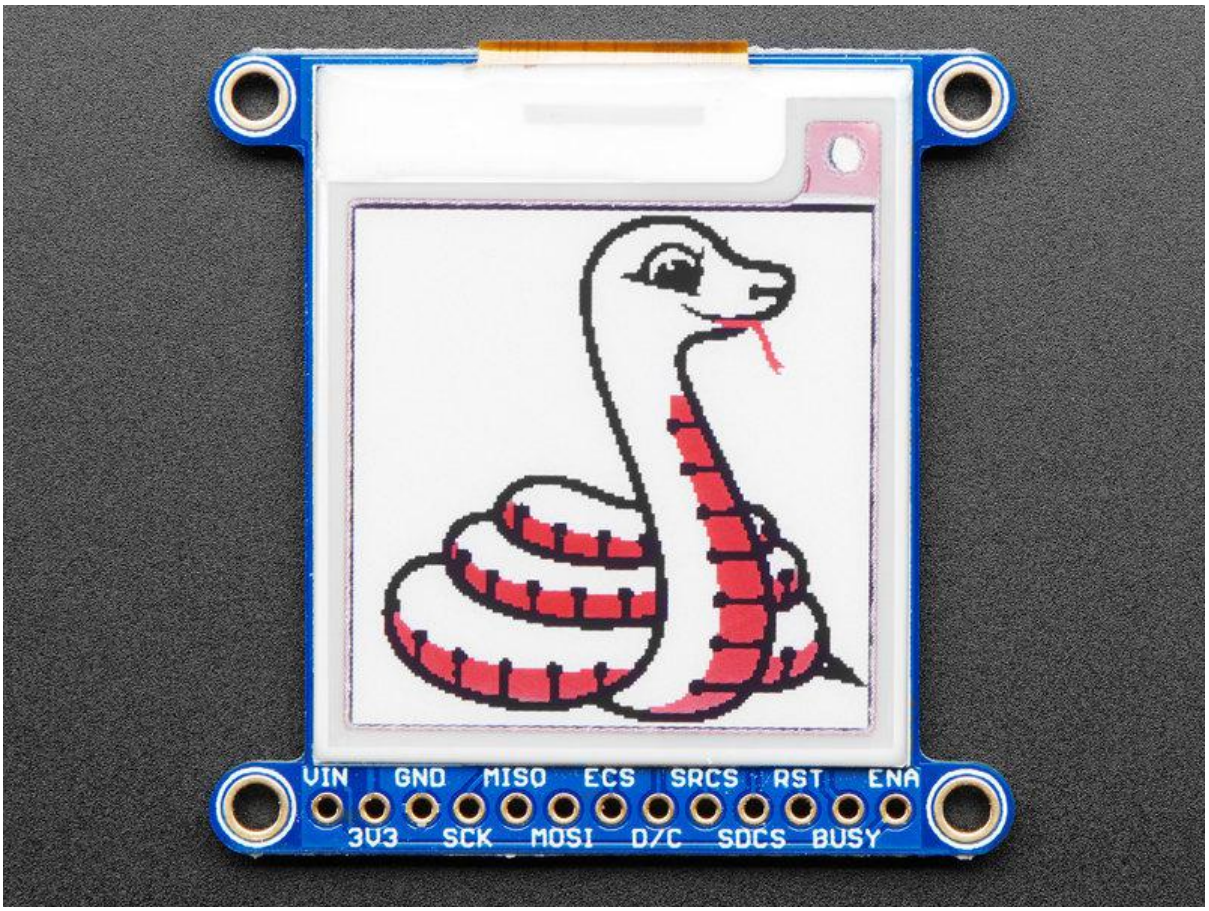
Adafruit eInk Display Breakouts and FeatherWings

Created by lady ada



https://learn.adafruit.com/adafruit-eink-display-breakouts

Last updated on 2021-12-09 03:20:06 PM EST

Table of Contents

Overview 5

Pinouts 8

- eInk Breakout Friend 9
- Power Pins 10
- Data Control Pins 11
- FeatherWing Connections 12

Shield Pinouts 13

- Power Pins 14
- Data Pins 14
- Buttons 14

Assembly 16

- Assembly 16
- Add the E-Ink Display 17
- And Solder! 18

Usage & Expectations 19

Arduino Setup 20

Arduino Code 21

- Wiring 21
- FeatherWing Connection 22
- Load First Demo 23
- Load Graphics Test Demo 24
- Unnecessary Pins 25

Drawing Bitmaps 25

Arduino Library Documentation 27

Adafruit GFX Library 28

CircuitPython Code 28

- CircuitPython Microcontroller Wiring 28
- CircuitPython eInk displayio Library Installation 29
- Usage 31

Python Code 36

- Wiring 36
- Setup 37
- Python Installation of EPD Library 37
- Download font5x8.bin 37
- DejaVu TTF Font 37
- Pillow Library 38
- Usage 38
- Tri-Color Example 39
- Monochrome Example 40
- Tri-Color Bitmap Example 41

- Full Example Code 45
- Image Drawing with Pillow 46
- Drawing Shapes and Text with Pillow 51

Python Docs 55

2.9" Grayscale eInk FeatherWing 56

Pinouts 57
- Power Pins 58
- Buttons 58
- Data Control Pins 59

Wiring 60
- FeatherWing Connection 60

Arduino Usage 61
- Configure Pins 61
- Configure Display Size 62
- Upload Sketch 62

Arduino Bitmaps 63

CircuitPython Code 65
- CircuitPython eInk displayio Library Installation 65
- Usage 66

Downloads 68
- Files 68
- Schematic 68
- Fab Print 68

2.9" Tri-Color eInk 69

Wiring 70
- Breakout Wiring 70
- FeatherWing Connection 70
- Python Wiring 71

Arduino Usage 71
- FeatherWing Wiring 72
- Breakout Wiring 72

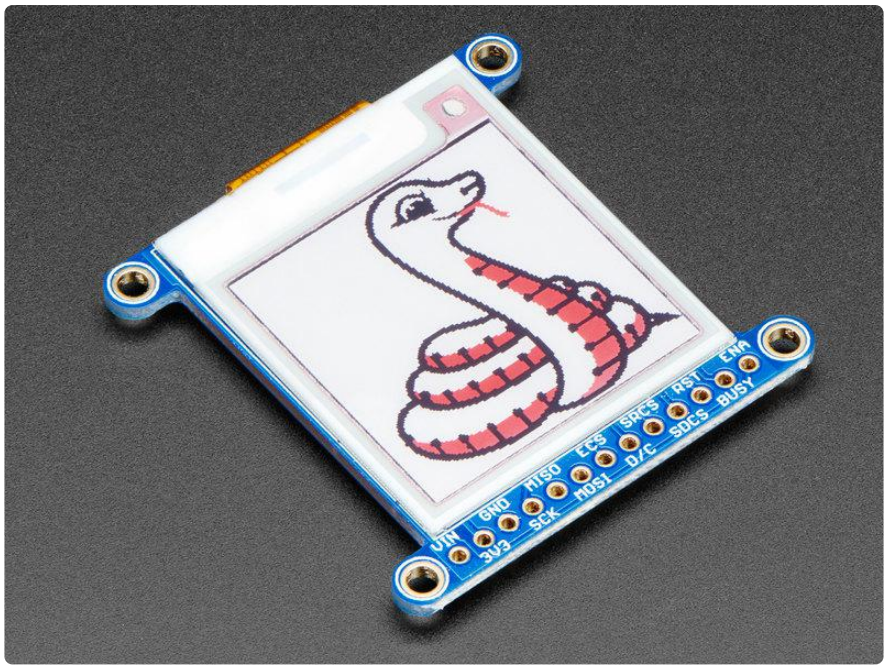
Downloads 73
- Files 73
- Schematic & Fabrication Prints 74
- Shared schematic for 1.54" 2.13" and 2.7" Breakouts 74
- 2.9 Inch Display 76
- eInk Friends 77
- 2.7" Shield 79

Overview



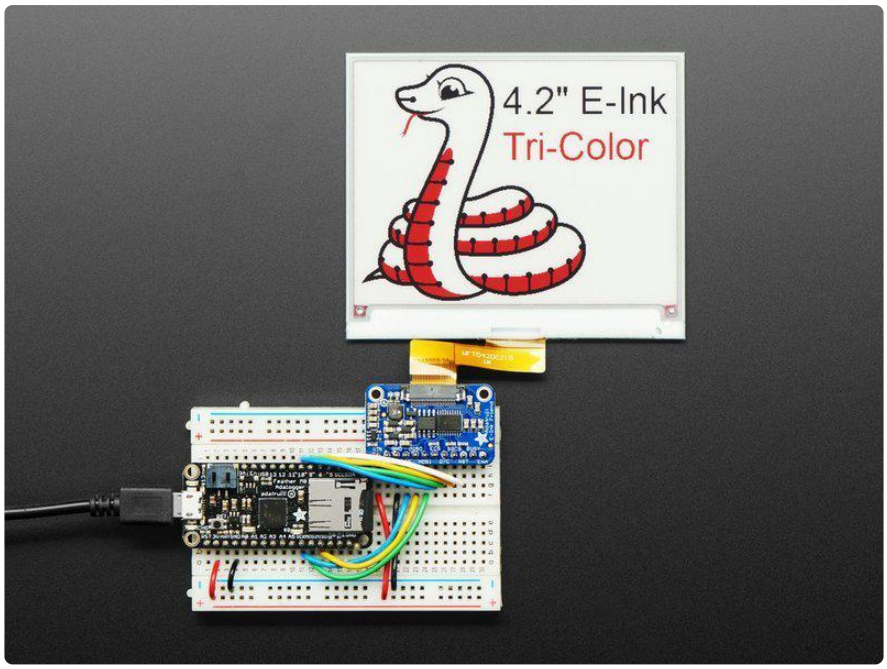
Easy e-paper finally comes to microcontrollers, with these breakouts, shields and friends that are designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those new-fangled 'e-readers' like the Kindle or Nook. They have gigantic electronic paper 'static' displays - that means the image stays on the display even when power is completely disconnected. The image is also high contrast and very daylight readable. It really does look just like printed paper!

We've liked these displays for a long time, but they were never designed for makers to use. Finally, we decided to make our own!

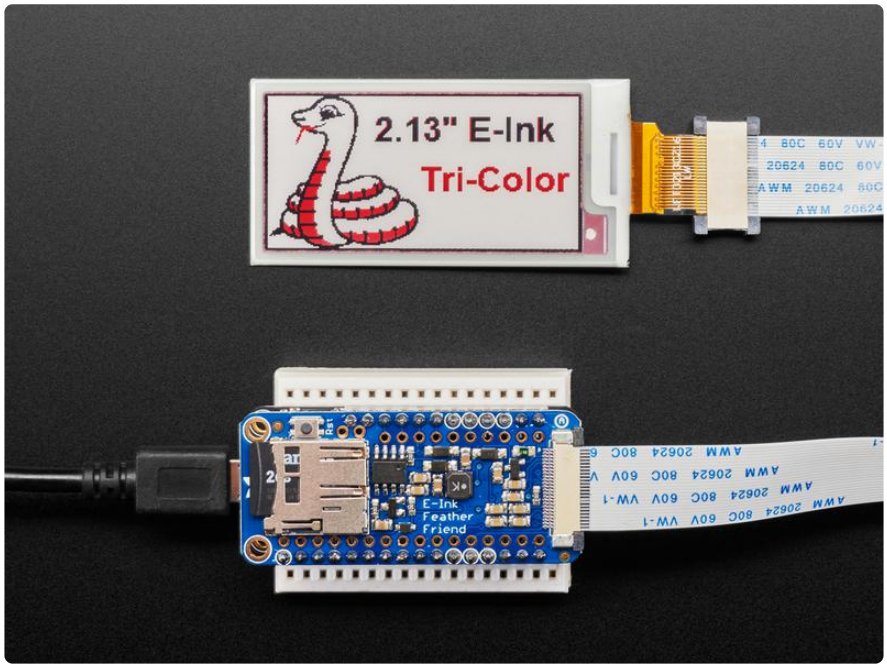


We have multiple tri-color displays. They have black and red ink pixels and a white-ish background. Using our Arduino library, you can create a 'frame buffer' with what pixels you want to have activated and then write that out to the display. Most simple breakouts leave it at that. But if you do the math, using even the smallest 1.54" display: 152 x 152 pixels x 2 colors = 5.7 KBytes. Which won't fit into many microcontroller memories. Heck, even if you do have 32KB of RAM, why waste 6KB?

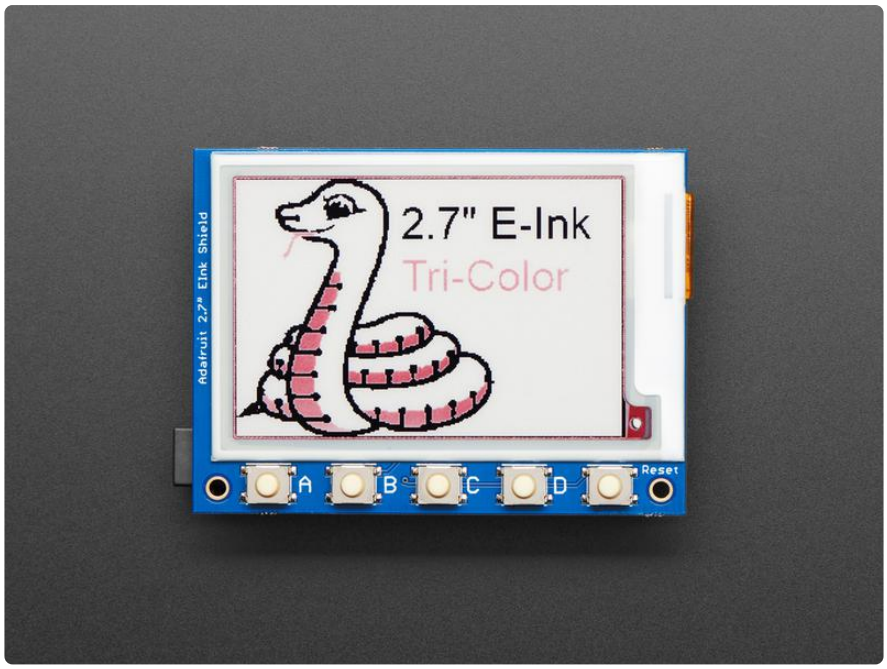
So we did you a favor and tossed a small SRAM chip on the back. This chip shares the SPI port the eInk display uses, so you only need one extra pin. And, no more frame-buffering! You can use the SRAM to set up whatever you want to display, then shuffle data from SRAM to eInk when you're ready. The library we wrote does all the work for you (https://adafru.it/BRK), you can just interface with it as if it were an Adafruit_GFX compatible display (https://adafru.it/BRK).



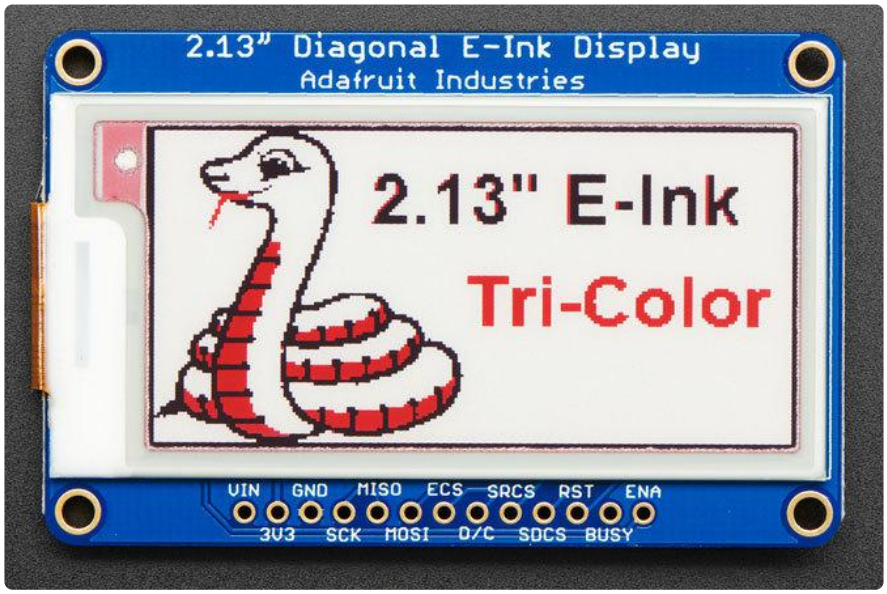
On the EInk Friends and Breakouts, for ultra-low power usages, the onboard 3.3V regulator has the Enable pin brought out so you can shut down the power to the SRAM, MicroSD and display.



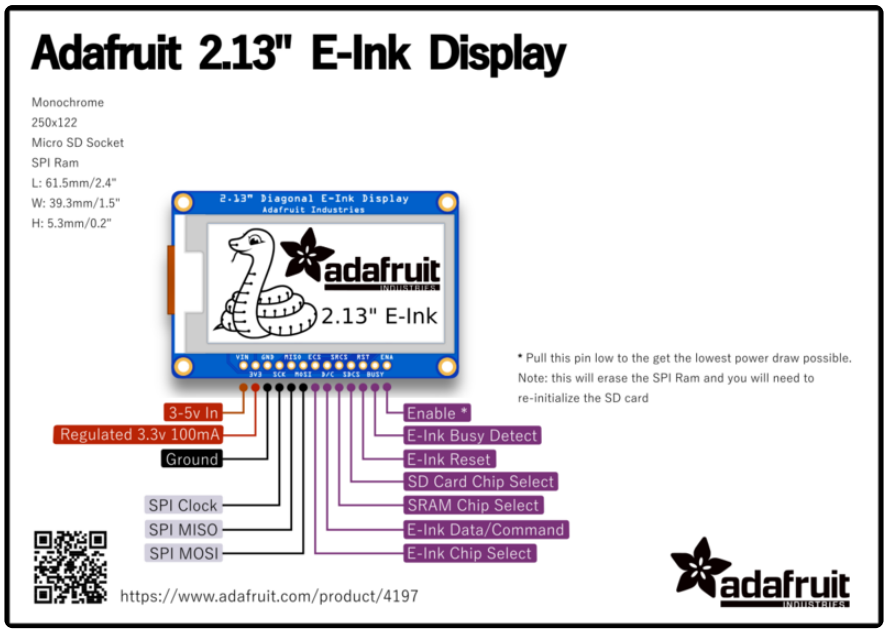
On the Breakouts and Shields, We even tossed on a MicroSD socket so you can store images, text files, whatever you like to display. Everything is 3 or 5V logic safe so you can use it with any and all microcontrollers.

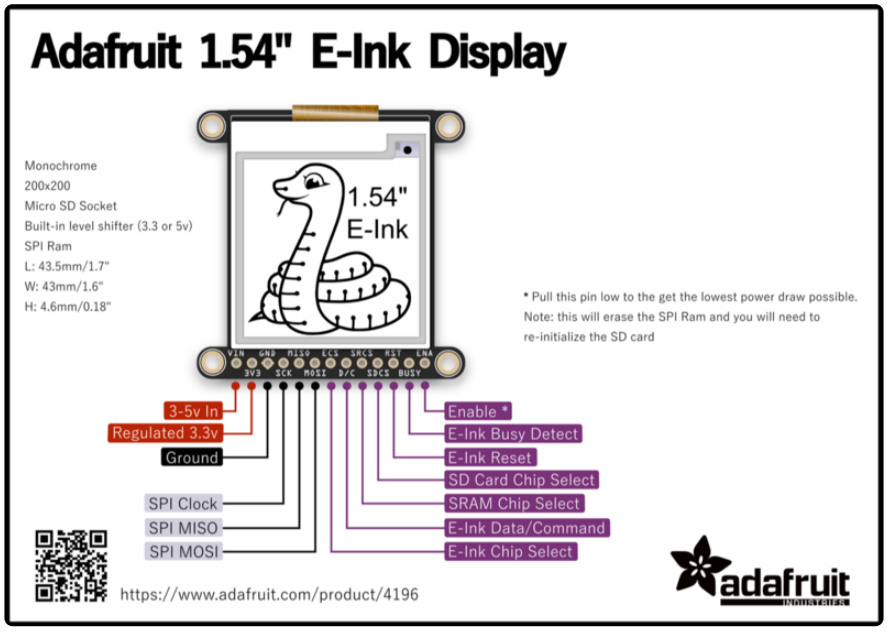


Pinouts



This e-Paper display uses SPI to receive image data. Since the display is SPI, it was easy to add two more SPI devices to share the bus - an SPI SRAM chip and SPI-driven SD card holder. There's quite a few pins and a variety of possible combinations for control depending on your needs

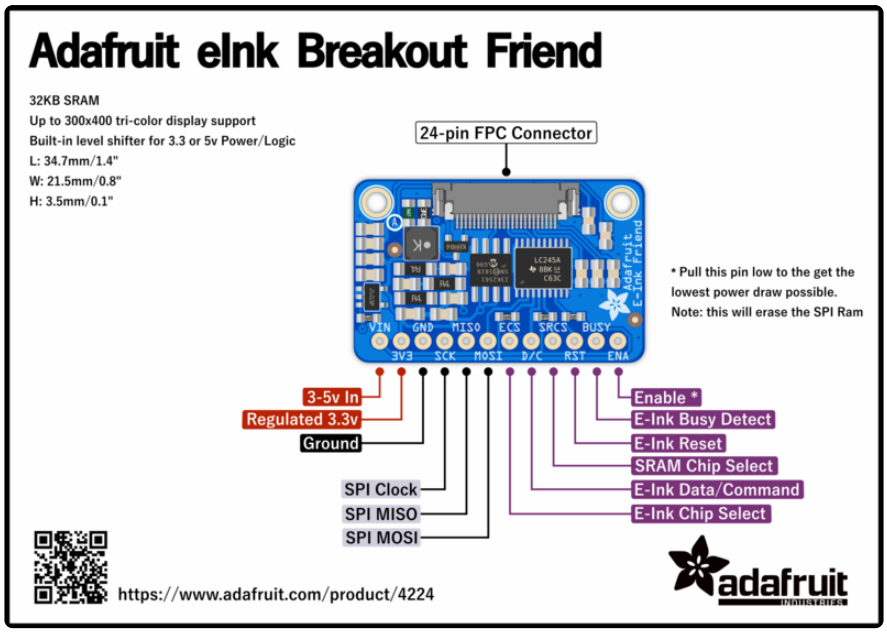




The pin outs are identical for the 1.54", 2.13" and 2.7" E-Ink display!

eInk Breakout Friend

Connect a bare eInk display to this breakout to use it!



