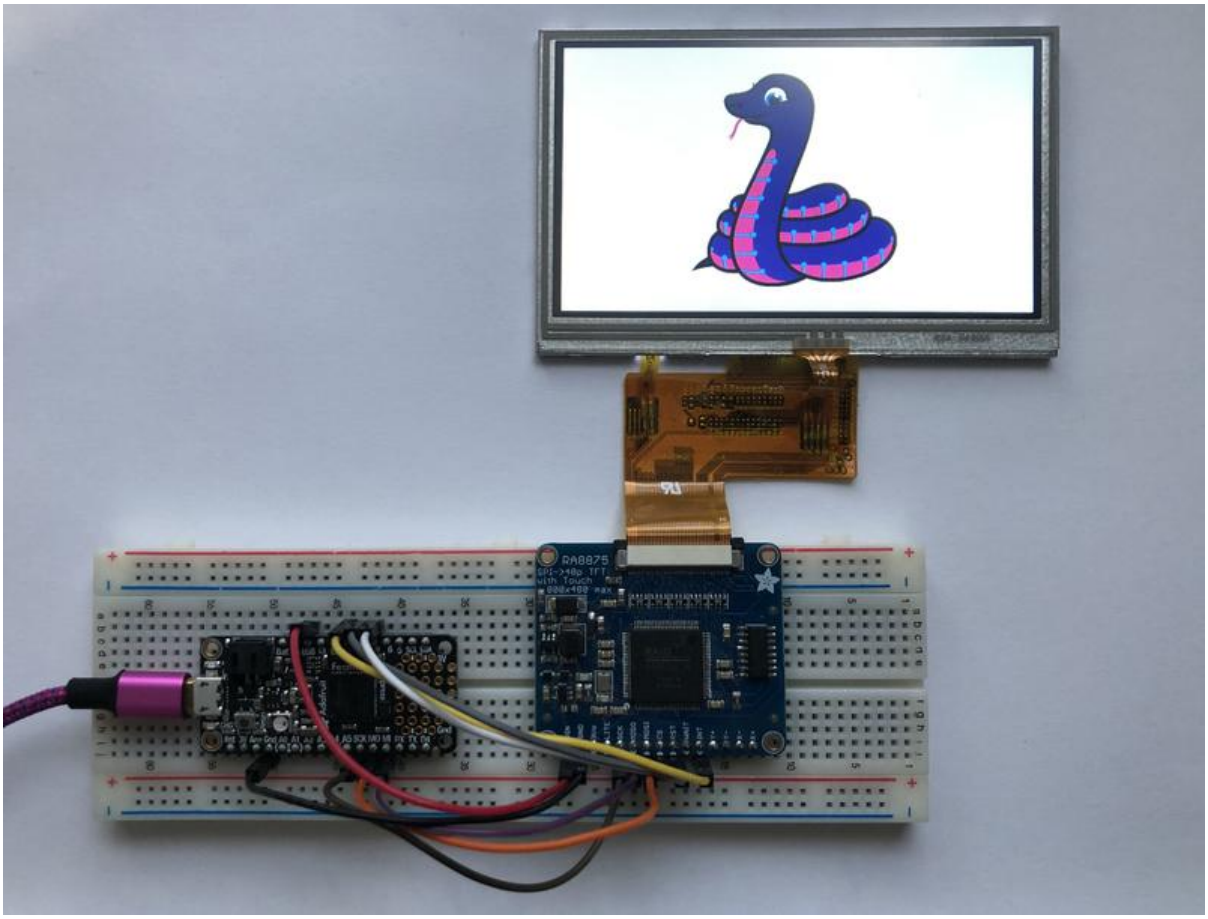
RA8875 Touch Display Driver Board

Created by Melissa LeBlanc-Williams

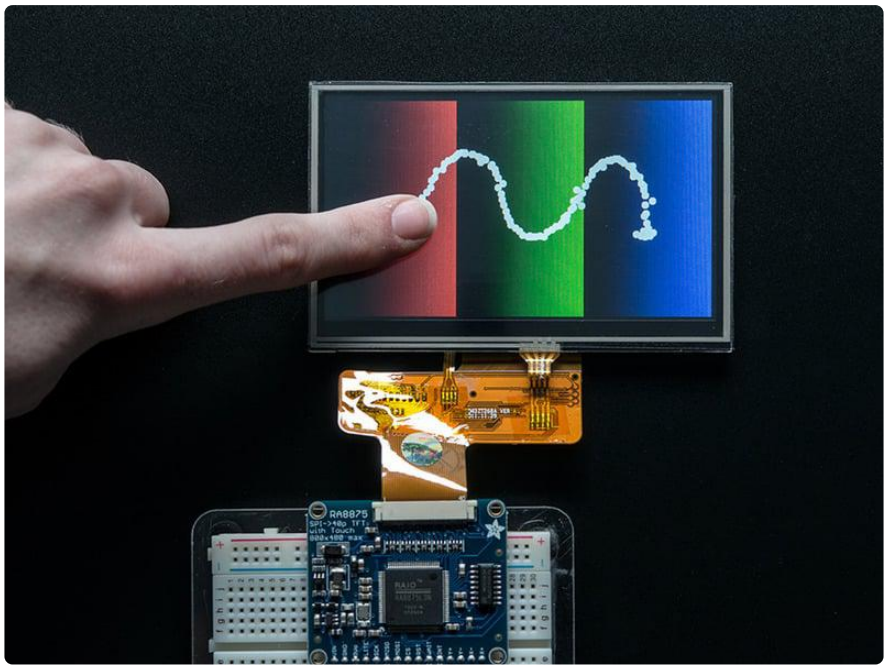


https://learn.adafruit.com/ra8875-touch-display-driver-board

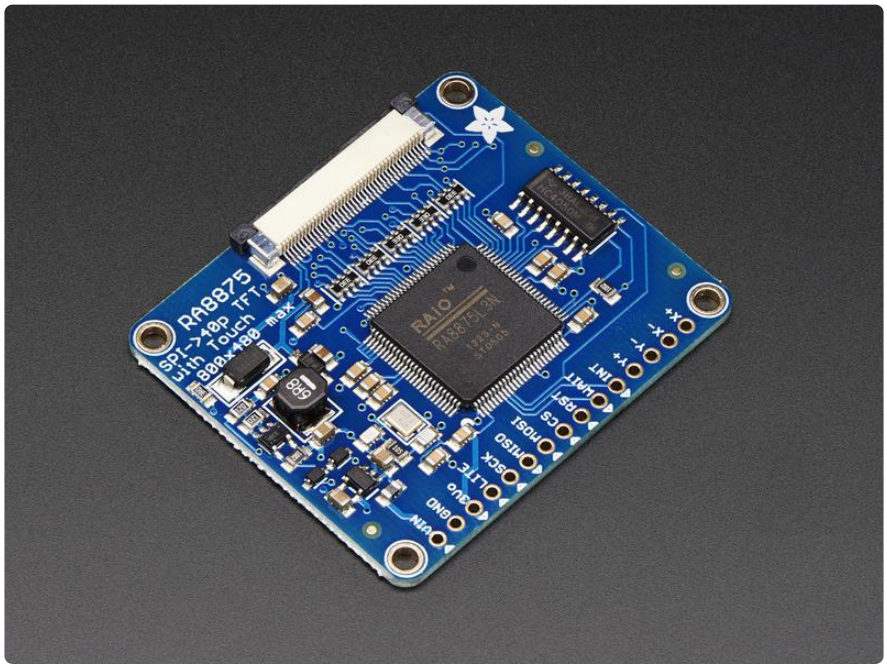
Last updated on 2022-12-01 03:29:40 PM EST

Table of Contents

Overview

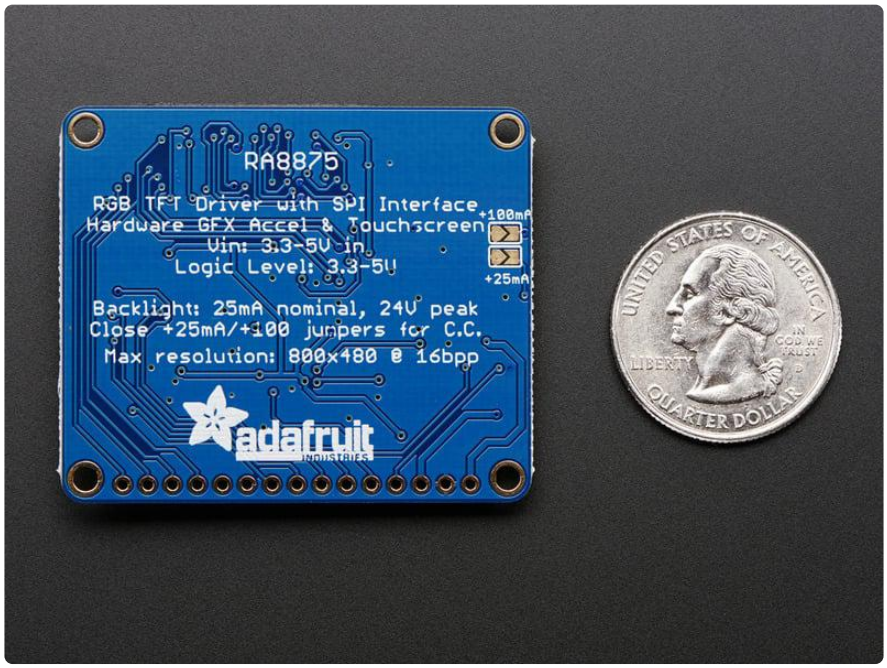


Have you gazed longingly at large TFT displays - you know what I'm talking about here, 4", 5" or 7" TFTs with up to 800x480 pixels. Then you look at your micro-controller project, but there's no way it can control a display like that, one that requires 60Hz refresh and 4 MHz pixel clocking. Heck, it doesn't even have enough pins. I suppose you could move to ARM core processors with TTL display drivers built in but you've already got all these shields working and anyways you like small micros you've got.

