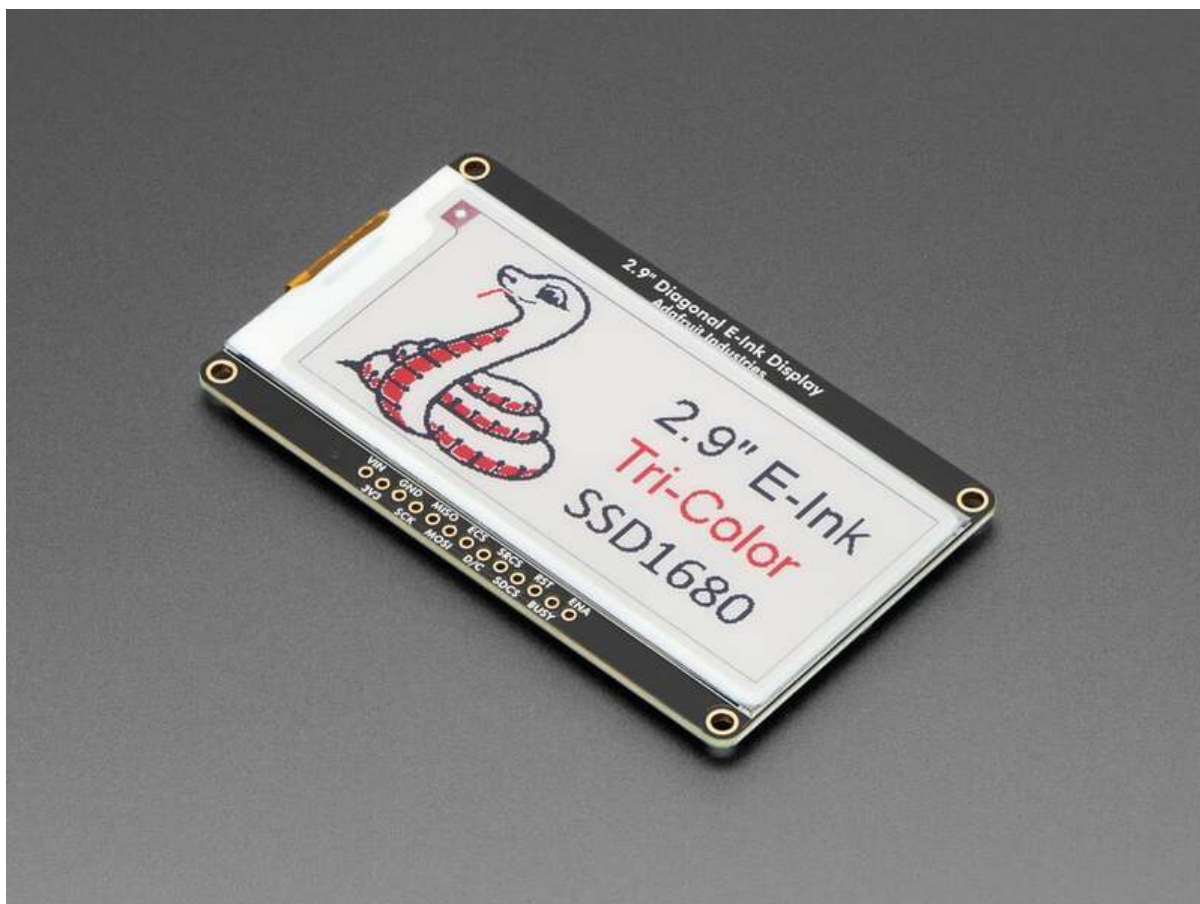




# Adafruit 2.9" eInk Display Breakouts and FeatherWings

Created by Melissa LeBlanc-Williams



<https://learn.adafruit.com/adafruit-2-9-eink-display-breakouts-and-featherwings>

Last updated on 2025-12-18 02:49:36 PM UTC

# Table of Contents

Overview	5
<ul style="list-style-type: none"><li>• <a href="#">2.9" Tri-Color eInk Display and FeatherWing Revision</a></li><li>• <a href="#">We have multiple 2.9" EPD displays:</a></li><li>• <a href="#">2.9" Tri-Color eInk Display Revision History</a></li><li>• <a href="#">2.9" Tri-Color eInk FeatherWing Revision History</a></li></ul>	
Pinouts	9
<ul style="list-style-type: none"><li>• <a href="#">2.9" Tri-Color eInk Display and FeatherWing Revision</a></li><li>• <a href="#">eInk Breakout Friend</a></li><li>• <a href="#">Power Pins</a></li><li>• <a href="#">Data Control Pins</a></li><li>• <a href="#">FeatherWing Connections</a></li><li>• <a href="#">2.9" Tri-Color eInk FeatherWing Revision History</a></li><li>• <a href="#">FeatherWing Buttons</a></li></ul>	
EYESPI	14
<ul style="list-style-type: none"><li>• <a href="#">The EYESPI Connector and Cables</a></li><li>• <a href="#">Wiring Your EYESPI Display</a></li><li>• <a href="#">EYESPI Pins</a></li></ul>	
Plugging in an EYESPI Cable	19
Assembly	21
<ul style="list-style-type: none"><li>• <a href="#">Assembly</a></li><li>• <a href="#">Add the E-Ink Display</a></li><li>• <a href="#">And Solder!</a></li></ul>	
Wiring	24
<ul style="list-style-type: none"><li>• <a href="#">EYESPI Connector</a></li><li>• <a href="#">Breakout Wiring</a></li><li>• <a href="#">FeatherWing Connection</a></li><li>• <a href="#">Python Wiring</a></li></ul>	
Usage and Expectations	27
Arduino Setup	28
Arduino Usage	29
<ul style="list-style-type: none"><li>• <a href="#">2.9" Monochrome 296x128 Pixel Display</a></li><li>• <a href="#">2.9" Tri-Color 296x128 Pixel Display</a></li><li>• <a href="#">2.9" Tri-Color eInk Display and FeatherWing Revision</a></li><li>• <a href="#">2.9" Grayscale 296x128 Pixel Display</a></li><li>• <a href="#">Configure Pins</a></li><li>• <a href="#">Configure Display Type &amp; Size</a></li><li>• <a href="#">Upload Sketch</a></li></ul>	
Arduino Bitmaps	35
<ul style="list-style-type: none"><li>• <a href="#">Tri-Color and Mono Display Demo</a></li><li>• <a href="#">GrayScale Display Demo</a></li></ul>	
CircuitPython Monochrome Usage	40
<ul style="list-style-type: none"><li>• <a href="#">CircuitPython eInk displayio Library Installation</a></li></ul>	

- [Image File](#)
- [Monochrome Display Usage](#)

## CircuitPython Grayscale Usage 44

---

- [CircuitPython eInk displayio Library Installation](#)
- [Image File](#)
- [Grayscale Display Usage](#)

## CircuitPython Tri-Color Usage 47

---

- [2.9" Tri-Color eInk Display and FeatherWing Revision](#)
- [CircuitPython eInk displayio Library Installation](#)
- [Image File](#)
- [Adafruit\\_CircuitPython\\_UC8151D](#)
- [UC8151D Display Usage](#)
- [Adafruit\\_CircuitPython\\_IL0373](#)

## Python Setup 57

---

- [Setup Virtual Environment](#)
- [Install Adafruit\\_Blinka](#)
- [Python Installation of EPD Library](#)
- [Download font5x8.bin](#)
- [DejaVu TTF Font](#)
- [Pillow Library](#)
- [Chip Enable Lines](#)

## Python Usage 59

---

- [2.9" Tri-Color eInk Display and FeatherWing Revision](#)
- [Monochrome Example](#)
- [Tri-Color Example](#)
- [Full Example Code](#)
- [Bitmap Example](#)
- [Image Drawing with Pillow](#)
- [Drawing Shapes and Text with Pillow](#)

## WipperSnapper Wiring & Test 80

---

- [What is WipperSnapper](#)
- [Wiring](#)
- [Usage](#)

## Downloads 89

---

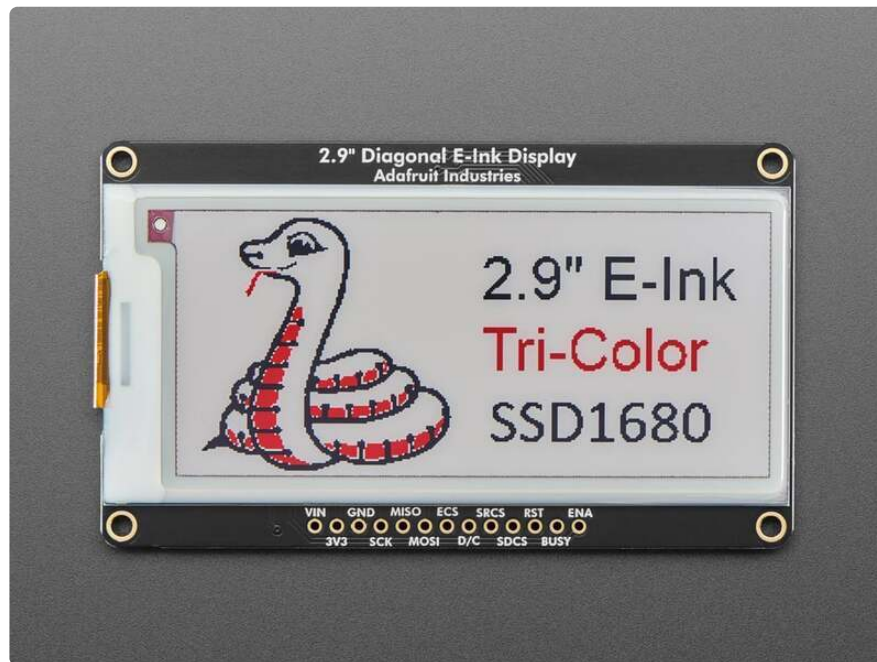
- [2.9" Tri-Color eInk Display Revision History](#)
- [2.9" Tri-Color eInk FeatherWing Revision History](#)
- [Files](#)
- [Schematic & Fabrication Prints](#)
- [2.9" FeatherWing](#)
- [eInk Friends](#)





---

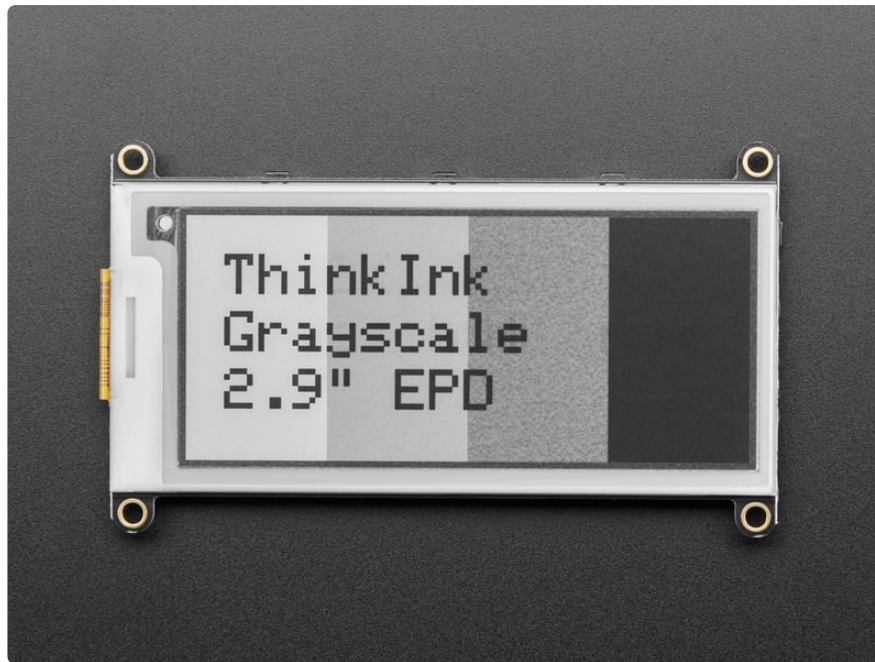
# Overview



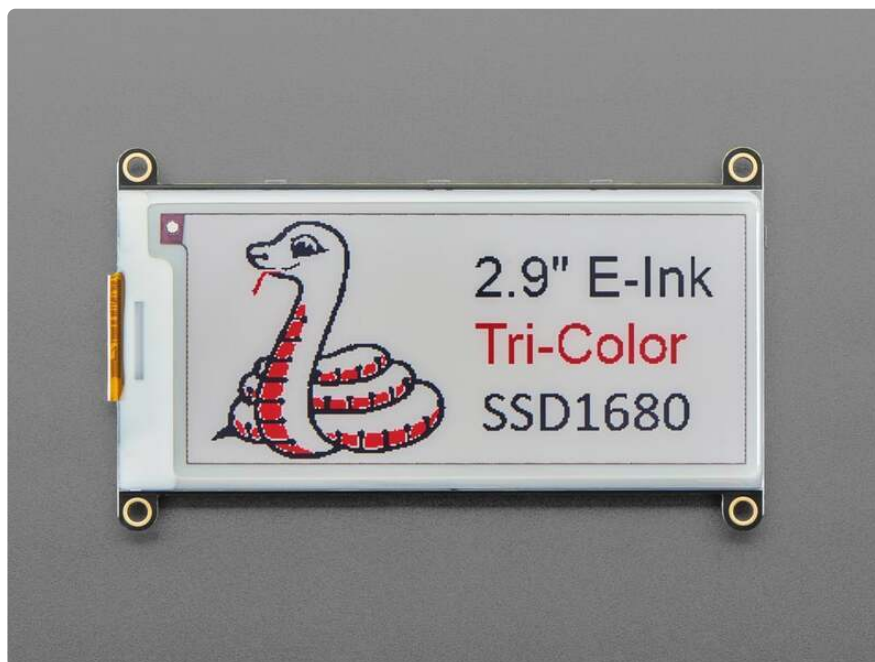
## 2.9" Tri-Color eInk Display and FeatherWing Revision

- As of June 20, 2025 - The tri-color display and FeatherWing have been updated (old ones discontinued) to now use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!

Easy e-paper finally comes to microcontrollers with these breakouts, shields and friends that are designed to make it a breeze to add a monochrome, tri-color, or grayscale 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!



Adafruit has 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 2.9" EPD displays:  
Breakouts and Flexibles

- We have a newer [296x128 Tri-Color display \(http://adafru.it/1028\)](http://adafru.it/1028) with the UC8151D chipset, which has black and red ink pixels and a white-ish background.

- The older tri-color breakout, which had the same Product ID, had the IL0373 chipset. This display is no longer offered.
- The [2.9" monochrome flexible display \(http://adafru.it/4262\)](http://adafru.it/4262) has a resolution of 296x128 and is flexible. For this display, you will probably want to pick up an [e-Ink Breakout Friend \(http://adafru.it/4224\)](http://adafru.it/4224) or [e-Ink Feather Friend \(http://adafru.it/4446\)](http://adafru.it/4446). There is an [extension cable \(http://adafru.it/4230\)](http://adafru.it/4230) available for this type of connection also.

## FeatherWings

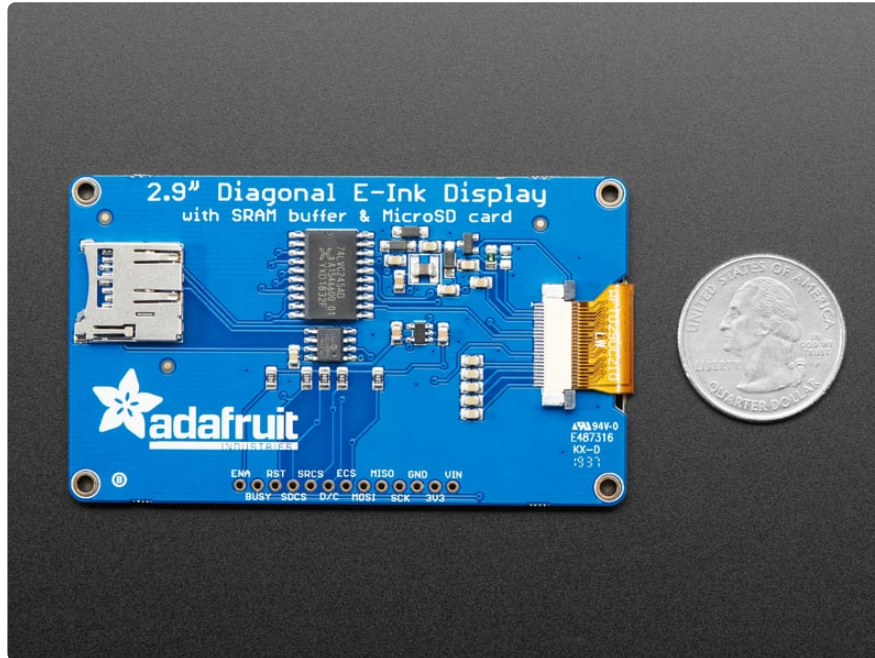
- The newer [296x128 Tri-Color FeatherWing \(http://adafru.it/4778\)](http://adafru.it/4778) with the SSD1675 chipset.
- The older tri-color FeatherWing with the IL0373 has the same display as the older breakout. This display is also no longer offered.
- The grayscale FeatherWing features 4 levels of grayscale. We have a [296x128 Grayscale FeatherWing \(http://adafru.it/4777\)](http://adafru.it/4777).



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,  $296 \times 128 \text{ pixels} \times 2\text{-bits-per-pixel} = 9.5 \text{ KBytes}$ . 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 elnk 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 elnk when you're ready. [The library we wrote does all the work for you \(https://adafru.it/BRK\)](https://adafru.it/BRK), you can just interface with it as if it were an [Adafruit\\_GFX compatible display \(https://adafru.it/BRK\)](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 added on a MicroSD socket to the breakouts and FeatherWings 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 common Maker microcontrollers.

## 2.9" Tri-Color elnk Display Revision History

- As of June 20, 2025 - The display has been updated (old ones discontinued) to use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!
- As of May 22th, 2024 – We've updated this PCB with a new EYESPI connector along with the Adafruit Penguin to make a lovely and legible silkscreen.
- As of November 16, 2022 - this revision has been updated to have a 2.9" Tri-Color elnk display only with a UC8151D chipset.



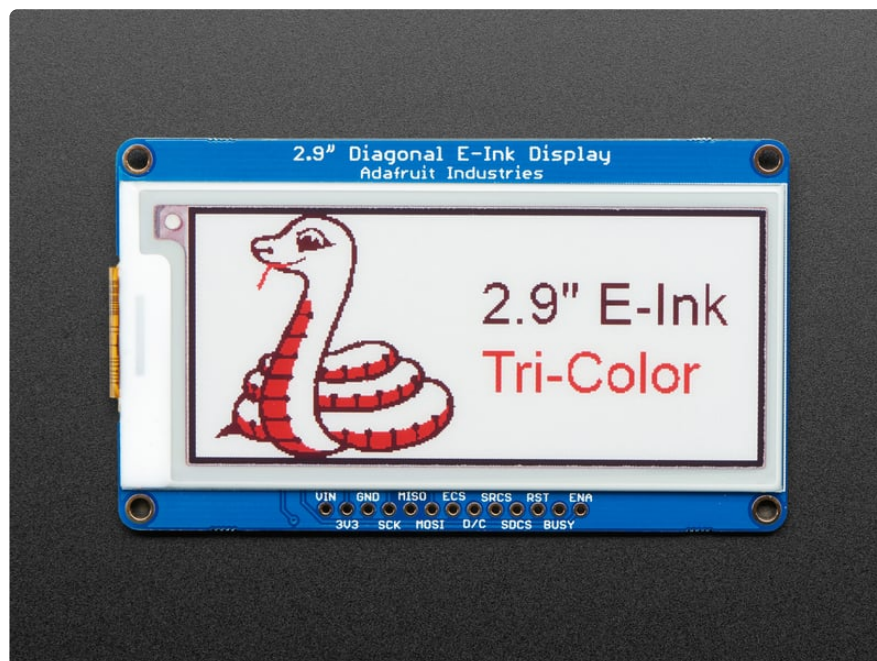
## 2.9" Tri-Color eInk FeatherWing Revision History

- As of June 24th, 2025 – The display has been updated (old one discontinued) it now uses a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!
- As of Sep 8, 2023 – we've updated this PCB with Adafruit Penguin to make a lovely and legible silkscreen - you may get the new PCB or the older version with vector fonts - both are identical other than the fancy silkscreen.

## Pinouts

### 2.9" Tri-Color eInk Display and FeatherWing Revision

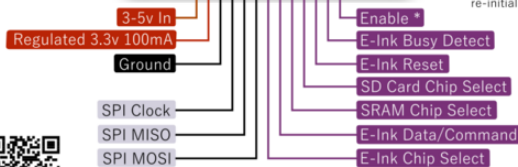
- As of June 20, 2025 - The tri-color display and FeatherWing have been updated (old ones discontinued) to now use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!



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

## Adafruit 2.13" E-Ink Display

Monochrome  
250x122  
Micro SD Socket  
SPI Ram  
L: 61.5mm/2.4"  
W: 39.3mm/1.5"  
H: 5.3mm/0.2"



\* Pull this pin low to get the lowest power draw possible.  
Note: this will erase the SPI Ram and you will need to re-initialize the SD card



<https://www.adafruit.com/product/4197>



pinouts are identical for the 2.13" and 2.9" E-Ink display!