Power Pins



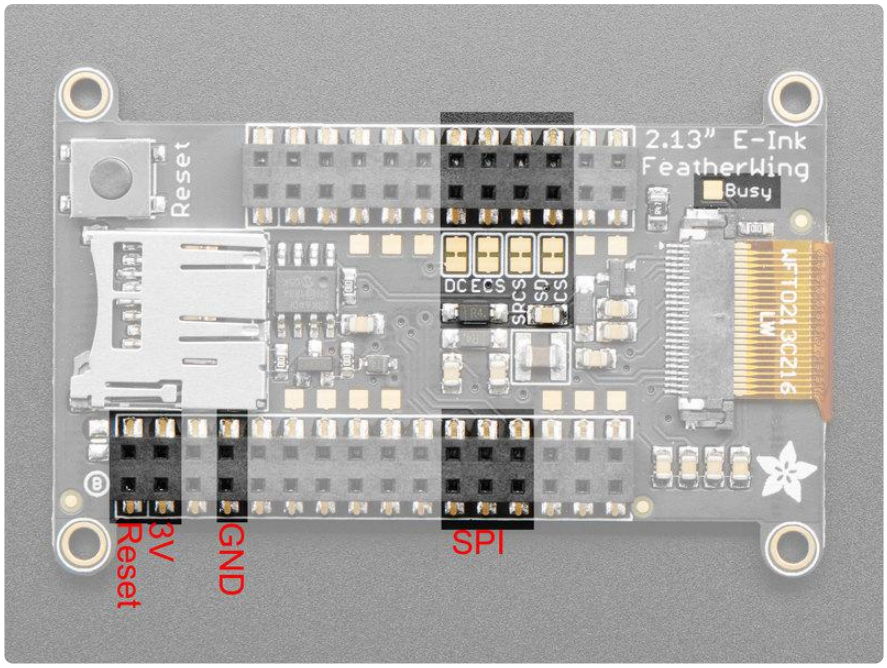
- 3-5V / Vin - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- 3.3V out - this is the 3.3V output from the onboard regulator, you can 'borrow' about 100mA if you need to power some other 3.3V logic devices
- GND - this is the power and signal ground pin
- ENAble - This pin is all the way on the right. It is connected to the enable pin on the onboard regulator that powers everything. If you want to really have the lowest possible power draw, pull this pin low! Note that if you do so you will cut power to the eInk display but also the SPI RAM (thus erasing it) and the SD card (which means you'll have to re-initialize it when you re-power

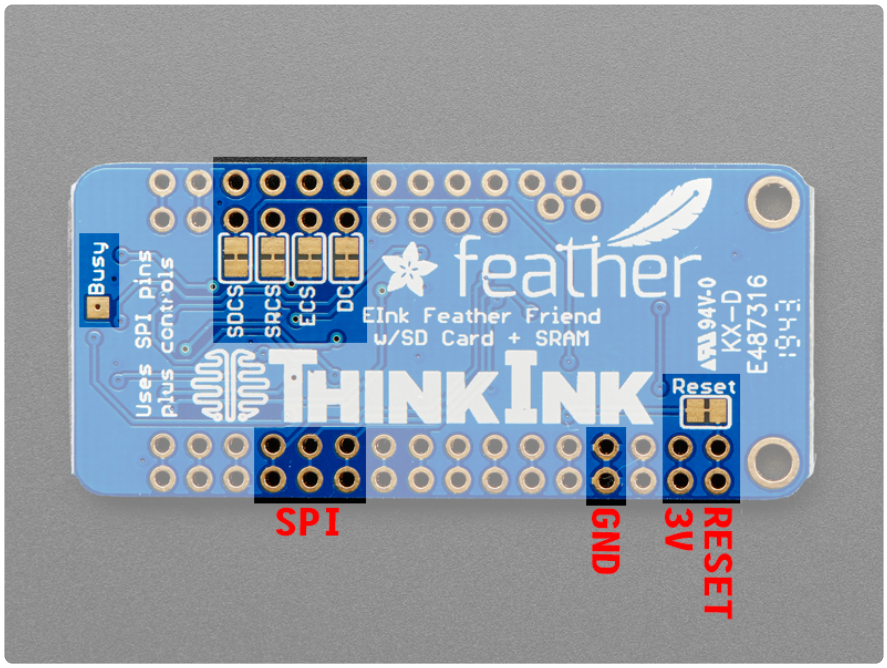
Data Control Pins



- SCK - this is the SPI clock input pin, required for e-Ink, SRAM and SD card
- MISO - this is the SPI Microcontroller In Serial Out pin, its used for the SD card and SRAM. It isn't used for the e-Ink display which is write-only, however you'll likely be using the SRAM to buffer the display so connect this one too!
- MOSI - this is the SPI Microcontroller Out Serial In pin, it is used to send data from the microcontroller to the SD card, SRAM and e-Ink display
- ECS - this is the E-Ink Chip Select, required for controlling the display
- D/C - this is the e-Ink Data/Command pin, required for controlling the display
- SRCS - this is the SRAM Chip Select, required for communicating with the onboard RAM chip.
- SDCS - this is the SD card Chip Select, required for communicating with the onboard SD card holder. You can leave this disconnected if you aren't going to access SD cards
- RST - this is the E-Ink ReSeT pin, you may be able to share this with your microcontroller reset pin but if you can, connect it to a digital pin.
- BUSY - this is the e-Ink busy detect pin, and is optional if you don't want to connect the pin (in which case the code will just wait an approximate number of seconds)

FeatherWing Connections





The FeatherWing eInk Display and eInk Feather Friend are a little more compact but have just about the same pins as the breakout

- SPI MOSI/MISO/SCK are on the FeatherWing SPI connection pads

SD CS, SRAM CS, EINK CS and DC are in order after the two I2C pins. The numbers of the pins these correspond to will differ from board to board. However, on 32u4/328p/M0/M4/nRF52840 and many other boards you will see the following connections

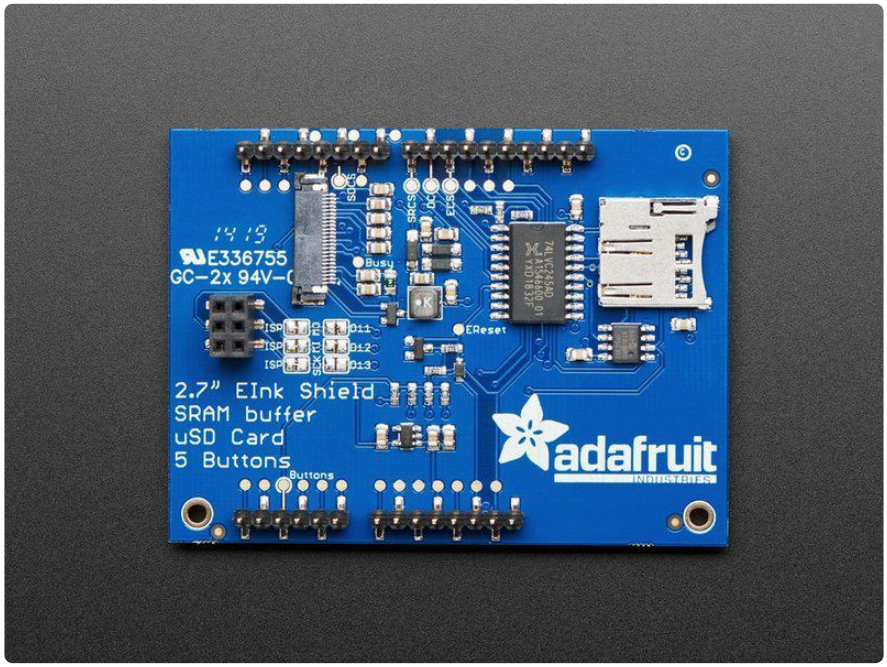
- SD CS to Pin D5
- SRAM CS to Pin D6
- EINK CS to Pin D9
- EINK DC to Pin D10

If you do not plan to use the SD card, you can cut the trace to SD CS. Likewise for SRAM CS.

The Reset pin for the E-Ink display is connected to an auto-reset circuit and also to the Feather Reset pin, so it will reset when you press the reset button.

The Busy pin is available on a breakout pad, you can solder it to a wire and connect to a pin if you need it - we figure most people will just use a fixed delay.

Shield Pinouts



The 2.7" EInk Shield is a little special in that the pins are fixed, so we'll document that here.

Power Pins

- 5V - this pin on the Arduino is used to generate the 3V logic level for the EInk chip, level shifter and boost converter.
- GND - connected for power and logic reference
- IORef - this pin is connected to the level shifter and pullups. On modern Arduino boards it is connected to the logic level of the board (3V or 5V)

Data Pins

- SCK, MISO, MOSI - The 3 SPI logic pins are connected through the 2x3 socket header which is compatible with any Arduino board. If you have an Arduino board without the 2x3 headers, you can cut the jumpers and connect the solder jumper traces to D13, D12 and D11 respectively.
- ECS (EInk Chip Select) - this is connected to D10
- DC (EInk Data/Command) - this is connected to D9
- SCS (SRAM Chip Select) - this is connected to D8
- SDCS (SD Card Chip Select) - this is connected to D5

The BUSY pin is not used on the 2.7" display (it doesn't do anything anyways)

The RESET pin is connected to the microcontroller reset pin, but is available on a pad labeled EReset if you want to toggle it yourself!

Buttons

The 4 buttons on the front are connected through a resistor divider to A3 you can use this function to determine what button was pressed:

```
int8_t readButtons(void) {
  uint16_t reading = analogRead(A3);
  //Serial.println(reading);

  if (reading &gt; 600) {
    return 0; // no buttons pressed
  }
  if (reading &gt; 400) {
    return 4; // button D pressed
  }
  if (reading &gt; 250) {
    return 3; // button C pressed
  }
  if (reading &gt; 125) {
    return 2; // button B pressed
  }
```

```
  return 1; // Button A pressed
}
```

Here's a simple test example for an Arduino with standard pin numbers:

```
/***************************************************
  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_ThinkInk.h"

#define EPD_DC      9 // can be any pin, but required!
#define EPD_CS      10  // can be any pin, but required!
#define EPD_BUSY    -1  // can set to -1 to not use a pin (will wait a fixed delay)
#define SRAM_CS     8  // can set to -1 to not use a pin (uses a lot of RAM!)
#define EPD_RESET   -1  // can set to -1 and share with chip Reset (can't deep
sleep)

// 2.7" Tricolor Featherwing or Breakout with IL91874 chipset
ThinkInk_270_Tricolor_C44 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);


void setup() {
  Serial.begin(115200);
  while (!Serial) { delay(10); }
  Serial.println("Adafruit EPD full update test in red/black/white");
  display.begin(THINKINK_TRICOLOR);
  display.setRotation(2);
}

void loop() {
  Serial.println("Banner demo");
  display.clearBuffer();
  display.setTextSize(3);
  display.setCursor((display.width() - 144)/2, (display.height() - 24)/2);
  display.setTextColor(EPD_BLACK);
  display.print("Tri");
  display.setTextColor(EPD_RED);
  display.print("Color");
  display.display();

  delay(15000);

  Serial.println("Color rectangle demo");
  display.clearBuffer();
  display.fillRect(display.width()/3, 0, display.width()/3, display.height(),
EPD_BLACK);
  display.fillRect((display.width()*2)/3, 0, display.width()/3, display.height(),
EPD_RED);
  display.display();

  delay(15000);

  Serial.println("Text demo");
  // large block of text
  display.clearBuffer();
  display.setTextSize(1);
  testdrawtext("Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur
adipiscing ante sed nibh tincidunt feugiat. Maecenas enim massa, fringilla sed
malesuada et, malesuada sit amet turpis. Sed porttitor neque ut ante pretium vitae
malesuada nunc bibendum. Nullam aliquet ultrices massa eu hendrerit. Ut sed nisi
```

```
lorem. In vestibulum purus a tortor imperdiet posuere. ", EPD_BLACK);
  display.display();

  delay(15000);

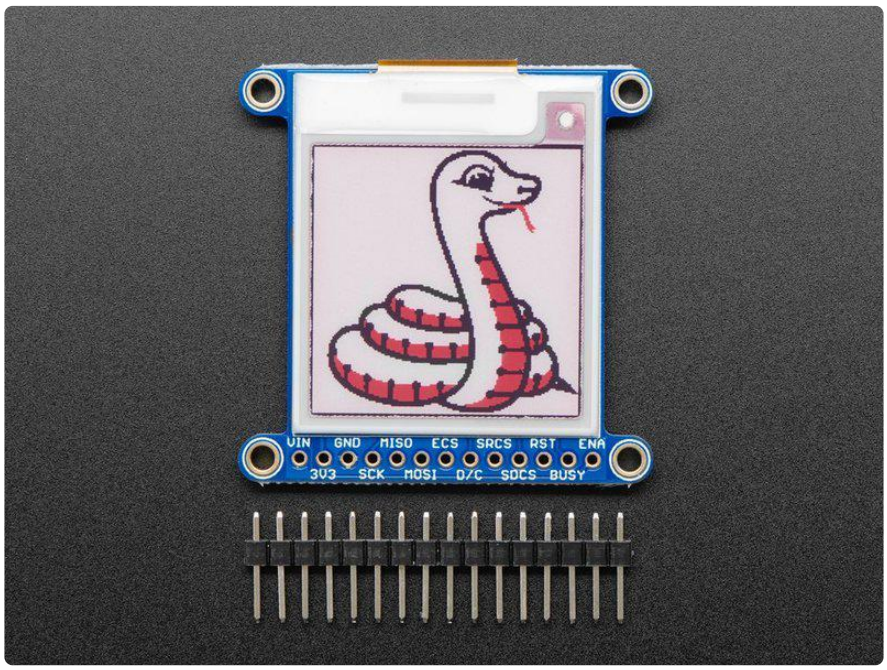
  display.clearBuffer();
  for (int16_t i=0; i&lt;display.width(); i+=4) {
    display.drawLine(0, 0, i, display.height()-1, EPD_BLACK);
  }

  for (int16_t i=0; i&lt;display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, EPD_RED);
  }
  display.display();

  delay(15000);
}

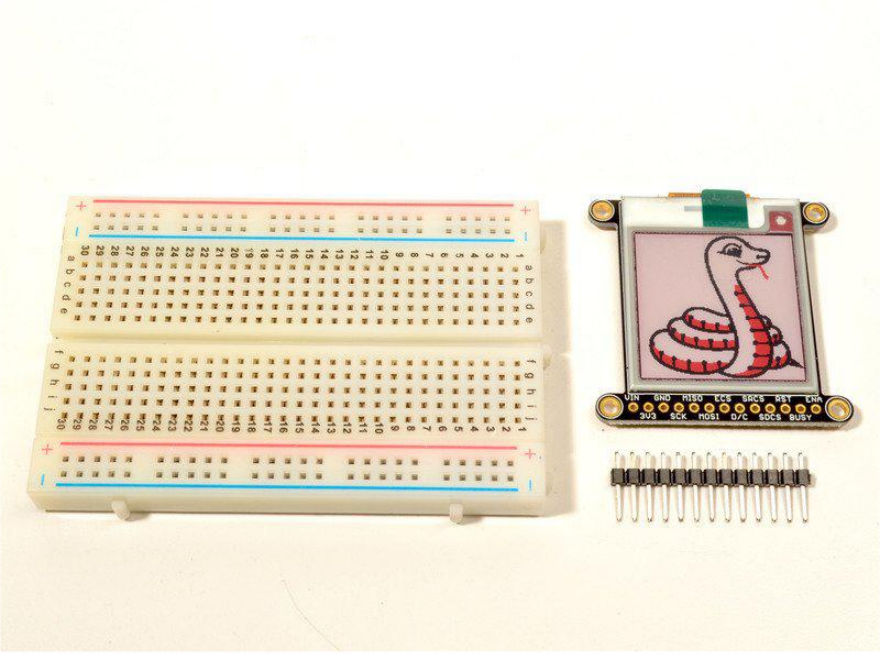

void testdrawtext(char *text, uint16_t color) {
  display.setCursor(0, 0);
  display.setTextColor(color);
  display.setTextWrap(true);
  display.print(text);
}
```

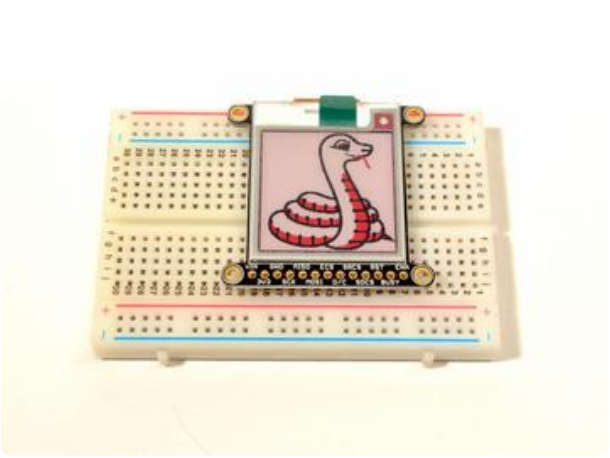
Assembly



Assembly

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

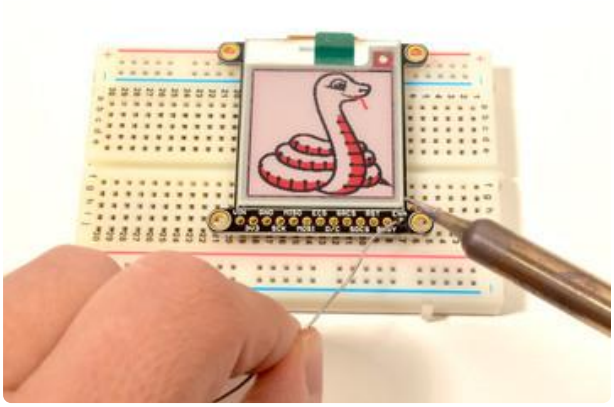


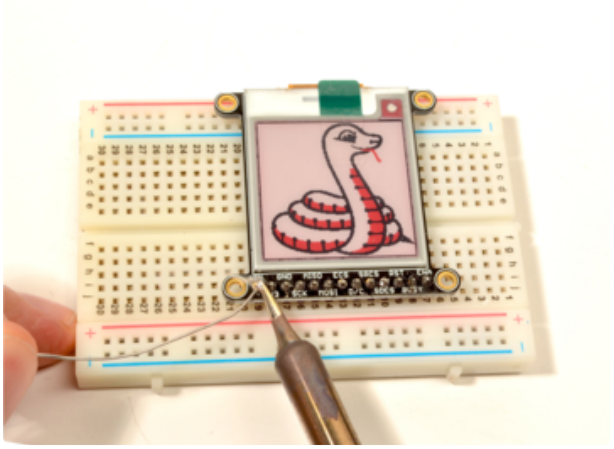


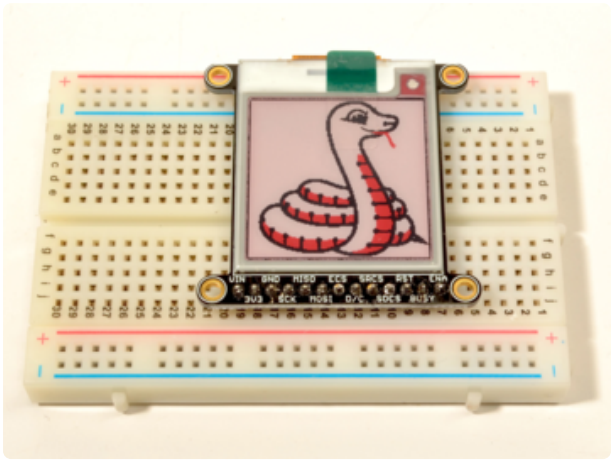
Add the E-Ink Display

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

And Solder!