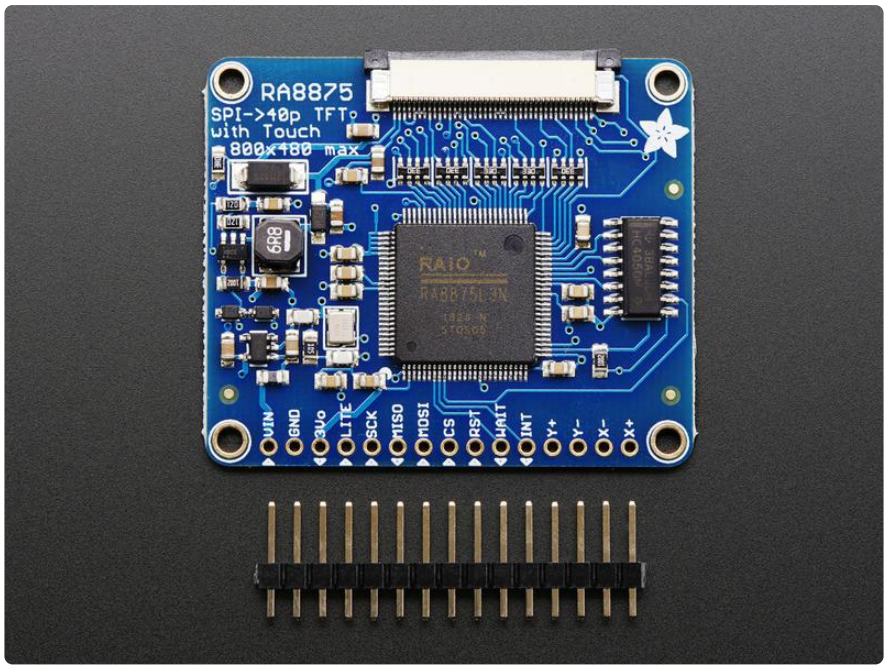


What if I told you there was a driver chip that could fulfill those longings? A chip that can control up 800x480 displays, and heck, a resistive touchscreen as well. All you need to give up is 5 or so SPI pins. Would you even believe me? Well, sit down because this product may shock you.

The RA8875 is a powerful TFT driver chip. It is a perfect match for any chip that wants to draw on a big TFT screen but doesn't quite have the oomph (whether it be hardware or speed). Inside is 768KB of RAM, so it can buffer the display (and depending on the screen size also have double overlaying). The interface is SPI with a very basic register read/write method of communication (no strange and convoluted packets). The chip has a range of hardware-accelerated shapes such as lines, rectangles, triangles, ellipses, built in and round-rects. There is also a built in English/ European font set (see the datasheet section 7-4-1 for the font table) This makes it possible to draw fast even over SPI.

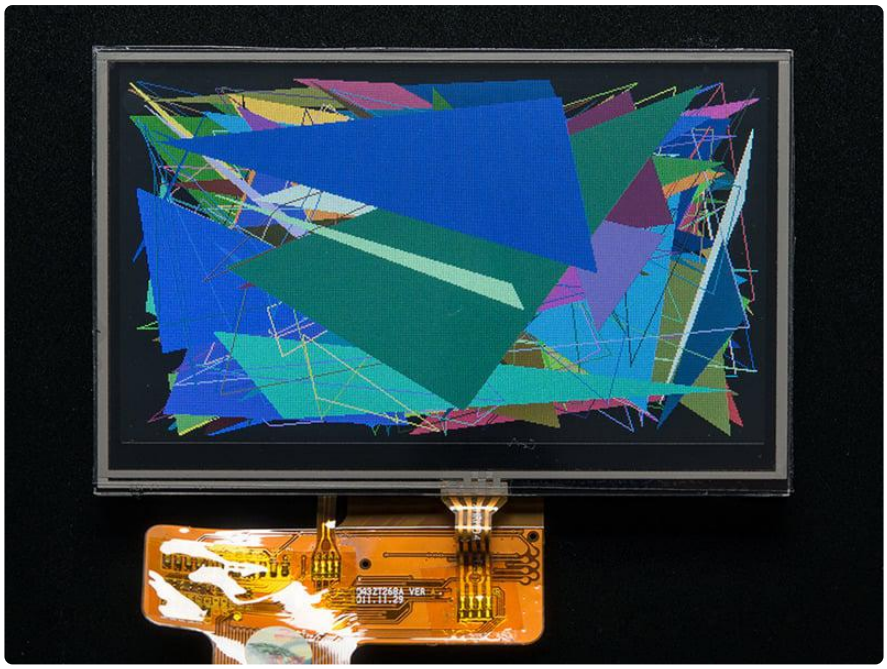


The RA8875 can also handle standard 4-wire resistive touchscreens over the same SPI interface to save you pins. There's an IRQ pin that you can use to help manage touch interrupts. The touchscreen handler isn't the most precise driver we've used, so we broke out the X/Y pins so you can connect them up to something like the STMPE610 which is a very classy touchscreen controller. (http://adafru.it/1571)



On the PCB we have the main chip, level shifting so you can use safely with 3-5V logic. There is also a 3V regulator to provide clean power to the chip and the display. For the backlight, we put a constant-current booster that can provide 25mA or 50mA at up to 24V. The connector to the screen is a classic '40 pin' connector. All the 40-pin TFT's in the Adafruit shop are known to work well. There are other 40-pin displays that have different pinouts or backlight management and these may not work - they may even damage the driver or TFT if the boost converter pushes 24V into the display logic pins! For that reason, we only recommend the displays we've tested and sell here.

Each order comes with an assembled, tested RA8875 breakout and a stick of header. You'll also need to purchase a 40-pin TFT screen. We currently have 4.3", 5.0" and 7.0" screens available.

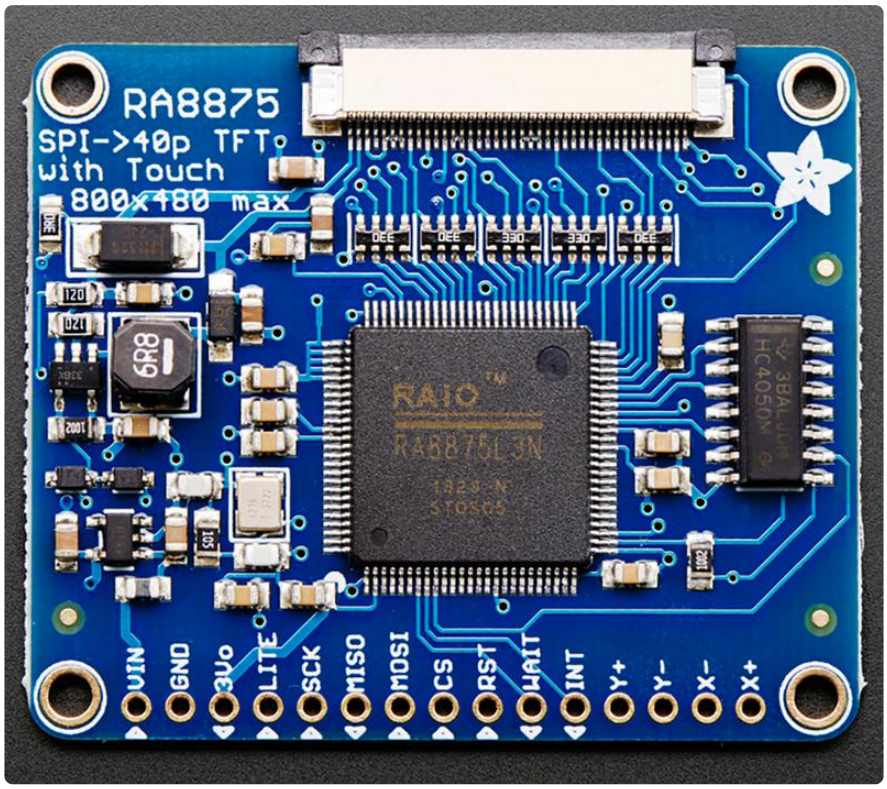


To get you started we've written a graphics library that handles the basic interfacing, drawing and reading functions. Download the Adafruit RA8875 library from github () a nd install as described in our tutorial. () Connect a 40 pin TFT to the FPC port and wire up the SPI interface to an Arduino as described in the example code. Once started you'll be able to see the graphic/text demo and then touch the screen to 'paint'. For more advanced details on what the RA8875 can do (and it can do a lot) check the datasheet.