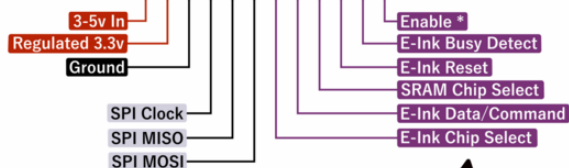
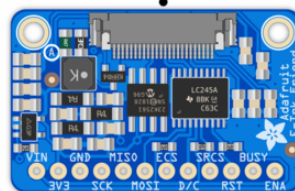
## eInk Breakout Friend

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

## Adafruit eInk Breakout Friend

32KB SRAM  
Up to 300x400 tri-color display support  
Built-in level shifter for 3.3 or 5v Power/Logic  
L: 34.7mm/1.4"  
W: 21.5mm/0.8"  
H: 3.5mm/0.1"

24-pin FPC Connector



\* Pull this pin low to get the lowest power draw possible.  
Note: this will erase the SPI Ram



<https://www.adafruit.com/product/4224>

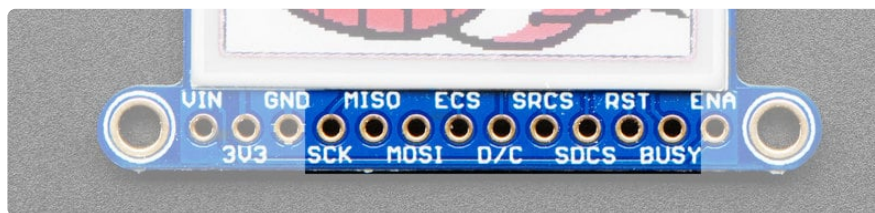


# Power Pins



- **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



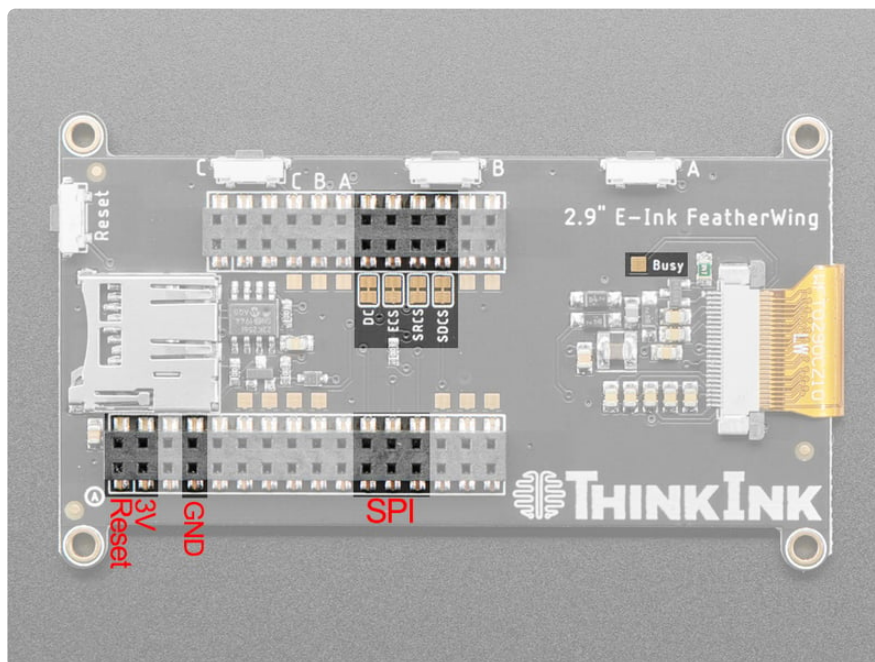
- **SCK** - this is the SPI clock input pin, required for e-Ink, SRAM and SD card
- **MISO** - this is the SPI Microcontroller In Serial Out pin, its used for the SD card and SRAM. It isn't used for the e-Ink display which is write-only, however you'll likely be using the SRAM to buffer the display so 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 **C**hip **S**elect, required for controlling the display
- **D/C** - this is the e-Ink **D**ata/**C**ommand pin, required for controlling the display
- **SRCS** - this is the **S**RAM **C**hip **S**elect, 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)

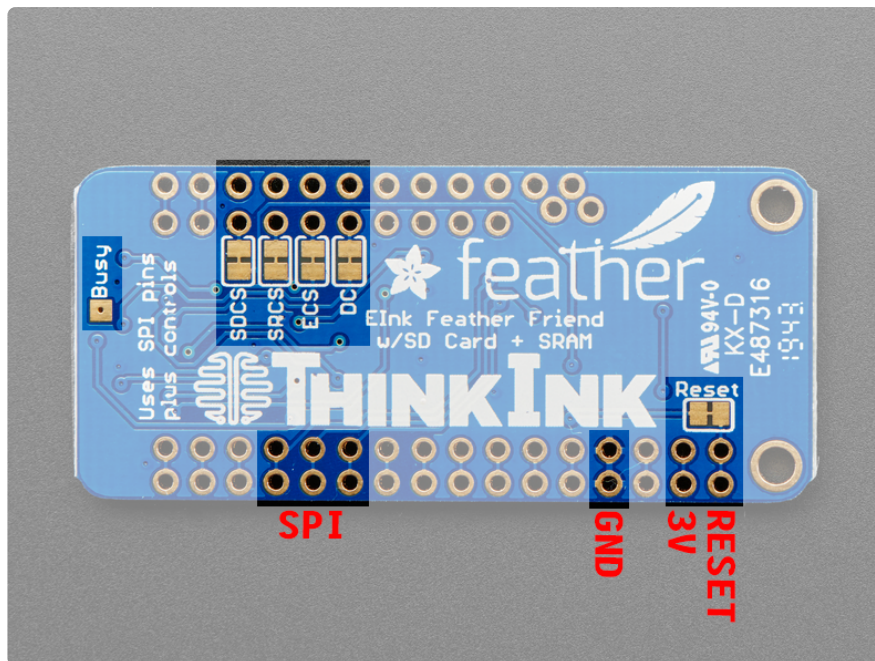
## FeatherWing Connections

### 2.9" Tri-Color eInk FeatherWing Revision History

- As of June 24th, 2025 – The display has been updated (old one discontinued) it now uses a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!
- As of Sep 8, 2023 – we've updated this PCB with Adafruit Pinguin to make a lovely and legible silkscreen - you may get the new PCB or the older version with vector fonts - both are identical other than the fancy silkscreen.







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.

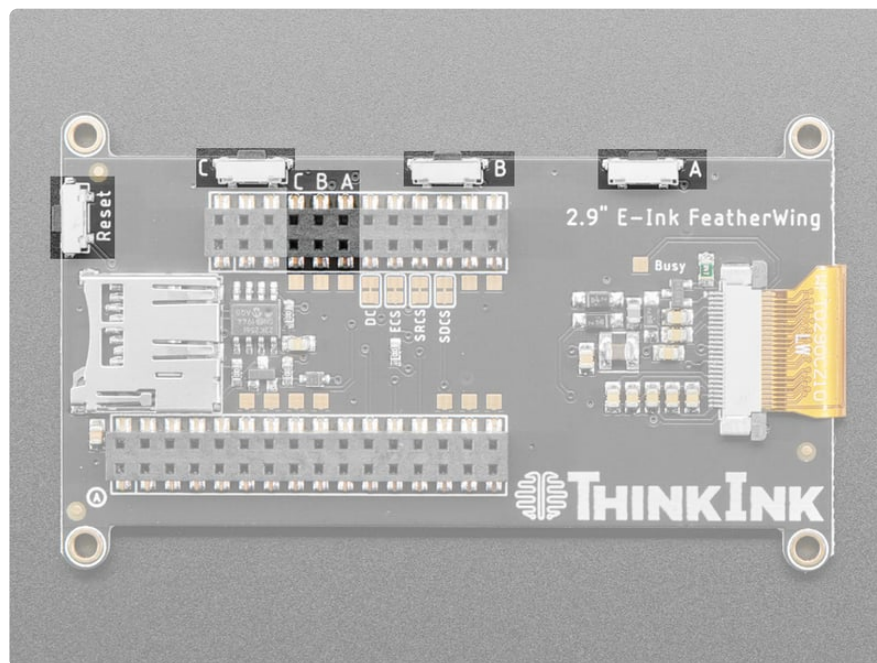
The **Reset** pin for the E-Ink display is connected to an auto-reset circuit and also to the Feather Reset pin, so it will reset when you press the reset button.

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.

## FeatherWing Buttons

The 2.9" eInk FeatherWings also feature a few buttons. The exact pin the buttons are connected to can vary by the specific feather.

- **A** - this is button A and is connected to Pin **D11** on many Feathers, but may vary.
- **B** - this is button B and is connected to Pin **D12** on many Feathers, but may vary.
- **C** - this is button C and is connected to Pin **D13** on many Feathers, but may vary.
- **Reset** - this button will reset the attached Feather.

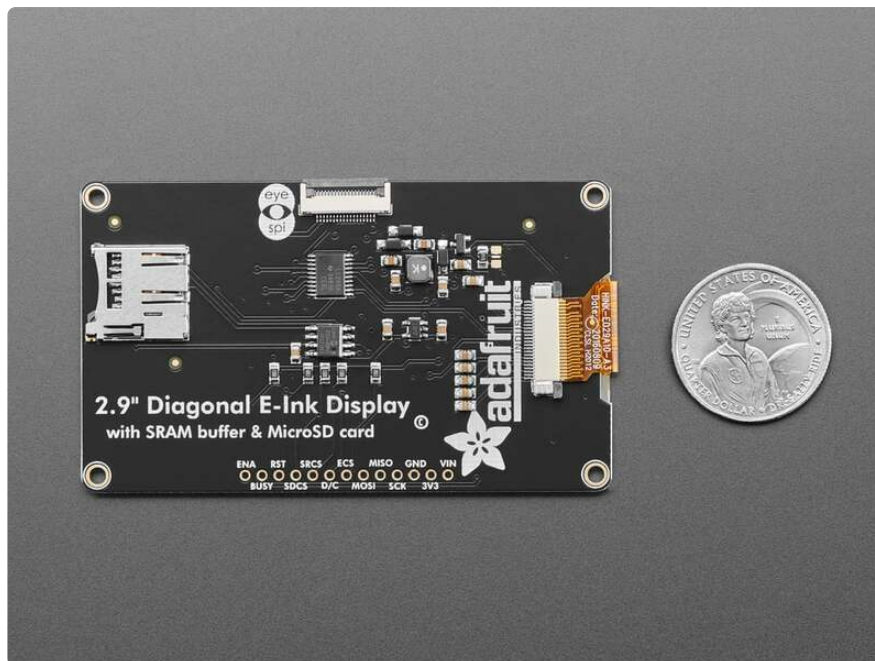


---

## EYESPI



Currently, only the 2.9" Tri-Color eInk display has been updated to have an SPI connector.

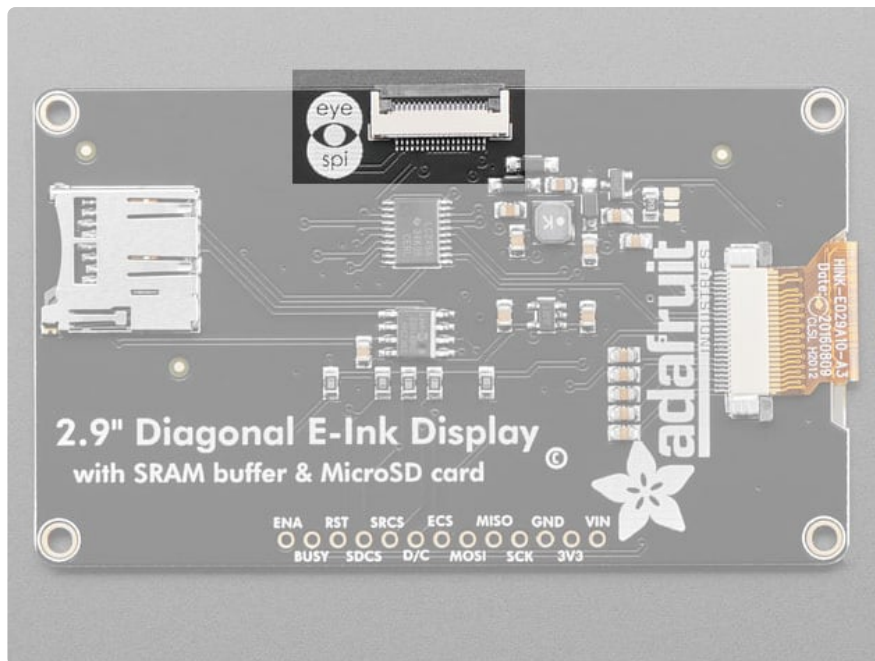


This display now comes with an **EYESPI connector**. This connector allows you to connect your display without soldering. There are [EYESPI cables](https://adafruit.it/18eT) (<https://adafruit.it/18eT>) available in multiple lengths, which means you can find one to fit any project. This is especially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

## The EYESPI Connector and Cables

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

The EYESPI connector location on this display is indicated below.



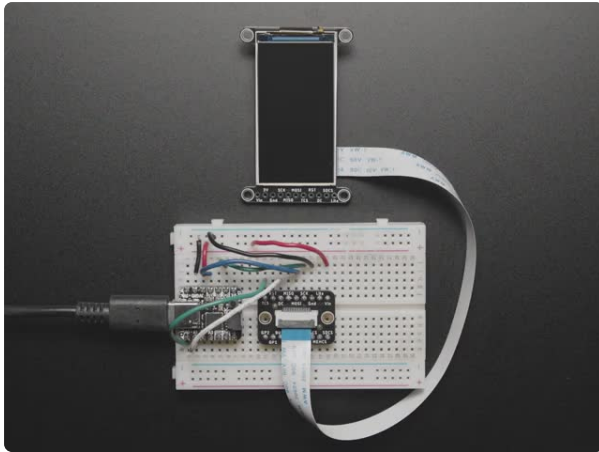
The EYESPI cables are 18 pin 0.5mm pitch flex cables. They are ~9.6mm wide, and designed to fit perfectly into the EYESPI connector. Adafruit currently offers EYESPI cables in three different lengths: [50mm](http://adafru.it/5462) (<http://adafru.it/5462>), [100mm](http://adafru.it/5239) (<http://adafru.it/5239>),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



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

## Wiring Your EYESPI Display

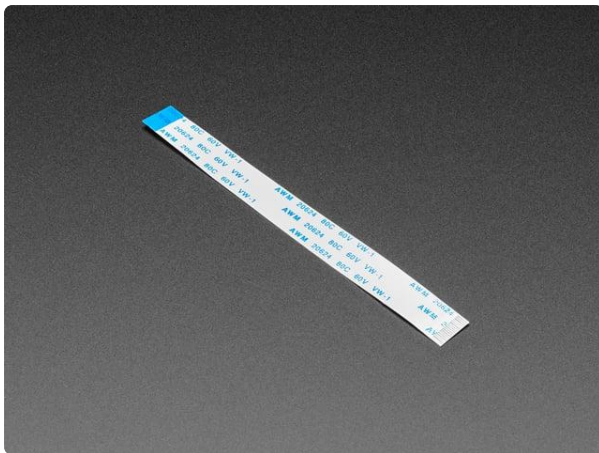
Wiring your EYESPI display to a microcontroller via the EYESPI connector requires the EYESPI breakout board and an EYESPI cable.



### Adafruit EYESPI Breakout Board - 18 Pin FPC Connector

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

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



### EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type

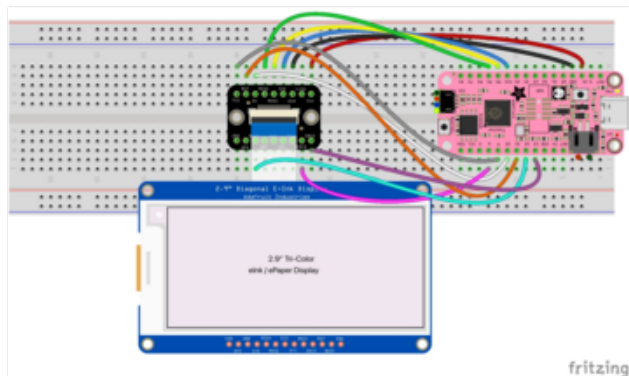
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...

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

The following example shows how to connect the 2.9" eInk display to a Feather RP2040 using the EYESPI breakout board.

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





Breakout Vin to Feather Vin (red wire)  
 Breakout Gnd to Feather Gnd (black wire)  
 Breakout SCK to Feather SCK (blue wire)  
 Breakout MOSI to Feather MOSI (yellow wire)  
 Breakout MISO to Feather MISO (green wire)  
 Breakout MEMCS to Feather pin 6 (pink wire)  
 Breakout TCS to Feather pin 9 (grey wire)  
 Breakout DC to Feather pin 10 (white wire)  
 Breakout RST to Feather pin 11 (orange wire)  
 Breakout BUSY to Feather pin 12 (cyan wire)  
 Breakout SDCS to Feather pin 13 (purple wire)

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

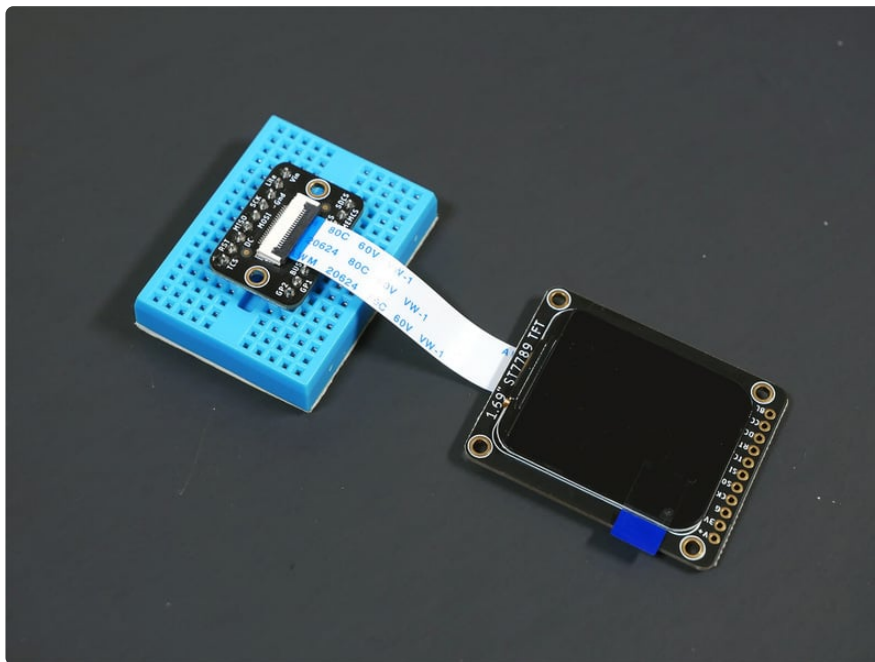
## EYESPI Pins

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

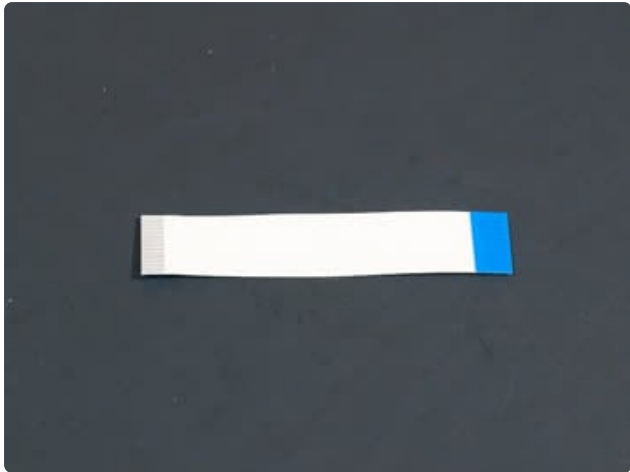
- **Vin** - This is the power pin. To power the board (and thus your display), connect to the same power as the logic level of your microcontroller, e.g. for a 3V micro like a Feather, use 3V, and for a 5V micro like an Arduino, use 5V
- **GND** - This is common ground for power and logic
- **SCK** - This is the SPI clock input pin
- **MOSI** - This is the SPI MOSI (**M**icrocontroller **O**ut / **S**erial **I**n) pin. It is used to send data from the microcontroller to the SD card and/or display

- **MISO** - This is the SPI MISO (**M**icrocontroller **I**n / **S**erial **O**ut) pin. It's used for the SD card. It isn't used for the display because it's write-only. It is 3.3V logic out (but can be read by 5V logic)
  - **TCS** - eInk Chip Select pin
  - **DC** - Data/command pin
  - **MEMCS** - **M**emory **C**hip **S**elect, required for communicating with the onboard RAM chip
  - **RST** - This is the display reset pin. Connecting to ground resets the display! It's best to have this pin controlled by the library so the display is reset cleanly, but you can also connect it to the microcontroller's Reset pin, which works for most cases. Often, there is an automatic-reset chip on the display which will reset it on power-up, making this connection unnecessary in that case
  - **BUSY** - e-Ink busy detect pin
  - **SDCS** - This is the SD card chip select pin. This pin is required for communicating with the SD card holder onboard the connected display
- 

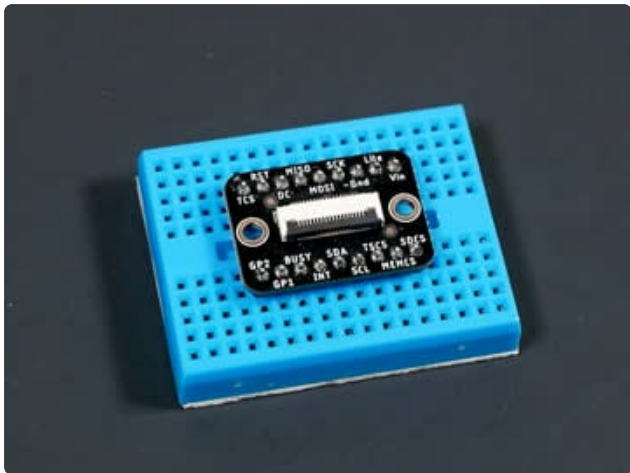
## Plugging in an EYESPI Cable



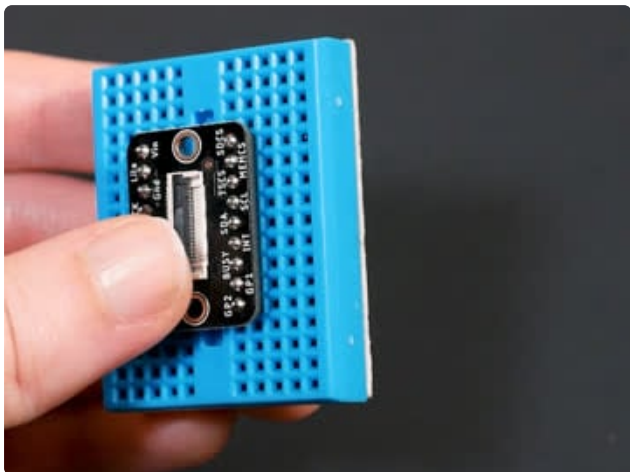
You can connect an EYESPI compatible display to the EYESPI breakout board using an EYESPI cable. An EYESPI cable is an 18 pin flexible PCB (FPC). The FPC can only be connected properly in one orientation, so be sure to follow the steps below to ensure that your display and breakout are plugged in properly.



