2.2" TFT Display
Created by lady ada
Overview

This lovely little display breakout is the best way to add a small, colorful and bright display to any project. Since the display uses 4-wire SPI to communicate and has its own pixel-addressable frame buffer, it can be used with every kind of microcontroller. Even a very small one with low memory and few pins available!

NOTE: This tutorial no longer covers the 176x220 pixel version of the display - only the newer 320x240 pixel version. Chances are you DO NOT have the older version!
The 2.2" display has 320x240 color pixels. Unlike the low cost "Nokia 6110" and similar LCD displays, which are CSTN type and thus have poor color and slow refresh, this display is a true TFT! The TFT driver (ILI9340) can display full 16-bit color. And the LCD will always come with the same driver chip so there's no worries that your code will not work from one to the other.

The breakout has the TFT display soldered on (it uses a delicate flex-circuit connector) as well as a ultra-low-dropout 3.3V regulator and a 3/5V level shifter so you can use it with 3.3V or 5V power and logic. We also had a little space so we placed a microSD card holder so you can easily load full color bitmaps from a FAT16/FAT32 formatted microSD card. The microSD card is not included but you can pick one up here (http://adafruit.it/102).
This color display uses SPI to receive image data. That means you need at least 4 pins - clock, data in, TFT CS and D/C. If you'd like to have SD card usage too, add another 2 pins - data out and card cs. However, there's a couple other pins you may want to use, let's go thru them all!

- **GND** - this is the power and signal ground pin
- **Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **D/C** - this is the TFT SPI data or command selector pin. Use 3-5V logic level
- **RESET** - this is the TFT reset pin. Connect to ground to reset the TFT! 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 Arduino Reset pin, which works for most cases. There is an automatic-reset chip connected so it will reset on power-up. Use 3-5V logic level
- **SD Card CS / SDCS** - this is the SD card chip select, used if you want to read from the SD card. Use 3-5V logic level
- **LCD_CS** - this is the TFT SPI chip select pin. Use 3-5V logic level
- **MOSI** - this is the SPI Master Out Slave In pin, it is used to send data from the microcontroller to the SD card and/or TFT. Use 3-5V logic level
- **MISO** - this is the SPI Master In Slave Out pin, it's used for the SD card. It isn't used for the TFT display which is write-only. It is 3.3V logic out (but can be read by 5V logic)
- **CLK** - this is the SPI clock input pin. Use 3-5V logic level
- **Backlit** - 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. Use 3-5V logic level
Assembly

Start by connecting a piece of header to the display. This will make breadboarding much easier. Break off a piece of 0.1" header 9 pins long and place it into a breadboard, long pins facing down into the breadboard.

Place the display on top

Solder all the pins
Arduino Wiring

There are two ways to wire up these displays:

**Software SPI is a more flexible method** (you can use any pins on the Arduino) and **hardware SPI is much faster** (4-8x faster) but you are required to use the hardware SPI pins.

Since the display is quite large, we found that drawing would seem really slow if using 'software' SPI. For that reason, we'll show primarily how to wire up using hardware SPI and then how you can change the pins if desired.

Hardware SPI means that we have to connect the **CLK** and **MOSI** pins to fixed digital pins.

On '328 and '168 Arduinos, **CLK** must connect to digital 13 and **MOSI** must connect to digital 11. If using an Arduino Mega, connect **CLK** to 52 and **MOSI** to 51. If you're using another kind of Arduino you'll need to use the SPI hardware port (https://adafru.it/iCE)

Digital 10 (53 on Arduino Mega) must also be an output (but doesn't need to be connected to any particular pin).

**Arduino UNO or Compatible Wiring**

We'll use the following pin connections:

- **GND** connects to ground - black wire
- **VIN** connects to +5V - red wire
- **DC (data/clock)** connects to digital 9 on Atmega328
- Skip **SDCS** (SD card chip select - used for SD card interfacing)
- **CS** (chip select) connects to digital 10 on Atmega328
- **MOSI** (data out) connects to digital 11 on Atmega328
- **SCK** (clock) connects to digital 13 on Atmega328
- Skip **MISO** (data in - used for SD card interfacing)

https://adafru.it/All
Wiring for Other Boards

We'll use the following pin connections:

- **GND** connects to ground - black wire
- **VIN** connects to +5V - red wire
- **DC** (data/clock) connects to digital 9
- Skip **SDCS** (SD card chip select - used for SD card interfacing)
- **CS** (chip select) connects to digital 10
- **MOSI** (data out) connects to **MOSI**
- **SCK** (clock) connects to **SCK**
- Skip **MISO** (data in - used for SD card interfacing)

You can later change the CS and RST pins but to match the tutorial, use this connection diagram.
Arduino Code

Once you have the display wired up, it's time to test your wiring by uploading the example code we have written. Again, we suggest using an Arduino to test.

Install Library

Go to the Arduino Library manager under Sketch -> Include Library -> Manage Libraries...

From within the Library manager, start by installing Adafruit GFX:

Then look for and install the Adafruit ILI9341 library

Note that this display has an ILI9340 but we still use the ILI9341 library, it's OK! The chips are nearly identical

One more! Look for and install Adafruit_ZeroDMA. That's the third and final library in this sequence.

You can read more about installing libraries in our tutorial (https://adafruit.it/aYG).
Run Graphics Test

Restart the Arduino IDE. You should now be able to select File > Examples > Adafruit_ILI9341 > graphicstest sketch. Upload the sketch to your Arduino wired as before.