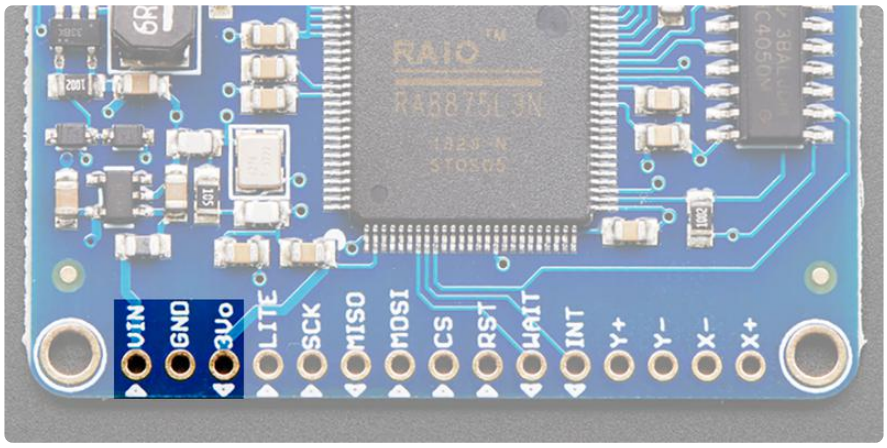
Please note! The RA8875 does not tri-state the MISO pin, it should not share that pin with any other SPI device (including an SD card reader) without the use of a 74HC125 or similar ()

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

Pinouts



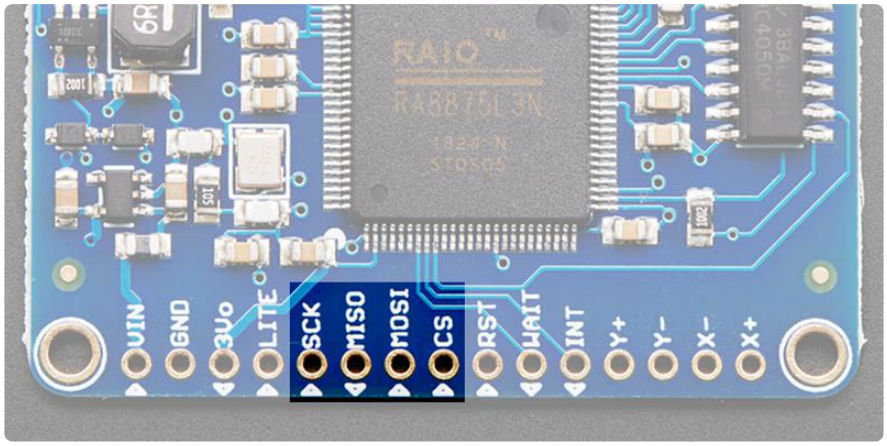
Power Pins



These are the pins that are involved with powering the RA8875:

- VIN - power input, connect to 3-5VDC.
- GND - power and signal ground. Connect to your power supply and microcontroller ground.
- 3Vo is the output from the onboard 3.3V regulator. If you have a need for a clean 3.3V output, you can use this! It can provide at least 100mA output.

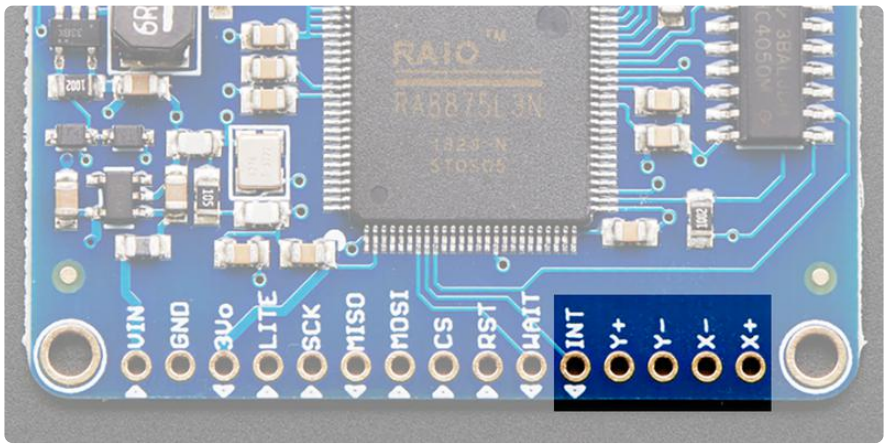
SPI Logic pins



All pins are 3-5 V compliant, so you can use with 3V or 5V micocontrollers

- SCK - this is the SPI clock pin, its an input to the chip
- MISO - this is the Microcontroller In Serial Out pin, for data sent from the RA8875 to your processor - this is 3.3V logic output, and can be read by 5V microcontrollers just fine. This pin does not tri-state when the CS pin is pulled low, so it cannot share an SPI bus without a tri-state chip like the 74HC125 or similar
- MOSI - this is the Microcontroller Out Serial In pin, for data sent from your processor to the RA8875
- CS - this is the chip select pin, drop it low to start an SPI transaction. Its an input to the chip. This pin is not pulled high by default! Please either set it high or add a pullup resistor to keep the chip disabled!

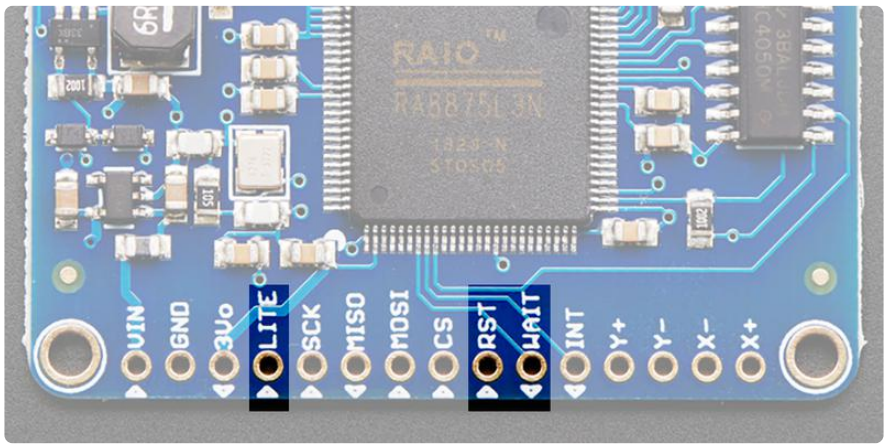
Touch Pins



These are the pins that are involved with the touch panel for the RA8875. The X an Y pairs of pins are ideal for reading touches using an external touch controller such as the STMPE610:

- INT - interrupt pin that goes high when the panel is being touched
- Y+ - touch panel Y positive signal
- Y- - touch panel Y negative signal
- X+ - touch panel X positive signal
- X- - touch panel X negative signal

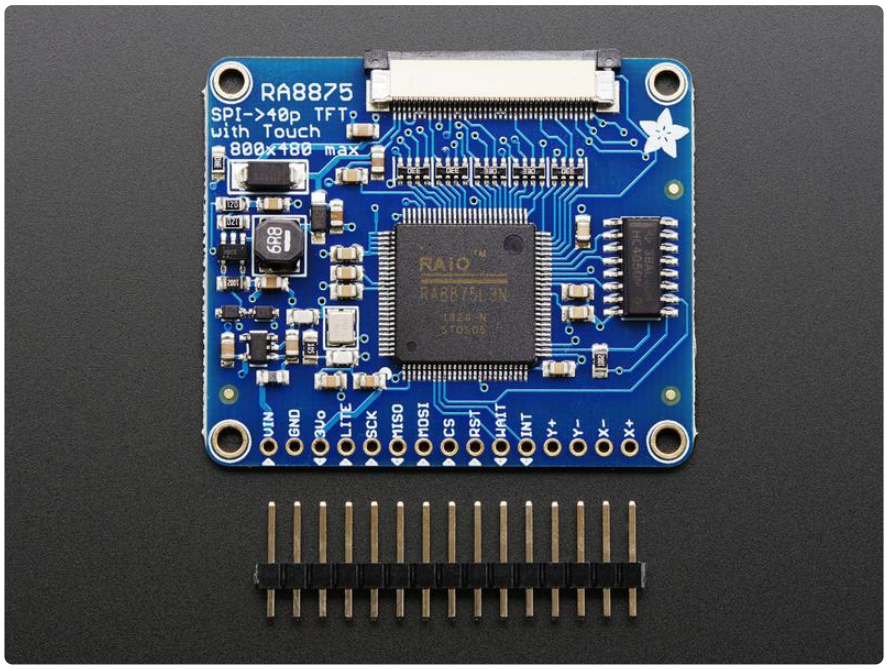
Other Pins

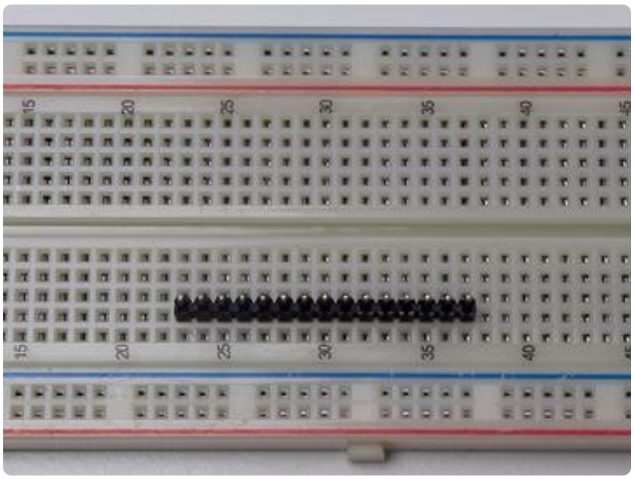


These are the remaining pins for the RA8875:

- LITE - PWM Signal used to externally control the Backlight
- RST - Reset line for the RA8875. It is active low meaning you would reset the RA8875 by pulling it this pin to ground.
- WAIT - output to indicate that the RA8875 is in a busy state. The RA8875 can't communicate with the microcontroller when the wait pin is active. It is active low and could be used by the microcontroller to poll busy status.

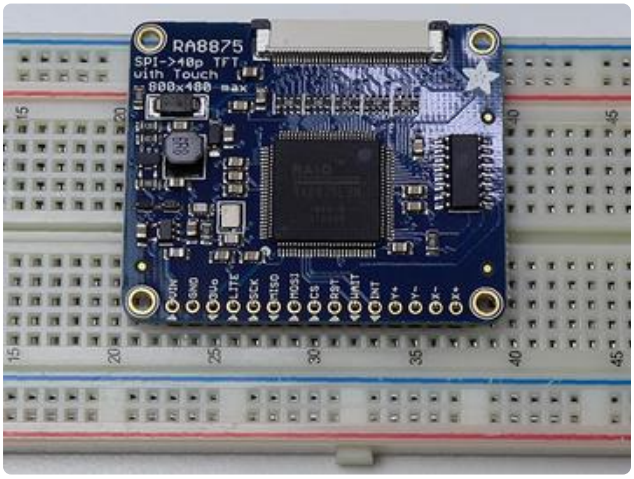
Assembly





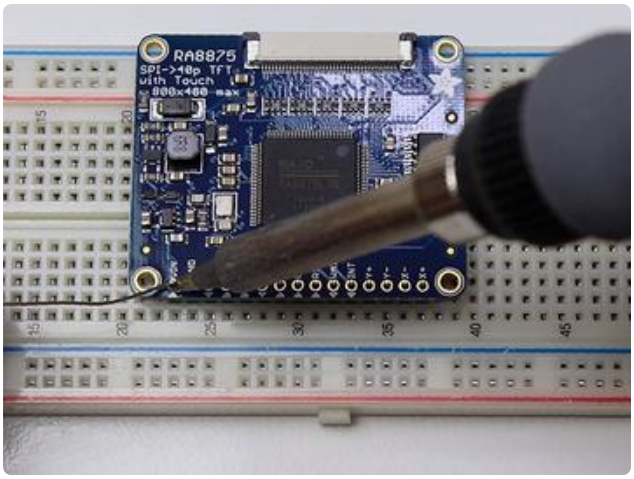
Prepare the header strip:

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



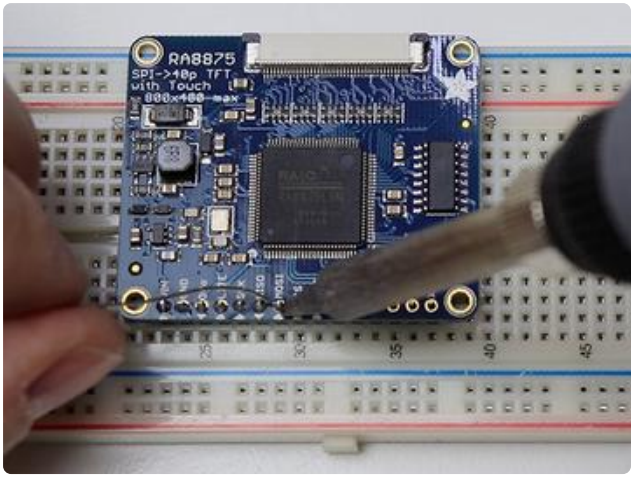
Add the breakout board:

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

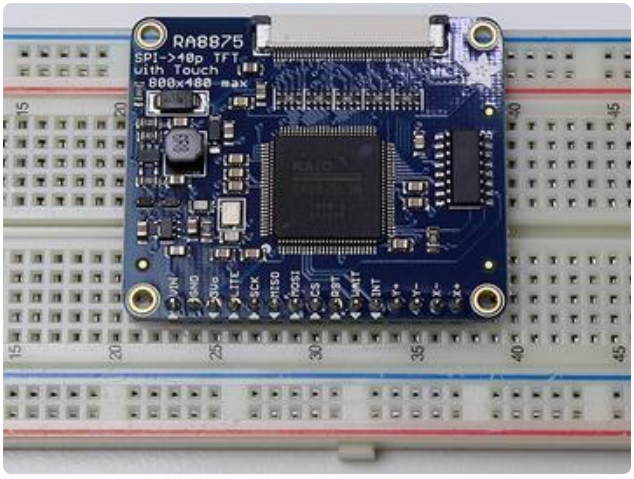


And Solder!

Be sure to solder all pins for reliable electrical contact.



(For tips on soldering, be sure to check out our Guide to Excellent Soldering ()).



Check your solder joints visually and continue onto the next steps

Connect the LCD

Very carefully slide the ears out, insert the cable with the gold contacts facing up, then slide the ears back in.

You're done!

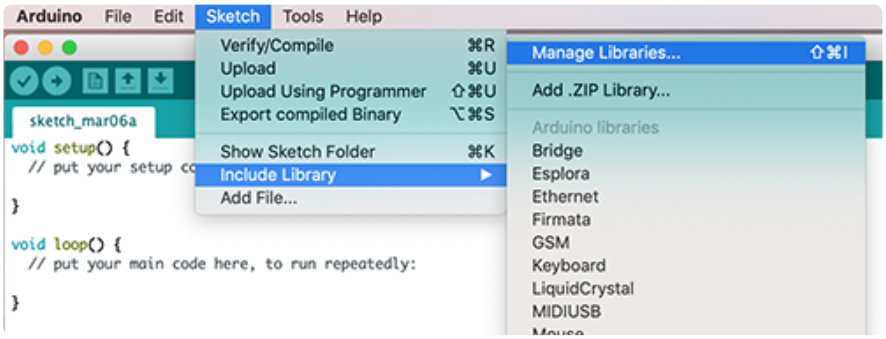
Continue on to the next steps.

Arduino Code

To get this display working on the Arduino, first we will need to install a couple of Arduino libraries.

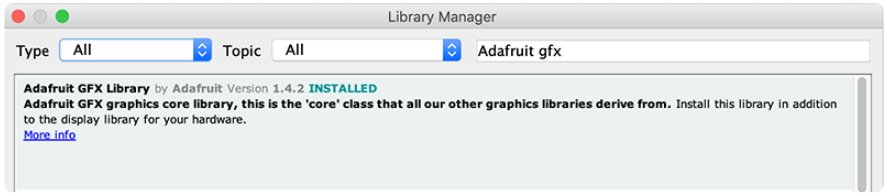
Install Arduino Libraries

Lets begin by installing all the libraries we need. Open up the library manager in Arduino IDE



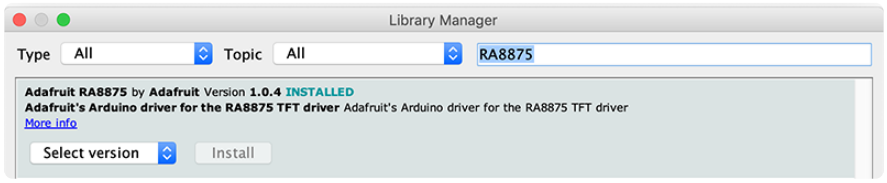
Search for and install the Following libraries:

Adafruit GFX



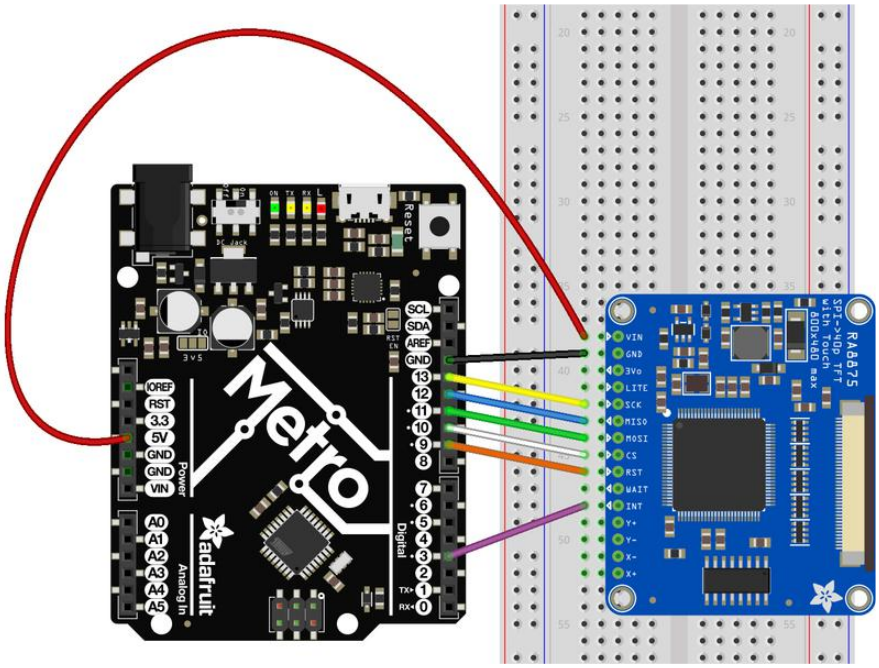
If using an earlier version of the Arduino IDE (pre-1.8.10), locate and install Adafruit_BusIO (newer versions handle this prerequisite automatically).

Adafruit RA8875



Once you have installed these, restart the Arduino IDE

Arduino Wiring



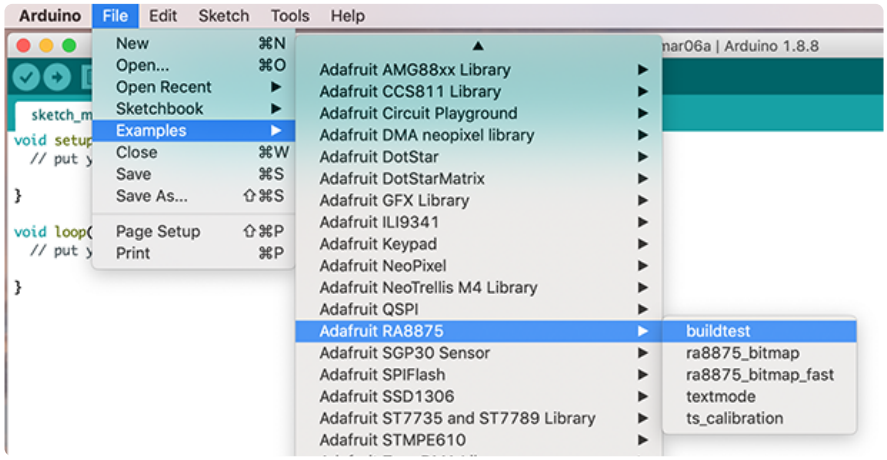
Start by connecting the power pins

- 3-5V Vin connects to the Arduino 5V pin
- GND connects to Arduino ground
- SCK connects to SPI clock. On Arduino Uno/Duemilanove/328-based, thats Digital 13. On Mega's, its Digital 52 and on Leonardo/Due/Metro M0/M4 it's ICSP-3 (See SPI Connections for more details ())
- MISO connects to SPI MISO. On Arduino Uno/Duemilanove/328-based, thats Digital 12. On Mega's, its Digital 50 and on Leonardo/Due/Metro M0/M4 it's ICSP-1 (See SPI Connections for more details ())
- MOSI connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, thats Digital 11. On Mega's, its Digital 51 and on Leonardo/Due/Metro M0/M4 it's ICSP-4 (See SPI Connections for more details ())
- CS connects to our SPI Chip Select pin. We'll be using Digital 10 but you can later change this to any pin
- RST connects to our Reset pin. We'll be using Digital 9 but you can later change this pin as well.
- INT connects to our Interrupt pin. We'll be using Digital 3 but you can later change this pin too.

That's it! You don't need to connect any of the other pins.

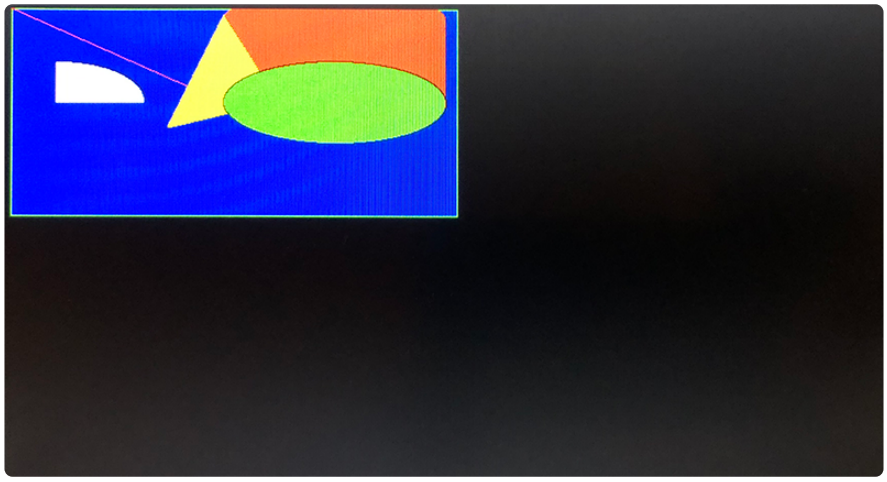
Build Test

Start by opening up the file -> examples -> Adafruit RA8875 -> buildtest:



Be sure to set the screen size in the sketch to the appropriate size and upload it to your Arduino

You'll see a graphics test program run, showing drawing lines, text, rectangles, ellipses, triangles, etc.

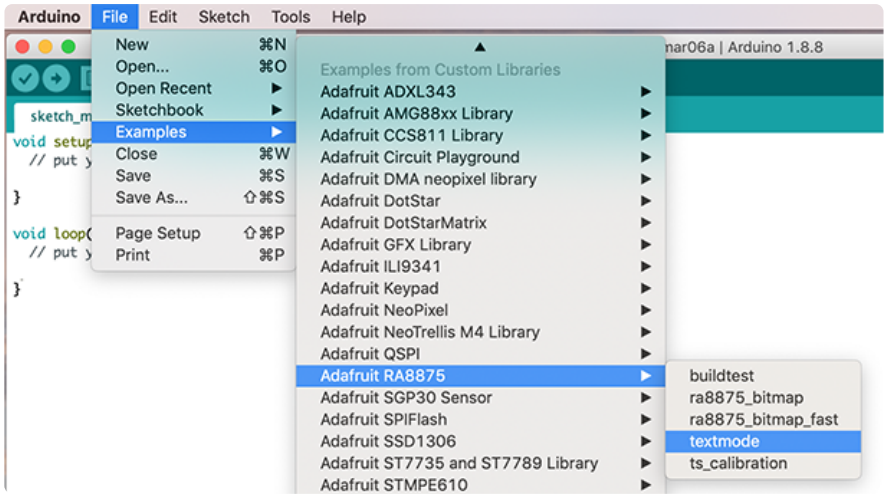


Feel free to touch the screen if your LCD Display is a touchscreen. The screen will start drawing dots where your finger was.

Text Mode

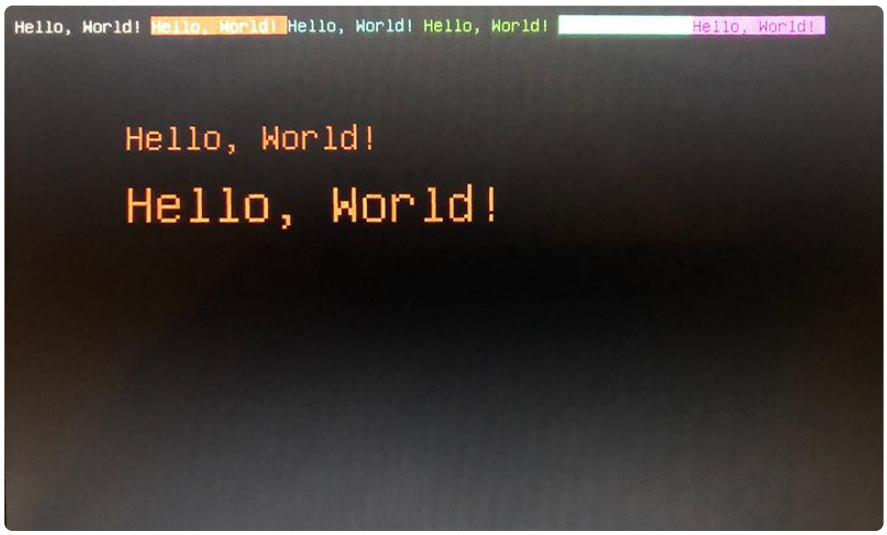
Unlike many displays, the RA8875 requires switching modes between drawing text and graphics. They can still be drawn on the screen at the same time, but you are required to manually set the appropriate mode for the type of function you wish to call.

