CPU)[1]
draw.text((x, y), MemUsage, font=font, fill="#00FF00")
y += font.getsize(MemUsage)[1]
draw.text((x, y), Disk, font=font, fill="#0000FF")
y += font.getsize(Disk)[1]
draw.text((x, y), Temp, font=font, fill="#FF00FF")
```

Finally, we write all the information out to the display using `disp.image()`. Since we are looping, we tell Python to sleep for `0.1` seconds so that the CPU never gets too busy.

```
disp.image(image)
time.sleep(.1)
```



Troubleshooting

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

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

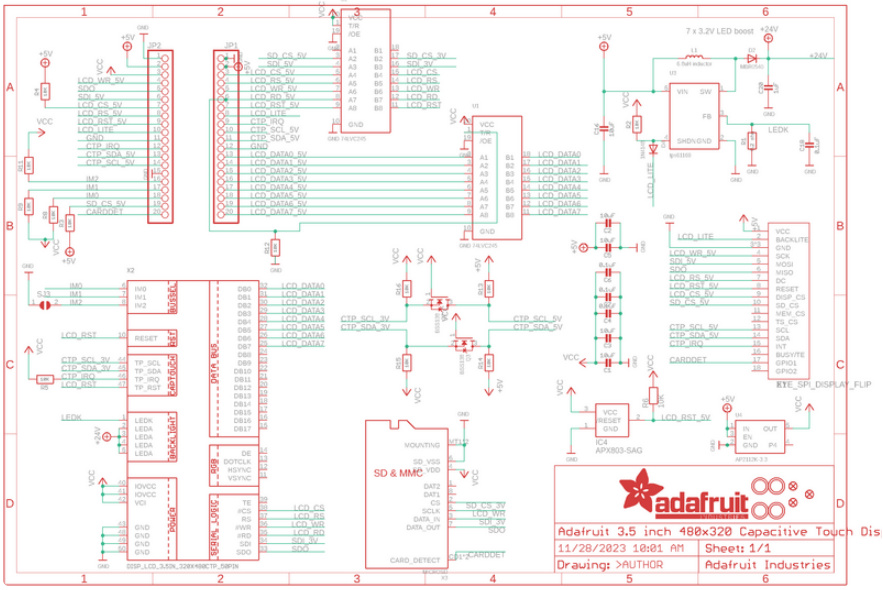
In Arduino, use **delay()** to add a few milliseconds before calling **tft.begin()**. Adjust the amount of delay as needed to see how little you can get away with for your specific setup.