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



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



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



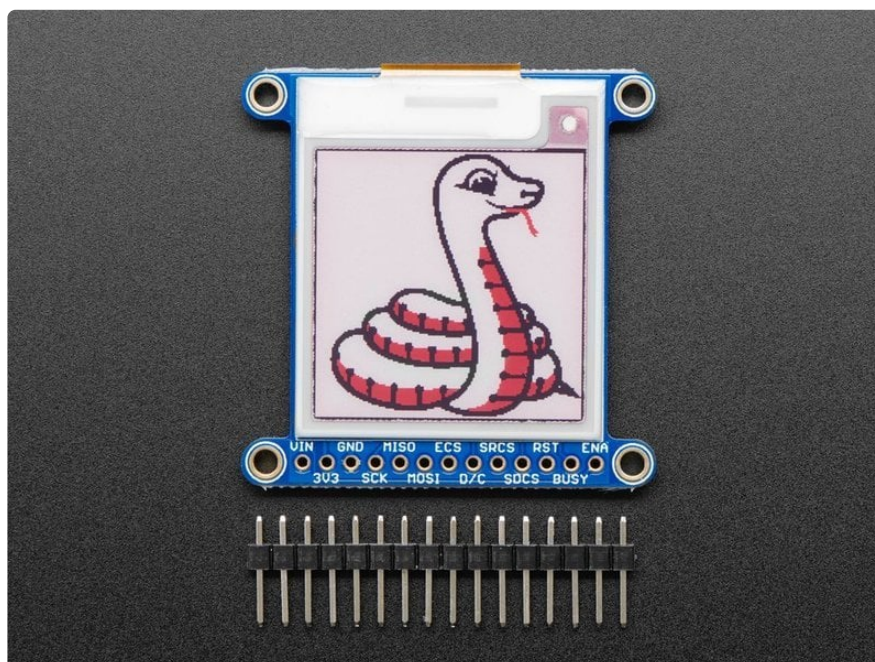


To secure the cable, lower the FPC connector latch onto the EYESPI cable.



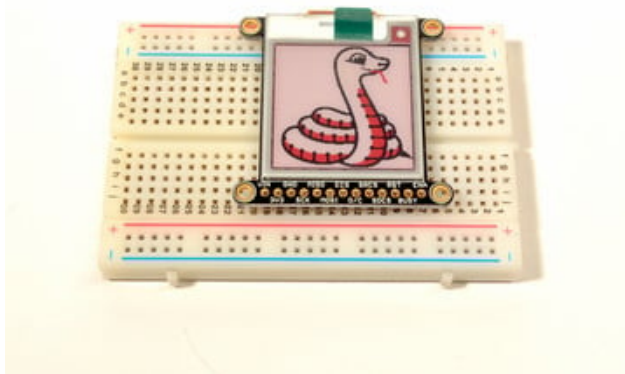
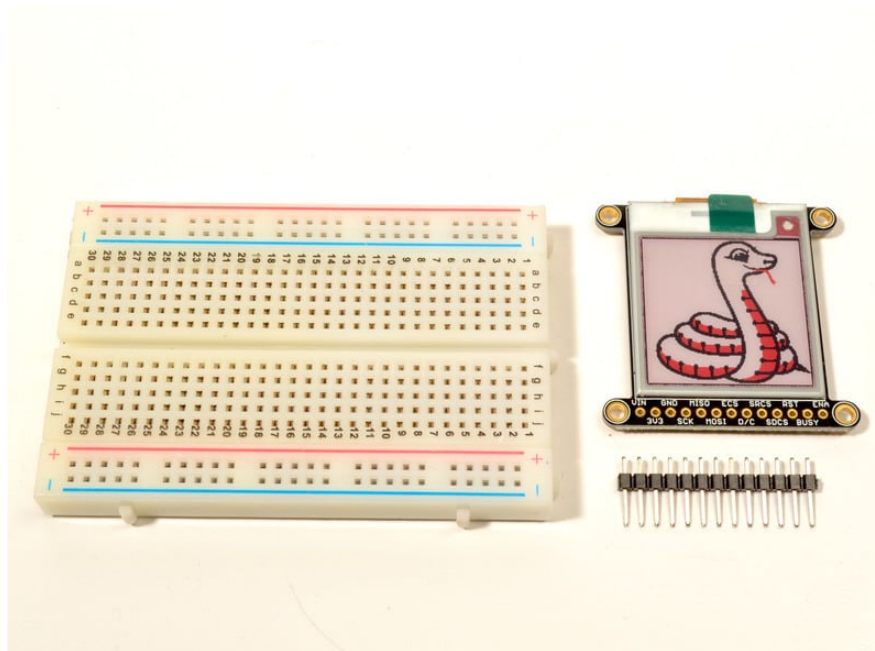
Repeat this process for the FPC connector on your display. Again, ensure that the **blue stripe** on either end of the cable is facing up.

## Assembly



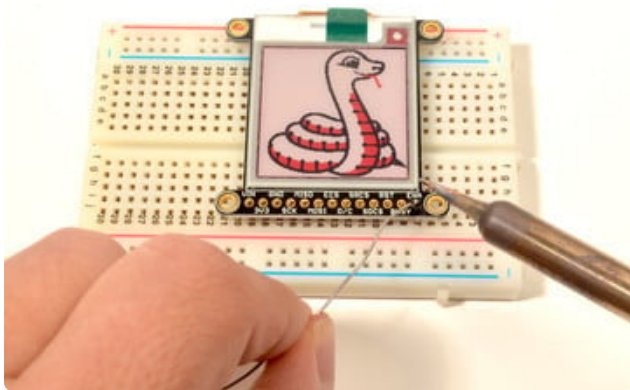
## Assembly

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

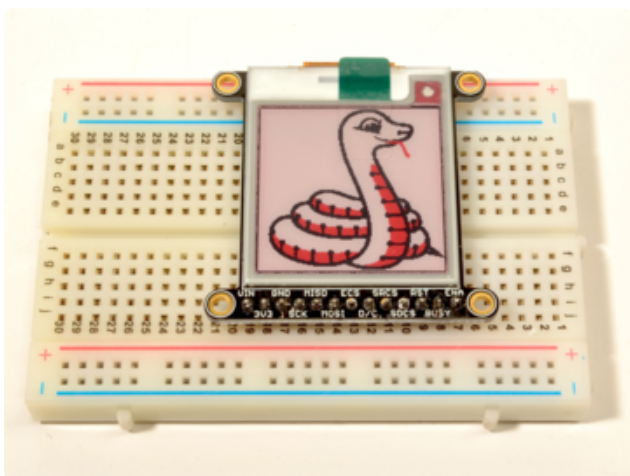
Place the board over the pins so that the short pins poke through the top of the breakout pads



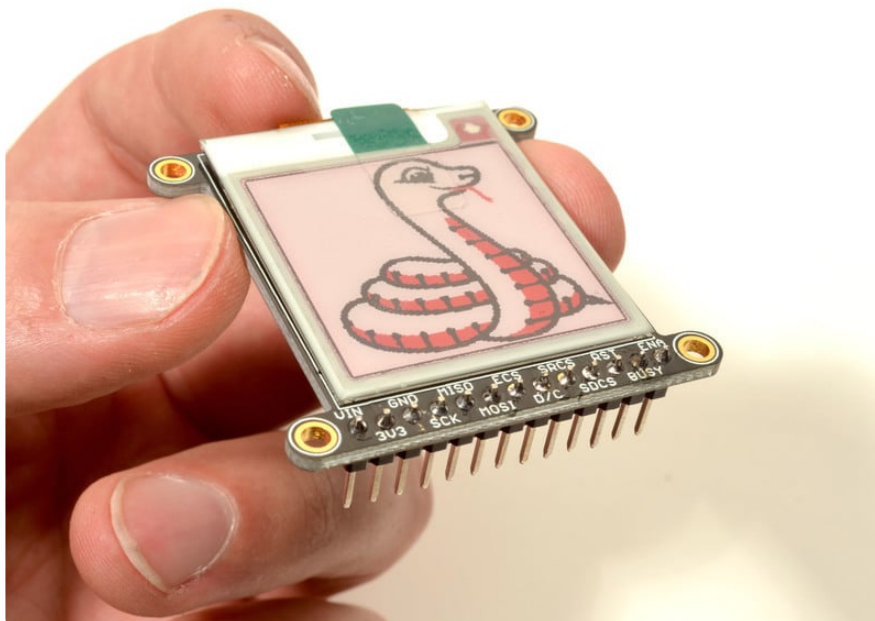
## And Solder!

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://adafruit.it/aTk) (<https://adafruit.it/aTk>)).



OK, you're done!



---

## Wiring

### EYESPI Connector

If you have one of the newer breakouts with an EYESPI connector on the back, you can use that with an 18-pin ribbon cable to connect it either directly to an adapter board such as the [EYESPI BFF for QT Py or Xiao](http://adafru.it/5772) (<http://adafru.it/5772>) or by using the [EYESPI Breakout](http://adafru.it/5613) (<http://adafru.it/5613>) and wiring that to your microcontroller. See the [EYESPI page](https://adafru.it/1a1u) (<https://adafru.it/1a1u>) for more information.

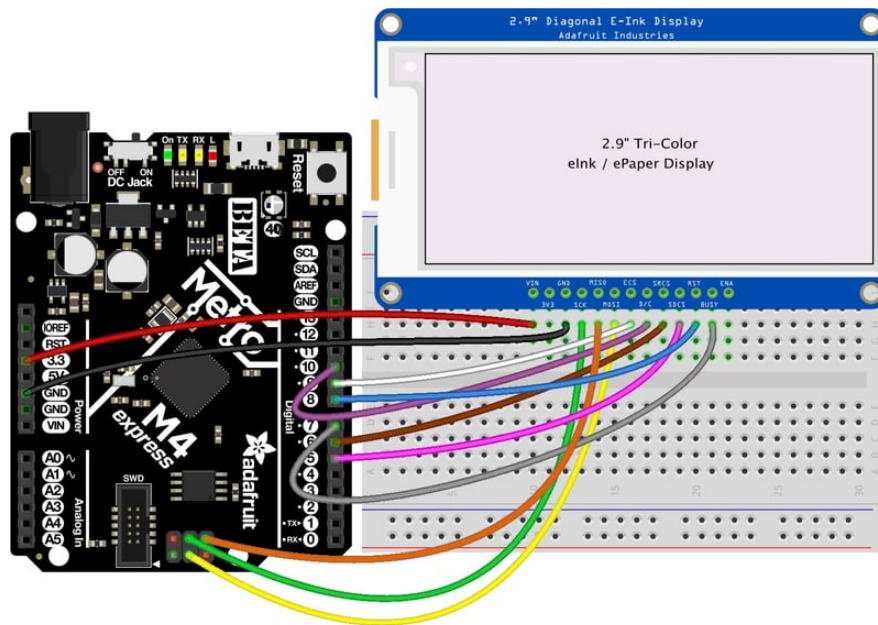
### Breakout Wiring

Though it looks like a lot of connections, wiring up an eInk breakout is pretty straightforward! Below shows using hardware SPI to connect it to an Adafruit Metro M4.

- **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**.



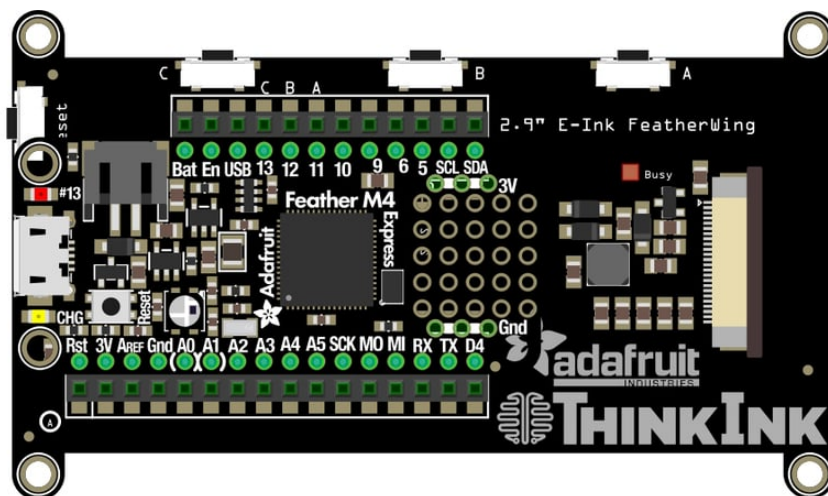
- **SRCS** connects to our SRAM Chip Select pin. We'll be using **Digital 6**.
- **RST** connects to our e-Ink reset pin. We'll be using **Digital 8**.
- **BUSY** connects to our e-Ink busy pin. We'll be using **Digital 7**.
- **SDCS** connects to our SD Card Chip Select pin. We'll be using **Digital 5**.



fritzing

## FeatherWing Connection

FeatherWing usage is easy, simply plug your Feather board into the FeatherWing board.



fritzing

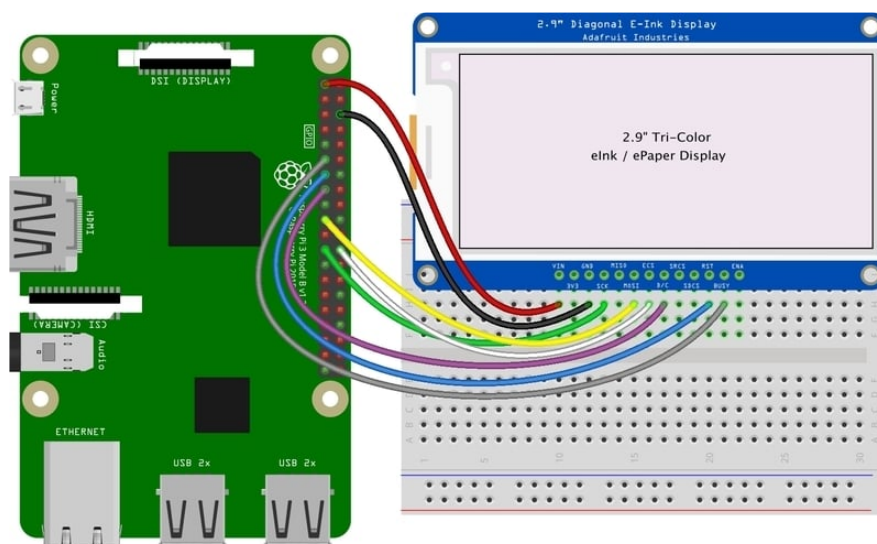
# Python Wiring



Wiring for the eInk friend breakout is the same as the Tri-Color breakout.

There are many Single Board Computers (SBC) so showing them all is impractical. The Raspberry Pi is the most common and it is shown below. Others are likely similar. The SPI peripheral may need to be enabled on such boards.

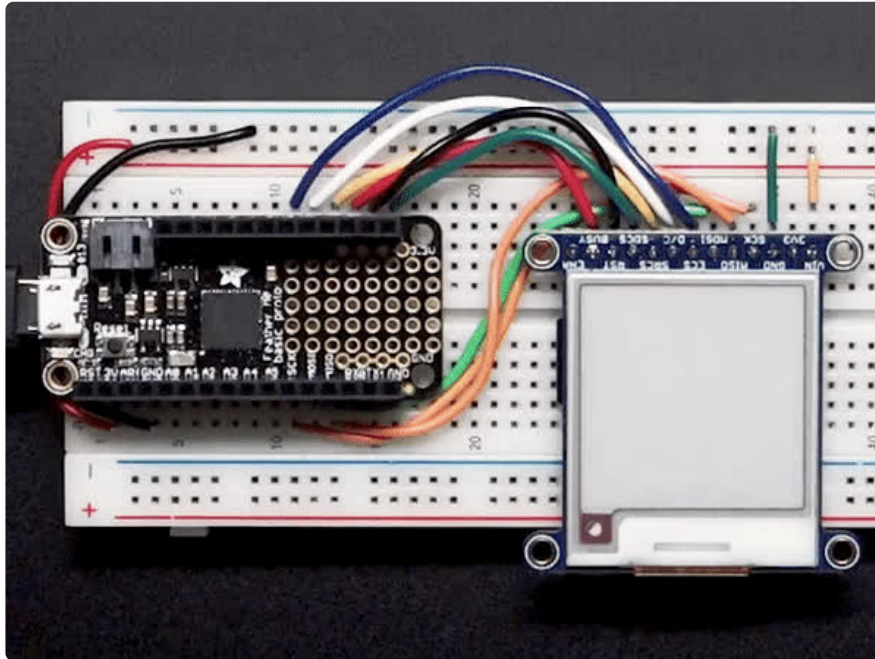
- Raspberry Pi 3.3 to display VIN
- Raspberry Pi GND to display GND
- Raspberry Pi SCLK to display SCK
- Raspberry Pi MOSI to display MOSI
- Raspberry Pi GPIO CE0 to display ECS
- Raspberry Pi GPIO 22 to display D/C
- Raspberry Pi GPIO 27 to display RST
- Raspberry Pi GPIO 17 to display BUSY



fritzing

---

# Usage and 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 many seconds to fully erase and replace an image**

There's also a recommended limit on refreshing - **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**



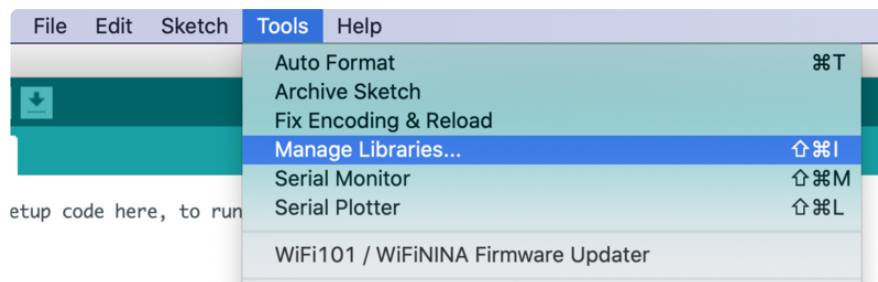
**Do not** update more than once every 180 seconds or you may permanently age the display

---

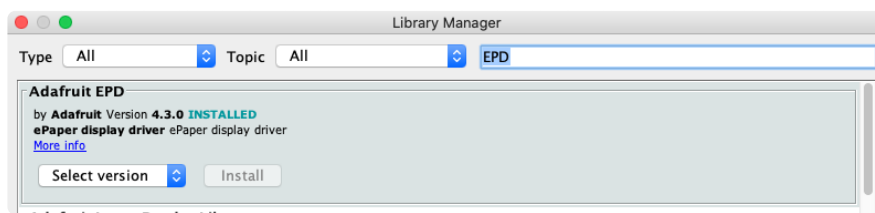
# Arduino Setup

To use the display, you will need to [install the Adafruit\\_EPD library \(code on our github repository\) \(https://adafru.it/BRK\)](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**

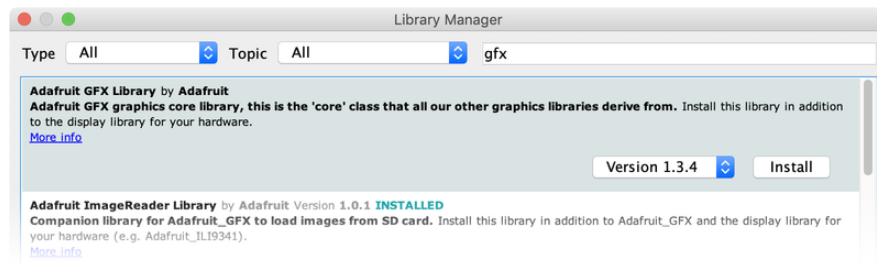


If you would like to draw bitmaps, do the same with **adafruit ImageReader**, click **Install**



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 Usage



There is where the differences in the tri-color/monochrome and chipset/ dimensions start mattering. Check carefully to make sure you are running the right example and creating the matching ThinkInk type for your display or you won't see anything happen on the EPD (or the image may be really weird looking).