Start by opening up the file -> examples -> Adafruit RA8875 -> textmode



Just like the previous example, be sure to set the screen size in the sketch to the appropriate size and upload it to your Arduino

You'll see a text mode test print out "Hello, World!" in a variety of colors and sizes.



CircuitPython Code

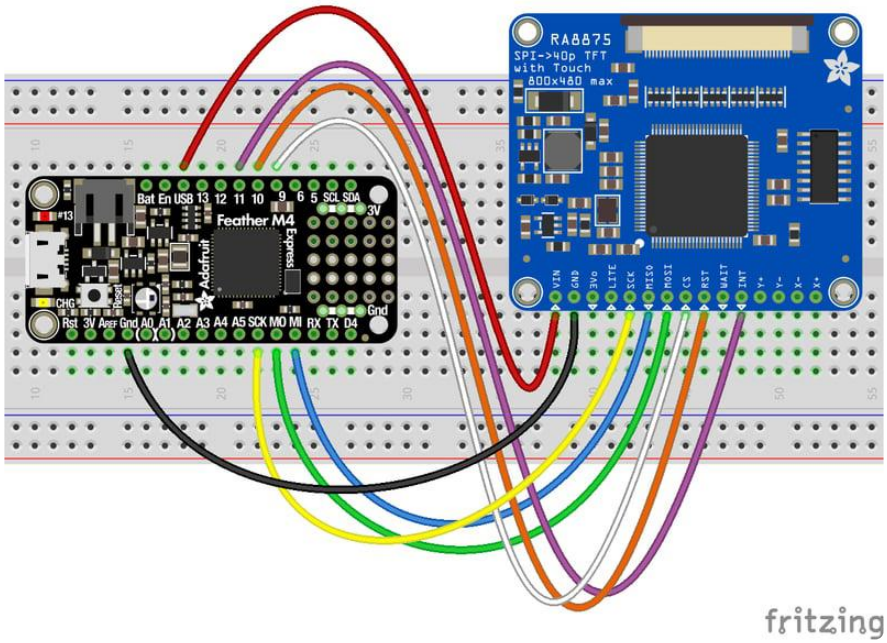
It is very easy to use the RA8875 with CircuitPython. We recommend using an M4 based board due to the size of library.

CircuitPython Microcontroller Wiring

Start by connecting the power pins

- 3-5V Vin connects to the Feather's USB pin
- GND connects to the Feather's ground
- SCK connects to SPI clock. On the Feather M4 Express, thats also SCK
- MISO connects to SPI MISO. On the Feather M4 Express, thats MI
- MOSI connects to SPI MOSI. On the Feather M4 Express, thats MO
- CS connects to our SPI Chip Select pin. We'll be using Digital 9 but you can later change this to any pin
- RST connects to our Reset pin. We'll be using Digital 10 but you can later change this pin as well.
- INT connects to our Interrupt pin. This pin is actually optional, but improves the accuracy. We'll be using Digital 11 but you can later change this pin too.

That's it! You don't need to connect any of the other pins.



Library Installation

You'll need to install the Adafruit CircuitPython RA8875 () library on your CircuitPython board.

You will also need to install the Adafruit CircuitPython BusDevice () library which is the only dependency.

First, make sure you are running the latest version of Adafruit CircuitPython () for your board.

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

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

- adafruit_ra8875
- adafruit_bus_device

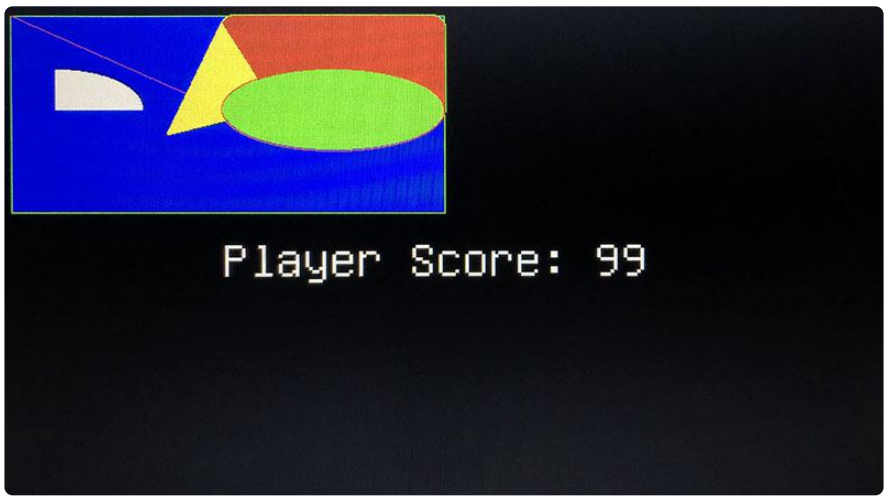
Before continuing make sure your board's lib folder or root filesystem has the adafruit_ra8875 and adafruit_bus_device folders copied over.

Usage

To demonstrate the usage of the display, we'll use the example python script included with the library.

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

# Quick test of RA8875 with Feather M4
import time
import busio
import digitalio
import board

from adafruit_ra8875 import ra8875
from adafruit_ra8875.ra8875 import color565

BLACK = color565(0, 0, 0)
RED = color565(255, 0, 0)
BLUE = color565(0, 255, 0)
GREEN = color565(0, 0, 255)
YELLOW = color565(255, 255, 0)
CYAN = color565(0, 255, 255)
MAGENTA = color565(255, 0, 255)
WHITE = color565(255, 255, 255)

# Configuration for CS and RST pins:
cs_pin = digitalio.DigitalInOut(board.D9)
rst_pin = digitalio.DigitalInOut(board.D10)
int_pin = digitalio.DigitalInOut(board.D11)

# Config for display baudrate (default max is 6mhz):
BAUDRATE = 6000000

# Setup SPI bus using hardware SPI:
spi = busio.SPI(clock=board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Create and setup the RA8875 display:
display = ra8875.RA8875(spi, cs=cs_pin, rst=rst_pin, baudrate=BAUDRATE)
display.init()

display.fill(RED)
time.sleep(0.500)
display.fill(YELLOW)
time.sleep(0.500)
display.fill(BLUE)
time.sleep(0.500)
display.fill(CYAN)
time.sleep(0.500)
display.fill(MAGENTA)
time.sleep(0.500)
display.fill(BLACK)
display.circle(100, 100, 50, BLACK)
display.fill_circle(100, 100, 49, BLUE)

display.fill_rect(10, 10, 400, 200, GREEN)
display.rect(10, 10, 400, 200, BLUE)
display.fill_round_rect(200, 10, 200, 100, 10, RED)
display.round_rect(200, 10, 200, 100, 10, BLUE)
display.pixel(10, 10, BLACK)
display.pixel(11, 11, BLACK)
display.line(10, 10, 200, 100, RED)
display.fill_triangle(200, 15, 250, 100, 150, 125, YELLOW)
```

```
display.triangle(200, 15, 250, 100, 150, 125, BLACK)
display.fill_ellipse(300, 100, 100, 40, BLUE)
display.ellipse(300, 100, 100, 40, RED)
display.curve(50, 100, 80, 40, 2, BLACK)
display.fill_curve(50, 100, 78, 38, 2, WHITE)

display.txt_set_cursor(display.width // 2 - 200, display.height // 2 - 20)
display.txt_trans(WHITE)
display.txt_size(2)
testvar = 99
display.txt_write("Player Score: " + str(testvar))

display.touch_init(int_pin)
display.touch_enable(True)

x_scale = 1024 / display.width
y_scale = 1024 / display.height

# Main loop:
while True:
    if display.touched():
        coords = display.touch_read()
        display.fill_circle(
            int(coords[0] / x_scale), int(coords[1] / y_scale), 4, MAGENTA
        )
        display.txt_color(WHITE, BLACK)
        display.txt_set_cursor(display.width // 2 - 220, display.height // 2 - 20)
        display.txt_size(2)
        display.txt_write(
            "Position ("
            + str(int(coords[0] / x_scale))
            + ", "
            + str(int(coords[1] / y_scale))
            + ")"
        )
```



We start out by importing any libraries we want to use in our code. This includes a special function that converts 24-bit color into 16-bit color which is what this display uses.

```
import time
import busio
import digitalio
import board
```

```
import adafruit_ra8875.ra8875 as ra8875
from adafruit_ra8875.ra8875 import color565
```

Next, we define a bunch of colors for ease of reuse.

```
BLACK = color565(0, 0, 0)
RED = color565(255, 0, 0)
BLUE = color565(0, 255, 0)
GREEN = color565(0, 0, 255)
YELLOW = color565(255, 255, 0)
CYAN = color565(0, 255, 255)
MAGENTA = color565(255, 0, 255)
WHITE = color565(255, 255, 255)
```

Next, we initialize SPI and the display. 6 MHz was about the fastest that it would stably run in CircuitPython. Initialization is much easier in CircuitPython. By default this runs for a display of 800x480 pixels in size. If you have a display of a different size such as the 480x272, you could pass the width and height parameters into the constructor.