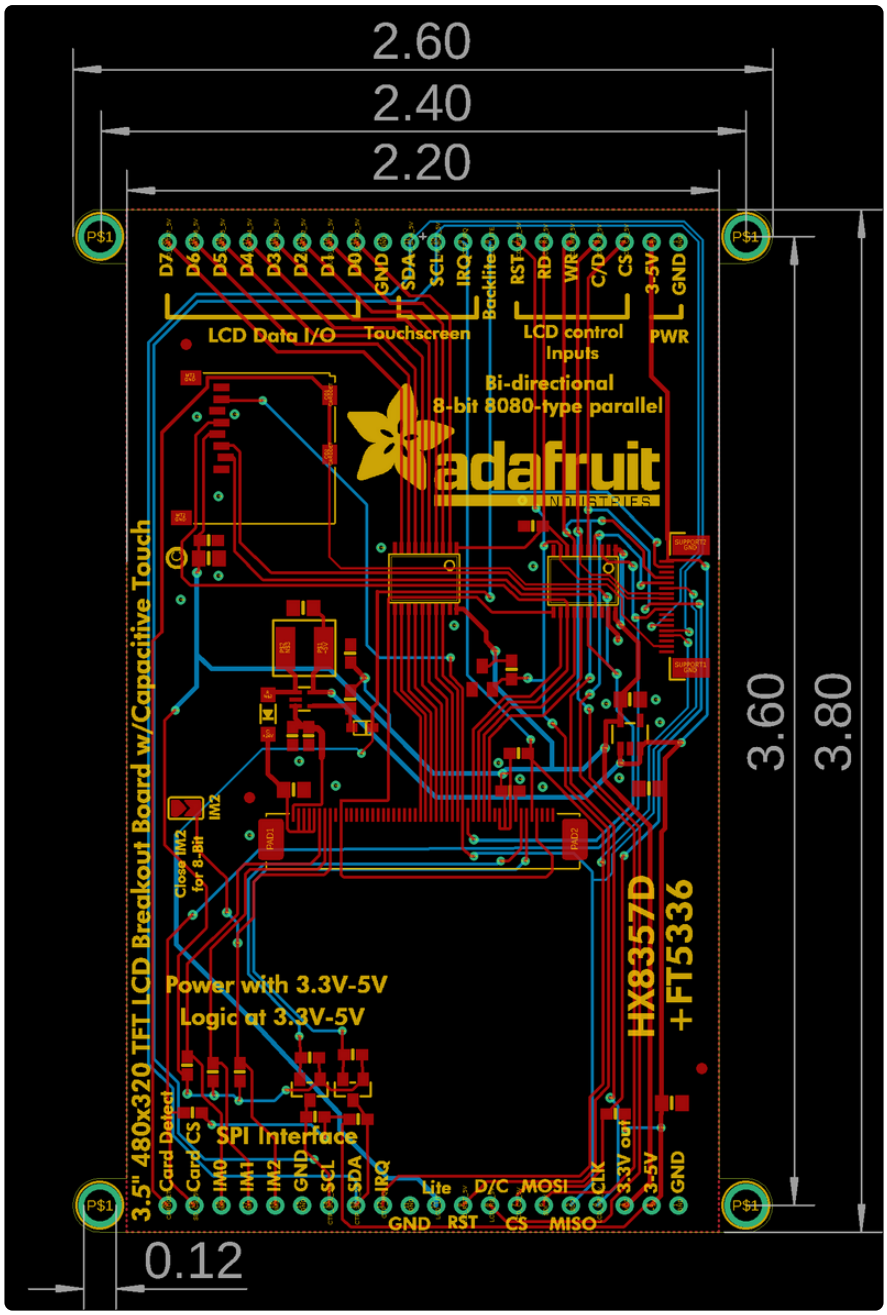
Downloads

Datasheets and Files

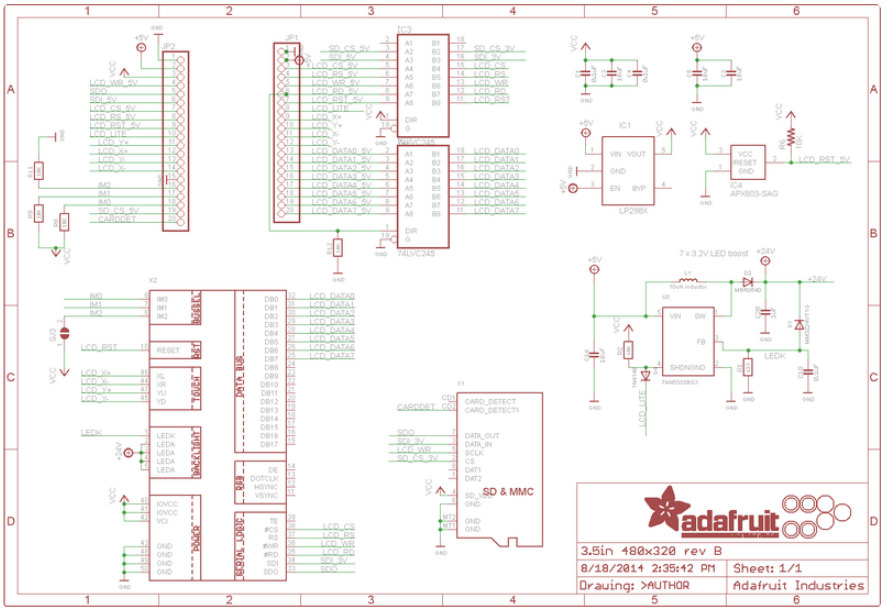
- Datasheet for the HX8357D chipset controller (https://adafru.it/dQQ)
- Datasheet for the 3.5" TFT display (raw) (https://adafru.it/dR4)
- FT5336 Capacitive Touch Controller Datasheet (https://adafru.it/19bA)
- EagleCAD PCB files on GitHub (https://adafru.it/pBE)
- Resistive Touch Fritzing object in Adafruit Fritzing library (https://adafru.it/19bB)
- Capacitive Touch Fritzing object in Adafruit Fritzing library (https://adafru.it/19bC)

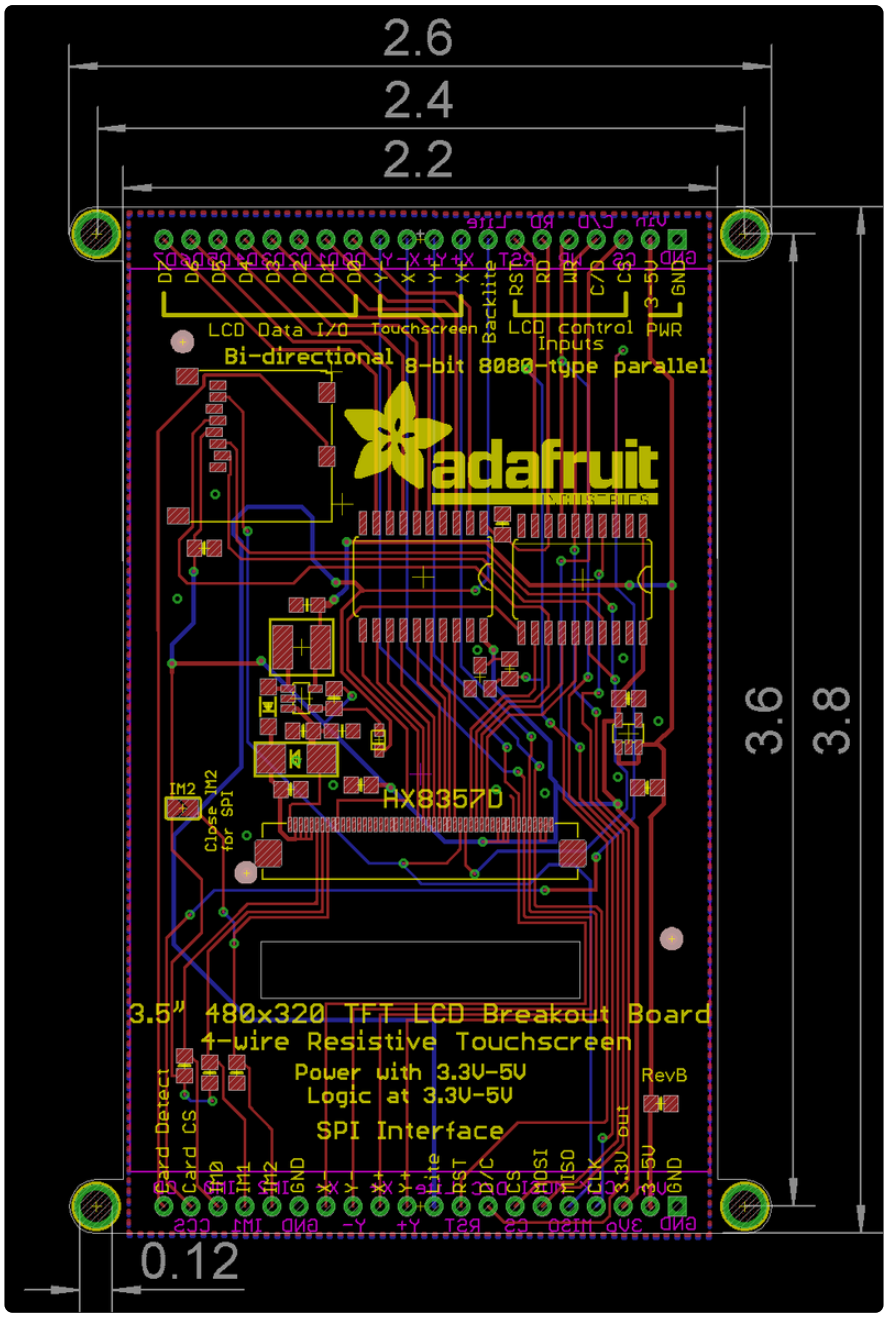
Schematic and Fab Print - Capacitive Touch




2.60
2.40
2.20
3.60
3.80
0.12
3.5" 480x320 TFT LCD Breakout Board w/Capacitive Touch