### 2.9" Monochrome 296x128 Pixel Display

For the 296x128 Flexible Display, below is a monochrome demo.

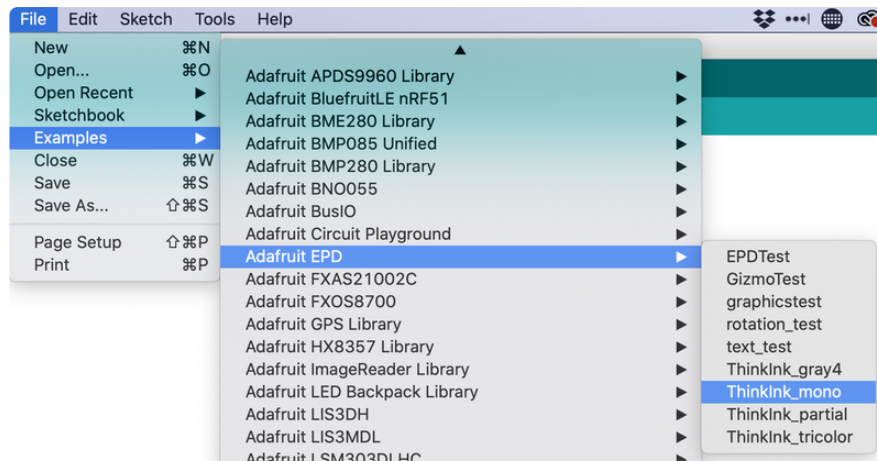


#### [2.9" Flexible 296x128 Monochrome eInk / ePaper Display](https://www.adafruit.com/product/4262)

Woah, the cyber-future is here! Flexible E-Ink has been demo'd at high-tech events for years but now you can actually get your paws on it. This display is true E-Ink / E-Paper,...

<https://www.adafruit.com/product/4262>

Open up **File→Examples→Adafruit\_EPD→ThinkInk\_mono**



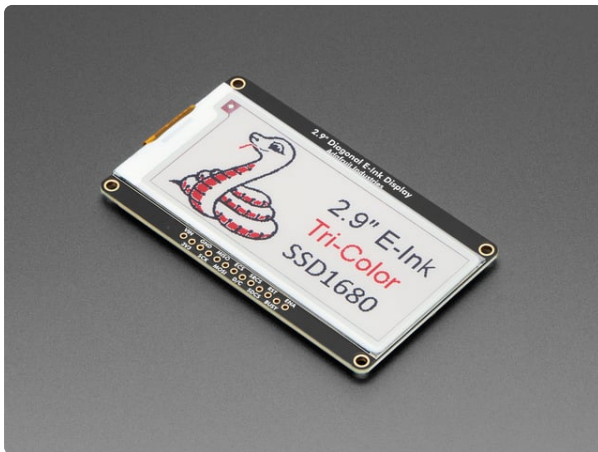
## 2.9" Tri-Color 296x128 Pixel Display

For the 296x128 Tri-Color display, below is a tri-color demo.



### 2.9" Tri-Color eInk Display and FeatherWing Revision

- As of June 20, 2025 - The tri-color display and FeatherWing have been updated (old ones discontinued) to now use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!



### Adafruit 2.9" Red/Black/White eInk Display Breakout - THINK INK

Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those...

<https://www.adafruit.com/product/1028>



## Adafruit 2.9" Tri-Color eInk / ePaper Display FeatherWing

Easy e-paper comes to your Feather with this breakout that's designed to make it a breeze to add a tri-color eInk display.

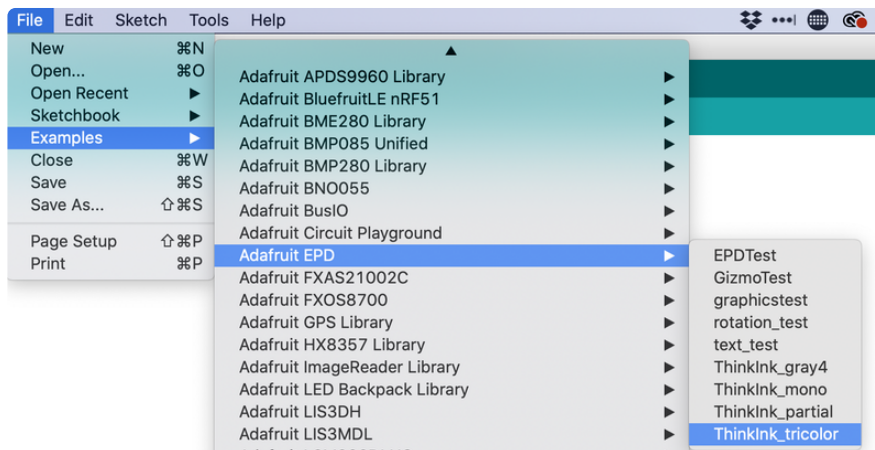
Chances are you've seen one of those...

<https://www.adafruit.com/product/4778>



demo updates every 15 seconds which is fine for demonstrating the functionality for a short time, but we recommend not updating more often than seconds if you are planning on running any code long term.

Open up **File**→**Examples**→**Adafruit\_EPD**→**ThinkInk\_tricolor**



## 2.9" Grayscale 296x128 Pixel Display

For the 296x128 Grayscale display, below is the grayscale demo.

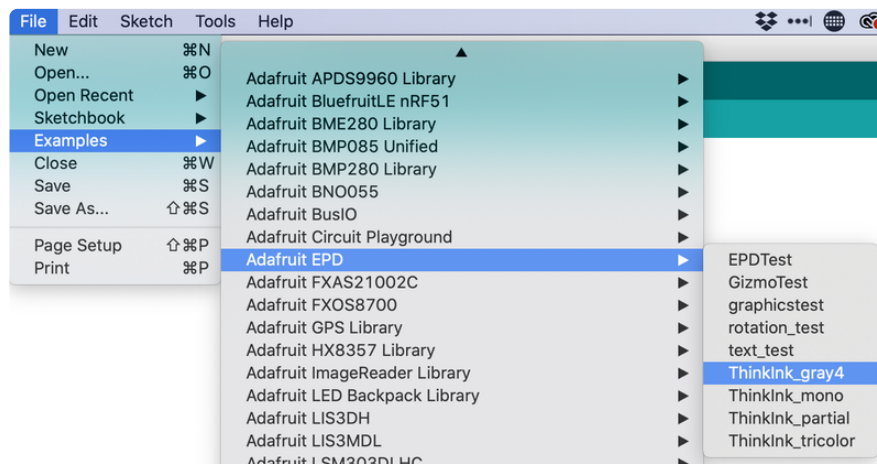


## Adafruit 2.9" Grayscale eInk / ePaper Display FeatherWing

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...

<https://www.adafruit.com/product/4777>

Open up **File→Examples→Adafruit\_EPD→ThinkInk\_gray4**



## Configure Pins



the FeatherWing you must be sure both EPD\_RESET and EPD\_BUSY are to -1 since neither of these lines are connected or the E-Ink will not update.

No matter what display you have, you will need to verify that your pins match your wiring. 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)
```

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

If you wired the display differently than on the wiring page, adjust the pin numbers accordingly.

## Configure Display Type & Size

Find the part of the script where you can pick which display is going to be used. The elnk displays are made up a combination of a Chipset and a Film in different sizes. Adafruit has narrowed it down to just a few choices between the size of the display, chipset, and film based on available combinations. In the sketch, it is sorted by size, so it's easy to find your display.

You will need to uncomment the appropriate initializer and and leave any other type commented:

- For the [2.9" 296x128 Monochrome Flexible Display](http://adafru.it/4262) (<http://adafru.it/4262>), you will use the `ThinkInk_290_Mono_M06` display initializer.
- For the **SSD1680** chipset version of the [2.9" 296x128 Tri-Color breakout](http://adafru.it/1028) (<http://adafru.it/1028>) or [2.9" 296x128 Tri-Color FeatherWing](http://adafru.it/4778) (<http://adafru.it/4778>), you will use the `ThinkInk_290_Tricolor_Z94` display initializer.
- For the **UC8151D** chipset version of the [2.9" 296x128 Tri-Color breakout](http://adafru.it/1028) (<http://adafru.it/1028>), you will use the `ThinkInk_290_Tricolor_Z13` display initializer.
- For the **IL0373** chipset version of the [2.9" 296x128 Tri-Color breakout](http://adafru.it/1028) (<http://adafru.it/1028>) or [2.9" 296x128 Tri-Color FeatherWing](http://adafru.it/4778) (<http://adafru.it/4778>), you will use the `ThinkInk_290_Tricolor_Z10` display initializer.
- For the [2.9" 296x128 Grayscale breakout](http://adafru.it/4086) (<http://adafru.it/4086>) , you will use the `ThinkInk_290_Grayscale4_T5` display initializer.

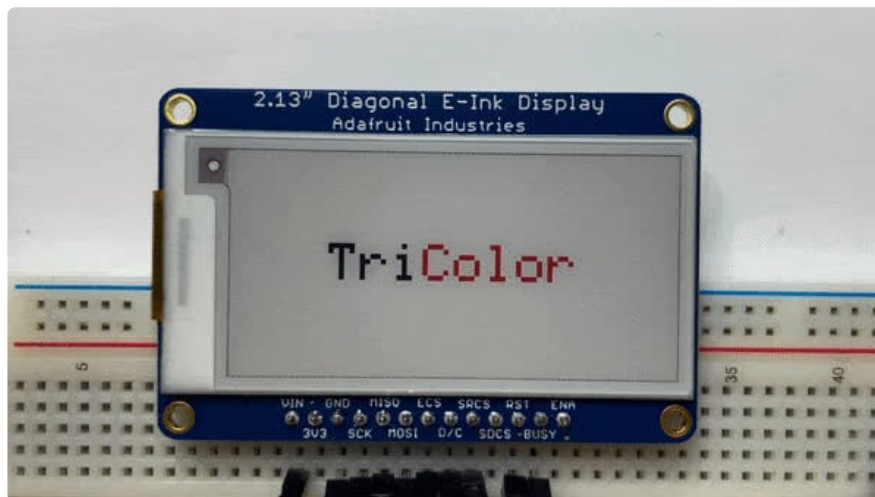
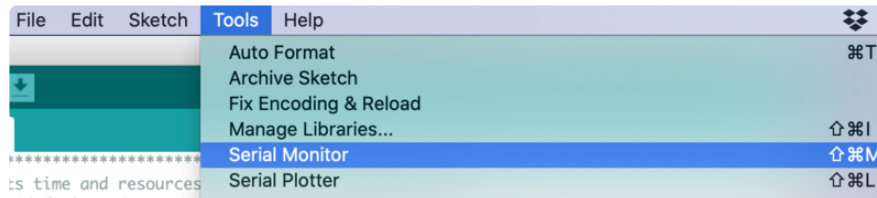
For example, for the mono 296x128 flexible display, uncomment this line, and comment any other line that is creating a ThinkInk display object

```
// 2.9" Monochrome displays with 296x128 pixels and UC8151D chipset  
//ThinkInk_290_Mono_M06 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);
```

# Upload Sketch

After checking the pinouts and the display type one more time, go ahead and upload the sketch to your board. Once it is done uploading, open the **Serial Monitor**.

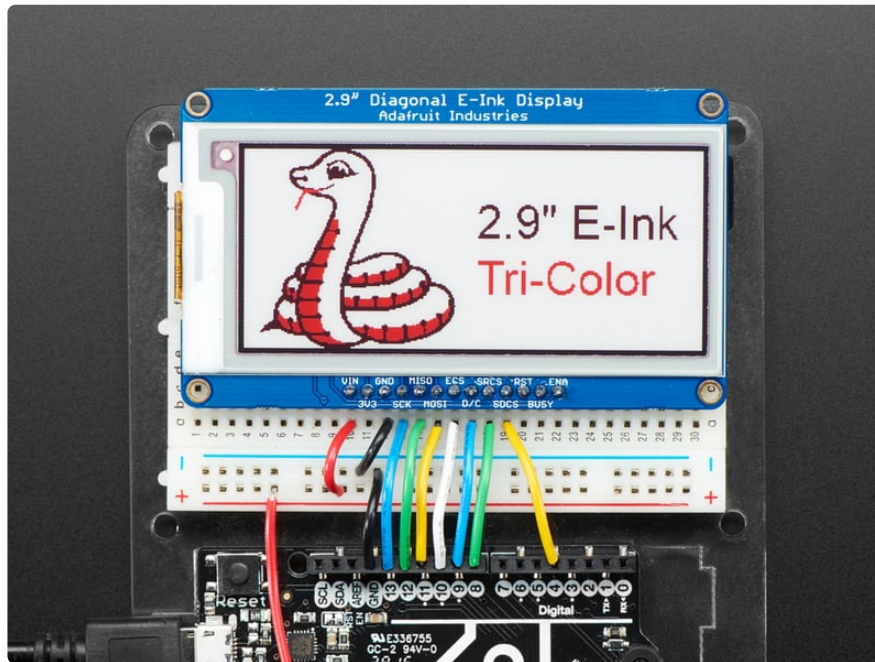
The display should start running a series of display tests





---

# Arduino Bitmaps



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

## Tri-Color and Mono Display Demo

The Tri-color and Mono demo uses a single bitmap. The Blinka bitmaps below is used for the demonstration. Select the one that is correct for your display:



<https://adafru.it/19f2>



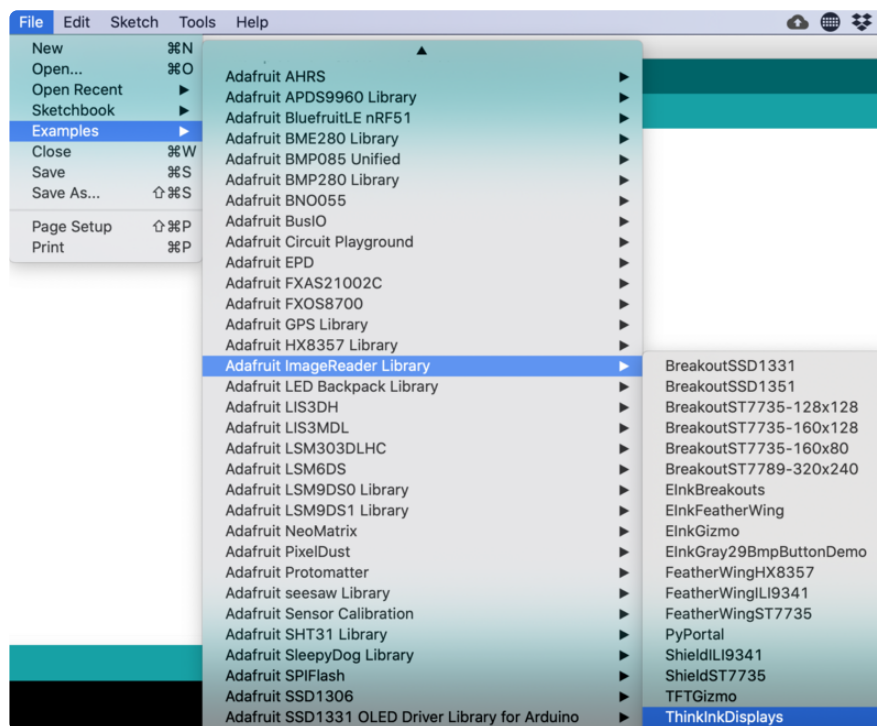
## 2.9" E-Ink Tri-Color

<https://adafru.it/19f3>

Download the **blinka.bmp** file and place it 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→ThinkInkDisplays** example



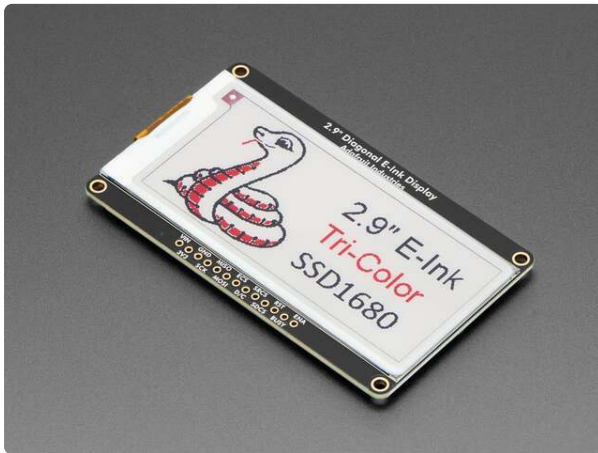
There are just a couple of things you may need to change in this file.

If you are using the FeatherWing, set **EPD\_RESET** and **EPD\_BUSY** to -1 otherwise it will wait indefinitely.



You'll need to change the initializer for your display. These options are heavily commented on the previous [Arduino Usage page \(https://adafru.it/1alq\)](https://adafru.it/1alq) in this guide.

Upload to your board and you should see an image of Blinky appear.

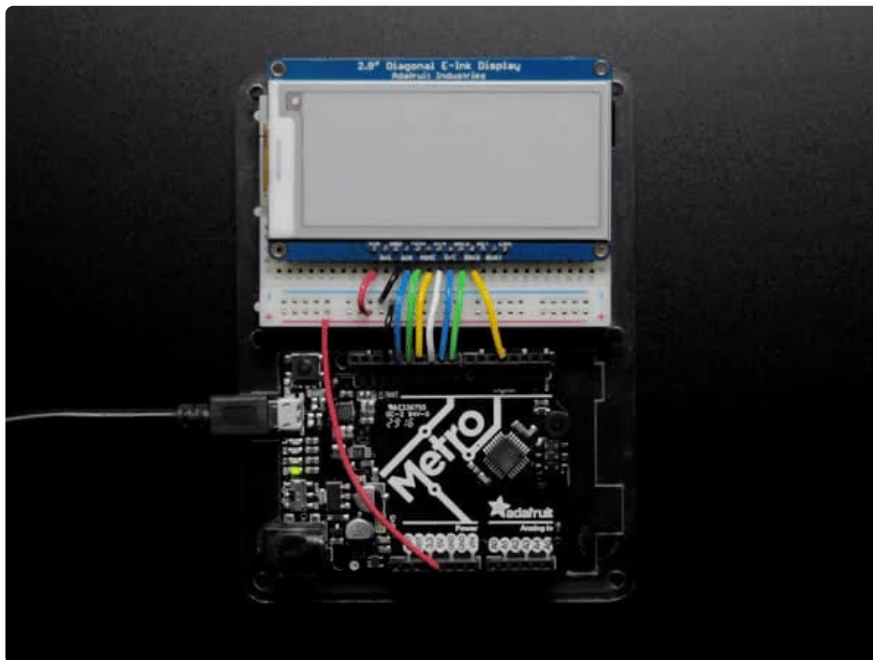


Adafruit 2.9" eInk Display Breakouts and FeatherWings

By M. LeBlanc-Williams

▮ [Arduino Usage](https://adafru.it/1alq)

<https://learn.adafruit.com/adafruit-2-9-eink-display-breakouts-and-featherwings/arduino-usage>

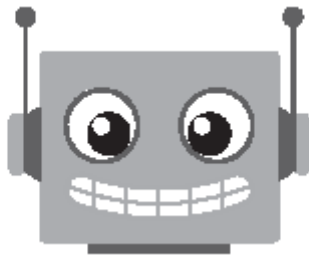


## GrayScale Display Demo

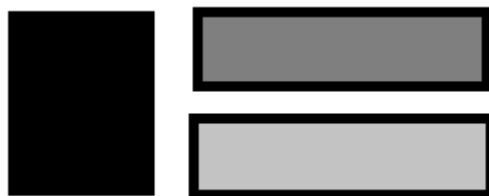
The 2.9" Grayscale display can show a max of 296x128 pixels. The grayscale demo uses three bitmaps. These bitmaps may be used for fun in the demo:



<https://adafru.it/OdE>



<https://adafru.it/OdF>

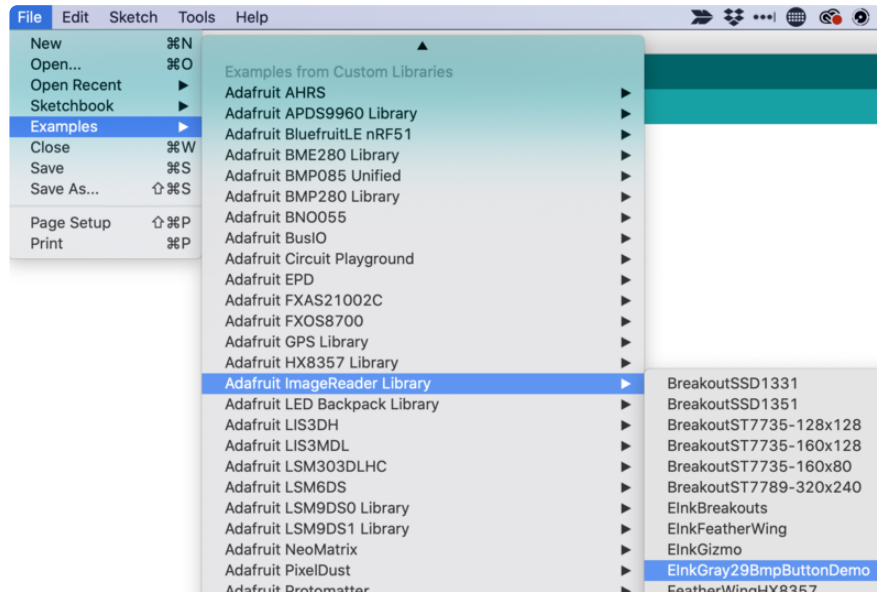


<https://adafru.it/OdG>

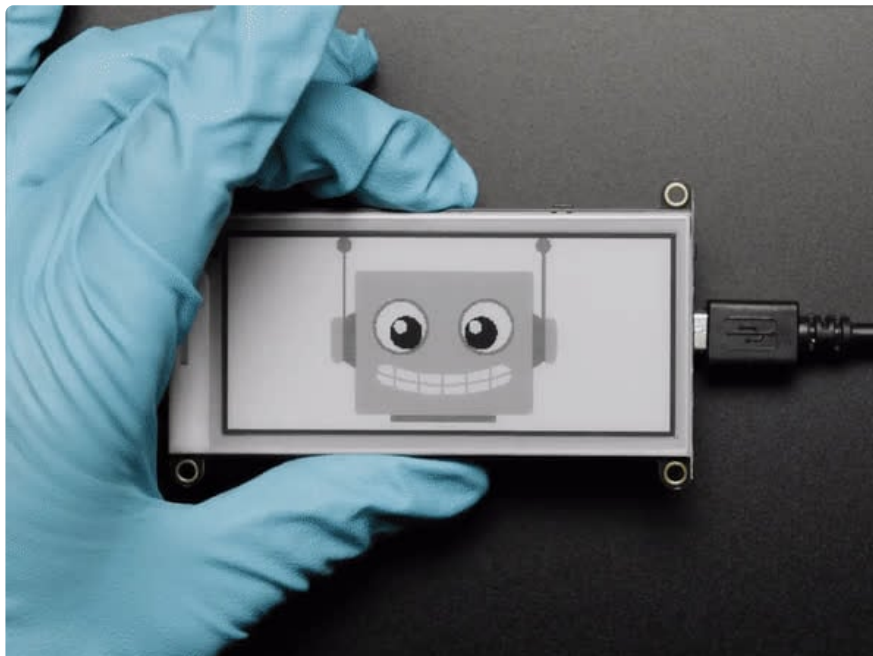
Rename the files to **panda\_head.bmp**, **adabot\_head.bmp**, and **29gray4.bmp** and place them into the base directory of a microSD card and insert it into the microSD socket in the breakout.

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.



If you want to later use your own image, use an image editing tool and crop your image to no larger than 296 pixels wide and 128 pixels high. Save it as a 24-bit color

**BMP** file - it must be 24-bit color format to work, even if it was originally a 16-bit color image - because of the way BMPs are stored and displayed!

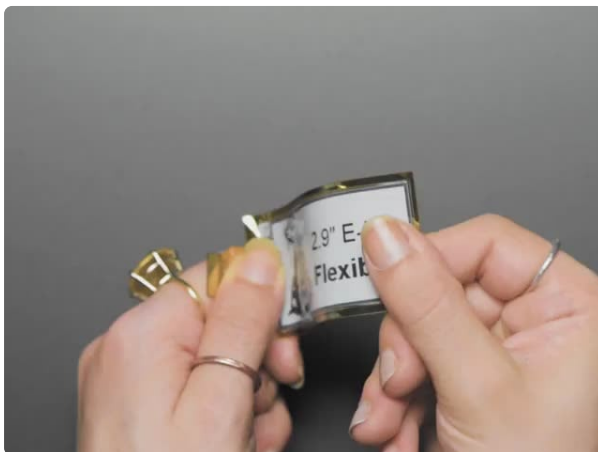
---

## CircuitPython Monochrome Usage



There is where the differences in the tri-color/monochrome and chipset/resolutions start mattering. Check carefully to make sure you are running the right example and creating the matching library type for your display or you won't see anything happen on the EPD (or the image may be really weird looking).

This page goes over how to use the **2.9" Monochrome eInk** displays with CircuitPython.



### [2.9" Flexible 296x128 Monochrome eInk / ePaper Display](https://www.adafruit.com/product/4262)

Woah, the cyber-future is here! Flexible E-Ink has been demo'd at high-tech events for years but now you can actually get your paws on it. This display is true E-Ink / E-Paper,...

<https://www.adafruit.com/product/4262>

## 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) (<https://adafru.it/Amd>) for your board. You will need the latest version of CircuitPython.

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) (<https://adafru.it/zdx>). The introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<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 **lib** folder on your **CIRCUITPY** drive. The displayio driver contains the initialization

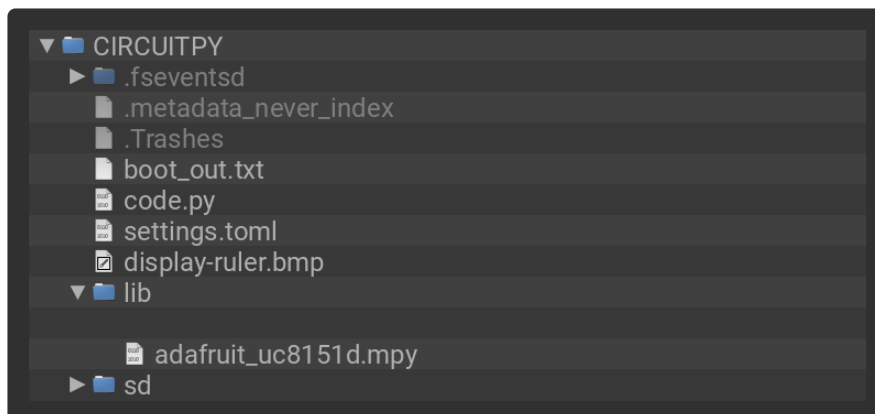
codes specific to your display that are needed to for it to work. Since there is more than one driver, you will need to copy the correct file over. Here is a list of each of the displays and the correct driver for that display.



se the elnk displays with displayio, you will need to use the latest version ircuitPython and a board that can fit **displayio** . See the Support Matrix determine if **displayio** is available on a given board: [https://circuitpython.readthedocs.io/en/latest/shared-bindings/support\\_matrix.html](https://circuitpython.readthedocs.io/en/latest/shared-bindings/support_matrix.html)

## Adafruit\_CircuitPython\_UC8151D

The flexible monochrome displays use the **Adafruit\_CircuitPython\_UC8151D** library. To easily get all the required files and libraries, you can click the **Download Project Bundle** link at the top of the example code below. Just unzip, open the folder that corresponds to the version of CircuitPython you have installed, and copy the contents to the **CIRCUITPY** drive.



## Image File

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

<https://adafru.it/Tsa>

Copy **display-ruler.bmp** into the root directory of your **CIRCUITPY** drive.



# Monochrome Display Usage

In the examples folder for your UC8151C displayio driver, there should be a test for your display which is listed here:

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2021 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

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

Supported products:
* Adafruit Flexible 2.9" Monochrome
* https://www.adafruit.com/product/4262
"""
# pylint: disable=no-member

import time

import board
import displayio
from fourwire import FourWire

import adafruit_uc8151d

displayio.release_displays()

# This pinout works on a Feather M4 and may need to be altered for other boards.
spi = board.SPI() # Uses SCK and MOSI
epd_cs = board.D9
epd_dc = board.D10
epd_reset = board.D5
epd_busy = None

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

display = adafruit_uc8151d.UC8151D(
    display_bus, width=296, height=128, rotation=90, busy_pin=epd_busy
)

g = displayio.Group()

pic = displayio.OnDiskBitmap("/display-ruler.bmp")
t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
g.append(t)

# Place the display group on the screen
display.root_group = 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)
```