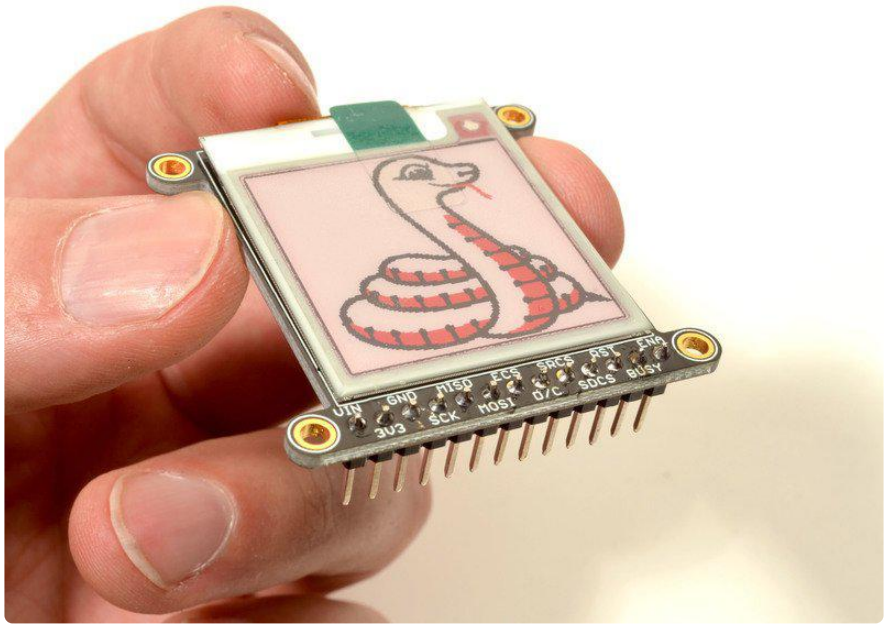




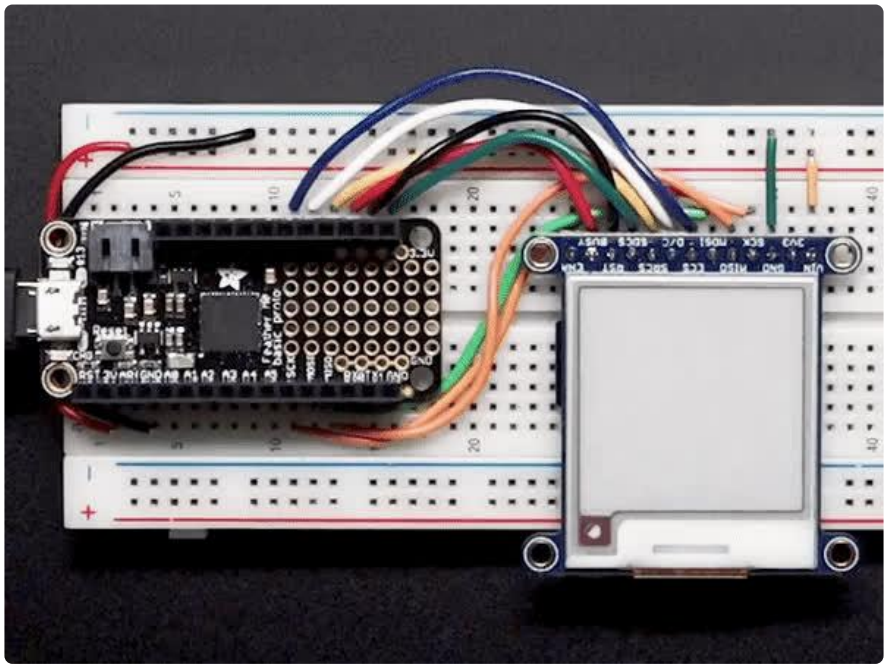
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out the Guide to Excellent Soldering (https://adafru.it/aTk)).

OK, you're done!



Usage & Expectations



One thing to remember with these small e-Ink screens is that its very slow compared to OLEDs, TFTs, or even 'memory displays'. It will take may seconds to fully erase and replace an image

There's also a recommended limit on refeshing - you shouldn't refresh or change the display more than every 3 minutes (180 seconds).

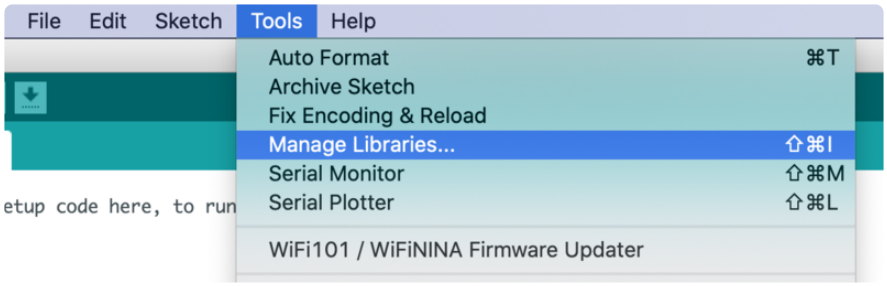
You don't have to refresh often, but with tri-color displays, the larger red ink dots will slowly rise, turning the display pinkish instead of white background. To keep the background color clear and pale, refresh once a day

Do not update more than once every 180 seconds or you may permanently damage the display

Arduino Setup

To use the display, you will need to install the Adafruit_EPD library (code on our github repository) (https://adafru.it/BRK). It is available from the Arduino library manager so we recommend using that.

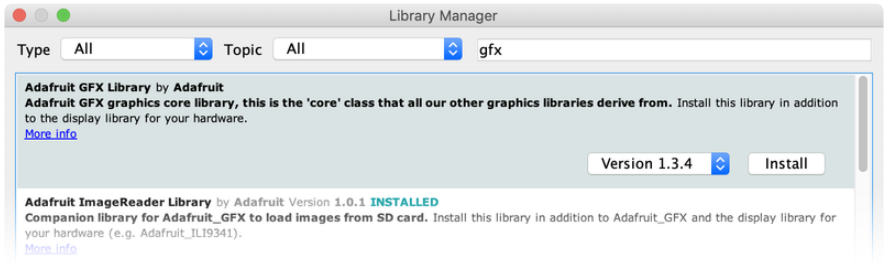
From the IDE open up the library manager...



And type in adafruit EPD to locate the library. Click Install

If you would like to draw bitmaps, do the same with adafruit ImageReader, click Install

Do the same to install the latest adafruit GFX library, click Install



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

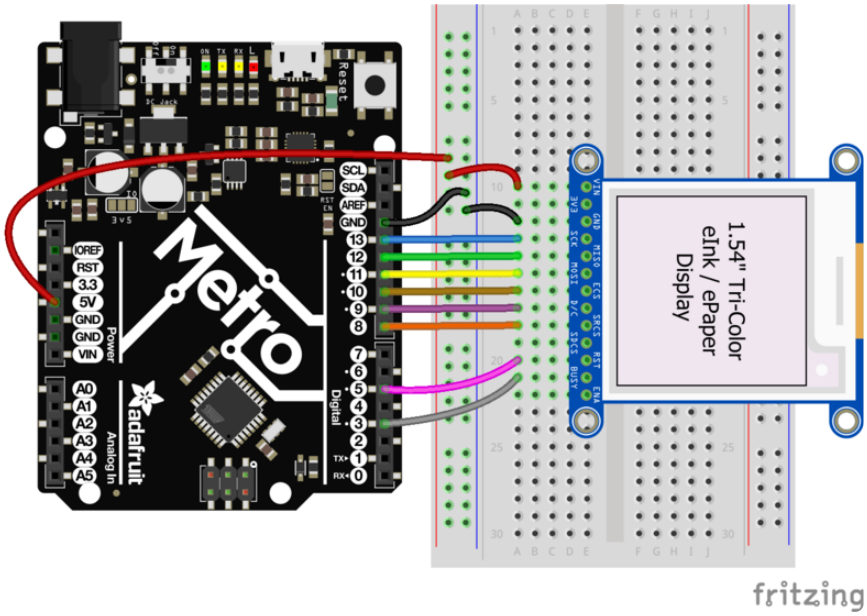
Arduino Code

Do not update more than once every 180 seconds or you may permanently damage the display

Wiring

Wiring up the display in SPI mode is pretty easy as there's not that many pins! We'll be using hardware SPI, but you can also use software SPI (any pins) later.

The pin outs are identical for the 1.54", 2.13" and 2.7" E-Ink display!



Start by connecting the power pins

- 3-5V Vin connects to the microcontroller board's 5V or 3.3V power supply pin
- GND connects to ground

Required SPI Pins

These use the hardware SPI interface and is required so check your microcontroller board to see which pins are hardware SPI

- CLK connects to SPI clock. On Arduino Uno/Duemilanove/328-based, thats Digital 13. (For other Arduino-compatibles See SPI Connections for more details (https://adafru.it/d5h))
- MISO connects to SPI MISO. On Arduino Uno/Duemilanove/328-based, thats Digital 12. (For other Arduino-compatibles See SPI Connections for more details (https://adafru.it/d5h))
- MOSI connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, thats Digital 11. (For other Arduino-compatibles See SPI Connections for more details (https://adafru.it/d5h))

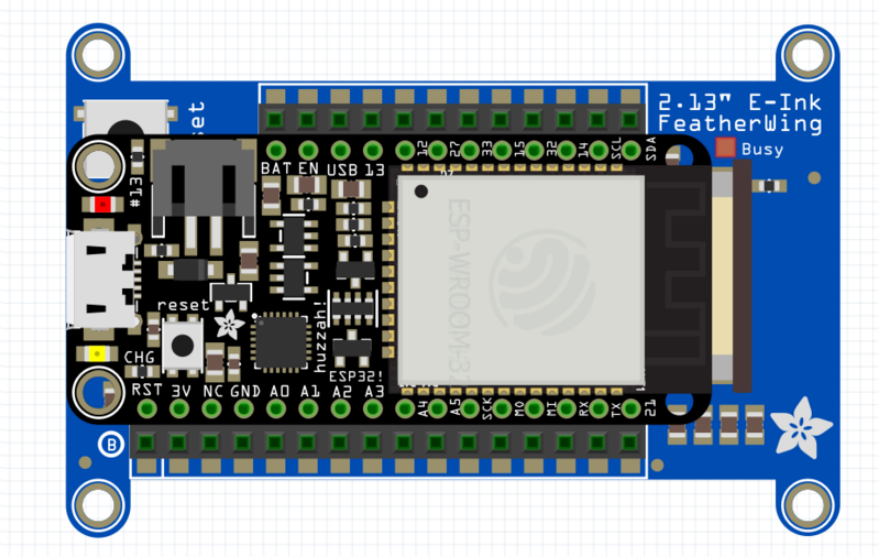
Other Digital I/O Pins

These can be set in the sketch to any pins you like but to follow the exact example code we'll use the following:

- ECS connects to our e-Ink Chip Select pin. We'll be using Digital 10 but you can later change this to any pin
- D/C connects to our e-Ink data/command select pin. We'll be using Digital 9 but you can later change this pin too.
- SRCS connects to our SRAM Chip Select pin. We'll be using Digital 8 but you can later change this to any pin
- RST connects to our e-Ink reset pin. We'll be using Digital 5 but you can later change this pin too.
- BUSY connects to our e-Ink busy pin. We'll be using Digital 3 but you can later change this pin too.

FeatherWing Connection

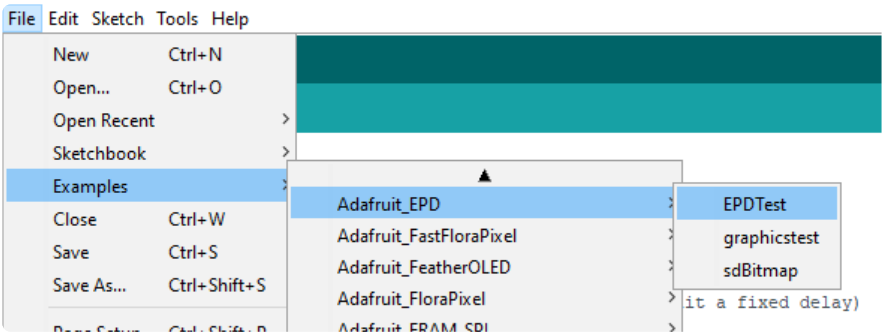
FeatherWing usage is easy, simply plug your Feather into the Wing



Load First Demo

Open up File→Examples→Adafruit_EPD→EPDTest

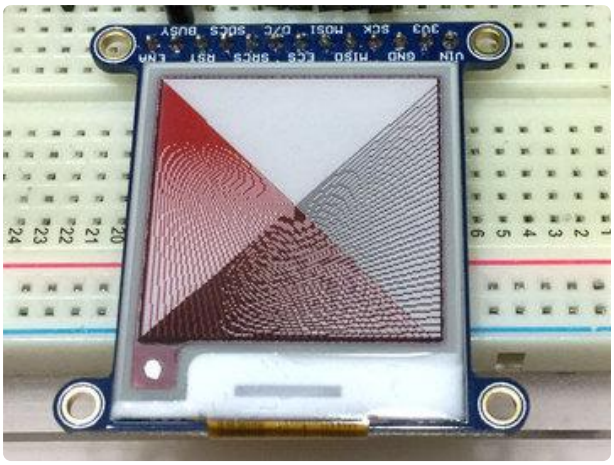
If you have the FeatherWing open up File→Examples→Adafruit_EPD→FeatherWingTest



At the top of the sketch find the lines that look like:

```
/* Uncomment the following line if you are using 1.54" tricolor EPD */
Adafruit_IL0373 display(152, 152 ,EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);
/* Uncomment the following line if you are using 2.13" tricolor EPD */
//Adafruit_IL0373 display(212, 104 ,EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);
/* Uncomment the following line if you are using 2.7" tricolor EPD */
//Adafruit_IL91874 display(264, 176 ,EPD_DC, EPD_RESET, EPD_CS, SRAM_CS);
```

And uncomment the matching object for the screen chipset and resolution you will be using. Then upload to your microcontroller wired up to the display



You will see the display flash a bunch and then a set of black and red lines will appear like shown on the left.

If you see the lines, your wiring is good! If not, go back and check your wiring to make sure its correct. If you didn't use the default pins, change them in the sketch

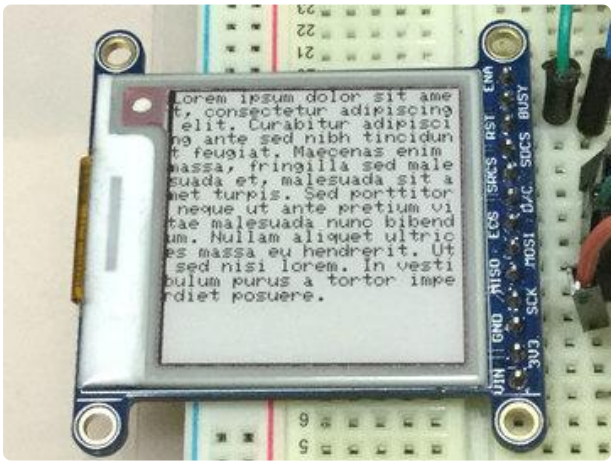
Load Graphics Test Demo

Open up File→Examples→Adafruit_EPD→graphicstest and upload to your microcontroller wired up to the display

If you're using a FeatherWing, use the pin definitions from the top of FeatherWingTest, for example:

```
#ifdef ESP8266
#define SD_CS 2
#define SRAM_CS 16
#define EPD_CS 0
#define EPD_DC 15
#endif
```

and copy those into the top of the graphics test sketch



This time you will see the display going through a range of tests, from pixels, lines, text circles etc.

This shows all the different shapes and techniques you can use that come with the Adafruit GFX library! Unlike most e-paper displays, where you can only draw an image, the built in SRAM lets you have full control over what shows up on the eInk screen.

Don't forget, after you call drawLine() or print() to display lines or text or other graphics, you must call display() to make the e-Ink display show the changes. Since this takes a few seconds, only do it once you've drawn everything you need.

Unnecessary Pins

Once you've gotten everything working you can experiment with removing the RST and BUSY pins. We recommend tying RST to your microcontroller's Reset line so the eInk display is reset when the microcontrollers is. The busy pin makes startup a little faster but we don't find it to be essential

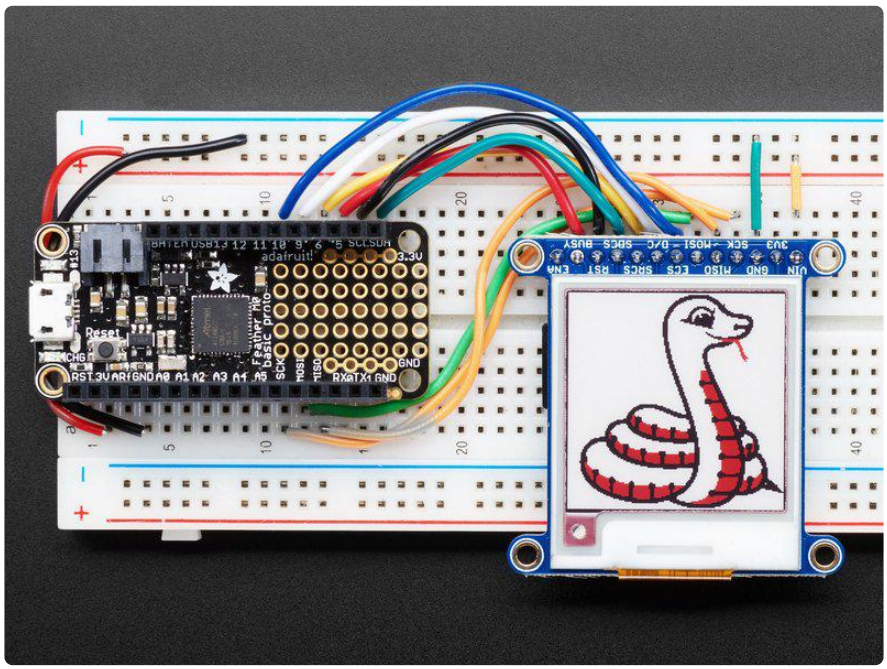
You can set the code as below to remove control of those pins from the Adafruit_EPD library:

```
#define EPD_RESET   -1 // can set to -1 and share with microcontroller Reset!
#define EPD_BUSY    -1 // can set to -1 to not use a pin (will wait a fixed delay)
```

Thus saving you two pins!

Note that the 2.7" Tri-color display works best if you have a reset pin, it really likes being reset before sending data, so we recommend keeping it.

Drawing Bitmaps



You may need a board with more memory such as the Feather M4 or Metro M4 to handle the memory requirements of drawing a bitmap.

Not only can you draw shapes but you can also load images from the SD card, perfect for static images!

The 1.54" display can show a max of 152x152 pixels. Lets use this Blinka bitmap as our demo:



Click here to download blinka.bmp

https://adafru.it/BTa

For the 2.13" display, use this image instead

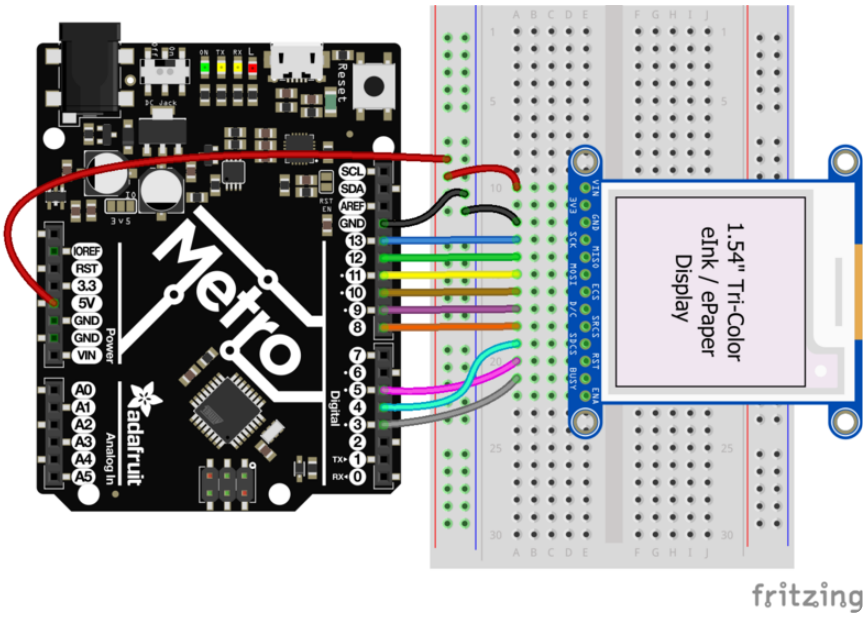


2.13" Sized bitmap

https://adafru.it/EaE

Rename the file blinka.bmp and place into the base directory of a microSD card and insert it into the microSD socket in the breakout.

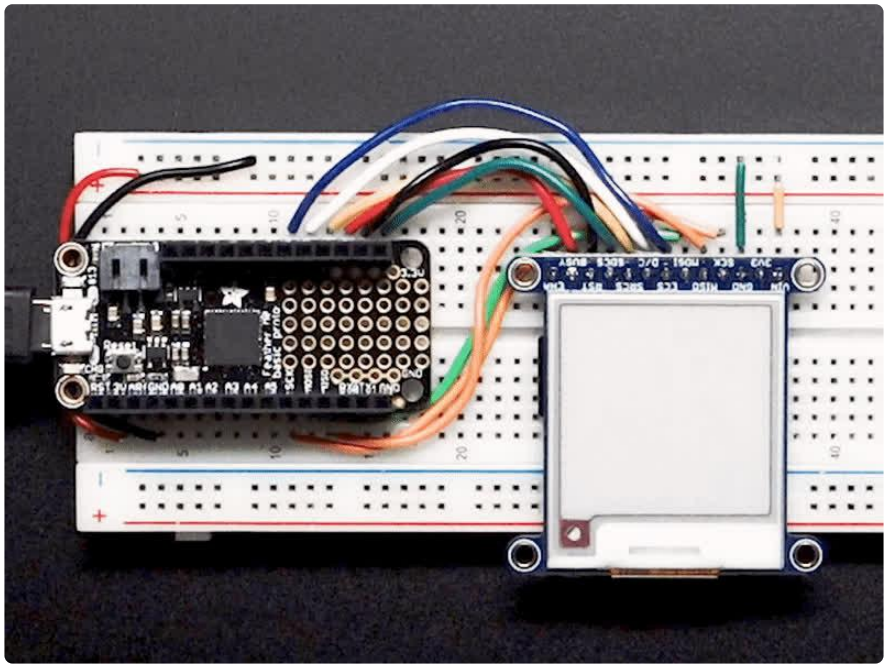
One extra wire is required, for SDCS which is the SD card Chip Select. We'll connect that to pin #4 but you can use any pin.