```
# Configuration for CS and RST pins:
cs_pin = digitalio.DigitalInOut(board.D9)
rst_pin = digitalio.DigitalInOut(board.D10)
int_pin = digitalio.DigitalInOut(board.D11)

# Config for display baudrate (default max is 6mhz):
BAUDRATE = 6000000

# Setup SPI bus using hardware SPI:
spi = busio.SPI(clock=board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Create and setup the RA8875 display:
display = ra8875.RA8875(spi, cs=cs_pin, rst=rst_pin, baudrate=BAUDRATE)
display.init()
```

Next, we fill the screen with a few different colors.

```
display.fill(RED)
time.sleep(0.500)
display.fill(YELLOW)
time.sleep(0.500)
display.fill(BLUE)
time.sleep(0.500)
display.fill(CYAN)
time.sleep(0.500)
display.fill(MAGENTA)
time.sleep(0.500)
```

After that, we clear the screen and draw a bunch of different shapes to demonstrate the hardware accelerated shape drawing functions.

```
display.fill(BLACK)
display.circle(100, 100, 50, BLACK)
display.fill_circle(100, 100, 49, BLUE)
```

```
display.fill_rect(11, 11, 398, 198, GREEN)
display.rect(10, 10, 400, 200, BLUE)
display.fill_round_rect(200, 10, 200, 100, 10, RED)
display.round_rect(199, 9, 202, 102, 12, BLUE)
display.pixel(10, 10, BLACK)
display.pixel(11, 11, BLACK)
display.line(10, 10, 200, 100, RED)
display.triangle(200, 15, 250, 100, 150, 125, BLACK)
display.fill_triangle(200, 16, 249, 99, 151, 124, YELLOW)
display.ellipse(300, 100, 100, 40, RED)
display.fill_ellipse(300, 100, 98, 38, BLUE)
display.curve(50, 100, 80, 40, 2, BLACK)
display.fill_curve(50, 100, 78, 38, 2, WHITE)
```

Next, we draw some text on the screen. Please note that unlike the Arduino library, mode changing is automatically handled in the library.

```
display.txt_set_cursor(display.width // 2 - 200, display.height // 2 - 20)
display.txt_trans(WHITE)
display.txt_size(2)
testvar = 99
display.txt_write("Player Score: " + str(testvar))
```

Finally, touch is enabled and in the main loop we read the coordinates of the text, display where the touch occurred on the screen and draw circles in that location.

```
display.touch_init(int_pin)
display.touch_enable(True)

x_scale = 1024 / display.width
y_scale = 1024 / display.height

# Main loop:
while True:
    if display.touched():
        coords = display.touch_read()
        display.fill_circle(int(coords[0]/x_scale), int(coords[1]/y_scale), 4,
MAGENTA)
        display.txt_color(WHITE, BLACK)
        display.txt_set_cursor(display.width // 2 - 220, display.height // 2 - 20)
        display.txt_size(2)
        display.txt_write("Position (" + str(int(coords[0]/x_scale)) + ", " +
                          str(int(coords[1]/y_scale)) + ")")
```

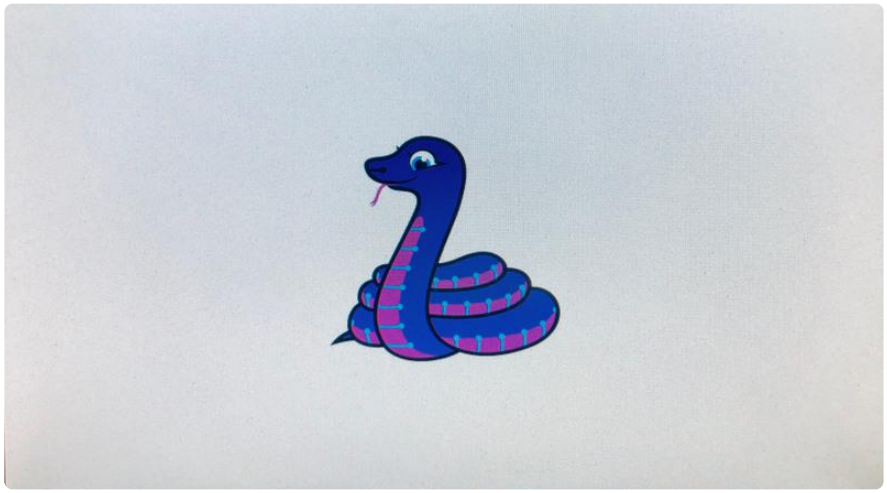
Loading a Bitmap

In this next example, we'll decode and load a bitmap that is stored as a file on the MicroController. Although the image is included in the examples folder of the library, we have provided a link to it for your convenience.

Click here to Download ra8875_blinka.bmp

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

# Quick bitmap test of RA8875 with Feather M4
import struct

import busio
import digitalio
import board

from adafruit_ra8875 import ra8875
from adafruit_ra8875.ra8875 import color565

WHITE = color565(255, 255, 255)

# Configuration for CS and RST pins:
cs_pin = digitalio.DigitalInOut(board.D9)
rst_pin = digitalio.DigitalInOut(board.D10)

# Config for display baudrate (default max is 6mhz):
BAUDRATE = 8000000

# Setup SPI bus using hardware SPI:
spi = busio.SPI(clock=board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Create and setup the RA8875 display:
display = ra8875.RA8875(spi, cs=cs_pin, rst=rst_pin, baudrate=BAUDRATE)
display.init()
display.fill(WHITE)


def convert_555_to_565(rgb):
    return (rgb & 0x7FE0) << 1 | 0x20 | rgb & 0x001F


class BMP:
    def __init__(self, filename):
        self.filename = filename
        self.colors = None
        self.data = 0
        self.data_size = 0
        self.bpp = 0
        self.width = 0
        self.height = 0
        self.read_header()

    def read_header(self):
        if self.colors:
            return
        with open(self.filename, "rb") as f:
            f.seek(10)
            self.data = int.from_bytes(f.read(4), "little")
            f.seek(18)
            self.width = int.from_bytes(f.read(4), "little")
            self.height = int.from_bytes(f.read(4), "little")
            f.seek(28)
            self.bpp = int.from_bytes(f.read(2), "little")
            f.seek(34)
            self.data_size = int.from_bytes(f.read(4), "little")
            f.seek(46)
            self.colors = int.from_bytes(f.read(4), "little")

    def draw(self, disp, x=0, y=0):
        print("{:d}x{:d} image".format(self.width, self.height))
        print("{:d}-bit encoding detected".format(self.bpp))
        line = 0
        line_size = self.width * (self.bpp // 8)
        if line_size % 4 != 0:
```

```
            line_size += 4 - line_size % 4
        current_line_data = b""
        with open(self.filename, "rb") as f:
            f.seek(self.data)
            disp.set_window(x, y, self.width, self.height)
            for line in range(self.height):
                current_line_data = b""
                line_data = f.read(line_size)
                for i in range(0, line_size, self.bpp // 8):
                    if (line_size - i) < self.bpp // 8:
                        break
                    if self.bpp == 16:
                        color = convert_555_to_565(line_data[i] | line_data[i + 1]
<< 8)
                    if self.bpp in (24, 32):
                        color = color565(
                            line_data[i + 2], line_data[i + 1], line_data[i]
                        )
                    current_line_data = current_line_data + struct.pack(">H", color)
                disp.setxy(x, self.height - line + y)
                disp.push_pixels(current_line_data)
            disp.set_window(0, 0, disp.width, disp.height)


bitmap = BMP("/ra8875_blinka.bmp")
x_position = (display.width // 2) - (bitmap.width // 2)
y_position = (display.height // 2) - (bitmap.height // 2)
bitmap.draw(display, x_position, y_position)
```



Let's dive in and take a look at how the code works. Just like the previous example, we start out by importing the libraries we need. In this case, we also need struct to help with color encoding.

```
import busio
import digitalio
import board

import adafruit_ra8875.ra8875 as ra8875
from adafruit_ra8875.ra8875 import color565
try:
    import struct
except ImportError:
    import ustruct as struct
```

White is the only color we will use, so we define that here.

```
WHITE = color565(255, 255, 255)
```

Next, we initialize the screen and set the background to white.

```
# Configuration for CS and RST pins:
cs_pin = digitalio.DigitalInOut(board.D9)
rst_pin = digitalio.DigitalInOut(board.D10)

# Config for display baudrate (default max is 6mhz):
BAUDRATE = 6000000

# Setup SPI bus using hardware SPI:
spi = busio.SPI(clock=board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Create and setup the RA8875 display:
display = ra8875.RA8875(spi, cs=cs_pin, rst=rst_pin, baudrate=BAUDRATE)
display.init()
display.fill(WHITE)
```

We have a helper function next for decoding 16-bit bitmaps into the correct bit structure. Even though we are using a 24-bit bmp in this example, if you would like to replace the image with one of your own and it happens to be a 16-bit bitmap, it will still work.

```
def convert_555_to_565(rgb):
    return (rgb &amp;amp; 0x7FE0) &amp;lt;&amp;lt; 1 | 0x20 | rgb &amp;amp; 0x001F
```