## Configure and Upload

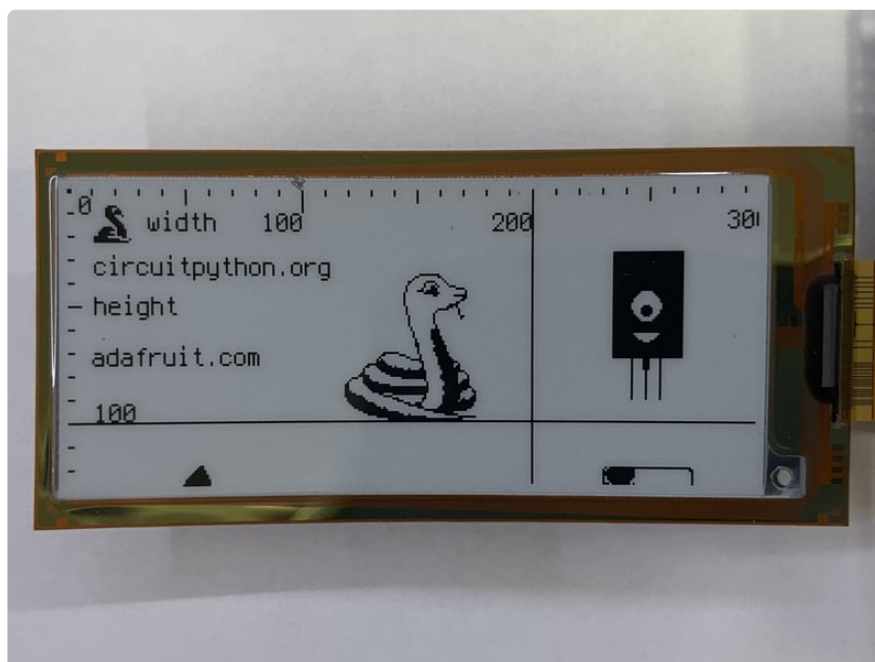
You will want to change the `epd_reset` and `epd_busy` to **the correct values**. If you are using the [eInk Breakout Friend](http://adafru.it/4224) (<http://adafru.it/4224>) and wired it up as shown on the Wiring page, you will want to change it to these values:

```
epd_reset = board.D8
epd_busy  = board.D7
```

If you are using the [eInk FeatherWing Friend](http://adafru.it/4446) (<http://adafru.it/4446>), you will want to change both of these values to **None**:

```
epd_reset = None
epd_busy  = None
```

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



---

# CircuitPython Grayscale Usage



There is where the differences in the tri-color/monochrome and chipset/ dimensions start mattering. Check carefully to make sure you are running the right example and creating the matching library type for your display or you won't see anything happen on the EPD (or the image may be really weird looking).

This page goes over how to use the **2.9" Grayscale eInk** displays with CircuitPython.



## [Adafruit 2.9" Grayscale eInk / ePaper Display 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...

<https://www.adafruit.com/product/4777>

## 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) (<https://adafru.it/Amd>) for your board. You will need the latest version of CircuitPython.

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) (<https://adafru.it/zdx>). The introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<https://adafru.it/ABU>) for both Express and non-Express boards.

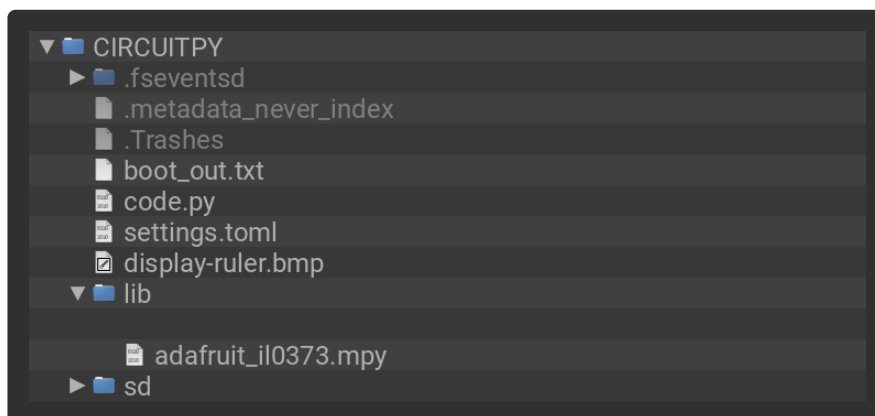
You will need to copy the appropriate displayio driver from the bundle **lib** folder to a **lib** folder on your **CIRCUITPY** drive. The displayio driver contains the initialization codes specific to your display that are needed for it to work. Since there is more than one driver, you will need to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



se the elnk displays with displayio, you will need to use the 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](https://circuitpython.readthedocs.io/en/latest/shared-bindings/support_matrix.html)

## Adafruit\_CircuitPython\_IL0373

The GrayScale FeatherWings use the **Adafruit\_CircuitPython\_IL0373** library. To easily get all the required files and libraries, you can click the **Download Project Bundle** link at the top of the example code below. Just unzip, open the folder that corresponds to the version of CircuitPython you have installed, and copy the contents to the **CIRCUITPY** drive.



## Image File

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

<https://adafru.it/Tsa>

Copy **display-ruler.bmp** into the root directory of your **CIRCUITPY** drive.

# Grayscale Display Usage

In the examples folder for your IL0373 displayio driver, there should be a test for your display which is 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 board
import busio
import displayio
import fourwire

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 = fourwire.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()

pic = displayio.OnDiskBitmap("/display-ruler.bmp")
t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
g.append(t)

display.root_group = g

display.refresh()

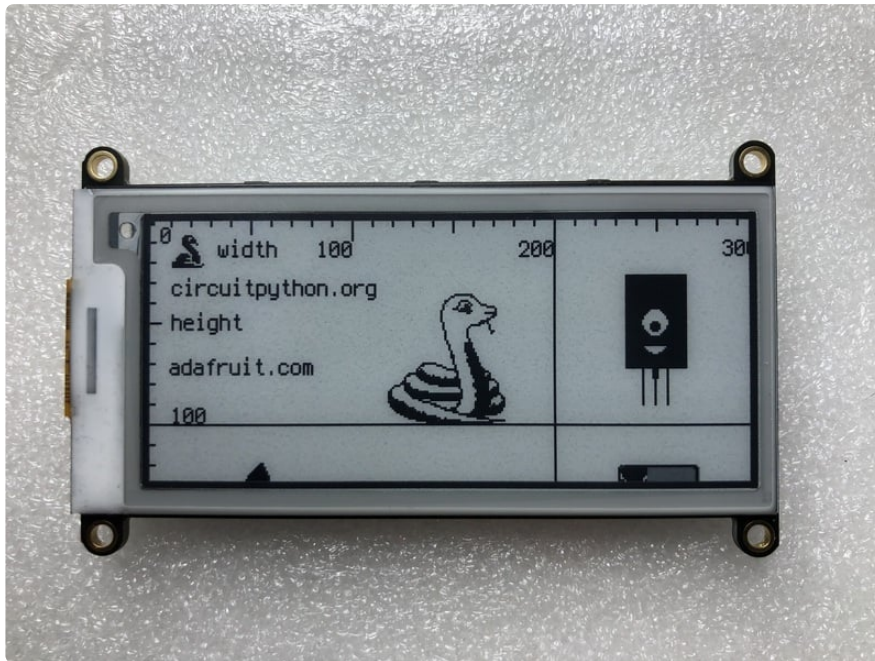
print("refreshed")
time.sleep(120)
```

You will want to change the `epd_reset` and `epd_busy` to the correct values. If you wired it up as shown on the Wiring page, you will want to change it to these values:



```
epd_reset = board.D8
epd_busy = board.D7
```

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



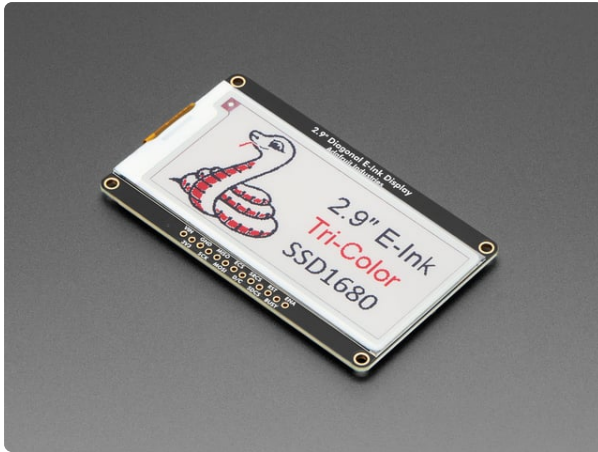
## CircuitPython Tri-Color Usage

! Here is where the differences in the tri-color/monochrome and chipset/ dimensions start mattering. Check carefully to make sure you are running the right example and creating the matching library type for your display or you won't see anything happen on the EPD (or the image may be really weird looking).

This page goes over how to use the **2.9" Tri-Color eInk** displays with CircuitPython.

### 2.9" Tri-Color eInk Display and FeatherWing Revision

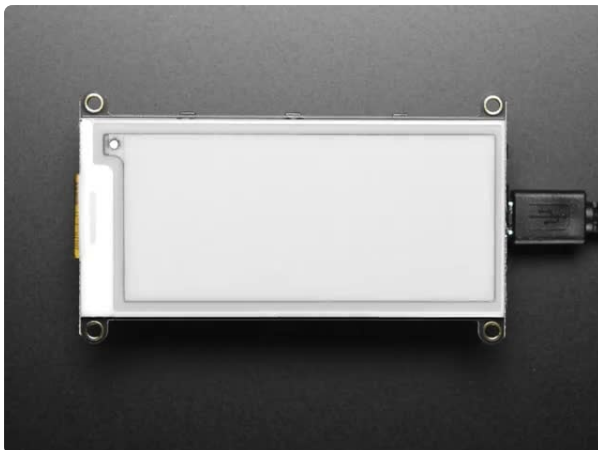
- As of June 20, 2025 - The tri-color display and FeatherWing have been updated (old ones discontinued) to now use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!



### Adafruit 2.9" Red/Black/White eInk Display Breakout - THINK INK

Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those...

<https://www.adafruit.com/product/1028>



### Adafruit 2.9" Tri-Color eInk / ePaper Display FeatherWing

Easy e-paper comes to your Feather with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those...

<https://www.adafruit.com/product/4778>

## 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) (<https://adafru.it/Amd>) for your board. You will need the latest version of CircuitPython.

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) (<https://adafru.it/zdx>). The introduction guide has [a great page on how to install the library bundle](https://adafru.it/ABU) (<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 **lib** 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.



se the elnk displays with displayio, you will need to use the 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](https://circuitpython.readthedocs.io/en/latest/shared-bindings/support_matrix.html)

## Image File

All of the examples below use the same bitmap image. You'll need the **display-ruler.bmp** bitmap file on your CIRCUITPY drive.

<https://adafru.it/Tsa>

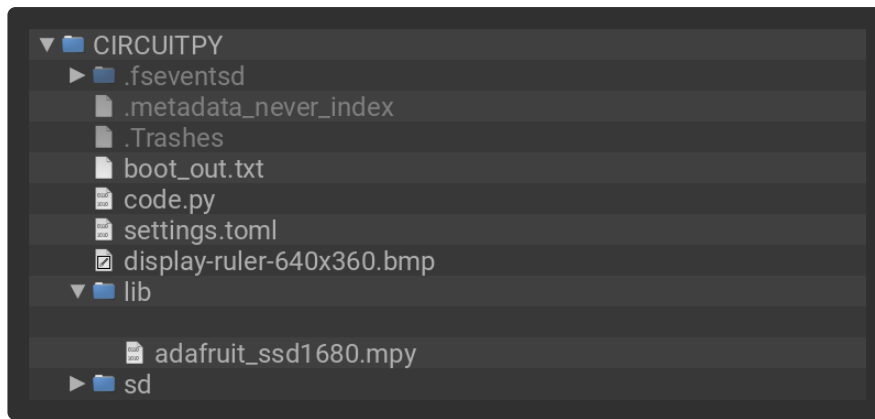
Copy **display-ruler.bmp** into the root directory of your **CIRCUITPY** drive.

**Adafruit\_CircuitPython\_SSD1680**



As of June 20, 2025 - The displays have been updated to use the SSD1680

As of June 20, 2025, the 2.9" Tri-Color display and FeatherWing use the **Adafruit\_CircuitPython\_SSD1680** library. To easily get all the required files and libraries, you can click the **Download Project Bundle** link at the top of the example code below. Just unzip, open the folder that corresponds to the version of CircuitPython you have installed, and copy the contents to the **CIRCUITPY** drive.



## Display Usage

```
# SPDX-FileCopyrightText: 2017 Scott Shawcroft, written for Adafruit Industries
# SPDX-FileCopyrightText: Copyright (c) 2023 Jose D. Montoya
#
# SPDX-License-Identifier: Unlicense

"""Simple test script for Adafruit 2.9" Tri-Color eInk Display Breakout
Supported products:
* Adafruit 2.9" Tri-Color eInk Display Breakout
* https://www.adafruit.com/product/1028
"""

import time

import board
import displayio
from fourwire import FourWire

import adafruit_ssd1680

displayio.release_displays()

# This pinout works on a Metro M4 and may need to be altered for other boards.
spi = board.SPI() # Uses SCK and MOSI
epd_cs = board.D9
epd_dc = board.D10
epd_reset = board.D5
epd_busy = board.D6

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

display = adafruit_ssd1680.SSD1680(
    display_bus,
    width=296,
    height=128,
    highlight_color=0xFF0000,
    rotation=270,
)

g = displayio.Group()