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

Open the file->examples->Adafruit_ImageReader->EInkBreakouts example

Upload to the upload & you will see Blinka appear!



Arduino Library Documentation

Arduino Library Documentation (https://adafru.it/BST)

Adafruit GFX Library

Adafruit GFX Library (https://adafru.it/doL)

CircuitPython Code

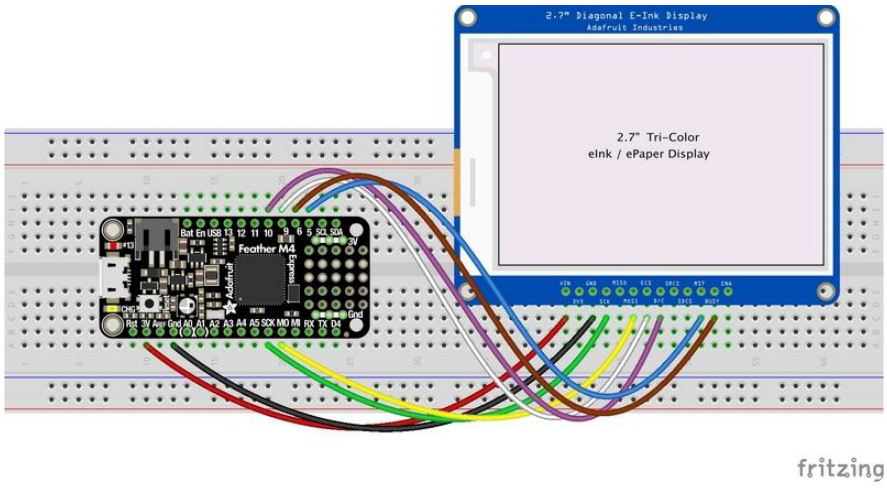
Do not update more than once every 180 seconds or you may permanently damage the display

CircuitPython Microcontroller Wiring

Using eInk displays with `displayio` is really easy. First, wire up your eInk breakout as shown below. All displays have the same pinout, so if your display differs from the one in the Fritzing diagram, you can wire it up the same way.

Breakout Wiring

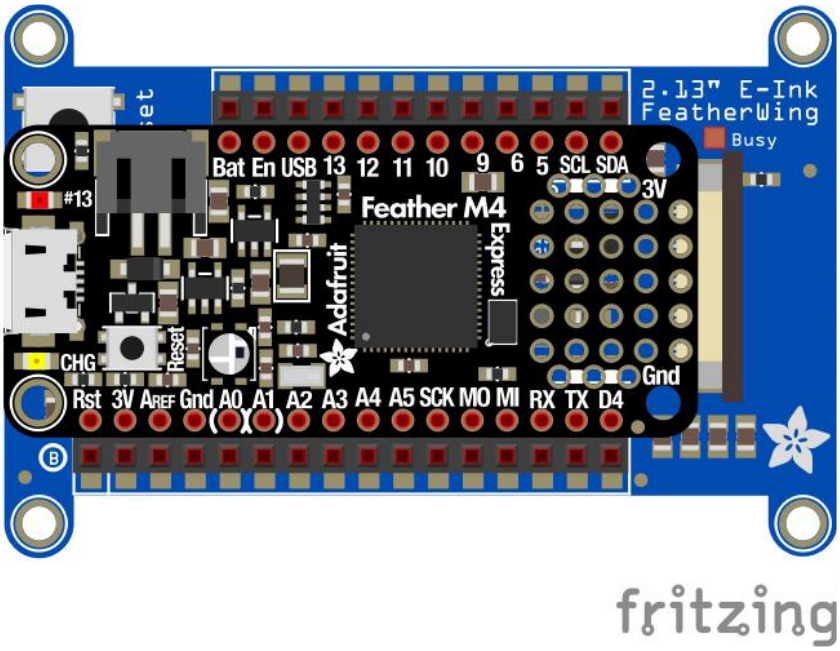
- Feather 3V to display VIN
- Feather GND to display GND
- Feather SCK to display SCK
- Feather MOSI to display MOSI
- Feather D10 to display D/C
- Feather D9 to display ECS
- Feather D6 to display BUSY
- Feather D5 to display RST



Download Fritzing Diagram

https://adafru.it/GdH

FeatherWing Wiring



Download Fritzing Diagram

https://adafru.it/GdI

To use the eInk displays with displayio, you will need to use the latest version of CircuitPython 5.0 and board that can fit `displayio`. See the Support Matrix to determine if `displayio` is available on a given board: https://circuitpython.readthedocs.io/en/latest/shared-bindings/support_matrix.html

CircuitPython eInk displayio Library Installation

To use displayio, you will need to install the appropriate library for your display.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board. You will need CircuitPython version 5.0 or later.

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.

You will need to copy the appropriate displayio driver from the bundle lib folder to a li b folder on your CIRCUITPY drive. The displayio driver contains the initialization codes specific to your display that are needed to for it to work. Since there is more than one driver, you will need to copy the correct file over. Here is a list of each of the displays and the correct driver for that display.

Adafruit_CircuitPython_IL0373

- 1.54” Tri-Color eInk
- 2.13” Tri-Color eInk
- 2.13” Tri-Color eInk FeatherWing
- 2.13” Flexible Monochrome eInk
- 2.9” Flexible Monochrome eInk
- 2.9” Tri-Color eInk

Copy the `adafruit_il0373.mpy` file from the bundle to the lib folder on your CIRC UITPY drive.

Adafruit_CircuitPython_SSD1608

- 1.54” Monochrome eInk

Copy the `adafruit_ssd1608.mpy` file from the bundle to the lib folder on your CIRC UITPY drive.

Adafruit_CircuitPython_SSD1675

- 2.13” Monochrome eInk
- 2.13 Monochrome eInk FeatherWing

Copy the `adafruit_ssd1675.mpy` file from the bundle to the lib folder on your CIRC UITPY drive.

Adafruit_CircuitPython_IL91874

- 2.7” Tri-Color eInk

Copy the `adafruit_il91874.mpy` file from the bundle to the lib folder on your CIRC UITPY drive.

Adafruit_CircuitPython_IL0398

- 4.2” Tri-Color eInk

Copy the `adafruit_il0398.mpy` file from the bundle to the lib folder on your CIRC UITPY drive.

Usage

To show you how to use the eInk with displayio, we'll show you how to draw a bitmap onto it. First start by downloading display-ruler.bmp

Download display-ruler.bmp

https://adafru.it/UIa

Next copy display-ruler.bmp into the root directory of your CIRCUITPY drive.

In the examples folder for your displayio driver, there should be a test for your display, which will all be similar, but include specific parameters such as the width and height of the display. In this example, we will examine the 2.9" Tri-color breakout test. Here is the code in its entirety.

This code is specific to the 2.9" breakout and may not work with other displays! Look at the CircuitPython Bundle for examples specific to your display.

```
import time
import board
import displayio

# Make sure your display driver is uncommented
import adafruit_il0373
#import adafruit_il91874
#import adafruit_ssd1608
#import adafruit_ssd1675
#import adafruit_il0398

# Set based on your display
FLEXIBLE = False
TRICOLOR = True
ROTATION = 90

# Used to ensure the display is free in CircuitPython
displayio.release_displays()

# Define the pins needed for display use
# This pinout is for a Feather M4 and may be different for other boards
# For the Metro/Shield, esc is board.D10 and dc is board.D9
spi = board.SPI()  # Uses SCK and MOSI
ecs = board.D9
```

```
dc = board.D10
rst = board.D5    # set to None for FeatherWing/Shield
busy = board.D6   # set to None for FeatherWing/Shield

if TRICOLOR:
    highlight = 0xff0000 #third color is red (0xff0000)
else:
    highlight = 0x000000

# Create the displayio connection to the display pins
display_bus = displayio.FourWire(spi, command=dc, chip_select=ecs,
                                 reset=rst, baudrate=1000000)

time.sleep(1)  # Wait a bit

# Create the display object
#display = adafruit_ssd1608.SSD1608(display_bus, width=200, height=200,   # 1.54"
HD Monochrome
#display = adafruit_ssd1675.SSD1675(display_bus, width=122, height=250,   # 2.13"
HD Monochrome
#display = adafruit_il91874.IL91874(display_bus, width=264, height=176,   # 2.7"
Tri-color
#display = adafruit_il0398.IL0398(display_bus, width=400, height=300,     # 4.2"
Tri-color
#display = adafruit_il0373.IL0373(display_bus, width=152, height=152,     # 1.54"
Tri-color
#display = adafruit_il0373.IL0373(display_bus, width=296, height=128,
swap_rams=FLEXIBLE, # 2.9" Tri-color OR Flexible Monochrome
display = adafruit_il0373.IL0373(display_bus, width=212, height=104,
swap_rams=FLEXIBLE, # 2.13" Tri-color OR Flexible Monochrome
                                 busy_pin=busy, rotation=ROTATION,
                                 highlight_color=highlight)

# Create a display group for our screen objects
g = displayio.Group()

# Display a ruler graphic from the root directory of the CIRCUITPY drive
f = open("/display-ruler.bmp", "rb")

pic = displayio.OnDiskBitmap(f)
# Create a Tilegrid with the bitmap and put in the displayio group
t = displayio.TileGrid(pic, pixel_shader=displayio.ColorConverter())
g.append(t)

# Place the display group on the screen
display.show(g)

# Refresh the display to have it actually show the image
# NOTE: Do not refresh eInk displays sooner than 180 seconds
display.refresh()
print("refreshed")

time.sleep(180)
```

We start by importing the libraries that we need. In this case we need `time` for adding delays, board the pin definitions, and of course `displayio`.

```
import time
import board
import displayio
```

Next you want to uncomment the import statement for the correct driver for your display. This should match the file you copied over earlier. In our case, the 2.9" uses the `adafruit_il0373` driver, so we can leave it as is.

```
# Make sure your display driver is uncommented
import adafruit_il0373
#import adafruit_il91874
#import adafruit_ssd1608
#import adafruit_ssd1675
#import adafruit_il0398
```

Next we want to set these variables based on your display. If you have a flexible display, you would want to change `FLEXIBLE` to `True`. If you have a monochrome display, you would want to change `TRICOLOR` to `False`. If you would like to change the rotation, you can do that here as well.

```
# Set based on your display
FLEXIBLE = False
TRICOLOR = True
ROTATION = 90
```

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 assign the Pins to use. Note that we are not using the SRAM on the eInk display, so we only need to connect the SCK and MOSI SPI pins. We set the SPI object to the board's SPI with the easy shortcut function `board.SPI()`. We also have the ePaper Display Chip Select and Data/Command pins.

For the breakout boards only, we define the Reset and Busy pins, otherwise you would change these to None in the case of a shield or FeatherWing.

```
spi = board.SPI()  # Uses SCK and MOSI
ecs = board.D9
dc = board.D10
rst = board.D5    # can be None to not use this pin
busy = board.D6   # can be None to not use this pin
```

In the next section, we set the highlight color to either red or black based on whether you have a monochrome or tri-color display. You can leave this alone.

```
if TRICOLOR:
    highlight = 0xff0000 #third color is red (0xff0000)
```

```
else:
    highlight = 0x000000
```

In the next line, we set the display bus to FourWire which makes use of the SPI bus. We pass it the `D/C`, and `CS` pins, which are also usually found on TFT displays and if this is a breakout, we also pass in the `reset` pin.

We set the baudrate to 1 MHz instead of the default 24 MHz because the ePaper displays are not about the speed. They are about the patience of waiting many seconds for them to change and the infrequent updates.

After that, we pause for 1 second. Remember, patience.

```
display_bus = displayio.FourWire(spi, command=epd_dc, chip_select=epd_cs,
                                 reset=epd_reset, baudrate=1000000)
time.sleep(1)
```

Next is the initializer. You will want to uncomment the one appropriate to your display. For the 2.9" display, we would want to comment out the line with 2.13" Tri-color OR Flexible Monochrome next to it and uncomment the line with 2.9" Tri-color OR Flexible Monochrome next to it.

```
# Create the display object
#display = adafruit_ssd1608.SSD1608(display_bus, width=200, height=200,   # 1.54"
HD Monochrome
#display = adafruit_ssd1675.SSD1675(display_bus, width=122, height=250,   # 2.13"
HD Monochrome
#display = adafruit_il91874.IL91874(display_bus, width=264, height=176,   # 2.7"
Tri-color
#display = adafruit_il0398.IL0398(display_bus, width=400, height=300,     # 4.2"
Tri-color
#display = adafruit_il0373.IL0373(display_bus, width=152, height=152,     # 1.54"
Tri-color
#display = adafruit_il0373.IL0373(display_bus, width=296, height=128,
swap_rams=FLEXIBLE, # 2.9" Tri-color OR Flexible Monochrome
display = adafruit_il0373.IL0373(display_bus, width=212, height=104,
swap_rams=FLEXIBLE, # 2.13" Tri-color OR Flexible Monochrome
                                 busy_pin=busy, rotation=ROTATION,
                                 highlight_color=highlight)
```

Next we create a couple of variables including a displayio group and a file handle to the display-ruler.bmp that you placed in your CIRCUITPY root folder. You did do that, right?

```
# Create a display group for our screen objects
g = displayio.Group()

# Display a ruler graphic from the root directory of the CIRCUITPY drive
f = open("/display-ruler.bmp", "rb")
```

Next we take the file handle and read the bitmap data into a `TileGrid` object. We also specify the pixel_shader to `displayio.ColorConverter()` because we want `displayio` to convert the image data into something that will look nice on the eInk. We take the TileGrid object and place it into the group.

```
pic = displayio.OnDiskBitmap(f)
# Create a Tilegrid with the bitmap and put in the displayio group
t = displayio.TileGrid(pic, pixel_shader=displayio.ColorConverter())
g.append(t)
```

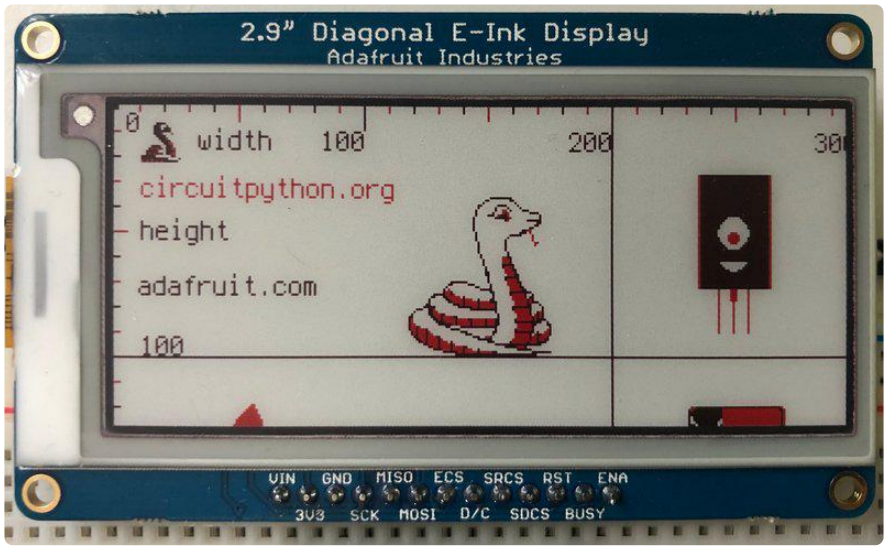
In the next line we tell the display to show everything in the group.

```
display.show(g)
```

Finally, we tell the display to refresh so that everything in memory is written out to the display.

```
display.refresh()
print("refreshed")
```

Your display will look something like this:



After that we tell it to pause for 180 seconds or three minutes before continuing where your display would show the REPL.

```
time.sleep(180)
```

Python Code

Wiring

It's easy to use eInk breakouts with Python and the Adafruit CircuitPython EPD (https://adafru.it/BTd) library. This library allows you to easily write Python code to control the display.

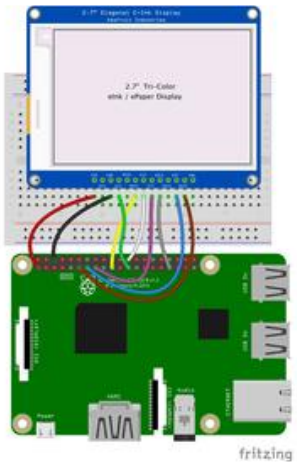
We'll cover how to wire the display to your Raspberry Pi. First assemble your display.

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 eInk. However, this is handy when you want to use the eInk display 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



- Raspberry Pi 3.3 to display VIN
- Raspberry Pi GND to display GND
- Raspberry Pi SCLK to display SCK
- Raspberry Pi MOSI to display MOSI
- Raspberry Pi GPIO CE0 to display ECS
- Raspberry Pi GPIO 22 to display D/C
- Raspberry Pi GPIO 27 to display RST
- Raspberry Pi GPIO 17 to display BUSY

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)!

Python Installation of EPD Library

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

- `sudo pip3 install adafruit-circuitpython-epd`

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`

Download font5x8.bin

This library also requires a font file to run! You can download it below. Before continuing, make sure the folder you are running scripts from contains the font5x8.bin file.

Download font5x8.bin

https://adafru.it/Gfb

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

Some of the examples also use 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`

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

Usage

To demonstrate the usage of the display we'll initialize it and draw some lines from the Python REPL.

Run the following code to import the necessary modules and set up the pin assignments:

```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD

spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
srcs = None
```

If you're using the 1.54" Tri-Color display, run the following code to initialize the display:

```
from adafruit_epd.il0373 import Adafruit_IL0373
display = Adafruit_IL0373(152, 152, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

If you're using the 2.13" Tri-Color display, run the following code to initialize the display:

```
from adafruit_epd.il0373 import Adafruit_IL0373
display = Adafruit_IL0373(104, 212, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

If you're using the 2.9" Tri-Color display, run the following code to initialize the display:

```
from adafruit_epd.il0373 import Adafruit_IL0373
display = Adafruit_IL0373(128, 296, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

If you're using the 2.7" Tri-Color display, run the following code to initialize the display:

```
from adafruit_epd.il91874 import Adafruit_IL91874
display = Adafruit_IL91874(176, 264, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

If you're using the 4.2" Tri-Color display, run the following code to initialize the display:

```
from adafruit_epd.il0398 import Adafruit_IL0398
display = Adafruit_IL0398(400, 300, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

If you're using the 1.54" HD Monochrome display, run the following code to initialize the display:

```
from adafruit_epd.ssd1608 import Adafruit_SSD1608
display = Adafruit_SSD1608(200, 200, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

Tri-Color Example

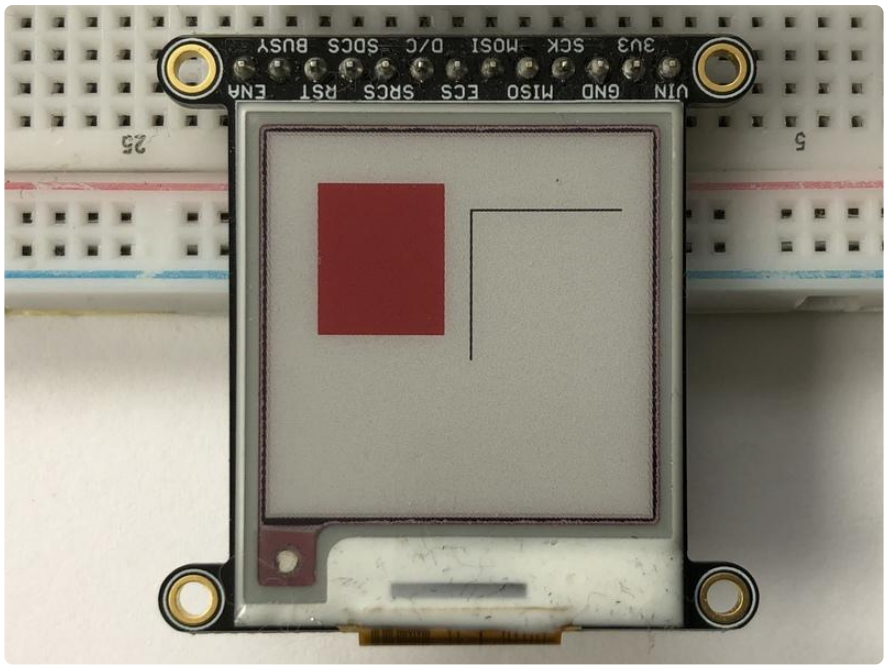
Now we can clear the screens buffer and draw some shapes. Once we're done drawing, we need to tell the screen to update using the `display()` method.

```
display.fill(Adafruit_EPD.WHITE)

display.fill_rect(20, 20, 50, 60, Adafruit_EPD.RED)
display.hline(80, 30, 60, Adafruit_EPD.BLACK)
display.vline(80, 30, 60, Adafruit_EPD.BLACK)

display.display()
```

Your display will look something like this:



Monochrome Example

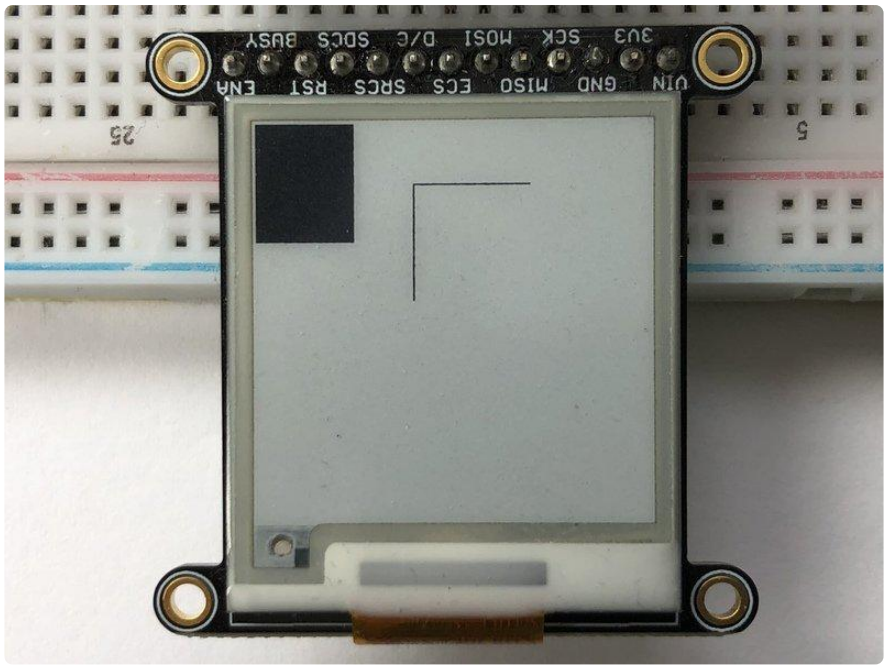
Now we can clear the screens buffer and draw some shapes. Once we're done drawing, we need to tell the screen to update using the `display()` method.

```
display.fill(Adafruit_EPD.WHITE)

display.fill_rect(0, 0, 50, 60, Adafruit_EPD.BLACK)
display.hline(80, 30, 60, Adafruit_EPD.BLACK)
display.vline(80, 30, 60, Adafruit_EPD.BLACK)

display.display()
```

Your eInk display should look similar to the image above, with a black rectangle instead of a red one.