After that, we define a BMP class that will read the headers, decode the bitmap, and draw it to the screen. It starts off by initializing all of the values to 0 or None.

```
class BMP(object):
    def __init__(self, filename):
        self.filename = filename
        self.colors = None
        self.data = 0
        self.data_size = 0
        self.bpp = 0
        self.width = 0
        self.height = 0
        self.read_header()
```

Then it goes through and reads the important information in the header at specific places in the file. In this case, the bytes are in little endian format which means the bytes are arranged with starting with the least significant byte values are at the smallest address. If you are interested, you can read more about endianness, on Wikipedia. ()

```
    def read_header(self):
        if self.colors:
            return
        with open(self.filename, 'rb') as f:
            f.seek(10)
            self.data = int.from_bytes(f.read(4), 'little')
            f.seek(18)
            self.width = int.from_bytes(f.read(4), 'little')
            self.height = int.from_bytes(f.read(4), 'little')
            f.seek(28)
            self.bpp = int.from_bytes(f.read(2), 'little')
            f.seek(34)
            self.data_size = int.from_bytes(f.read(4), 'little')
            f.seek(46)
            self.colors = int.from_bytes(f.read(4), 'little')
```

In the Draw function we read the appropriate amount of data depending on the file encoding, make sure it's in a format that the display understands, and push it to the display. Because the data is pushed directly out to the RA8875, the amount of memory required is pretty minimal.

```
    def draw(self, disp, x=0, y=0):
        print("{:d}x{:d} image".format(self.width, self.height))
        print("{:d}-bit encoding detected".format(self.bpp))
        line = 0
        line_size = self.width * (self.bpp//8)
        if line_size % 4 != 0:
            line_size += (4 - line_size % 4)
        current_line_data = b''
        with open(self.filename, 'rb') as f:
            f.seek(self.data)
            disp.set_window(x, y, self.width, self.height)
            for line in range(self.height):
                current_line_data = b''
                line_data = f.read(line_size)
                for i in range(0, line_size, self.bpp//8):
                    if (line_size-i) &lt; self.bpp//8:
                        break
                    if self.bpp == 16:
                        color = convert_555_to_565(line_data[i] | line_data[i+1]
&lt;&lt; 8)
                    if self.bpp == 24 or self.bpp == 32:
                        color = color565(line_data[i+2], line_data[i+1],
line_data[i])
                    current_line_data = current_line_data + struct.pack("&gt;H",
color)
                disp.setxy(x, self.height - line + y)
                disp.push_pixels(current_line_data)
            disp.set_window(0, 0, disp.width, disp.height)
```

Finally, we declare our class, figure out the center of the screen and draw the bitmap. This allows it to display on both the larger and smaller screens.

```
bitmap = BMP("/ra8875_blinka.bmp")
x_position = (display.width // 2) - (bitmap.width // 2)
y_position = (display.height // 2) - (bitmap.height // 2)
bitmap.draw(display, x_position, y_position)
```

Python Wiring and Usage

Wiring

It's easy to use display breakouts with Python and the Adafruit CircuitPython RA8875 ( ) 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.

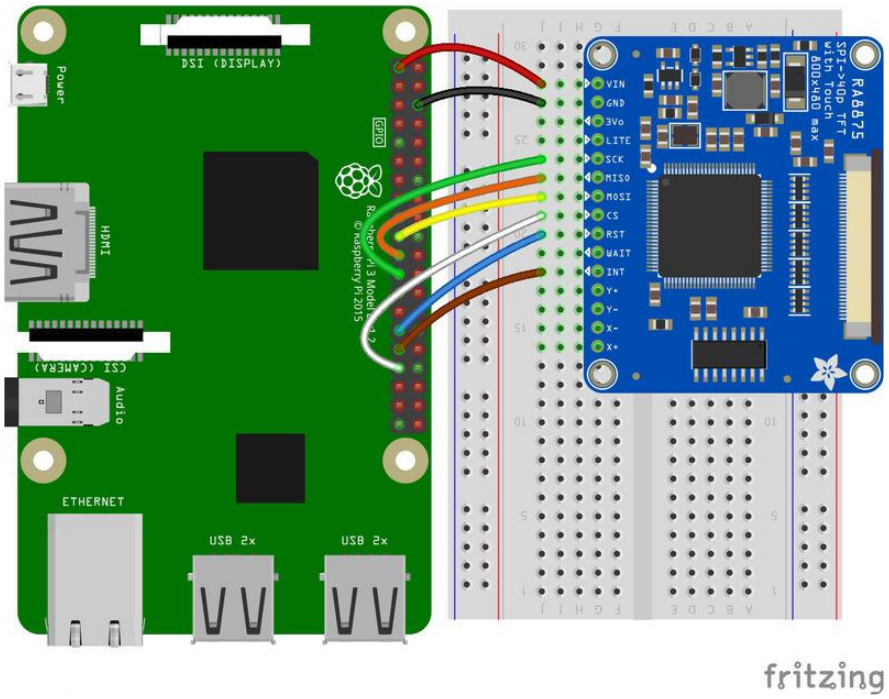
Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported ().

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

- 3-5V Vin connects to the Raspberry Pi's 3V pin
- GND connects to the Raspberry Pi's ground
- SCK connects to SPI clock. On the Raspberry Pi, thats SLCK
- MISO connects to SPI MISO. On the Raspberry Pi, thats also MISO
- MOSI connects to SPI MOSI. On the Raspberry Pi, thats also MOSI
- CS connects to our SPI Chip Select pin. We'll be using GPIO 13.
- RST connects to our Reset pin. We'll be using GPIO 5 but you can later change this pin later.
- INT connects to our Interrupt pin. This pin is actually optional, but improves the accuracy. We'll be using GPIO 6 but you can later change this pin too.



Download Fritzing Diagram

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 ()!

Python Installation of RA8875 Library

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

- `sudo pip3 install adafruit-circuitpython-ra8875`

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`

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

Usage

To use the RA8875 with the Raspberry Pi, you only need to make one or two changes to the CircuitPython examples in order to use them on the Pi.

Pin Mapping

First we need to change the Pin mapping because some of the pins that are used on the Feather are not available on the Raspberry Pi. Find the section with the following code:

```
cs_pin = digitalio.DigitalInOut(board.D9)
rst_pin = digitalio.DigitalInOut(board.D10)
int_pin = digitalio.DigitalInOut(board.D11)
```

and change it to the following:

```
cs_pin = digitalio.DigitalInOut(board.D13)
rst_pin = digitalio.DigitalInOut(board.D5)
int_pin = digitalio.DigitalInOut(board.D6)
```

Display Size

Next, the driver defaults to the 480x800 sized display. If you have a different size, you will need to add a couple of parameters. Find the line that looks like this:

```
display = ra8875.RA8875(spi, cs=cs_pin, rst=rst_pin, baudrate=BAUDRATE)
```

And go ahead and add a `width` and `height` parameter so if for instance, you had the 480x272 display, it would look like this:

```
display = ra8875.RA8875(spi, cs=cs_pin, rst=rst_pin, baudrate=BAUDRATE, width=480,
height=272)
```

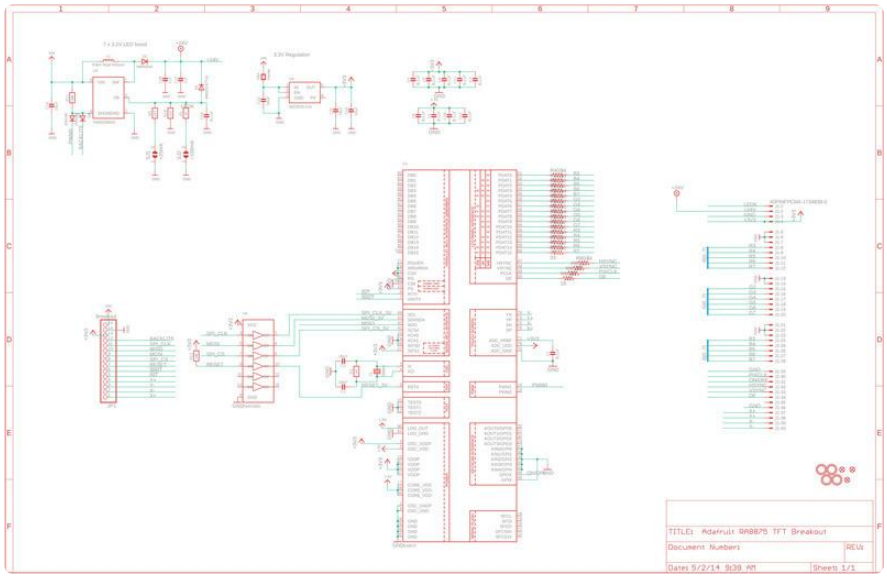
That should be it. You should be able to run both examples as described on the CircuitPython page.

Downloads

Files & Datasheets

- RA8875 Full Datasheet ()
- RA8875 App note ()
- Fritzing object in Adafruit Fritzing library ()
- EagleCAD PCB Files on GitHub ()

Schematic



Fabrication Print