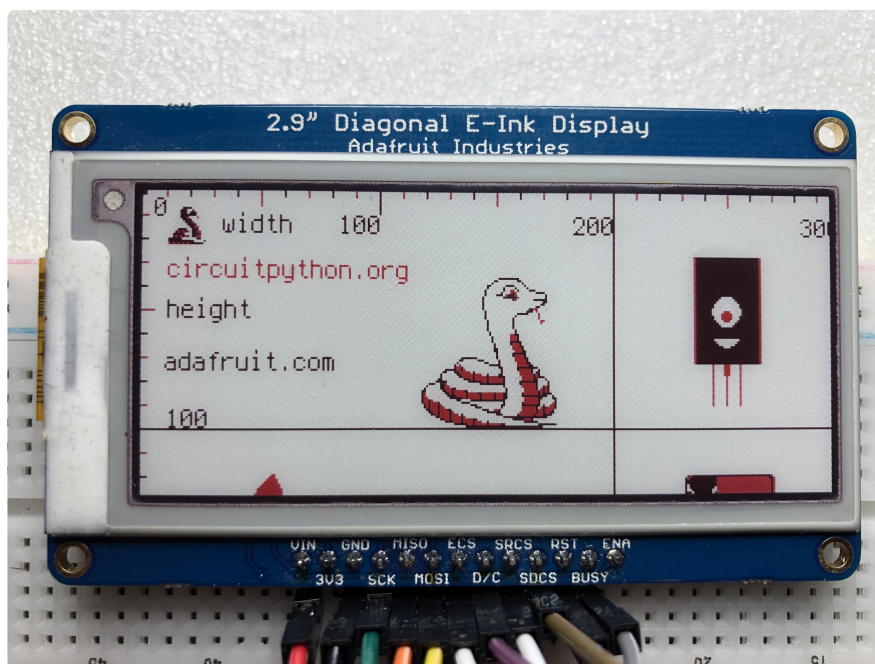
pic = displayio.OnDiskBitmap("/display-ruler-640x360.bmp")
t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
```

```
g.append(t)
display.root_group = g
display.refresh()
print("refreshed")
time.sleep(120)
```

For the FeatherWing, you will want to change the `epd_reset` and `epd_busy` values to `None`:

```
epd_reset = None
epd_busy = None
```

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



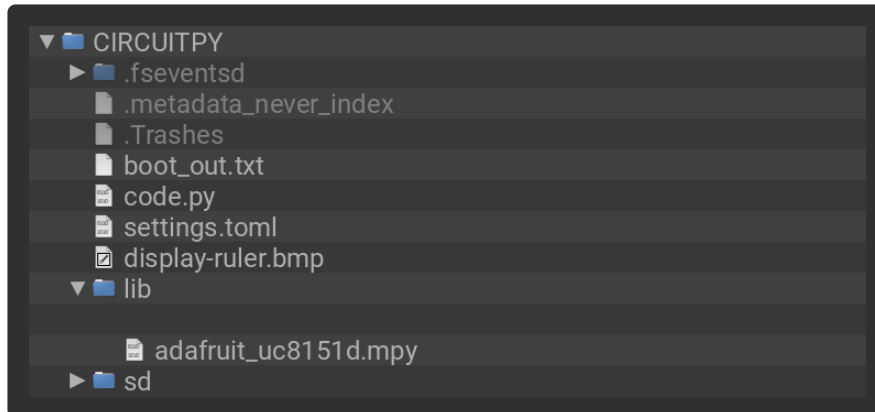
## Adafruit\_CircuitPython\_UC8151D



Between November 16, 2022 and June 20, 2025 the 2.9" Tri-Color elnk lay used the UC8151D chipset.



If you purchased the 2.9" Tri-Color elnk display between of November 16, 2022 and June 20, 2025, the display has the UC8151D chipset and uses the **Adafruit\_CircuitPython\_UC8151D** library. To easily get all the required files and libraries, you can click the **Download Project Bundle** link at the top of the example code below. Just unzip, open the folder that corresponds to the version of CircuitPython you have installed, and copy the contents to the **CIRCUITPY** drive.



## UC8151D Display Usage

In the examples folder for your UC8151D displayio driver, there should be a test for your display which is listed here:

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

"""Simple test script for Adafruit 2.9" 296x128 tri-color display
Supported products:
* Adafruit 2.9" Tri-Color Display Breakout
* https://www.adafruit.com/product/1028
"""

import time

import board
import displayio
from fourwire import FourWire

import adafruit_uc8151d

# 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
spi = board.SPI() # Uses SCK and MOSI
epd_cs = board.D9
epd_dc = board.D10
epd_reset = board.D5
epd_busy = board.D6

# Create the displayio connection to the display pins
display_bus = FourWire(spi, command=epd_dc, chip_select=epd_cs, reset=epd_reset,
baudrate=1000000)
```

```

time.sleep(1) # Wait a bit

# Create the display object - the third color is red (0xff0000)
display = adafruit_uc8151d.UC8151D(
    display_bus,
    width=296,
    height=128,
    rotation=270,
    busy_pin=epd_busy,
    highlight_color=0xFF0000,
)

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

# Display a ruler graphic from the root directory of the CIRCUITPY drive
pic = displayio.OnDiskBitmap("/display-ruler.bmp")
t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
g.append(t)

# Place the display group on the screen
display.root_group = 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)

```

You will want to change the `epd_reset` and `epd_busy` to the correct values. If you wired it up as shown on the Wiring page, you will want to change it to these values:

```

epd_reset = board.D8
epd_busy = board.D7

```

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

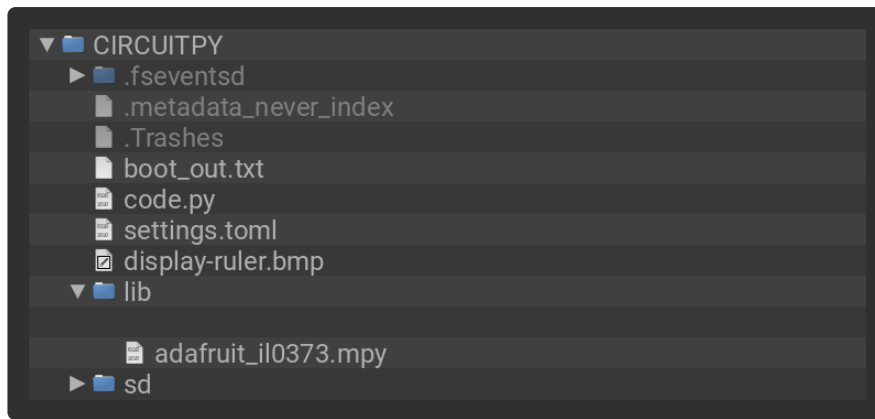


## Adafruit\_CircuitPython\_IL0373



- Before November 16, 2022, the 2.9" Tri-Color elnk display used the IL0373 chipset.
- Before June 24, 2025, the 2.9" Tri-Color elnk FeatherWing used the IL0373 chipset.

If you purchased the 2.9" Tri-Color elnk display before November 16, 2022, or the FeatherWing before June 20, 2025, they used the IL0373 chipset with the **Adafruit\_CircuitPython\_IL0373** library. To easily get all the required files and libraries, you can click the **Download Project Bundle** link at the top of the example code below. Just unzip, open the folder that corresponds to the version of CircuitPython you have installed, and copy the contents to the **CIRCUITPY** drive.



## IL0373 Display Usage

In the examples folder for your IL0373 displayio driver, there should be a test for your display which is listed here:

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

"""Simple test script for Adafruit 2.9" 296x128 tri-color display
Supported products:
* Adafruit 2.9" Tri-Color Display Breakout
* https://www.adafruit.com/product/1028
"""

import time

import board
import displayio
import fourwire

import adafruit_il0373

# 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
spi = board.SPI() # Uses SCK and MOSI
epd_cs = board.D9
epd_dc = board.D10
epd_reset = board.D5
epd_busy = board.D6

# Create the displayio connection to the display pins
display_bus = fourwire.FourWire(
    spi, command=epd_dc, chip_select=epd_cs, reset=epd_reset, baudrate=1000000
)
time.sleep(1) # Wait a bit

# Create the display object - the third color is red (0xff0000)
display = adafruit_il0373.IL0373(
    display_bus,
    width=296,
    height=128,
    rotation=270,
    busy_pin=epd_busy,
    highlight_color=0xFF0000,
)
```

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

# Display a ruler graphic from the root directory of the CIRCUITPY drive
pic = displayio.OnDiskBitmap("/display-ruler.bmp")
t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
g.append(t)

display.root_group = g

display.refresh()

print("refreshed")
time.sleep(180)
```

For the FeatherWing, you will want to change the `epd_reset` and `epd_busy` values to `None`:

```
epd_reset = None
epd_busy = None
```

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



---

# Python Setup

It's easy to use elnk breakouts with Python and the [Adafruit CircuitPython EPD](https://adafruit.it/BTd) (<https://adafruit.it/BTd>) library. This library allows you to easily write Python code to control the display.

Since there are 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://adafruit.it/BSN) (<https://adafruit.it/BSN>).



! this is not a kernel driver that will let you have the console appear on the . However, this is handy when you want to use the elnk display purely i 'user Python' code!



! can only use this technique with Linux/computer devices that have lware SPI support, and not all single board computers have an SPI device, heck before continuing

## Setup Virtual Environment

If you are installing on the Bookworm version of Raspberry Pi OS or later, you will need to install your python modules in a virtual environment. You can find more information in the [Python Virtual Environment Usage on Raspberry Pi](https://adafruit.it/19a5) (<https://adafruit.it/19a5>) guide. To Install and activate the virtual environment, use the following commands:

```
sudo apt install python3-venv
python -m venv env --system-site-packages
```

To activate the virtual environment:

```
source env/bin/activate
```



## Install Adafruit\_Blinka

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 \(https://adafru.it/BSN\)](https://adafru.it/BSN)!

## Python Installation of EPD Library

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

```
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.

<https://adafru.it/Xbr>

Alternatively, you can use **wget** to directly download the file to your pi:

```
wget https://github.com/adafruit/Adafruit_CircuitPython_framebuf/raw/main/examples/font5x8.bin
```

## 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
```

## Chip Enable Lines

Follow [these instructions](https://adafru.it/19fg) (<https://adafru.it/19fg>) for dealing with SPI chip enable line issues.

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

---

## Python Usage



⚠ this is not a kernel driver that will let you have the console appear on the . However, this is handy when you want to use the elnk display purely in 'user Python' code!



⚠ can only use this technique with Linux/computer devices that have I2C SPI support, and not all single board computers have an SPI device, check before continuing

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. You set the SRAM CS pin to **None** because the Raspberry Pi has lots of RAM, so you don't really need it.

```
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
```



Depending on the exact E-Ink display you're using, the driver and object initialization will differ a bit because Python must be told what chip driver to use and the size of the display.

Run the following code to initialize the 2.9" 296x128 Flexible Monochrome display:



#### 2.9" Flexible 296x128 Monochrome eInk / ePaper Display

Woah, the cyber-future is here! Flexible E-Ink has been demo'd at high-tech events for years but now you can actually get your paws on it. This display is true E-Ink / E-Paper,...

<https://www.adafruit.com/product/4262>

```
from adafruit_epd.uc8151d import Adafruit_UC8151D
display = Adafruit_UC8151D(128, 296, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
rst_pin=rst, busy_pin=busy)
display.set_black_buffer(1, True)
display.set_color_buffer(1, True)
```

If you have one of the older flexible displays, you can use the following code to initialize it:

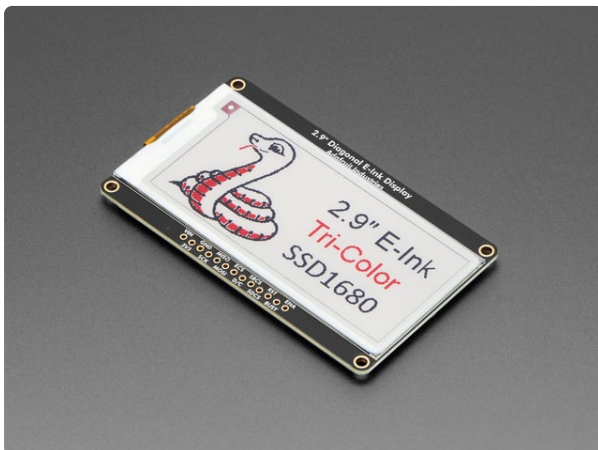
```
from adafruit_epd.il0373 import Adafruit_IL0373
display = Adafruit_IL0373(128, 296, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srscs,
rst_pin=rst, busy_pin=busy)
display.set_black_buffer(1, False)
display.set_color_buffer(1, False)
```

For the 2.9" 296x128 Tri-Color display:



## 2.9" Tri-Color eInk Display and FeatherWing Revision

- As of June 20, 2025 - The tri-color display and FeatherWing have been updated (old ones discontinued) to now use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!



### [Adafruit 2.9" Red/Black/White eInk Display Breakout - THINK INK](https://www.adafruit.com/product/1028)

Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those...

<https://www.adafruit.com/product/1028>

IL0373 Chipset:

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

SSD1680 Chipset:

```
display = Adafruit_SSD1680(128, 296, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srscs,
rst_pin=rst, busy_pin=busy)
```

## Monochrome Example

Now to clear the screen buffer and draw some shapes. Once done drawing, the code must tell the screen to update using the `display()` method.

```
display.rotation = 3
display.fill(Adafruit_EPDM.WHITE)

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

display.display()
```

## Tri-Color Example

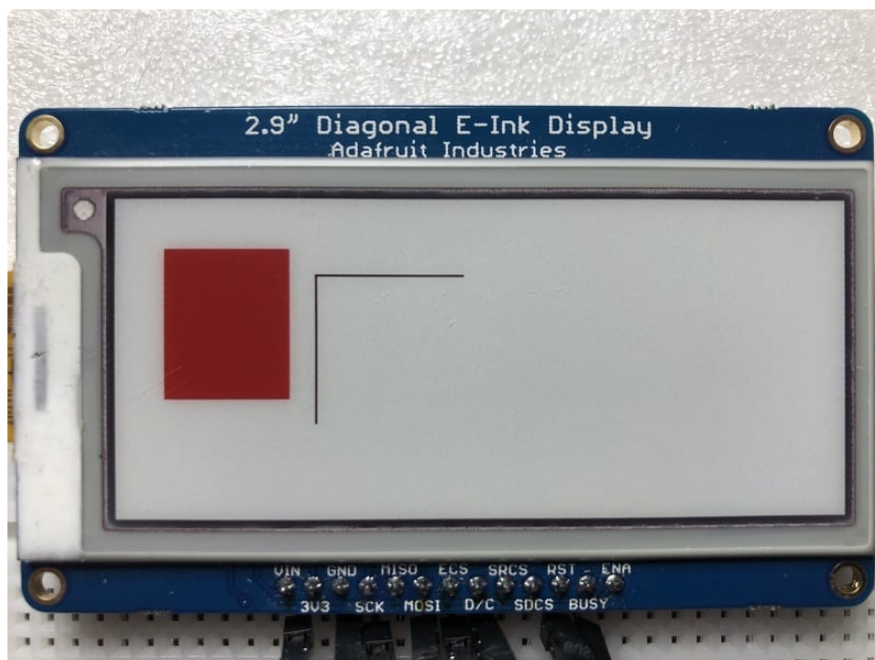
The Tri-Color example is almost the same as the monochrome example, except another color is added in. Once done drawing, the code needs to tell the screen to update using the `display()` method.

```
display.rotation = 3
display.fill(Adafruit_EPDM.WHITE)

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

display.display()
```

Your display will look something like this:



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

## Full Example Code

Here is the full example code.



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

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

import board
import busio
import digitalio

from adafruit_epd.ek79686 import Adafruit_EK79686
from adafruit_epd.epd import Adafruit_EPd
from adafruit_epd.il0373 import Adafruit_IL0373, Adafruit_IL0373_213_Flex_Mono
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.jd79661 import Adafruit_JD79661
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from adafruit_epd.ssd1680b import Adafruit_SSD1680B
from adafruit_epd.ssd1681 import Adafruit_SSD1681
from adafruit_epd.ssd1683 import Adafruit_SSD1683
from adafruit_epd.uc8151d import Adafruit_UC8151D
from adafruit_epd.uc8179 import Adafruit_UC8179

# 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_JD79661(122, 150, # 2.13" Quad-color 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_SSD1680B(122, 250 # 2.13" HD (Tri-color or mono) with
GDEY0213B74
# display = Adafruit_SSD1681(200, 200, # 1.54" 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_EK79686(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_UC8179(648, 480, # 5.83" mono 648x480 display
# display = Adafruit_UC8179(800, 480, # 7.5" mono 800x480 display
```



```

# display = Adafruit_IL0373(128, 296, # 2.9" Tri-color display IL0373
# display = Adafruit_IL0373_213_Flex_Mono(104, 212, # 2.13" mono flex display
# display = Adafruit_SSD1680(128, 296, # 2.9" Tri-color display SSD1680
# display = Adafruit_SSD1683(400, 300, # 4.2" 300x400 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,
)
""" display = Adafruit_UC8179(800, 480, # 7.5" tricolor 800x480 display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
    tri_color = True
)"""

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY OR!
# UC8179 5.83" or 7.5" monochrome displays
# 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
if type(display) == Adafruit_JD79661:
    WHITE = Adafruit_JD79661.WHITE
    BLACK = Adafruit_JD79661.BLACK
    RED = Adafruit_JD79661.RED
    YELLOW = Adafruit_JD79661.YELLOW
else:
    WHITE = Adafruit_EPD.WHITE
    BLACK = Adafruit_EPD.BLACK
    RED = Adafruit_EPD.RED

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

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

print("Draw lines")
if type(display) == Adafruit_JD79661:
    display.line(0, 0, display.width - 1, display.height - 1, YELLOW)
    display.line(0, display.height - 1, display.width - 1, 0, YELLOW)
else:
    display.line(0, 0, display.width - 1, display.height - 1, BLACK)
    display.line(0, display.height - 1, display.width - 1, 0, RED)

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

```

# 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.** The image below will be used on the 2.9" display. Click the button below to download the image and save it as **blinka.bmp** on your **Raspberry Pi**. The code uses a Tri-Color bitmap, but it should still work on a monochrome display.

<https://adafru.it/19f8>

Save the following code to your Raspberry Pi as **epd\_bitmap.py**.

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

import board
import busio
import digitalio

from adafruit_epd.ek79686 import Adafruit_EK79686
from adafruit_epd.epd import Adafruit_EPd
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from adafruit_epd.ssd1680b import Adafruit_SSD1680B
from adafruit_epd.ssd1681 import Adafruit_SSD1681
from adafruit_epd.ssd1683 import Adafruit_SSD1683
from adafruit_epd.uc8151d import Adafruit_UC8151D
from adafruit_epd.uc8179 import Adafruit_UC8179

# 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_SSD1680B(122, 250 # Newer 2.13" HD (Tri-color or mono)
with GDEY0213B74
# display = Adafruit_SSD1681(200, 200, # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264, # 2.7" Tri-color display
# display = Adafruit_EK79686(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_UC8179(648, 480, # 5.83" mono 648x480 display
# display = Adafruit_UC8179(800, 480, # 7.5" mono 800x480 display
# display = Adafruit_IL0373(128, 296, # 2.9" Tri-color display IL0373
```

```

# display = Adafruit_SSD1680(128, 296,          # 2.9" Tri-color display SSD1680
# display = Adafruit_SSD1683(400, 300,          # 4.2" 300x400 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,
)
""" display = Adafruit_UC8179(800, 480,          # 7.5" tricolor 800x480 display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
    tri_color = True
)"""

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY OR!
# UC8179 5.83" or 7.5" displays
# 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 writing, 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):
    try:
        f = open(filename, "rb")
    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, you will need to change a few pin definitions. 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)
```

Next, find the section that looks like this:

```
# 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,
)
```

Comment out these lines:

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

and **uncomment** the line that corresponds with your display.

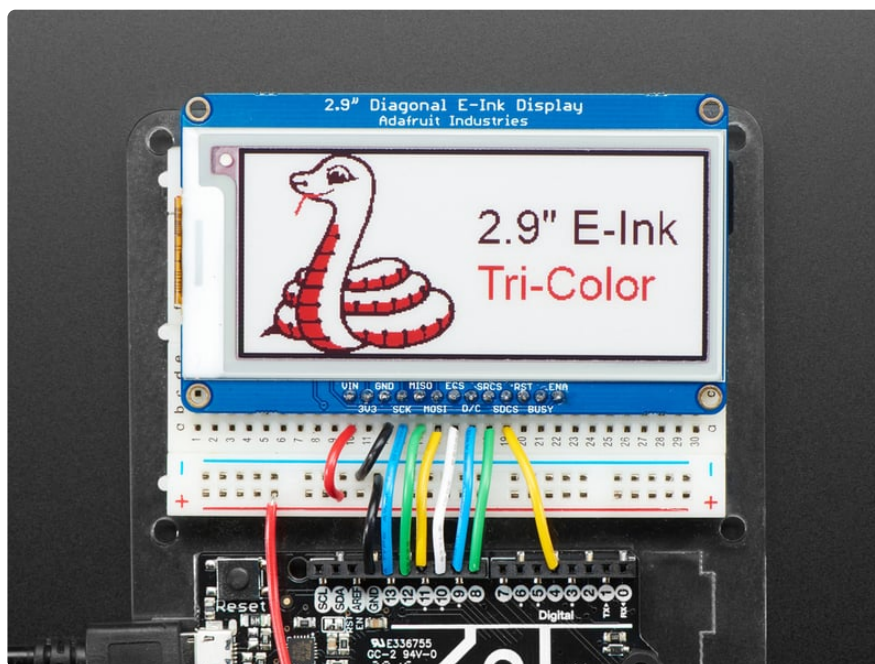
Next to tell the display the rotation setting desired. This can be a value between 0-3. For the 2.13" displays, a value of 3 seems to work well.

```
display.rotation = 3
```

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

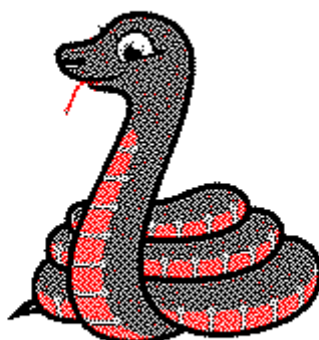
```
python3 epd_bitmap.py
```

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



## Image Drawing with Pillow

This example will use Pillow to resize and crop the image automatically and draw it on 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. This uses a PNG format file because it is a lossless format and won't introduce unexpected colors on the display.