That's all there is to drawing simple shapes with eInk displays and CircuitPython!

Tri-Color Bitmap Example

Here's a complete example of how to display a bitmap image on your display. Note that any .bmp image you want to display must be exactly the size of your display. We will be using the image below on the 1.54" display. Click the button below to download the image and save it as blinka.bmp on your Raspberry Pi.



Click here to download blinka for the 1.54" display

https://adafru.it/BTa

Save the following code to your Raspberry Pi as epd_bitmap.py.

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

import digitalio
```

```
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import


# create the spi device and pins we will need
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.D10)
dc = digitalio.DigitalInOut(board.D9)
srcs = digitalio.DigitalInOut(board.D7)  # can be None to use internal memory
rst = digitalio.DigitalInOut(board.D11)  # can be None to not use this pin
busy = digitalio.DigitalInOut(board.D12)  # can be None to not use this pin

# give them all to our driver
print("Creating display")
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
# display = Adafruit_UC8151D(128, 296,        # 2.9" mono flexible display
# display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, False)
# display.set_color_buffer(1, False)

# IF YOU HAVE A 2.9" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, True)
# display.set_color_buffer(1, True)

display.rotation = 0

FILENAME = "blinka.bmp"


def read_le(s):
    # as of this writting, int.from_bytes does not have LE support, DIY!
    result = 0
    shift = 0
    for byte in bytearray(s):
        result += byte << shift
        shift += 8
    return result


class BMPError(Exception):
    pass
```

```
def display_bitmap(epd, filename):  # pylint: disable=too-many-locals, too-many-
branches
    try:
        f = open(filename, "rb")  # pylint: disable=consider-using-with
    except OSError:
        print("Couldn't open file")
        return

    print("File opened")
    try:
        if f.read(2) != b"BM":  # check signature
            raise BMPError("Not BitMap file")

        bmpFileSize = read_le(f.read(4))
        f.read(4)  # Read & ignore creator bytes

        bmpImageoffset = read_le(f.read(4))  # Start of image data
        headerSize = read_le(f.read(4))
        bmpWidth = read_le(f.read(4))
        bmpHeight = read_le(f.read(4))
        flip = True

        print(
            "Size: %d\nImage offset: %d\nHeader size: %d"
            % (bmpFileSize, bmpImageoffset, headerSize)
        )
        print("Width: %d\nHeight: %d" % (bmpWidth, bmpHeight))

        if read_le(f.read(2)) != 1:
            raise BMPError("Not singleplane")
        bmpDepth = read_le(f.read(2))  # bits per pixel
        print("Bit depth: %d" % (bmpDepth))
        if bmpDepth != 24:
            raise BMPError("Not 24-bit")
        if read_le(f.read(2)) != 0:
            raise BMPError("Compressed file")

        print("Image OK! Drawing...")

        rowSize = (bmpWidth * 3 + 3) & ~3  # 32-bit line boundary

        for row in range(bmpHeight):  # For each scanline...
            if flip:  # Bitmap is stored bottom-to-top order (normal BMP)
                pos = bmpImageoffset + (bmpHeight - 1 - row) * rowSize
            else:  # Bitmap is stored top-to-bottom
                pos = bmpImageoffset + row * rowSize

            # print ("seek to %d" % pos)
            f.seek(pos)
            rowdata = f.read(3 * bmpWidth)
            for col in range(bmpWidth):
                b, g, r = rowdata[3 * col : 3 * col + 3]  # BMP files store RGB in
BGR
                if r < 0x80 and g < 0x80 and b < 0x80:
                    epd.pixel(col, row, Adafruit_EPD.BLACK)
                elif r >= 0x80 and g >= 0x80 and b >= 0x80:
                    pass  # epd.pixel(row, col, Adafruit_EPD.WHITE)
                elif r >= 0x80:
                    epd.pixel(col, row, Adafruit_EPD.RED)
    except OSError:
        print("Couldn't read file")
    except BMPError as e:
        print("Failed to parse BMP: " + e.args[0])
    finally:
        f.close()
    print("Finished drawing")
```

```
# clear the buffer
display.fill(Adafruit_EPD.WHITE)
display_bitmap(display, FILENAME)
display.display()
```

Before running it, we need to change a few pin definitions though. Find the section of code that looks like this:

```
ecs = digitalio.DigitalInOut(board.D10)
dc = digitalio.DigitalInOut(board.D9)
srcs = digitalio.DigitalInOut(board.D7)    # can be None to use internal memory
rst = digitalio.DigitalInOut(board.D11)    # can be None to not use this pin
busy = digitalio.DigitalInOut(board.D12)   # can be None to not use this pin
```

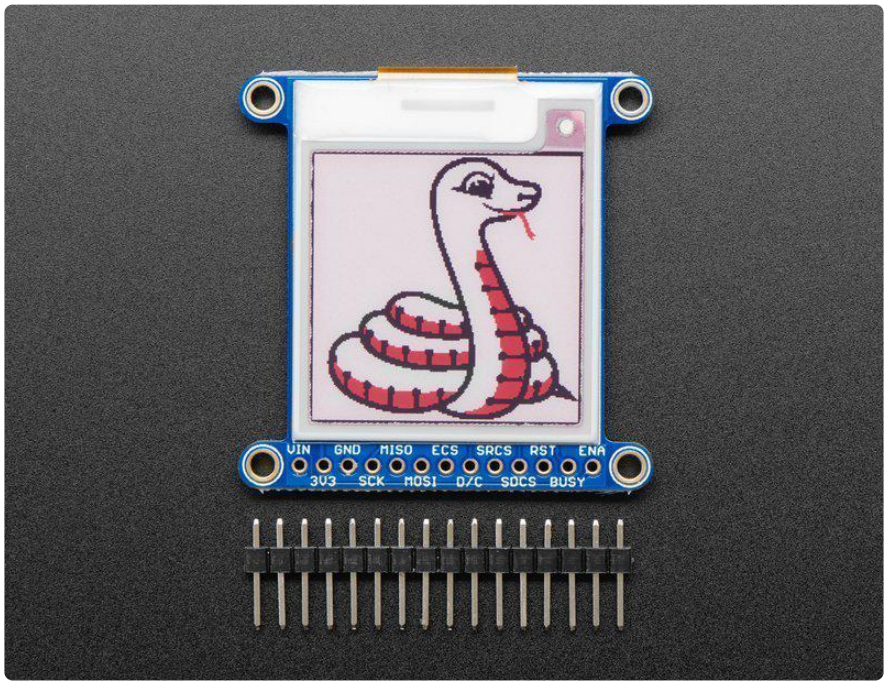
Change the pins to the following to match the wiring on the Raspberry Pi:

```
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
srcs = None
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
```

Now go to the command prompt on your Raspberry Pi and run the script with the following command:

```
python3 epd_bitmap.py
```

After a few seconds, your display should show this image:



Full Example Code

Here is the full example code.

To run the code sample below, you will need to change the pins the same way as you did in the Tri-color Bitmap Example.

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

import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import

# create the spi device and pins we will need
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.D12)
dc = digitalio.DigitalInOut(board.D11)
srcs = digitalio.DigitalInOut(board.D10)  # can be None to use internal memory
rst = digitalio.DigitalInOut(board.D9)  # can be None to not use this pin
busy = digitalio.DigitalInOut(board.D5)  # can be None to not use this pin

# give them all to our drivers
print("Creating display")
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
# display = Adafruit_UC8151D(128, 296,        # 2.9" mono flexible display
# display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, False)
# display.set_color_buffer(1, False)

# IF YOU HAVE A 2.9" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, True)
# display.set_color_buffer(1, True)
```

```
display.rotation = 1

# clear the buffer
print("Clear buffer")
display.fill(Adafruit_EPD.WHITE)
display.pixel(10, 100, Adafruit_EPD.BLACK)

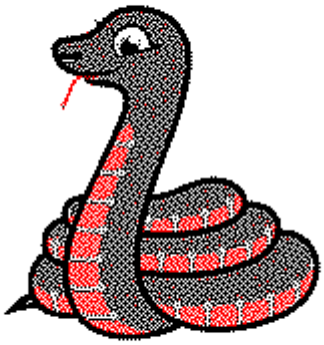
print("Draw Rectangles")
display.fill_rect(5, 5, 10, 10, Adafruit_EPD.RED)
display.rect(0, 0, 20, 30, Adafruit_EPD.BLACK)

print("Draw lines")
display.line(0, 0, display.width - 1, display.height - 1, Adafruit_EPD.BLACK)
display.line(0, display.height - 1, display.width - 1, 0, Adafruit_EPD.RED)

print("Draw text")
display.text("hello world", 25, 10, Adafruit_EPD.BLACK)
display.display()
```

Image Drawing with Pillow

In this image, we will use Pillow to resize and crop the image automatically and draw it the the ePaper Display. Pillow is really powerful and with it you can open and render additional file formats such as PNG or JPG. Let's start with downloading a PNG of blinka that has been adjusted down to 3 colors so it prints nicely on an ePaper Display. We are using PNG for this because it is a lossless format and won't introduce unexpected colors in.



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

```
# SPDX-FileCopyrightText: 2019 Melissa LeBlanc-Williams for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
Image resizing and drawing using the Pillow Library. For the image, check out the
associated Adafruit Learn guide at:
https://learn.adafruit.com/adafruit-eink-display-breakouts/python-code

"""

import digitalio
import busio
import board
from PIL import Image
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import


# create the spi device and pins we will need
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
srcs = None
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)

# give them all to our driver
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color or mono display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
# display = Adafruit_UC8151D(128, 296,        # 2.9" mono flexible display
# display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, False)
# display.set_color_buffer(1, False)

# IF YOU HAVE A 2.9" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, True)
# display.set_color_buffer(1, True)

display.rotation = 1

image = Image.open("blinka.png")

# Scale the image to the smaller screen dimension
image_ratio = image.width / image.height
screen_ratio = display.width / display.height
```

```
if screen_ratio < image_ratio:
    scaled_width = image.width * display.height // image.height
    scaled_height = display.height
else:
    scaled_width = display.width
    scaled_height = image.height * display.width // image.width
image = image.resize((scaled_width, scaled_height), Image.BICUBIC)

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

# Convert to Monochrome and Add dithering
# image = image.convert("1").convert("L")

# Display image.
display.image(image)
display.display()
```

So we start with our usual imports including a couple of Pillow modules and the ePaper display drivers.

```
import digitalio
import busio
import board
from PIL import Image, ImageDraw
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
```

That is followed by initializing the SPI bus and defining a few pins here. The reason we chose these is because they allow you to use the same code with the EPD bonnets if you chose to do so.

```
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
srcs = None
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
```

We wanted to make these examples work on as many displays as possible with very few changes. The 2.13" Tri-color display is selected by default. For other displays, go ahead and comment out the following lines:

```
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
```

and uncomment the line appropriate for your display.

```
#display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
#display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
#display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
#display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
#display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
#display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy
)
```

Uncomment the next two lines if you have a flexible display. This tells the library to change a couple of settings so that it is writing the correct colors to the correct places.

```
# IF YOU HAVE A FLEXIBLE DISPLAY (2.13" or 2.9") uncomment these lines!
#display.set_black_buffer(1, False)
#display.set_color_buffer(1, False)
```

Next we tall the display the rotation setting we want to use. This can be a value between `0-3`.

```
display.rotation = 1
```

Next we open the Blinka image, which we've named blinka.png, 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.png")
```

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.

Nearest actually gives a little better result with the Tri-color eInks, but loses detail with displaying a color image on the monochrome display, so we decided to go with the best balance.

```
image_ratio = image.width / image.height
screen_ratio = display.width / display.height
if screen_ratio &lt; image_ratio:
    scaled_width = image.width * display.height // image.height
    scaled_height = display.height
else:
    scaled_width = display.width
    scaled_height = image.height * display.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.

```
x = scaled_width // 2 - display.width // 2
y = scaled_height // 2 - display.height // 2
image = image.crop((x, y, x + display.width, y + display.height))
```

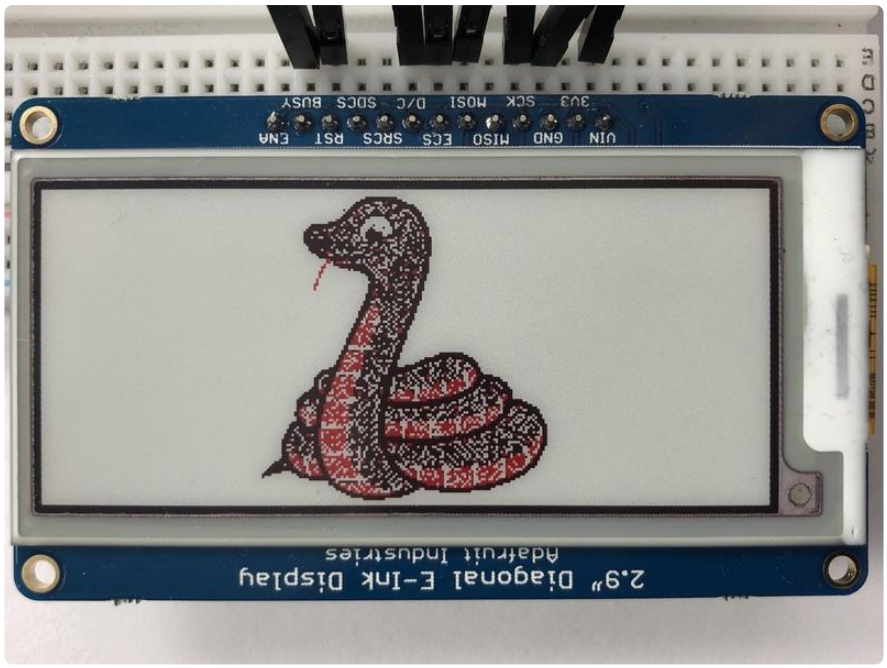
Finally, we take our `image`, draw it to the frame buffer and `display` it. At this point, the image should have the exact same dimensions at the display and fill it completely.

```
display.image(image)
display.display()
```

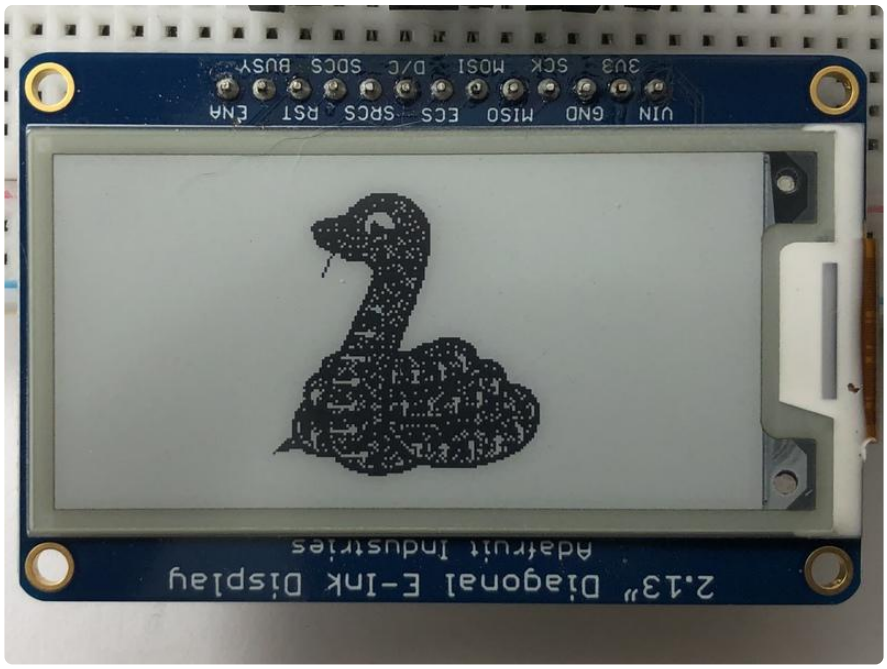
Now go to the command prompt on your Raspberry Pi and run the script with the following command:

```
python3 epd_pillow_image.py
```

After a few seconds, your display should show this image:



Here's what it looks like on a monochrome display:



Drawing Shapes and Text with Pillow

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. Go ahead and copy it onto your Raspberry Pi and save it as epd_pillow_demo.py. Here's the code for that.

```
# SPDX-FileCopyrightText: 2019 Melissa LeBlanc-Williams for Adafruit Industries
# SPDX-License-Identifier: MIT
```

```
"""
ePaper Display Shapes and Text demo using the Pillow Library.

"""

import digitalio
import busio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import

# First define some color constants
WHITE = (0xFF, 0xFF, 0xFF)
BLACK = (0x00, 0x00, 0x00)
RED = (0xFF, 0x00, 0x00)

# Next define some constants to allow easy resizing of shapes and colors
BORDER = 20
FONTSIZE = 24
BACKGROUND_COLOR = BLACK
FOREGROUND_COLOR = WHITE
TEXT_COLOR = RED

# create the spi device and pins we will need
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
srcs = None
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)

# give them all to our driver
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color or mono display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
# display = Adafruit_UC8151D(128, 296,        # 2.9" mono flexible display
# display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, False)
# display.set_color_buffer(1, False)

# IF YOU HAVE A 2.9" FLEXIBLE DISPLAY uncomment these lines!
# display.set_black_buffer(1, True)
# display.set_color_buffer(1, True)

display.rotation = 1
```

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

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

# Draw a filled box as the background
draw.rectangle((0, 0, display.width - 1, display.height - 1), fill=BACKGROUND_COLOR)

# Draw a smaller inner foreground rectangle
draw.rectangle(
    (BORDER, BORDER, display.width - BORDER - 1, display.height - BORDER - 1),
    fill=FOREGROUND_COLOR,
)

# 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(
    (display.width // 2 - font_width // 2, display.height // 2 - font_height // 2),
    text,
    font=font,
    fill=TEXT_COLOR,
)

# Display image.
display.image(image)
display.display()
```

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

```
import digitalio
import busio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
```

Next we define some colors that can be used with Pillow.

```
WHITE = (0xFF, 0xFF, 0xFF)
BLACK = (0x00, 0x00, 0x00)
RED = (0xFF, 0x00, 0x00)
```

After that, we create some parameters that are easy to change. If you had a smaller display for instance, you could reduce the `FONTSIZE` and `BORDER` parameters. 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. You could play around with

the colors as well. One thing to note is that on monochrome displays, the `RED` will show up as `BLACK`.

```
BORDER = 20
FONTSIZE = 24
BACKGROUND_COLOR = BLACK
FOREGROUND_COLOR = WHITE
TEXT_COLOR = RED
```

After that, the initializer and rotation sections are exactly the same as in the previous example. If you have are using a different display than the 2.13" Tri-color, go ahead and adjust your initializer as explained in the previous example. After that, 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.

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

draw = ImageDraw.Draw(image)
```

Next we clear whatever is on the screen by drawing a rectangle using the `BACKGROUND_COLOR` that takes up the full screen.

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

Next we will draw an inner rectangle using the `FOREGROUND_COLOR`. We use the `BORDER` parameter to calculate the size and position that we want to draw the rectangle.

```
draw.rectangle((BORDER, BORDER, display.width - BORDER - 1, display.height - BORDER
- 1),
               fill=FOREGROUND_COLOR)
```

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.

```
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.