Once uploaded, the Arduino should perform all the test display procedures! If you’re not seeing anything - first check if you have the backlight on, if the backlight is not lit something is wrong with the power/backlight wiring. If the backlight is lit but you see nothing on the display make sure you’re using our suggested wiring.
Adafruit GFX Library

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

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

Read more about it here! (https://adafruit.it/aPx)

Bitmaps

In this example, we'll show how to display a 220x176 pixel full color bitmap from a microSD card.
We have an example sketch in the library showing how to display full color bitmap images stored on an SD card. You'll need a microSD card such as this one (http://adafruit.it/102). You'll also need to be running Arduino 1.0 or later, as the SD library was updated.

You'll also need an image. We suggest starting with this bitmap of a rose (https://adafruit.it/cmm). If you want to later use your own image, use an image editing tool and crop your image to no larger than 160 pixels high and 128 pixels wide. Save it as a 24-bit color BMP file - it must be 24-bit color format to work, even if it was originally a 16-bit color image - because of the way BMPs are stored and displayed!

Names for bitmap files must not exceed 8 characters with a 3 character extension. "mybitmap.bmp" is fine. "myotherbitmap.bmp" is too long and will not be readable by the SD file system.

Copy the rose.bmp to the microSD card and insert it into the back of the breakout board.
Wire up the TFT according to the high-speed SPI diagram above. Test that your wiring is correct by uploading the graphics test sketch with the high speed SPI line uncommented and the flexible-low-speed wiring commented.

Once you are sure that the TFT is wired correctly, add the two wires for talking to the SD card. Connect **CDCS** (the unconnected pin in the middle) to digital pin 4 (you can change this later to any pin you want) that's the orange wire below. Connect **MISO** (last unconnected pin) to the Arduino's hardware SPI **MISO** pin, that's the white wire below. For Classic arduinos, this is pin 12. For Mega's this is pin 50. You can't change the **MISO** pin, its fixed in the chip hardware.
Now load the **bitmap** example sketch into the Arduino. It should display the parrot image. If you have any problems, check the serial console for any messages such as not being able to initialize the microSD card or not finding the image.
Bitmaps

There is a built in microSD card slot into the breakout, 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).

It's really easy to draw bitmaps. We have a library for it, Adafruit_ImageData, which can be installed through the Arduino Library Manager (Sketch→Include Library→Manage Libraries…). Enter “imageread” in the search field and the library is easy to spot:

Let's start by downloading this image of pretty flowers (pix by johngineer)
Copy `purple.bmp` into the base directory of a microSD card and insert it into the microSD socket in the breakout.

You'll need to connect up the SDCS pin to Digital 4 on your Arduino, and the MISO to MISO (or Digital #12 on an Uno) as well. In the below image, those are the extra purple & light blue wires.

You may want to try the SD library examples before continuing, especially one that lists all the files on the SD card.

Now upload the File→examples→Adafruit ImageReader Library→ShieldILI9341 example to your Arduino + breakout. You will see the flowers appear!
To make new bitmaps, make sure they are less than 240 by 320 pixels and save them in **24-bit BMP format**! They must be in 24-bit format, even if they are not 24-bit color as that is the easiest format for the Arduino. You can rotate images using the `setRotation()` procedure.

You can draw as many images as you want - don't forget the names must be less than 8 characters long. Just copy the BMP drawing routines below `loop()` and call

```c
bmpDraw(bmpfilename, x, y);
```

For each bitmap. They can be smaller than 320x240 and placed in any location on the screen.
CircuitPython Displayio Quickstart

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

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/drI). After that, the breakout should be ready to go.

Wiring the Breakout to the Feather

- **3.3V Vin** connects to the Feather 3V pin
- **GND** connects to Feather ground
- **CLK** connects to SPI clock. On the Feather that's SCK.
- **MISO** connects to SPI MISO. On the Feather that's MI
- **MOSI** connects to SPI MOSI. On the Feather that's MO
- **CS** connects to our SPI Chip Select pin. We'll be using Digital 9 but you can later change this to any pin
- **D/C** connects to our SPI data/command select pin. We'll be using Digital 10 but you can later change this pin too.
- **RST** connects to our reset pin. We'll be using Digital 6 but you can later change this pin too.
Required CircuitPython Libraries

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

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

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

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

- `adafruit_ili9341`

Before continuing make sure your board's lib folder or root filesystem has the `adafruit_ili9341` file copied over.

Code Example Additional Libraries

For the Code Example, you will need an additional library. We decided to make use of a library so the code didn't get overly complicated.

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

### CircuitPython Code Example

Temporarily unable to load content:

### Code Details

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

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

displayio.release_displays()
```

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.

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

display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.D6)
```

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.

```python
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.

```python
display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs, reset=board.D6)
```

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

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

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

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

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

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

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

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

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

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

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

```python
import adafruit_displayio_ssd1306
import displayio
import terminalio

# Create the display and basic group
splash = displayio.Group(max_size=10)

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

while True:
    pass
```

Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

Where to go from here

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

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

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

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

Files:

- Adafruit Fritzing Library (https://adafruit.it/aP3)
- ILI9340 (datasheet) (https://adafruit.it/CbV) controller with built in pixel-addressable video RAM buffer
- Display datasheet (https://adafruit.it/CbW)
- EagleCAD files on GitHub (https://adafruit.it/CbX)