Make sure you save it as **blinka.png** and place it in the same folder as your script. Here's the code to load onto the Raspberry Pi. Go ahead and copy it onto your Raspberry Pi and save it as **epd\_pillow\_image.py**.

```
# 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 board
import busio
import digitalio
from PIL import Image

from adafruit_epd.ek79686 import Adafruit_EK79686
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.jd79661 import Adafruit_JD79661
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from adafruit_epd.ssd1680b import Adafruit_SSD1680B
from adafruit_epd.ssd1681 import Adafruit_SSD1681
from adafruit_epd.ssd1683 import Adafruit_SSD1683
from adafruit_epd.uc8151d import Adafruit_UC8151D
from adafruit_epd.uc8179 import Adafruit_UC8179

# 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_JD79661(122, 150, # 2.13" Quad-color 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 or mono display
# display = Adafruit_SSD1680B(122, 250 # Newer 2.13" HD (Tri-color or mono)
with GDEY0213B74
# display = Adafruit_SSD1681(200, 200, # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264, # 2.7" Tri-color display
# display = Adafruit_EK79686(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_UC8179(648, 480, # 5.83" mono 648x480 display
# display = Adafruit_UC8179(800, 480, # 7.5" mono 800x480 display
# display = Adafruit_IL0373(128, 296, # 2.9" Tri-color display IL0373
# display = Adafruit_SSD1680(128, 296, # 2.9" Tri-color display SSD1680
# display = Adafruit_SSD1683(400, 300, # 4.2" 300x400 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,
)
""" display = Adafruit_UC8179(800, 480,          # 7.5" tricolor 800x480 display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
    tri_color = True
)"""

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY OR!
# UC8179 5.83" or 7.5" displays
# 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")

if type(display) == Adafruit_JD79661:
    # Create a palette with the 4 colors: Black, White, Red, Yellow
    # The palette needs 768 values (256 colors x 3 channels)
    palette = []

    # We'll map the 256 palette indices to our 4 colors
    # 0-63: Black, 64-127: Red, 128-191: Yellow, 192-255: White
    for i in range(256):
        if i < 64:
            palette.extend([0, 0, 0]) # Black
        elif i < 128:
            palette.extend([255, 0, 0]) # Red
        elif i < 192:
            palette.extend([255, 255, 0]) # Yellow
        else:
            palette.extend([255, 255, 255]) # White

    # Create a palette image
    palette_img = Image.new("P", (1, 1))
    palette_img.putpalette(palette)

    # Optional: Enhance colors before dithering for better results
    # from PIL import ImageEnhance

```

```
# enhancer = ImageEnhance.Color(image)
# image = enhancer.enhance(1.5) # Increase color saturation

# Quantize the image using Floyd-Steinberg dithering
image = image.quantize(palette=palette_img, dither=Image.FLOYDSTEINBERG)

# Convert back to RGB for the display driver
image = image.convert("RGB")

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

The code starts with library 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
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from adafruit_epd.ssd1681 import Adafruit_SSD1681
from adafruit_epd.uc8151d import Adafruit_UC8151D
```

That is followed by initializing the SPI bus and defining a few pins. The choices 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)
```

These examples work on as many displays as possible with very few changes. 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_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_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
)

```

Next change the rotation setting to **3**.

```
display.rotation = 3
```

Next is to open the Blinky image, which is named **blinky.png**, and it is assumed the file 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("blinky.png")
```

Here's where it starts to get interesting. It is desirable to scale the image so that it matches either the width or height of the display, depending on which is smaller, so that there may be some of the image to chop off when it is cropped. Start by calculating the width to height ratio of both the display and the image. If the height is the closer of the dimensions, you want to match the image height to the display height and let it be a bit wider than the display. Otherwise, you want to do the opposite.

Once you've figured out how to scale it, pass in the new dimensions and using a **Bicubic** rescaling method, the code reassigns 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 elinks, but loses detail with displaying a color image on the monochrome display, so this code uses the best balance.

```

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)
```

Next to figure the starting x and y points of the image to begin cropping so that the image ends up centered. 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, 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)).convert("RGB")
```

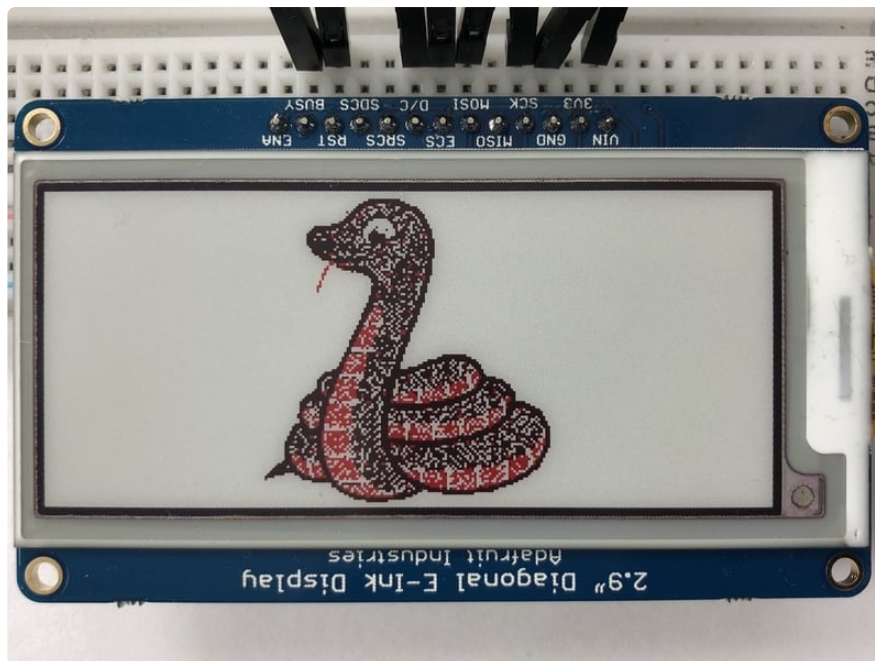
Finally, take the `image`, draw it to the frame buffer and `display` it. At this point, the image should have the exact same dimensions as the display and fill it completely.

```
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:



## Drawing Shapes and Text with Pillow

The next example takes 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 board
import busio
import digitalio
from PIL import Image, ImageDraw, ImageFont

from adafruit_epd.ek79686 import Adafruit_EK79686
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.jd79661 import Adafruit_JD79661
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from adafruit_epd.ssd1680b import Adafruit_SSD1680B
from adafruit_epd.ssd1681 import Adafruit_SSD1681
from adafruit_epd.ssd1683 import Adafruit_SSD1683
from adafruit_epd.uc8151d import Adafruit_UC8151D
from adafruit_epd.uc8179 import Adafruit_UC8179

# 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_JD79661(122, 150,
# display = Adafruit_SSD1608(200, 200,
# display = Adafruit_SSD1675(122, 250,
# display = Adafruit_SSD1680(122, 250,
# display = Adafruit_SSD1680B(122, 250
with GDEY0213B74
# display = Adafruit_SSD1681(200, 200,
# display = Adafruit_IL91874(176, 264,
# display = Adafruit_EK79686(176, 264,
# display = Adafruit_IL0373(152, 152,
# display = Adafruit_UC8151D(128, 296,
# display = Adafruit_UC8179(648, 480,
# display = Adafruit_UC8179(800, 480,
# display = Adafruit_IL0373(128, 296,
# display = Adafruit_SSD1680(128, 296,
# display = Adafruit_SSD1683(400, 300,
# display = Adafruit_IL0398(400, 300,
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,
)
""" display = Adafruit_UC8179(800, 480,
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
    tri_color = True
)"""

# IF YOU HAVE A 2.13" FLEXIBLE DISPLAY OR!
# UC8179 5.83" or 7.5" displays
# 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, use the imports, but this time include the `ImageDraw` and `ImageFont` Pillow modules to allow text rendering.

```

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
from adafruit_epd.ssd1680 import Adafruit_SSD1680
from adafruit_epd.ssd1681 import Adafruit_SSD1681
from adafruit_epd.uc8151d import Adafruit_UC8151D

```

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

```

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

```

After that, 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 it can adjust easily for different displays. You could play around with the colors as well. One thing to note is that on monochrome displays, **RED** will show up as **BLACK**.

For the 2.9" display, a **BORDER** value of **20** and a **FONTSIZE** value of **24** looks good.

```
BORDER = 10
FONTSIZE = 20
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. **Go ahead and adjust your initializer as explained in the previous example.** After that, create an **image** with the dimensions and use that to create a **draw** object. The **draw** object will have all of the drawing functions.

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

Next 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 to draw an inner rectangle using the **FOREGROUND\_COLOR**. Use the **BORDER** parameter to calculate the size and position to draw the rectangle.

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

Next to load a TTF font. The **DejaVuSans.ttf** font should come preloaded on your Pi in the location in the code. This will also make use of the **FONTSIZE** parameter discussed earlier.

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

Now to draw the text Hello World onto the center of the display. You may recognize the centering calculation was the same one used to center crop the image in the previous example. In this example though, the font size values are obtained 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, 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:



---

# WipperSnapper Wiring & Test

## What is WipperSnapper

WipperSnapper is a firmware designed to turn any WiFi-capable board into an Internet-of-Things device without programming a single line of code. WipperSnapper connects to [Adafruit IO \(https://adafru.it/fsU\)](https://adafru.it/fsU), a web platform designed ([by Adafruit! \(https://adafru.it/Bo5\)](https://adafru.it/Bo5)) to display, respond, and interact with your project's data.

Simply load the WipperSnapper firmware onto your board, add credentials, and plug it into power. Your board will automatically register itself with your Adafruit IO account.

From there, you can add components to your board such as buttons, switches, potentiometers, sensors, and more! Components are dynamically added to hardware, so you can immediately start interacting, logging, and streaming the data your projects produce without writing code.

If you've never used WipperSnapper, click below to read through the quick start guide before continuing.

<https://adafru.it/Vfd>

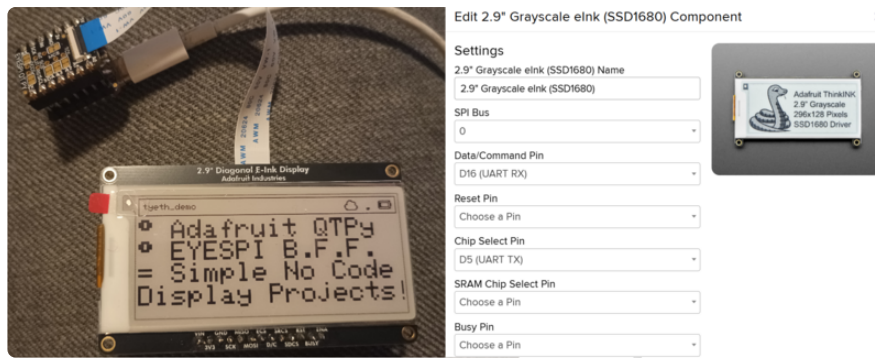
## Wiring

### EYESPI Connector

If you have one of the newer breakouts with an EYESPI connector on the back, you can use that with an 18-pin ribbon cable to connect it either directly to an adapter board such as the [EYESPI BFF for QT Py or Xiao \(http://adafru.it/5772\)](http://adafru.it/5772) or by using the [EYESPI Breakout \(http://adafru.it/5613\)](http://adafru.it/5613) and wiring that to your microcontroller.

See the [EYESPI pages \(https://adafru.it/1a1u\)](https://adafru.it/1a1u) for more information, then read the Breakout Wiring information below, and adjust that to your setup when it comes to configuring the pins in the New Component dialog.

If using the QTPy BFF, then it's as simple as selecting the **RX** pin for CS (chip select), and **TX** pin for D/C.

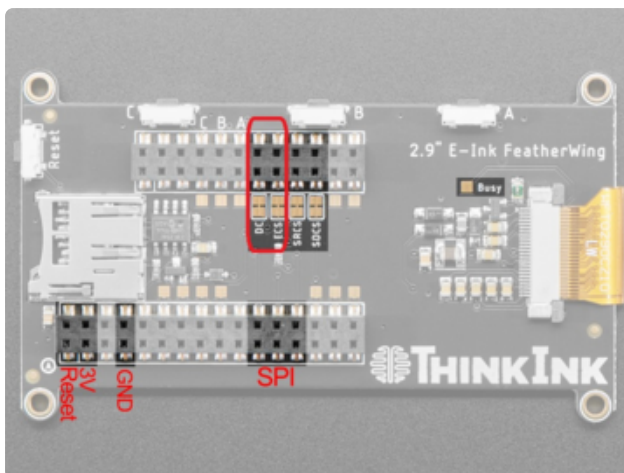


## FeatherWing Connection

FeatherWing usage is easy, simply plug your Feather board into the FeatherWing board.

To match up the pin numbers for your feather, visit the Pinout page on your board's Product Learn Guide.

You can quickly find it via the Docs link on the Wippersnapper devices page after setting up your board.



The E-Ink Featherwing uses the same pin layout as shown in the breakout wiring below, the two required pins on the shorter side (ECS+DC) are 4 unconnected pins away from the non-USB end, and the other side has a fixed layout for SPI (MOSI/ MISO/SCK) on Feathers.

If you wish to use the 3 A/B/C buttons, then confirm the pins for those on your board (marked C, B, A to the left of D/C on the featherwing image).

## Breakout Wiring

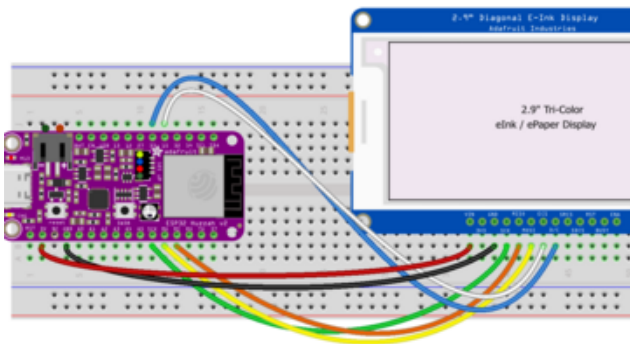
If you need help with soldering, then see the [Assembly page \(https://adafru.it/1ayb\)](https://adafru.it/1ayb), and this guide on [how to solder header pins \(https://adafru.it/RdJ\)](https://adafru.it/RdJ).

First, wire up the E-Ink display to your board as described below. Alternatively you may be able to use an EYESPI connector cable with a suitable socket (like on the [EYE](#)



[SPI BFF \(http://adafru.it/5772\)](http://adafru.it/5772) for QT Py & XIAO boards), or directly board-to-board if using the featherwing version. Later identify the pins by checking board+display pinout learn guide pages.

Here is an example of the 2.13" E-Ink wired to an [Adafruit ESP32 Feather V2 \(http://adafru.it/5400\)](http://adafru.it/5400) on a breadboard [with male Jumper wires \(2.54mm Dupont connectors\) \(https://adafru.it/ECb\)](https://adafru.it/ECb).



**VIN** connects to the Feather **3V** pin  
**GND** connects to Feather ground  
**SCK** connects to SPI clock. On the Feather ESP32v2 that's **SCK**.  
**MISO** connects to SPI MISO. On the Feather ESP32v2 that's **MI**  
**MOSI** connects to SPI MOSI. On the Feather ESP32v2 that's **MO**  
**ECS** connects to the E-Ink Chip Select pin. Use **Digital 15\***, the **white** wire.  
(\*or you can change this to any pin)  
**D/C** connects to SPI data/command select pin. Use **Digital 33\***, the **blue** wire.  
The **SRCS** (SRAM CS pin for display), **Busy** pin, and **Reset** pin are optional in this setup, but can provide improved performance and stability.

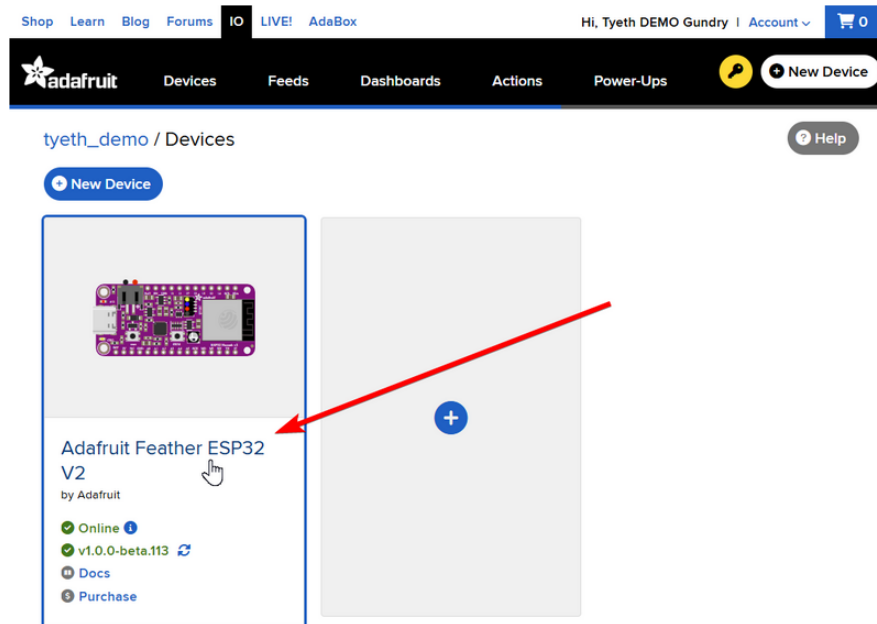
## Usage

Connect your board to Adafruit IO Wippersnapper and [navigate to the WipperSnapper board list \(https://adafru.it/TAu\)](https://adafru.it/TAu).

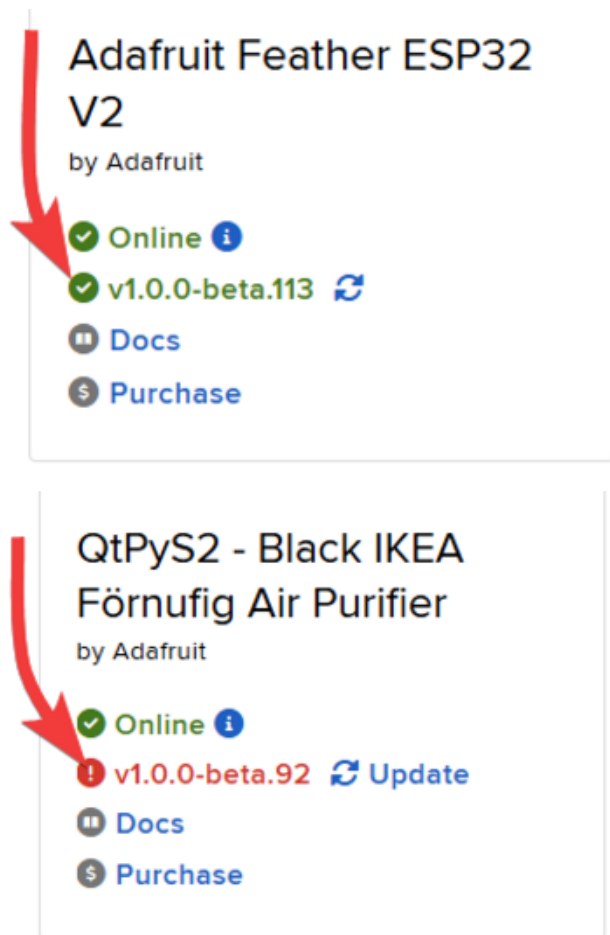
On that page, **select the WipperSnapper board you're using** to be brought to the board's interface page.

This guide page assumes you have **already installed WipperSnapper on your board**, and it is showing as "Online" on the Devices page.

it, then follow the quick start guide linked above, or visit the [New Device page](#) and follow the on-screen instructions.



If you do not see your board listed here - you need [to connect your board to Adafruit IO](https://adafru.it/Vfd) (<https://adafru.it/Vfd>) first.



On the device page, quickly **check that you're running the latest version of the WipperSnapper firmware.**

The device tile on the left indicates the version number of the firmware running on the connected board.

If the firmware version is green with a checkmark - continue with this guide. If the firmware version is red with an exclamation mark "!" - [update to the latest WipperSnapper firmware \(https://adafru.it/Vfd\)](https://adafru.it/Vfd) on your board before continuing.

Click the New Component button or the + button to bring up the component picker.