```
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
```

```
draw.text((display.width//2 - font_width//2, display.height//2 - font_height//2),
          text, font=font, fill=TEXT_COLOR)
```

Finally, just like before, we display the image.

```
display.image(image)
display.display()
```

Now go to the command prompt on your Raspberry Pi and run the script with the following command:

```
python3 epd_pillow_demo.py
```

After a few seconds, your display should show this image:



Python Docs

Python Docs (https://adafru.it/C4z)

2.9" Grayscale eInk FeatherWing



Easy e-paper comes to your Feather with this breakout that's designed to make it a breeze to add a monochrome eInk display. Chances are you've seen one of those new-fangled 'e-readers' like the Kindle or Nook. They have gigantic electronic paper 'static' displays - that means the image stays on the display even when power is completely disconnected. The image is also high contrast and very daylight readable. It really does look just like printed paper!

We've liked these displays for a long time, so wouldn't a custom e-paper FeatherWing with buttons make a ton of sense? This 'Wing is tested to work with all of our Feathers, from the ESP8266 to the M0. It has built in memory buffering so it can work with chips as small as the '32u4 and '328. It does use a lot of pins: the 3 SPI pins, and up to 4 control pins to manage the SD card slot and SRAM. Plus 3 optional buttons are available for Feathers with available pins.

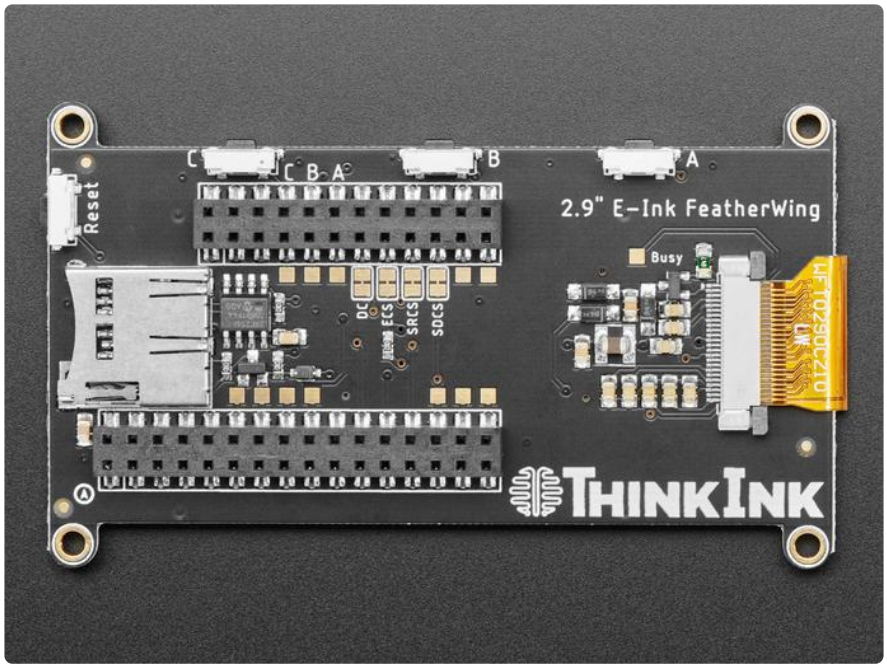
The FeatherWing sports a 2.9" grayscale display with 296x128 pixels. Each pixel can be white, light gray, dark gray or black. Compared to 'tri-color' displays with a red pigment, this display takes a lot less time to update, only about a second instead of 15 seconds!

Using our CircuitPython or Arduino libraries, you can create a 'frame buffer' with what pixels you want to have activated and then write that out to the display. Most simple breakouts leave it at that. But if you do the math, 296 x 128 pixels x 2-bits-per-pixel = 9.5 KBytes. Which won't fit into many microcontroller memories. Heck, even if you do have 32KB of RAM, why waste 9KB?

So we did you a favor and tossed a small SRAM chip on the back. This chip shares the SPI port the eInk display uses, so you only need one extra pin. And, no more frame-buffering! You can use the SRAM to set up whatever you want to display, then shuffle data from SRAM to eInk when you're ready. The library we wrote does all the work for you (https://adafru.it/BRK), you can just interface with it as if it were an Adafruit_GFX compatible display (https://adafru.it/BRK).

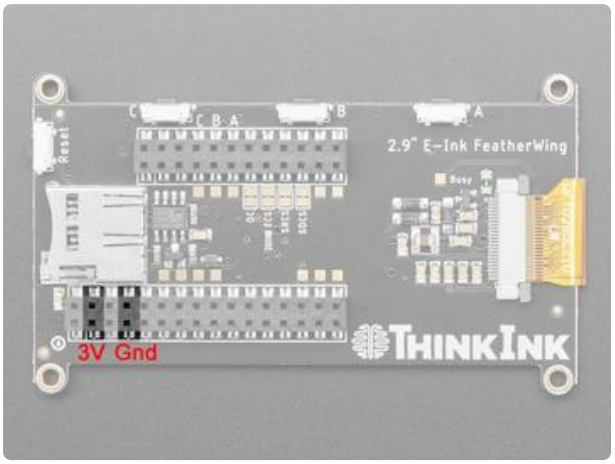
We even tossed on a MicroSD socket so you can store images, text files, whatever you like to display. Comes assembled and tested with socket headers that you can plug your Feather right into, no soldering required!

Pinouts



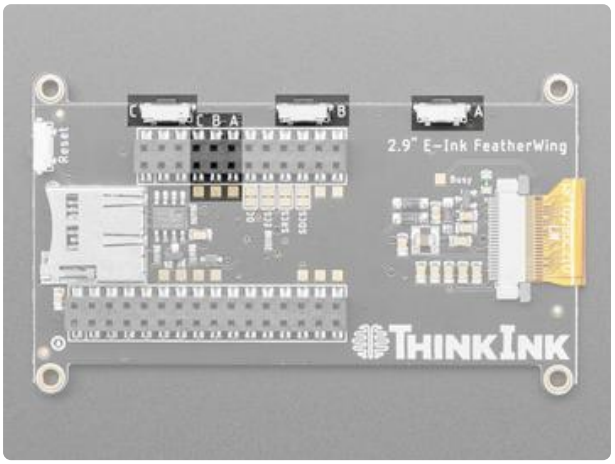
This e-Paper display uses SPI to receive image data. Since the display is SPI, it was easy to add two more SPI devices to share the bus - an SPI SRAM chip and SPI-driven SD card holder. There's quite a few pins and a variety of possible combinations for control depending on your needs.

Power Pins



- 3V - this is the power pin, and connects to the Feather 3V power supply output
- GND - this is the power and signal ground pin

Buttons

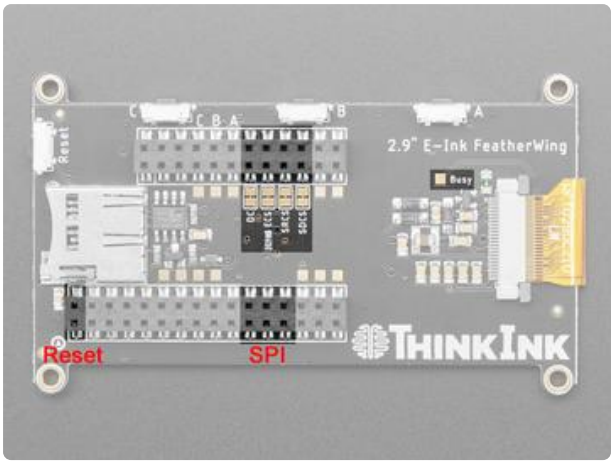


The buttons are connected to Digital Input pins. Pressing the button brings the pin low. There are no pull-up resistors connected, so you will need to enable those in software.

The numbers of the pins these correspond to will differ from board to board. However, on 32u4/328p/M0/M4/nRF52840 and many other boards you will see the following connections:

- A Button - this is connected to D11 of the feather
- B Button - this is connected to D12 of the feather
- C Button - this is connected to D13 of the feather

Data Control Pins



The FeatherWing uses SPI and some control pins for reading/writing data from the SD and then sending data to the E-Ink display

SPI data pins

- SCK - this is the SPI clock input pin, required for e-Ink, SRAM and SD card
- MISO - this is the SPI Microcontroller In Serial Out pin, its used for the SD card and SRAM. It isn't used for the e-Ink display which is write-only, however you'll likely be using the SRAM to buffer the display.
- MOSI - this is the SPI Microcontroller Out Serial In pin, it is used to send data from the microcontroller to the SD card, SRAM and e-Ink display

SPI control pins

- ECS - this is the E-Ink Chip Select, required for controlling the display
- SRCS - this is the SRAM Chip Select, required for communicating with the onboard RAM chip.
- SDCS - this is the SD card Chip Select, required for communicating with the onboard SD card holder. You can leave this disconnected if you aren't going to access SD cards

Other control pins

- D/C - this is the e-Ink Data/Command pin, required for controlling the display
- RST - this is connected to the microcontroller reset circuitry, so you don't need to worry about it in software.

Optional control pads:

- BUSY - this is the e-Ink busy detect pad and is optional if you don't want to connect the pin (in which case the code will just wait an approximate number of seconds). To use it, you will need to run a wire over to it.

SD CS, SRAM CS, EINK CS and DC are in order after the two I2C pins. The numbers of the pins these correspond to will differ from board to board. However, on 32u4/328p/M0/M4/nRF52840 and many other boards you will see the following connections

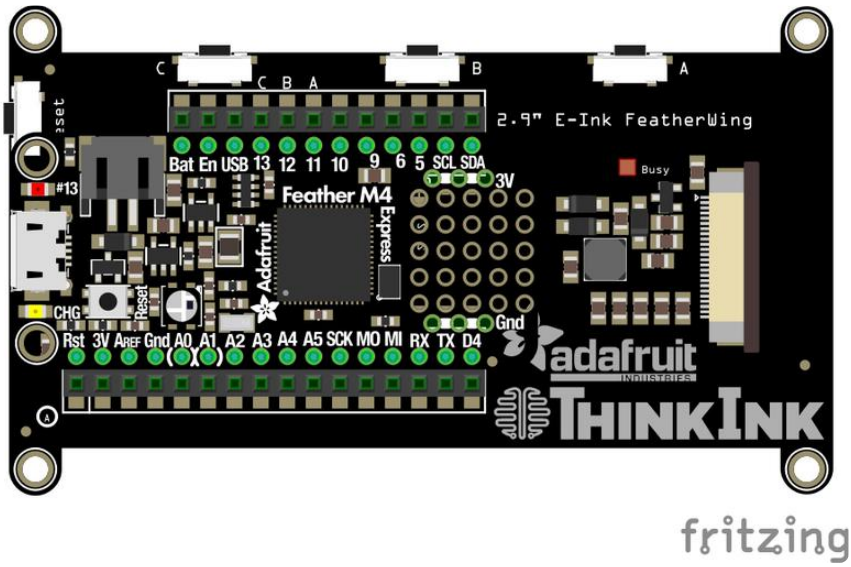
- SD CS to Pin D5
- SRAM CS to Pin D6
- EINK CS to Pin D9
- EINK DC to Pin D10

If you do not plan to use the SD card, you can cut the trace to SD CS.
Likewise if you do not plan to use the built in SRAM, say because you're in CircuitPython or if you are using Arduino and have a lot of RAM in your controller, you can cut the trace for SRAM CS.

Wiring

FeatherWing Connection

FeatherWing usage is easy, simply plug your Feather into the Wing



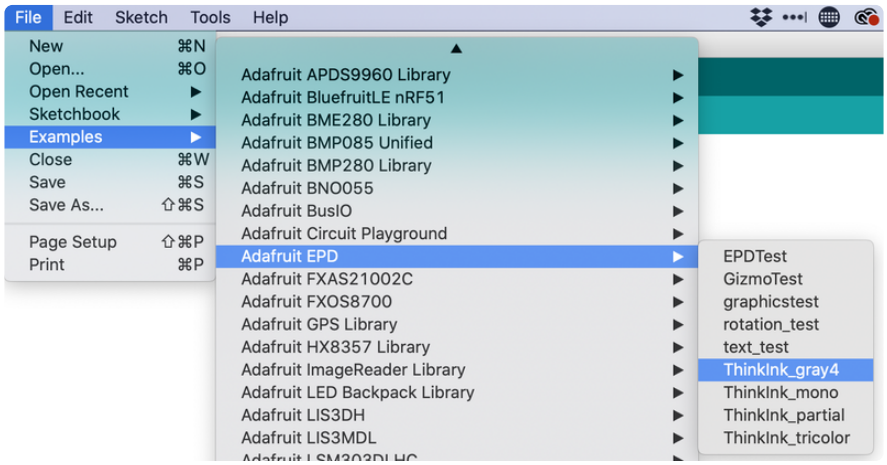
Download Fritzing Diagram

https://adafru.it/OdD

Arduino Usage

For this display we will run a 4-level grayscale demo

Open up File→Examples→Adafruit_EPD→ThinkInk_gray4



Configure Pins

At the top of the sketch find the lines that look like:

```
#define EPD_DC      10
#define EPD_CS      9
#define SRAM_CS     6
#define EPD_RESET   8 // can set to -1 and share with microcontroller Reset!
#define EPD_BUSY    7 // can set to -1 to not use a pin (will wait a fixed delay)
```

Change both `EPD_RESET` and `EPD_BUSY` to -1 since neither of these lines are connected on the FeatherWing.

You'll also need to update the CS and DC pins if you're not using a Feather M0 or M4. For example, on an ESP8266

```
#define SRAM_CS 16
#define EPD_CS 0
#define EPD_DC 15
```

Configure Display Size

Find the part of the script where you can pick which display is going to be used.

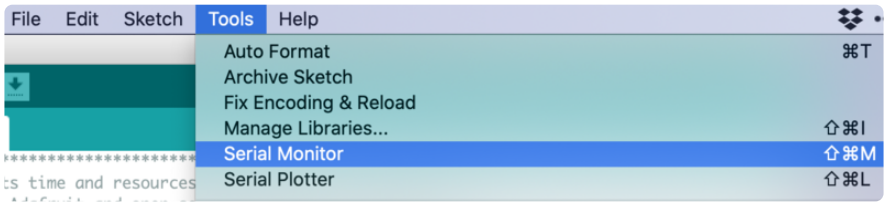
For the 2.9" Grayscale Featherwing, you will need to have `ThinkInk_290_Grayscale4_T5` uncommented, and any other type commented.

Fine this line and make sure it is not commented out:

```
// 2.9" Grayscale Featherwing or Breakout:
ThinkInk_290_Grayscale4_T5 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);
```

Upload Sketch

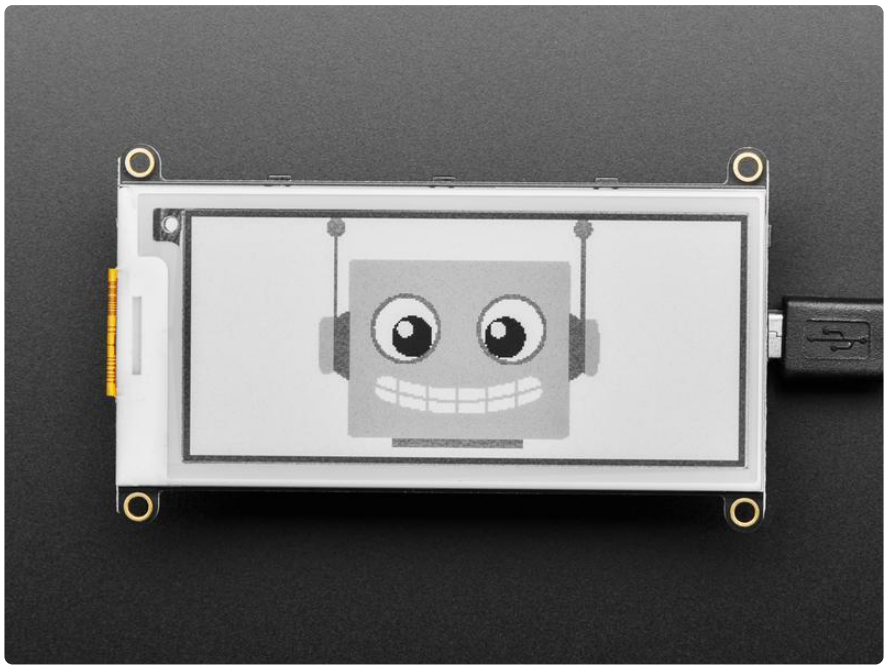
Go ahead and upload the sketch to your board. Once it is done uploading, open the S erial Monitor.



The display should start running a series of monochrome and grayscale tests.



Arduino Bitmaps



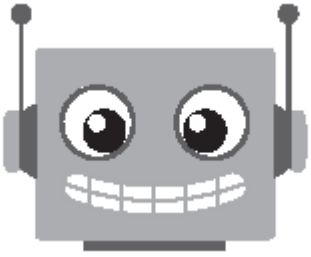
Not only can you draw shapes but you can also load images from the SD card, perfect for static images!

The 2.9" Grayscale display can show a max of 296x128 pixels. Lets use these three bitmaps for our demo:



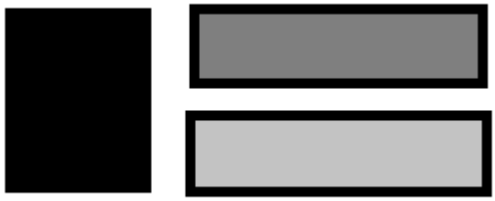
Download panda_head.bmp

https://adafru.it/OdE



Download adabot_head.bmp

https://adafru.it/OdF



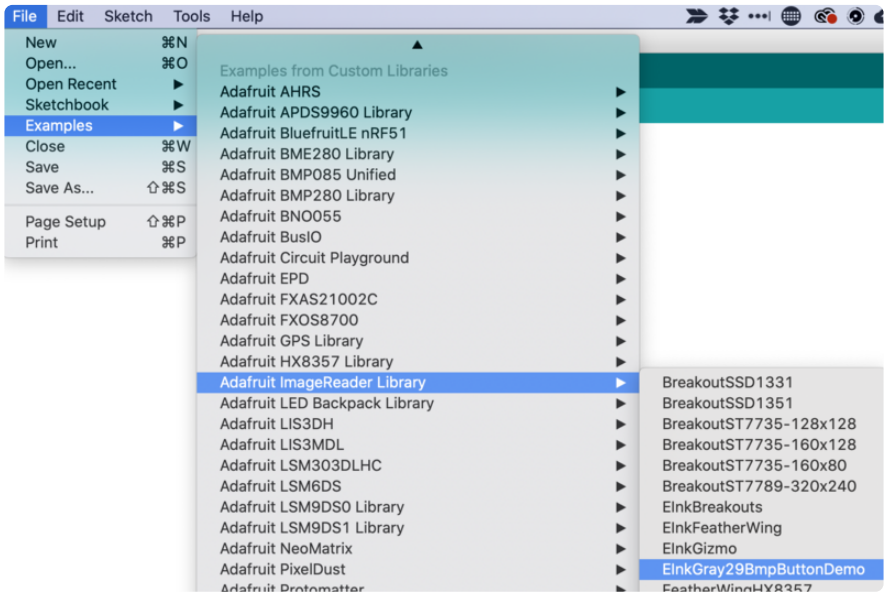
Download 29gray4.bmp

https://adafru.it/OdG

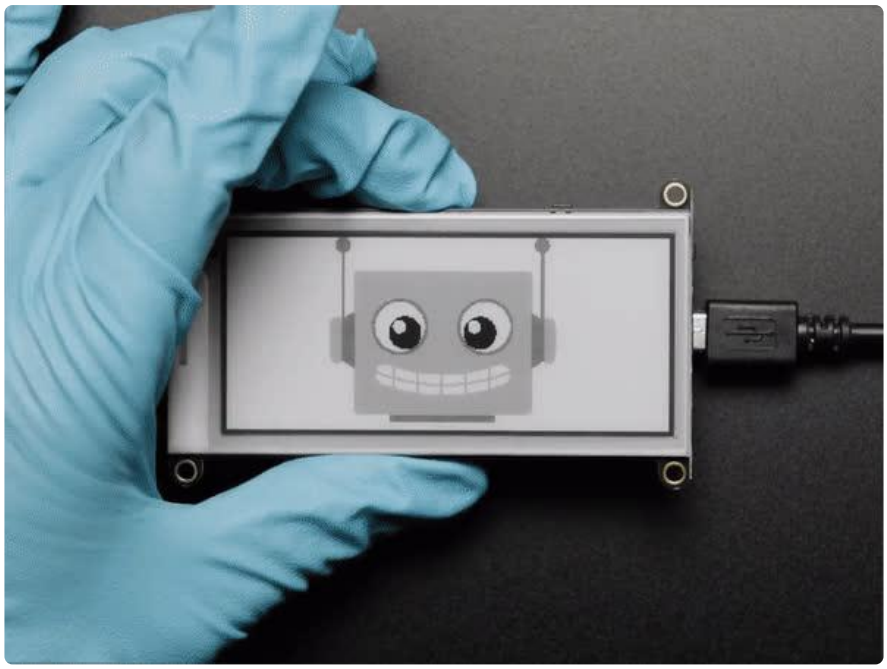
Rename the files to panda_head.bmp, adabot_head.bmp, and 29gray4.bmp and place them into the base directory of a microSD card and insert it into the microSD socket in the breakout.

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

Open the file->examples->Adafruit_ImageReader->EInkGray29BmpButtonDemo example



Upload to your board and press the buttons on top. You should see a different image appear for each button.



CircuitPython Code

CircuitPython eInk displayio Library Installation

To use displayio, you will need to install the appropriate library for your display.

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

To use the eInk displays with displayio, you will need to use the absolute latest version of CircuitPython and a board that can fit `displayio`. See the Support Matrix to determine if `displayio` is available on a given board: https://circuitpython.readthedocs.io/en/latest/shared-bindings/support_matrix.html

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.

You will need to copy the appropriate displayio driver from the bundle lib folder to a li b folder on your CIRCUITPY drive. The displayio driver contains the initialization codes specific to your display that are needed to for it to work. Since there is more than one driver, you will need to copy the correct file over. Here is a list of each of the displays and the correct driver for that display.

When downloading CircuitPython, for Grayscale support, you will need to choose Absolute Newest, choose your language, and then download the top-most link.

Adafruit_CircuitPython_IL0373

This display uses the Adafruit_CircuitPython_ILI0373 library. Copy the `adafruit_il0373.mpy` file from the bundle to the lib folder on your CIRCUITPY drive. You will want a recent version grayscale support was first added in version 1.3.1.

Usage

To show you how to use the eInk with displayio, we'll show you how to draw a bitmap onto it. First start by downloading display-ruler.bmp

Download display-ruler.bmp

https://adafru.it/Ula

Next copy display-ruler.bmp into the root directory of your CIRCUITPY drive.

In the examples folder for your displayio driver, there should be a test for your display which we have listed here:

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

"""Simple test script for 2.9" 296x128 grayscale display.

Supported products:
  * Adafruit 2.9" Grayscale
    * https://www.adafruit.com/product/4777
  """

import time
import busio
import board
import displayio
import adafruit_il0373

displayio.release_displays()

# This pinout works on a Feather M4 and may need to be altered for other boards.
spi = busio.SPI(board.SCK, board.MOSI)  # Uses SCK and MOSI
epd_cs = board.D9
epd_dc = board.D10

display_bus = displayio.FourWire(
    spi, command=epd_dc, chip_select=epd_cs, baudrate=1000000
)
time.sleep(1)

display = adafruit_il0373.IL0373(
```

```
    display_bus,
    width=296,
    height=128,
    rotation=270,
    black_bits_inverted=False,
    color_bits_inverted=False,
    grayscale=True,
    refresh_time=1,
)

g = displayio.Group()

with open("/display-ruler.bmp", "rb") as f:
    pic = displayio.OnDiskBitmap(f)
    # CircuitPython 6 & 7 compatible
    t = displayio.TileGrid(
        pic, pixel_shader=getattr(pic, "pixel_shader", displayio.ColorConverter())
    )
    # CircuitPython 7 compatible only
    # t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
    g.append(t)

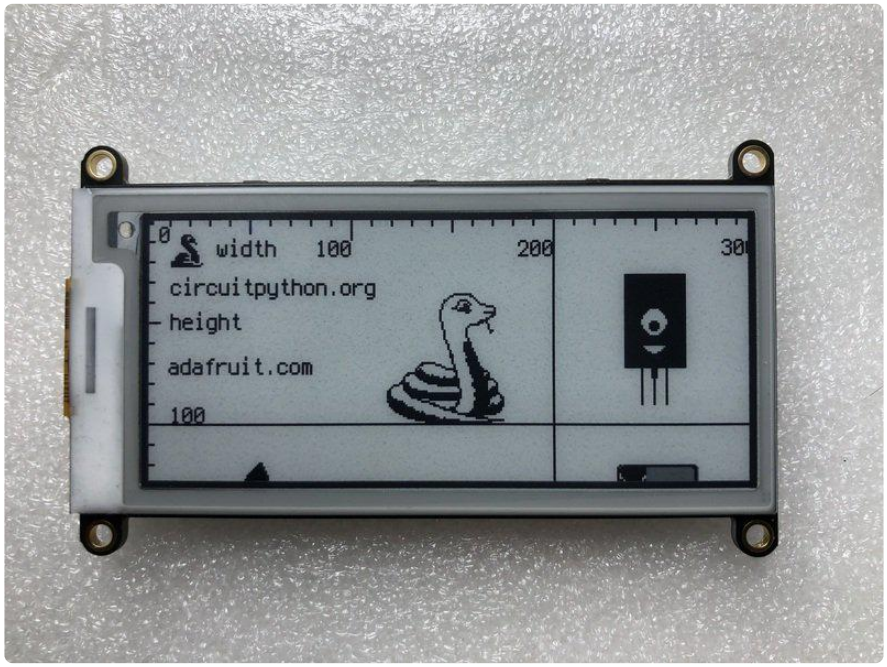
    display.show(g)

    display.refresh()

    print("refreshed")

    time.sleep(120)
```