LCD Data I/O
Touchscreen
LCD control Inputs
PWR
Bi-directional 8-bit 8080-type parallel
adafruit
INDUSTRIES
HX8357D
+FT5336
Power with 3.3V-5V Logic at 3.3V-5V
SPI Interface
Close IM2 for 8-Bit
IM2
Card Detect
Card CS
IM0
IM1
IM2
GND
SCL
SDA
IRQ
Lite
D/C
MOSI
CLK
3.3V out
3-5V
GND
RST
CS
MISO
D7
D6
D5
D4
D3
D2
D1
D0
GND
SDA
SCL
IRQ
Backlite
RST
RD
WR
C/D
CS
3-5V
GND
P$1

Schematic and Fab Print - Resistive Touch




2.6
2.4
2.2
3.6
3.8
0.12
LCD Data I/O
Touchscreen
Backlite
LCD control
Inputs
PWR
Bi-directional 8-bit 8080-type parallel
adafruit
HX8357D
3.5" 480x320 TFT LCD Breakout Board
4-wire Resistive Touchscreen
Power with 3.3V-5V
Logic at 3.3V-5V
SPI Interface
RevB
Card Detect
Card CS
IM2
IM1
IM0
GND
RST
D/C
CS
MOSI
MISO
CLK
3.3V_out
3-5V
Close for SPI