Adafruit IO supports a large amount of components, try searching for **Display** to see all the supported displays.



## Filtering and searching for components

Since WipperSnapper supports such a large number of components, you can use keyword filtering. Try searching for various keywords, like:

- component names: `aht20`, `servo`, `buzzer`, `button`, `neopixel`, etc
- sensor types: `light`, `temperature`, `pressure`, `humidity`, etc
- interface: `i2c`, `uart`, `ds18x20`, `pin`, `display`, etc (also I2C addresses e.g. `0x44`)
- vendor: `Adafruit`, `ASAIR`, `Infineon`, `Bosch`, `Honeywell`, `Sensirion`, etc
- display type / driver / family: `st7789`, `ili0373`, `sh1107`, `seg`, `lcd`

There are also product and documentation links for every component. Follow the links beneath the component descriptions to be taken to the appropriate product page, Learn Guide, or datasheet.

### New Component

Which component would you like to set up?

✕ SSD1680

Displaying 1 matching Components.



To quickly find your display, type `SSD1680` into the search bar, then select the **2.9" Grayscale eInk (SSD1680)** component.

On the component configuration page, the display's bus number and/or address should be listed along with the components settings.

Select all the pin options that match your wiring setup (leaving any drop-downs for any unattached pins unpopulated).

Finally choose any preferences like font size or alignment, then click **Create**.

## Create 2.9" Grayscale eInk (SSD1680) Component



### Settings

2.9" Grayscale eInk (SSD1680) Name

2.9" Grayscale eInk (SSD1680)

SPI Bus

0

Data/Command Pin

D33

Reset Pin

Choose a Pin

Chip Select Pin

D15

SRAM Chip Select Pin

Choose a Pin

Busy Pin

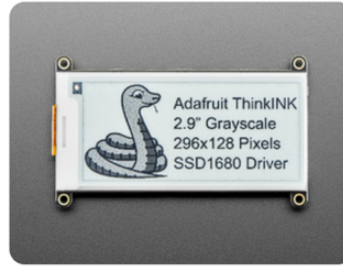
Choose a Pin

EPD Mode

Monochrome

Text Size

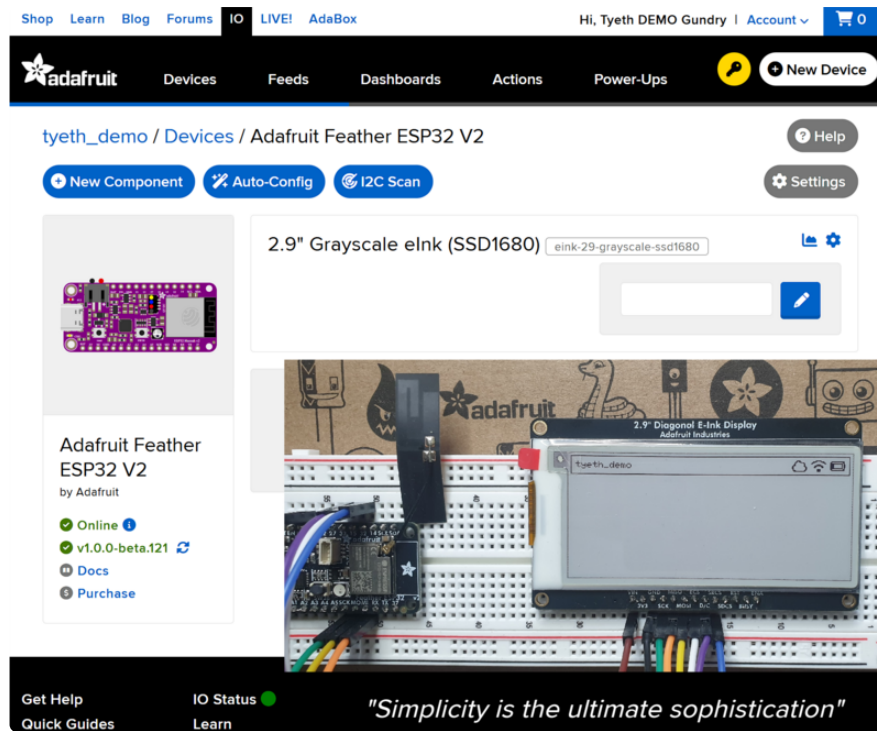
24px



[← Back to Component Type](#)

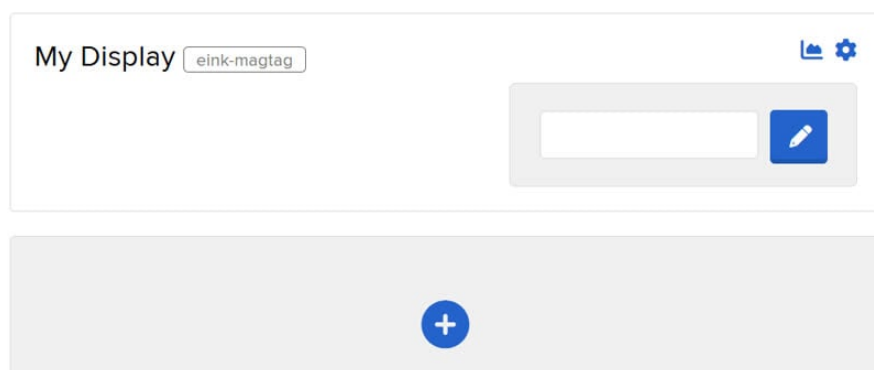
[Create Component](#)

After adding the component, your device page should show the newly created display component row, and the display attached to your board should refresh / turn on when the component is added.

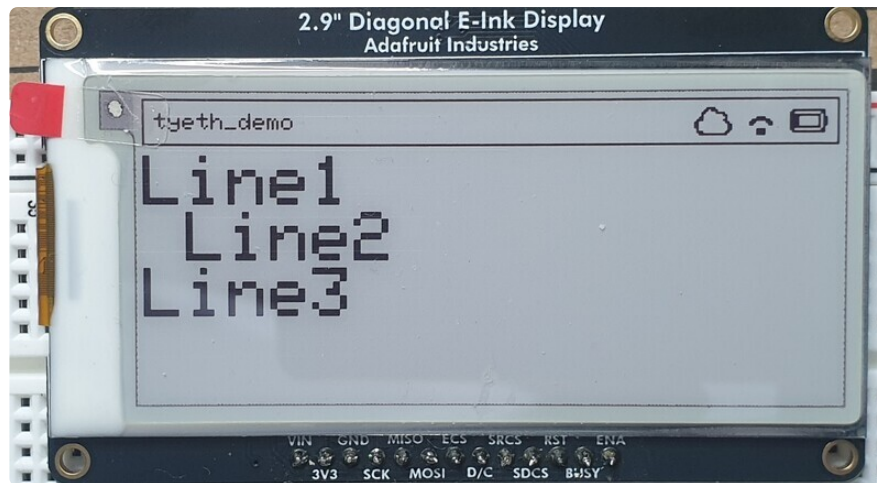


If appropriate, your display may also show a status bar with connection info (username, connection status, WiFi strength, and battery level).

To send a message to the display, click on the Pen button at the end of the display component row, then fill in the dialog and click **Send**:







To send a forced line break, use the backslash symbol ( `\` ) followed by the letter `n` for new line: `\n`



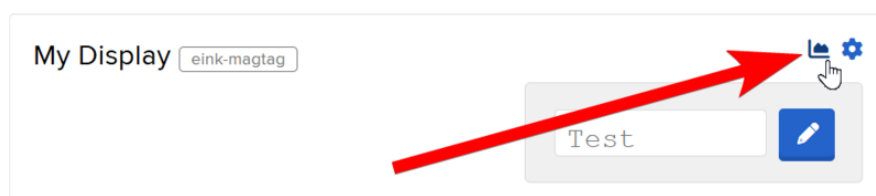
`Line1 \n Line2\nLine3`

The displays do accept true line breaks too (hitting the Enter/Return key in the text input box above, or sending `\n` or `\r\n` [via JSON](#)).

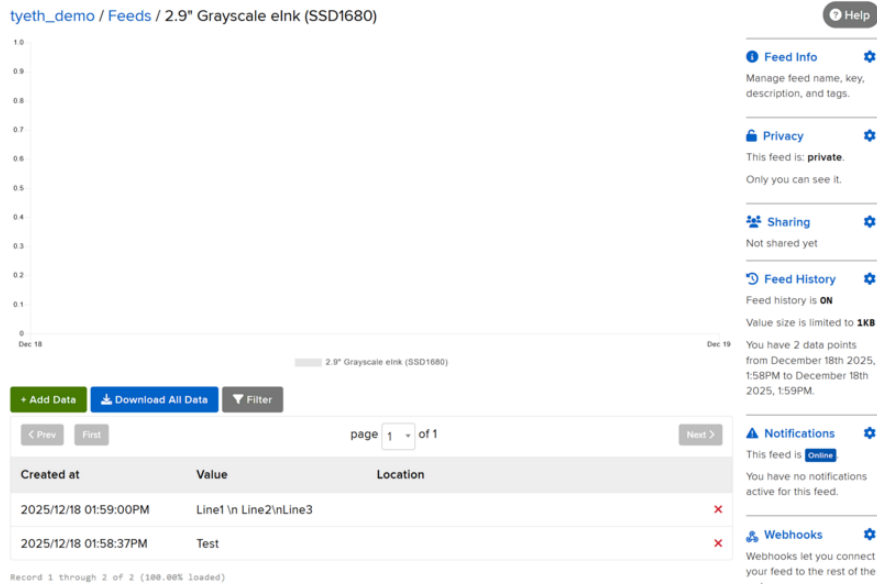
The display component is connected to a feed, which means that you can send data to the feed and have it immediately appear on screen.

This can be really useful when combined with automated [Actions](https://adafru.it/1axy) (<https://adafru.it/1axy>).

It also means there is a history of messages. To view the feed data that has been previously sent to the display, click on the graph icon at the end of the component row.



Here you can see the feed history and edit things about the feed such as the name, privacy, webhooks associated with the feed and more. If you want to learn more about how feeds work, [check out this page](https://adafru.it/10aZ) (<https://adafru.it/10aZ>).



Lastly, take note of the Settings Cog at the end of the component row, where you can alter the components configuration.

This will show you the same dialog as when you added the display, along with any advanced options. It's also where you can remove components.

Continue on with this guide, or maybe look at setting up an [Action \(https://adafru.it/YAT\)](https://adafru.it/YAT) to write to the display.

## Downloads

### 2.9" Tri-Color elnk Display Revision History

- As of June 20, 2025 - The display has been updated (old ones discontinued) to use a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!
- As of May 22th, 2024 – We've updated this PCB with a new EYESPI connector along with the Adafruit Penguin to make a lovely and legible silkscreen.
- As of November 16, 2022 - this revision has been updated to have a 2.9" Tri-Color elnk display only with a UC8151D chipset.

## 2.9" Tri-Color eInk FeatherWing Revision History

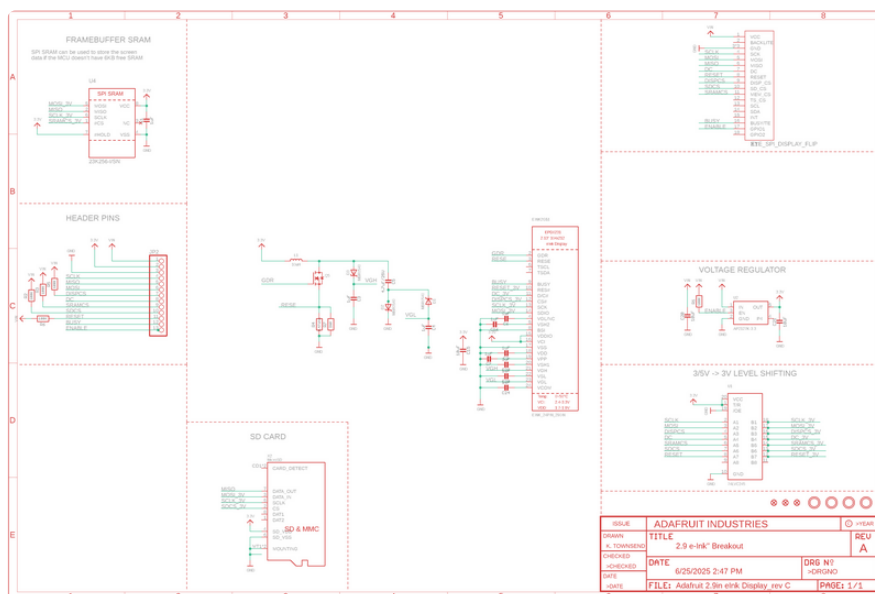
- As of June 24th, 2025 – The display has been updated (old one discontinued) it now uses a spiffy modern SSD1680 chip - we have support in CircuitPython/Python/Arduino for it!
- As of Sep 8, 2023 – we've updated this PCB with Adafruit Pinguin to make a lovely and legible silkscreen - you may get the new PCB or the older version with vector fonts - both are identical other than the fancy silkscreen.

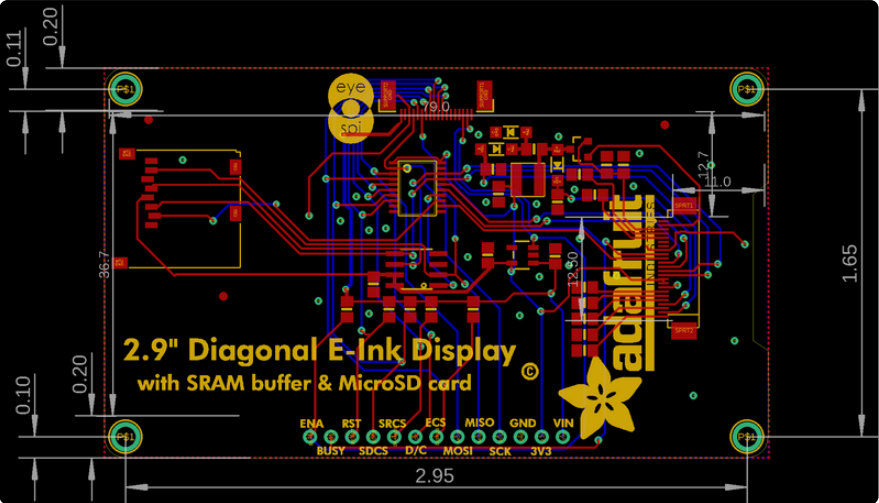
## Files

- [Fritzing object in Adafruit Fritzing Library \(https://adafru.it/aP3\)](https://adafru.it/aP3)
- [IL0376F E-Ink interface chip datasheet \(https://adafru.it/BRW\)](https://adafru.it/BRW)
- [UC8151D Datasheet \(https://adafru.it/1alu\)](https://adafru.it/1alu)
- [SSD1680 Datasheet \(https://adafru.it/1alt\)](https://adafru.it/1alt)
- [PCB Files on GitHub \(https://adafru.it/BRX\)](https://adafru.it/BRX)

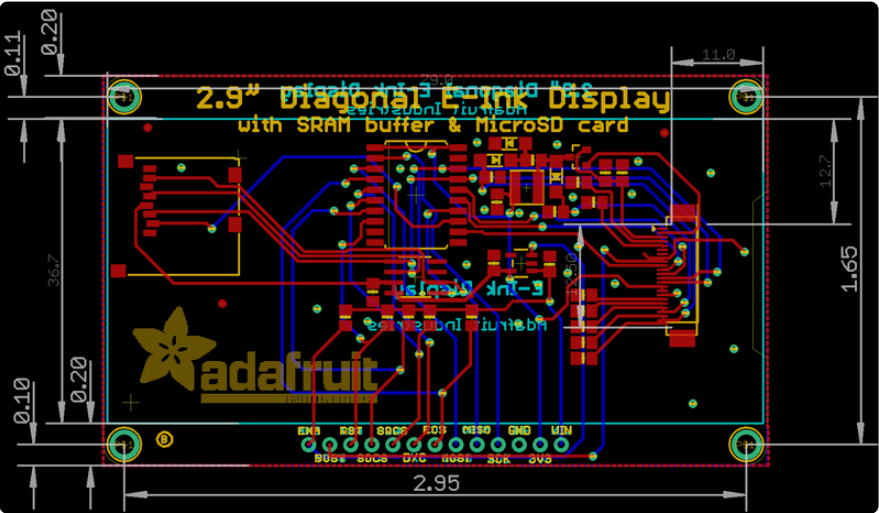
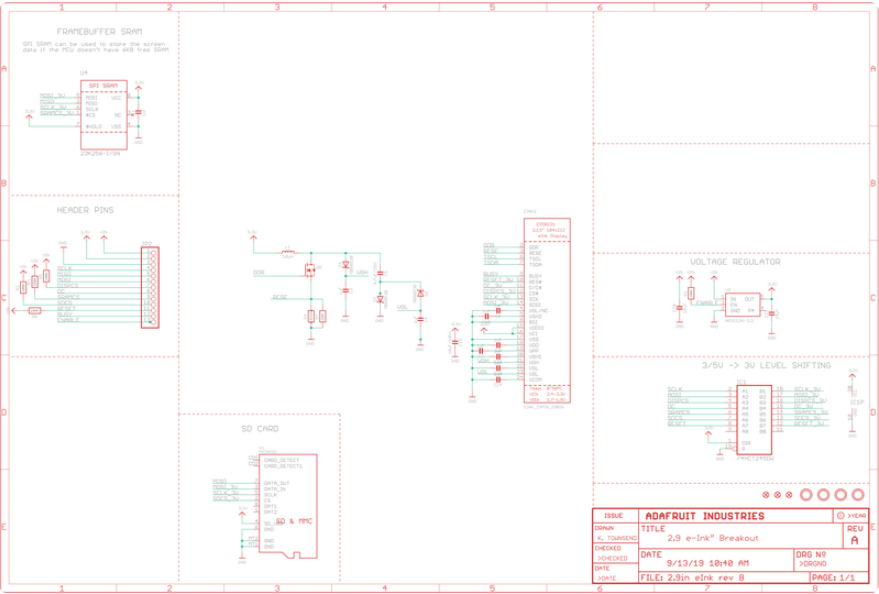
## Schematic & Fabrication Prints

Rev C

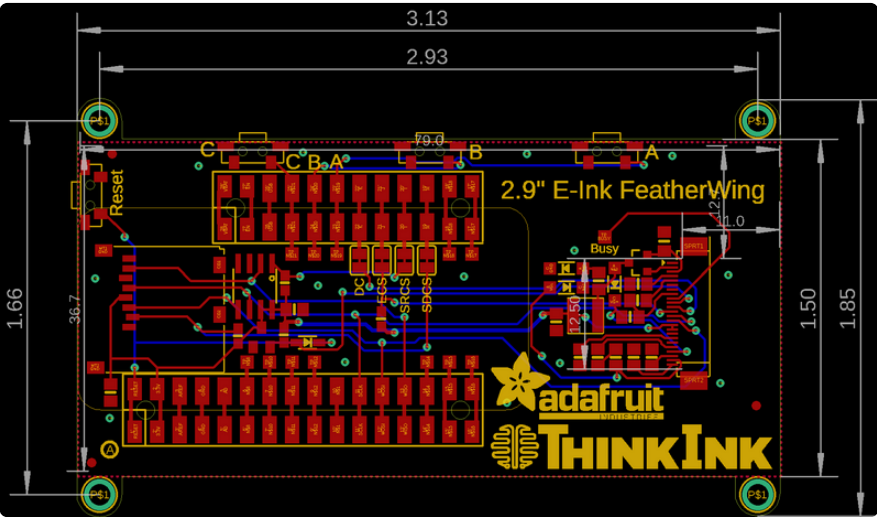
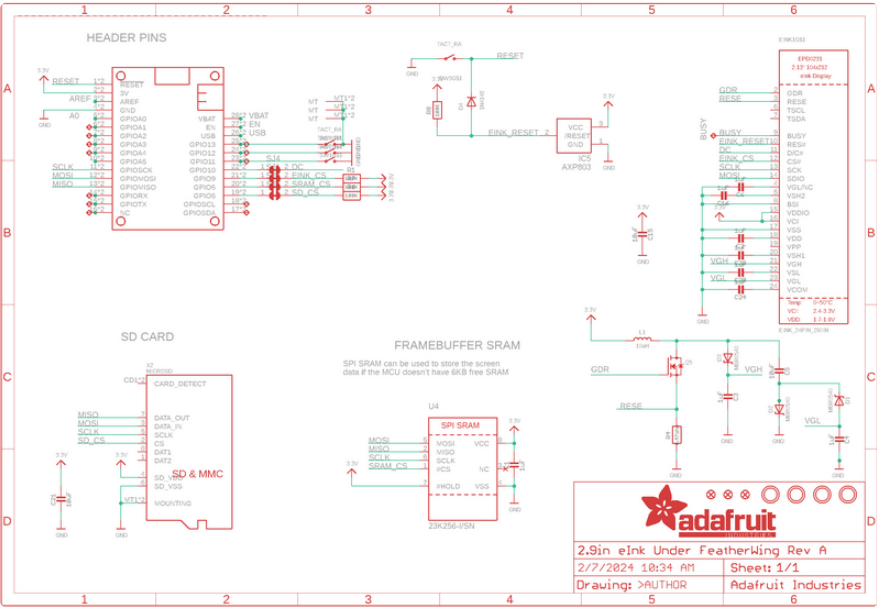




Rev B



2.9" FeatherWing



# eInk Friends