Save it to your CIRCUITPY drive as code.py and it should automatically run. Your display will look something like this:

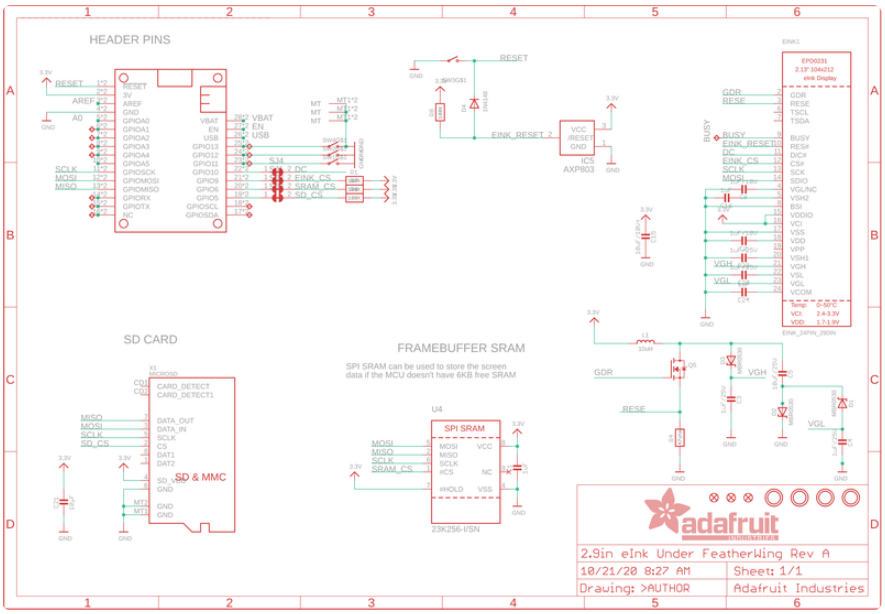


Downloads

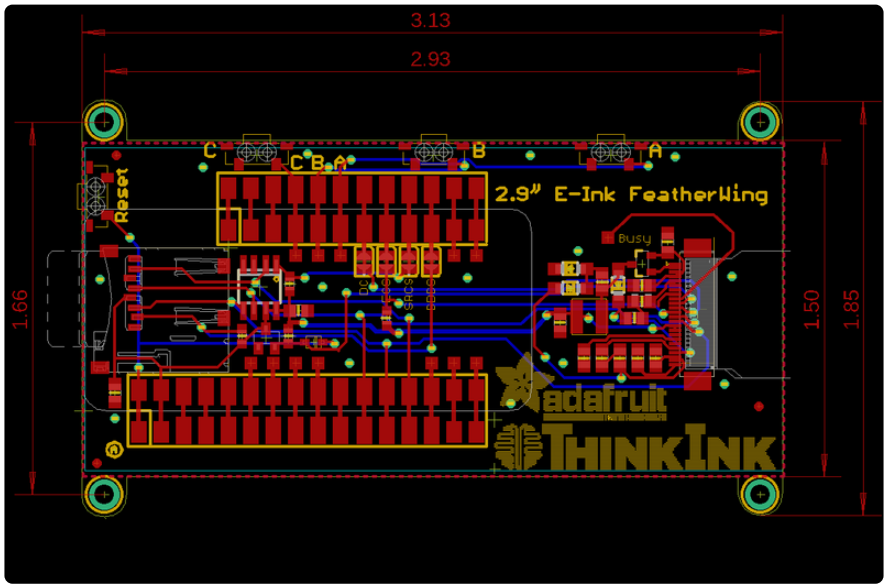
Files

- Fritzing object in Adafruit Fritzing Library (https://adafru.it/Pcq)
- IL0376F E-Ink interface chip datasheet (https://adafru.it/BRW)
- PCB Files on GitHub (https://adafru.it/BRX)

Schematic



Fab Print



2.9" Tri-Color eInk



This breakout has a 2.9" tri-color (red, black, and white) display. It has 296x128 black and red ink pixels and a white-ish background. Using our CircuitPython or Arduino libraries, you can create a 'frame buffer' with what pixels you want to have activated and then write that out to the display. Most simple breakouts leave it at that. But if you do the math, 296 x 128 pixels x 2 colors = 9.5 KBytes. Which won't fit into many microcontroller memories. Heck, even if you do have 32KB of RAM, why waste 10KB?

So we did you a favor and tossed a small SRAM chip on the back. This chip shares the SPI port the eInk display uses, so you only need one extra pin. And, no more frame-buffering! You can use the SRAM to set up whatever you want to display, then shuffle data from SRAM to eInk when you're ready. The library we wrote does all the work for you (https://adafru.it/BRK), you can just interface with it as if it were an Adafruit_GFX compatible display (https://adafru.it/BRK).

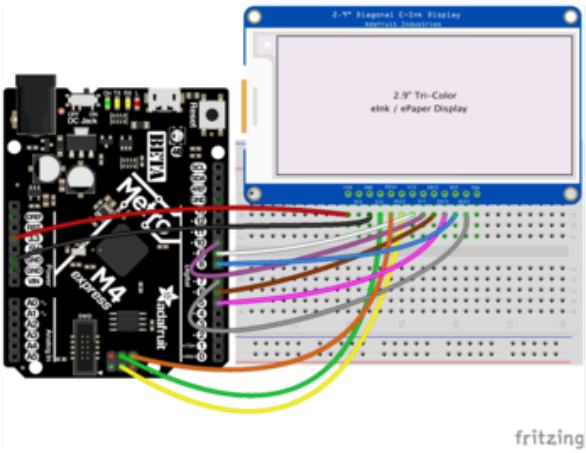
For ultra-low power usages, the onboard 3.3V regulator has the Enable pin brought out so you can shut down the power to the SRAM, MicroSD and display.

We even tossed on a MicroSD socket so you can store images, text files, whatever you like to display. Everything is 3 or 5V logic safe so you can use it with any and all microcontrollers.

Wiring

Breakout Wiring

Wiring up the display in SPI mode is pretty easy as there's not that many pins! We'll be using hardware SPI, but you can also use software SPI (any pins) later.



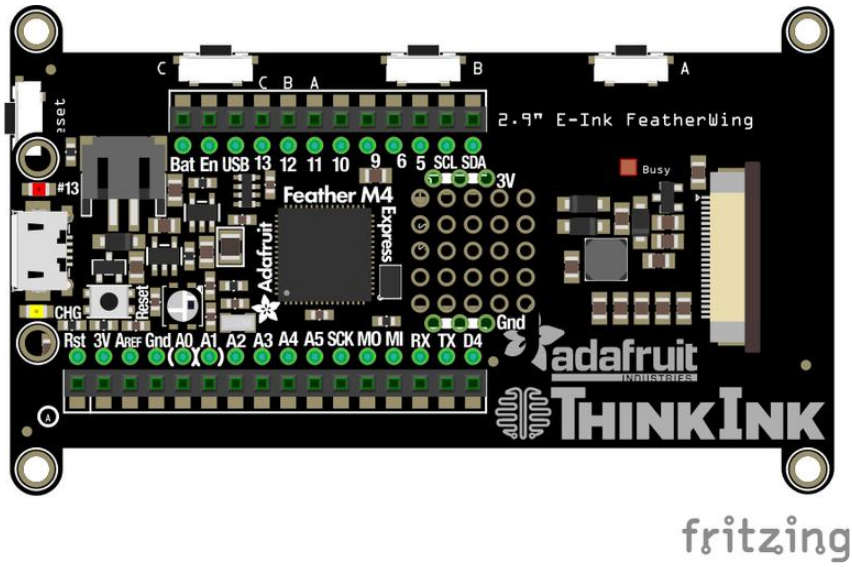
- Vin connects to the microcontroller board's 5V or 3.3V power supply pin
- GND connects to ground
- CLK connects to SPI clock. It's easiest to connect it to pin 3 of the ICSP header.
- MOSI connects to SPI MOSI. It's easiest to connect it to pin 4 of the ICSP header.
- MISO connects to SPI MISO. It's easiest to connect it to pin 1 of the ICSP header.
- ECS connects to our e-Ink Chip Select pin. We'll be using Digital 9
- D/C connects to our e-Ink data/ command select pin. We'll be using Digital 10.
- SRCS connects to our SRAM Chip Select pin. We'll be using Digital 6
- RST connects to our e-Ink reset pin. We'll be using Digital 8.
- BUSY connects to our e-Ink busy pin. We'll be using Digital 7.
- SDCS connects to our SD Card Chip Select pin. We'll be using Digital 5

Download Fritzing Diagram

https://adafru.it/Ofg

FeatherWing Connection

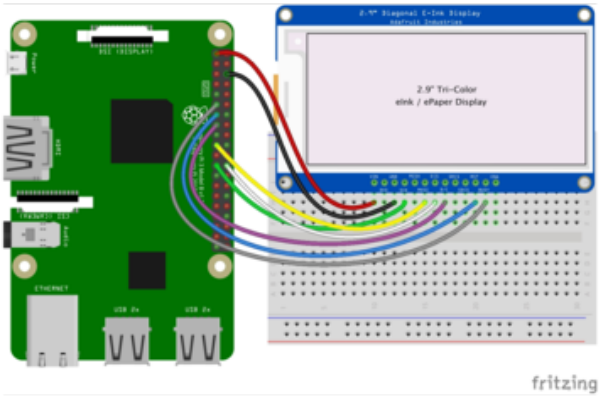
FeatherWing usage is easy, simply plug your Feather into the Wing



Download Fritzing Diagram

https://adafru.it/OdD

Python Wiring



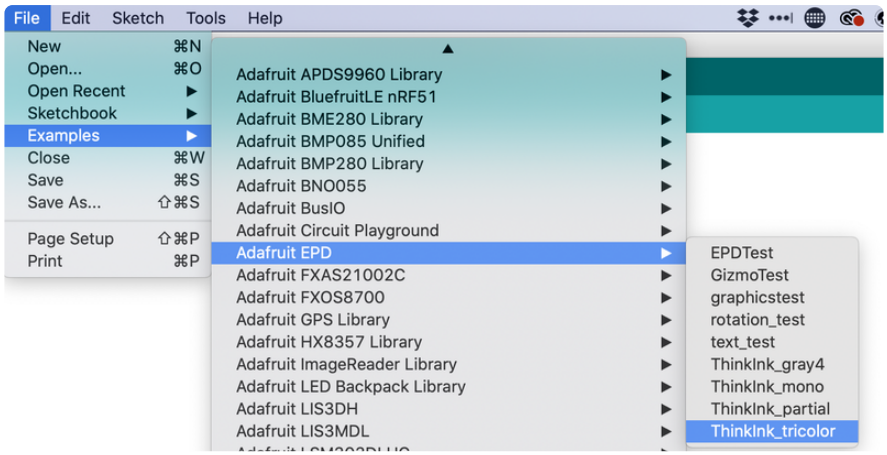
- Raspberry Pi 3.3 to display VIN
- Raspberry Pi GND to display GND
- Raspberry Pi SCLK to display SCK
- Raspberry Pi MOSI to display MOSI
- Raspberry Pi GPIO CE0 to display ECS
- Raspberry Pi GPIO 22 to display D/C
- Raspberry Pi GPIO 27 to display RST
- Raspberry Pi GPIO 17 to display BUSY

Download Fritzing Diagram

https://adafru.it/Ofh

Arduino Usage

Open up File→Examples→Adafruit_EPD→ThinkInk_tricolor



At the top of the sketch find the lines that look like:

```
#define EPD_CS      9
#define EPD_DC      10
#define SRAM_CS     6
#define EPD_RESET   8 // can set to -1 and share with microcontroller Reset!
#define EPD_BUSY    7 // can set to -1 to not use a pin (will wait a fixed delay)
```

FeatherWing Wiring

If you are using the FeatherWing, change both `EPD_RESET` and `EPD_BUSY` to -1 since neither of these lines are connected.

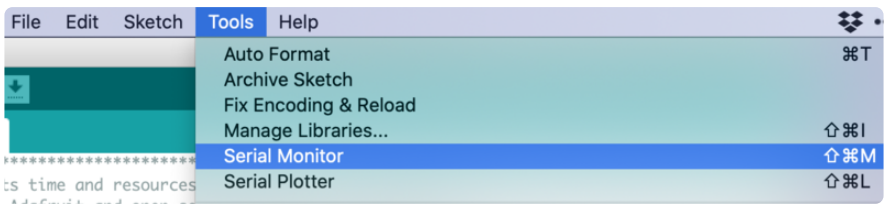
You'll also need to update the EPD_DC, EPD_CS and SRAM_CS pins if you're not using a Feather M0 or M4. For example, on an ESP8266

```
#define SRAM_CS 16
#define EPD_CS 0
#define EPD_DC 15
```

Breakout Wiring

If you are using the Breakout, just upload the sketch as it is.

Once it is done uploading, open the Serial Monitor.



The display should start running a series of monochrome and grayscale tests.



Downloads

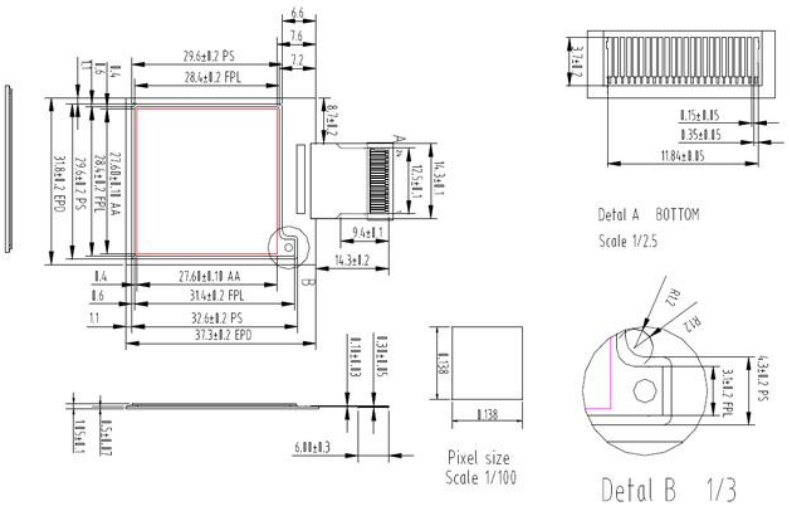
Files

- Fritzing object in Adafruit Fritzing Library (https://adafru.it/aP3)
- IL0376F E-Ink interface chip datasheet (https://adafru.it/BRW)
- PCB Files on GitHub (https://adafru.it/BRX)

SSD1675 driver datasheet

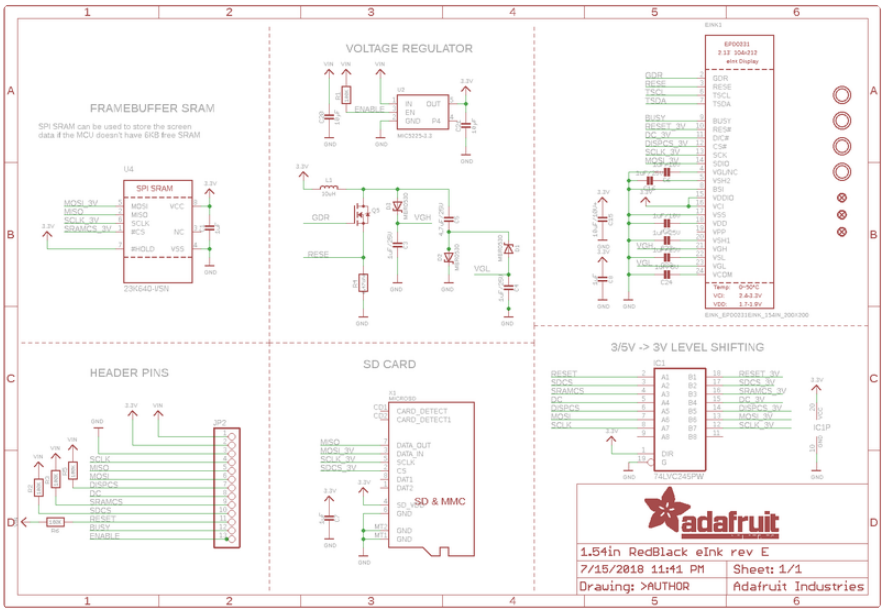
https://adafru.it/M5C

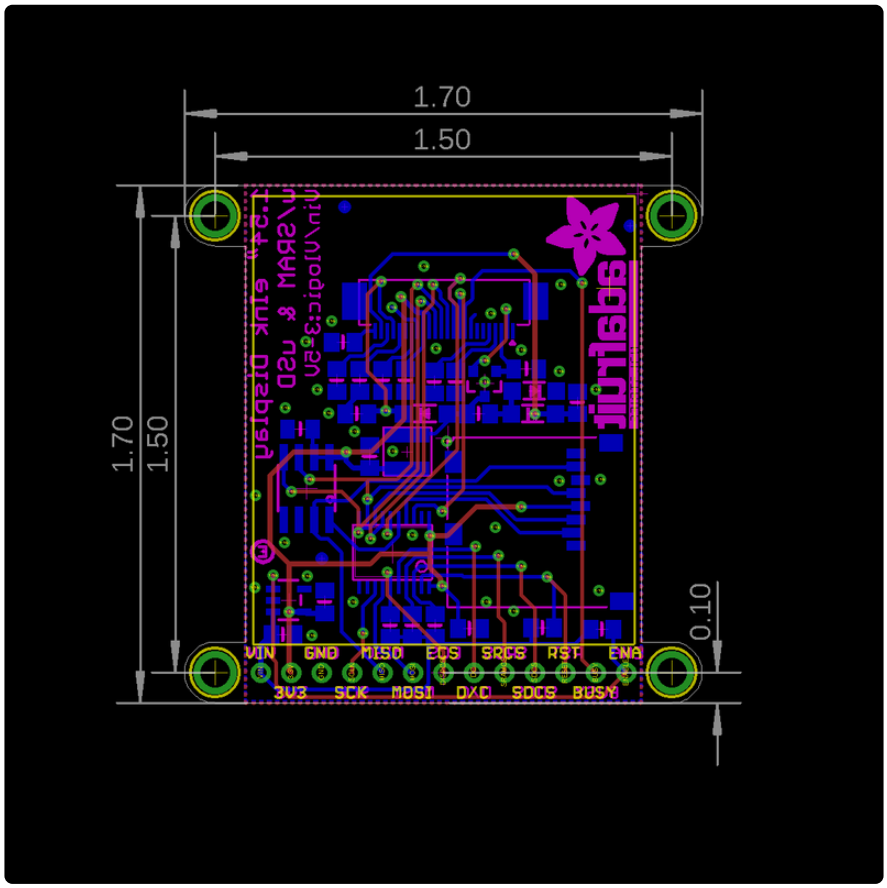
Display shape/outline:

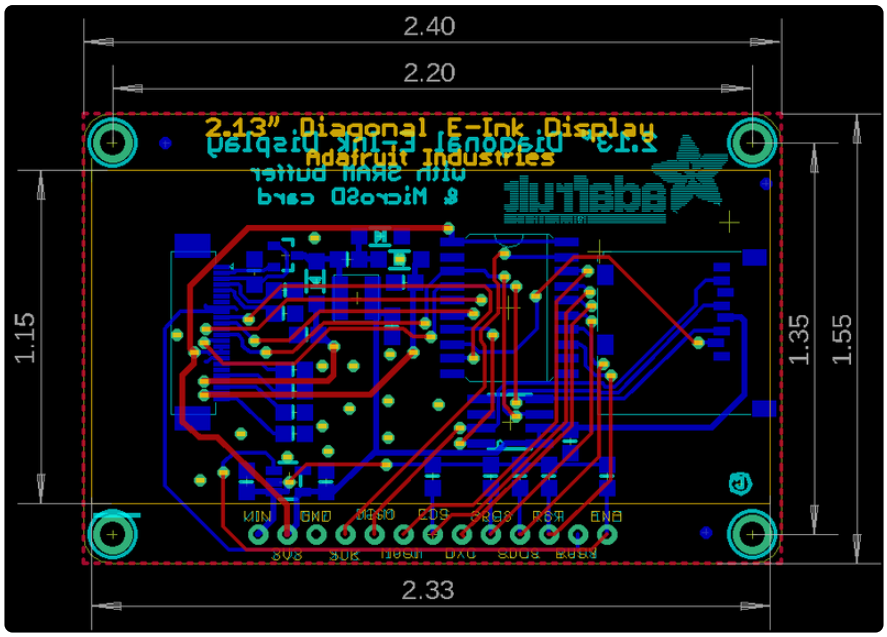


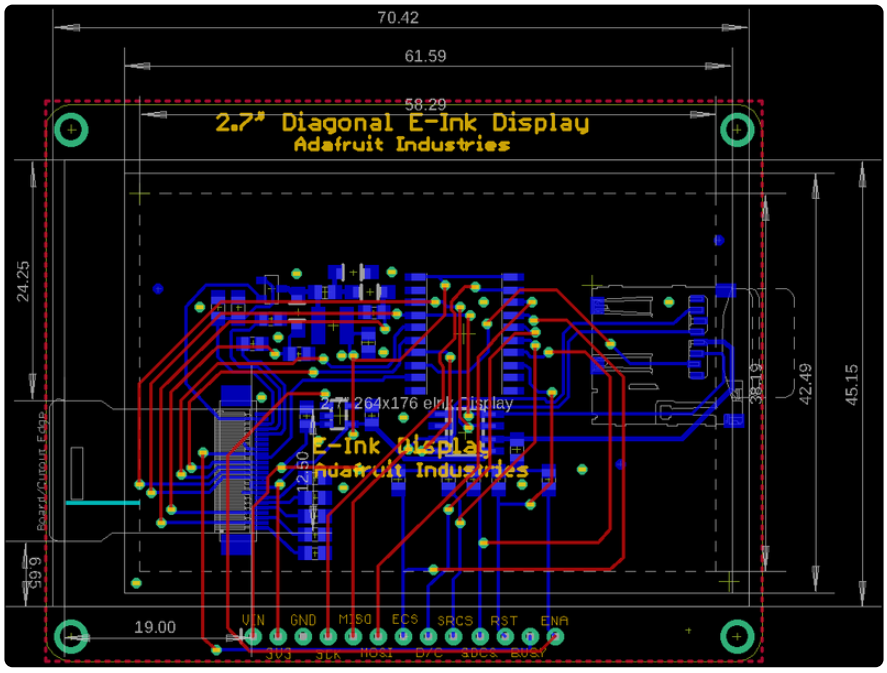
Schematic & Fabrication Prints

Shared schematic for 1.54" 2.13" and 2.7" Breakouts

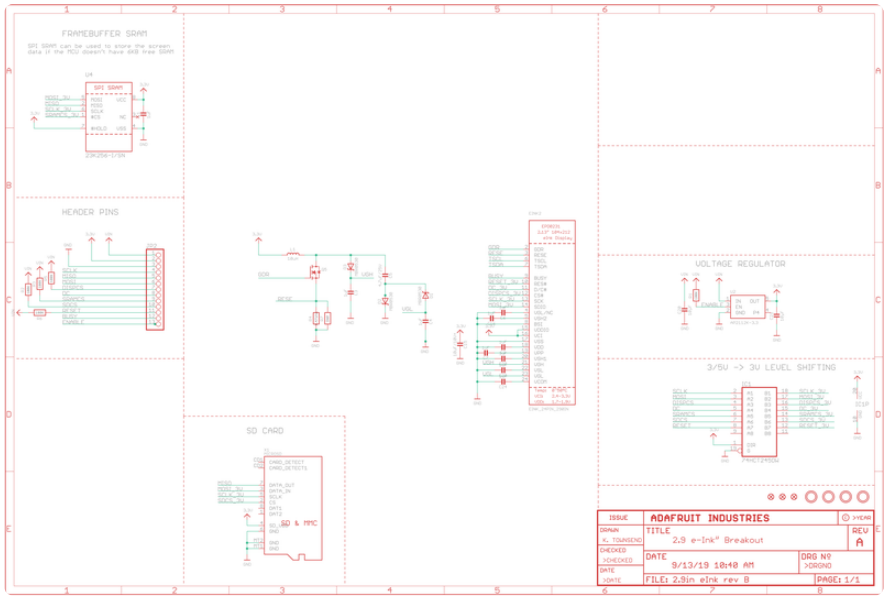


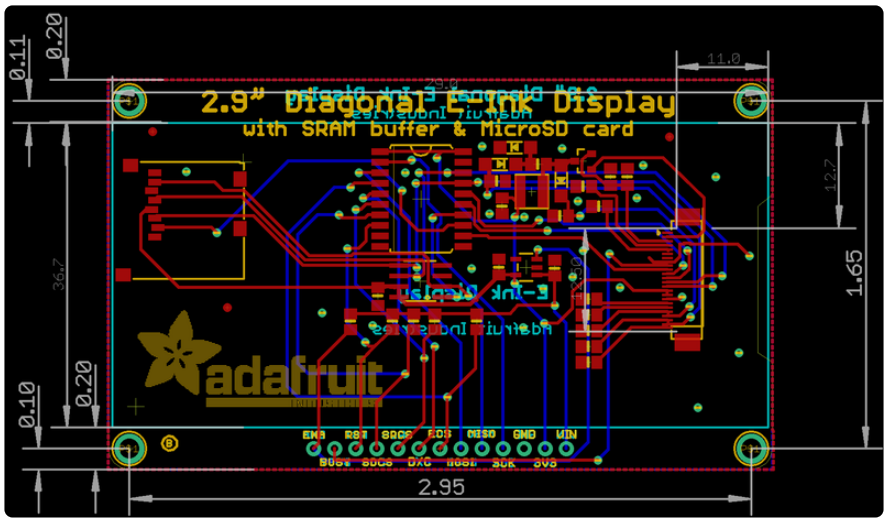



2.40
2.20
2.13" Diagonal E-Ink Display
Adafruit Industries
1.15
1.35
1.55
2.33

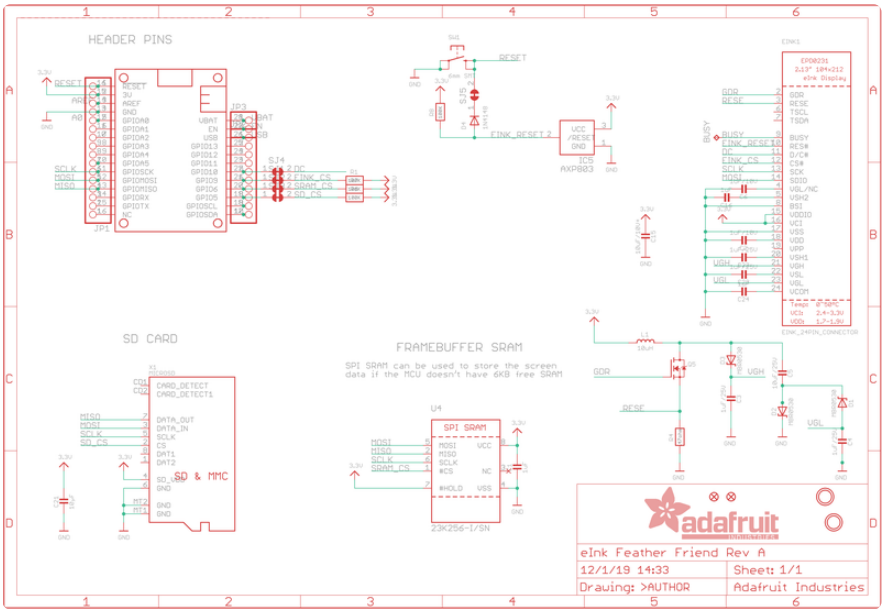

70.42
61.59
58.29
2.7" Diagonal E-Ink Display
Adafruit Industries
24.25
42.49
45.15
19.00
2.7" 264x176 eink display
E-Ink Display
Adafruit Industries

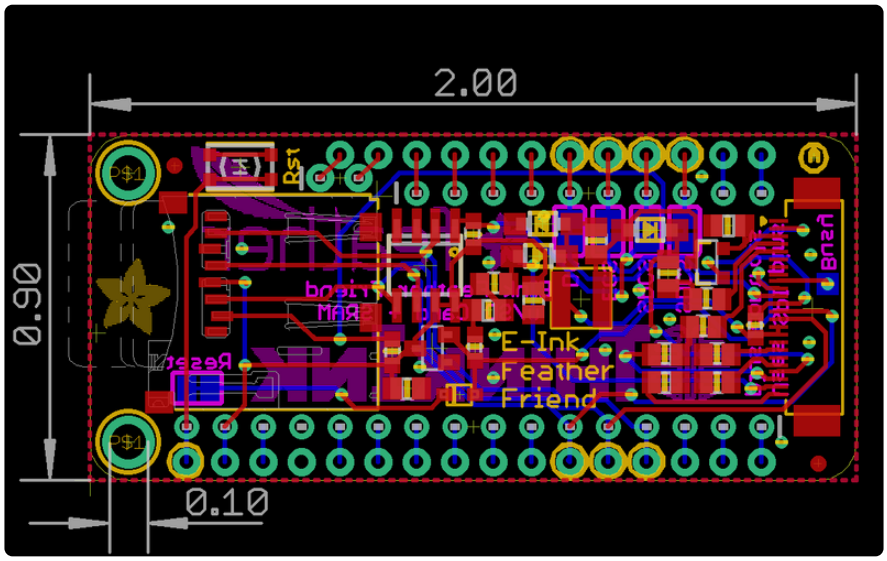
2.9 Inch Display

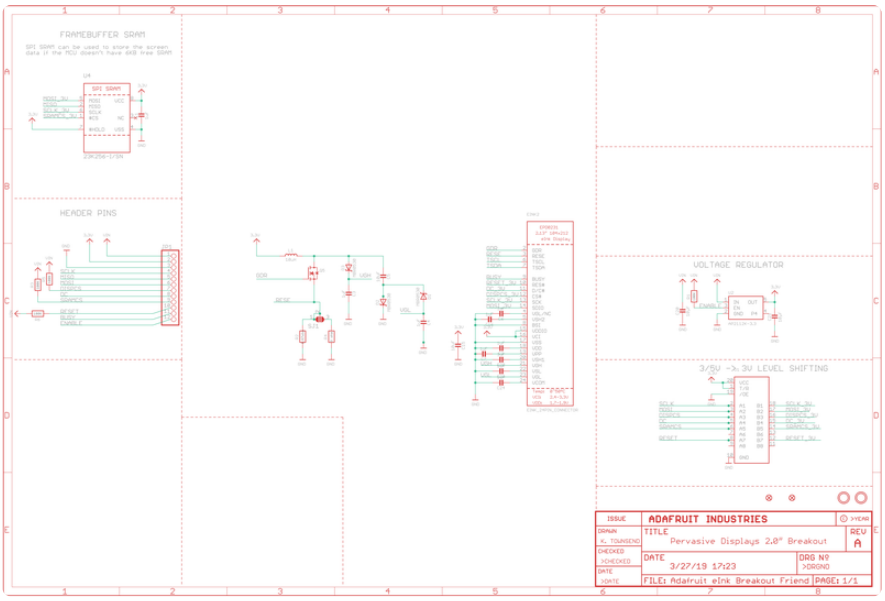


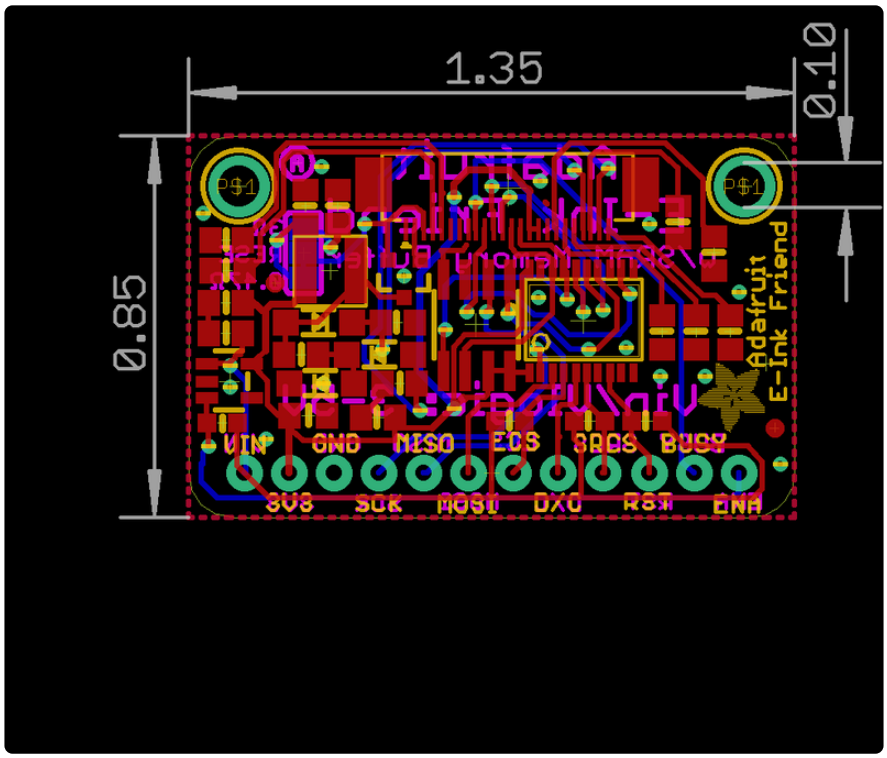


eInk Friends

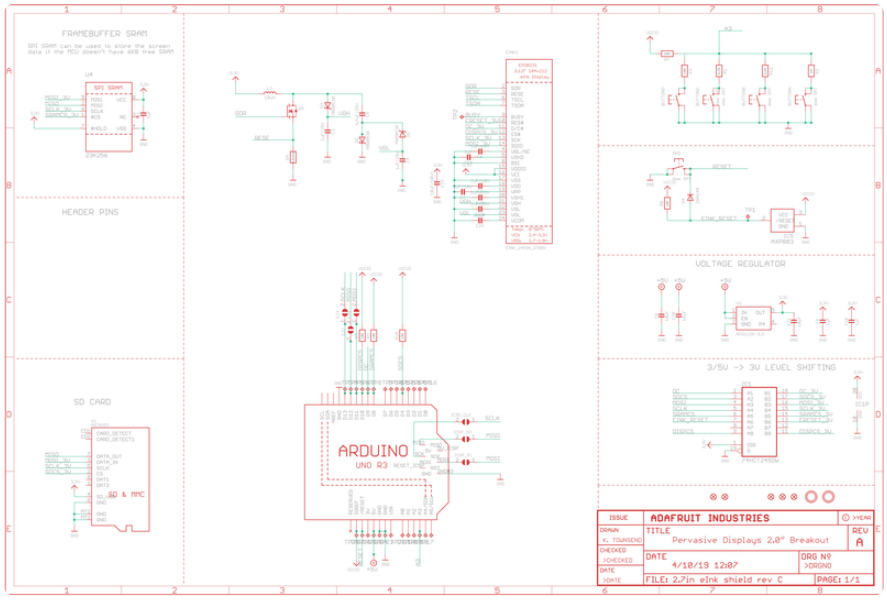


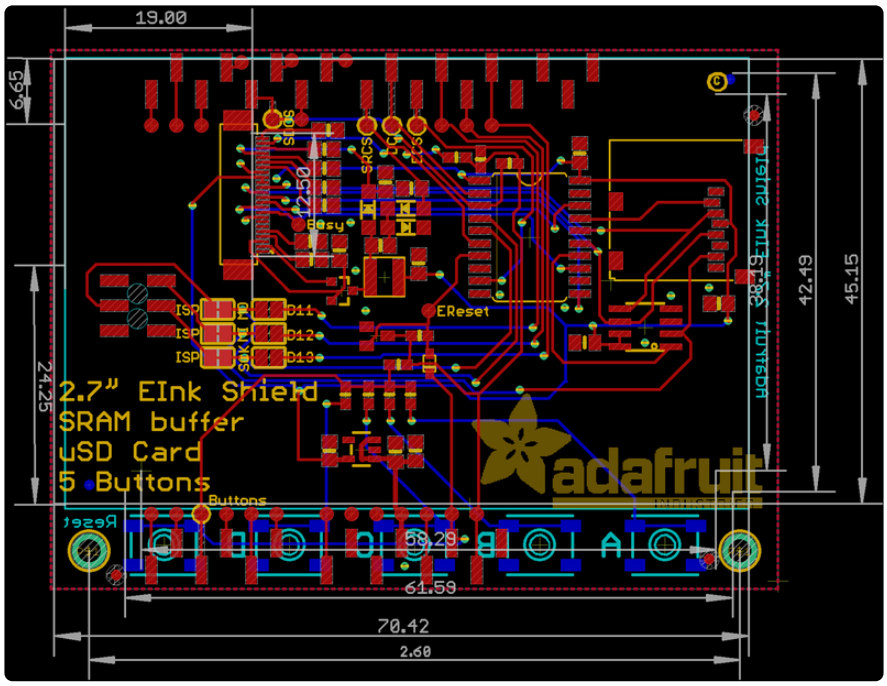



FRAMEBUFFER SRAM
HEADER PINS
VOLTAGE REGULATOR
3/5V ->3.3V LEVEL SHIFTING
ADAFRUIT INDUSTRIES
Pervasive Displays 2.0" Breakout
3/27/19 17:23
FILE: Adafruit eInk Breakout Friend
PAGE: 1/1


1.35
0.10
0.85
VIN
GND
MISO
SCK
MOSI
DXO
RST
BUSY
Adafruit
E-Ink Friend

2.7" Shield


FRAMEBUFFER SRAM
HEADER PINS
SD CARD
ARDUINO
UNO R3
VOLTAGE REGULATOR
3/5V -> 3V LEVEL SHIFTING
ADAFRUIT INDUSTRIES
Pervasive Displays 2.8" Breakout
4/10/19 12:07
FILE: 2.7in eInk shield rev C


19.00
6.65
24.25
42.19
45.15
28.19
12.50
2.54
ESD
SRAM
EReset
ISP
D11
D12
D13
2.7" EInk Shield
SRAM buffer
uSD Card
5 Buttons
Buttons
adafruit
68.20
61.50
70.42
2.60