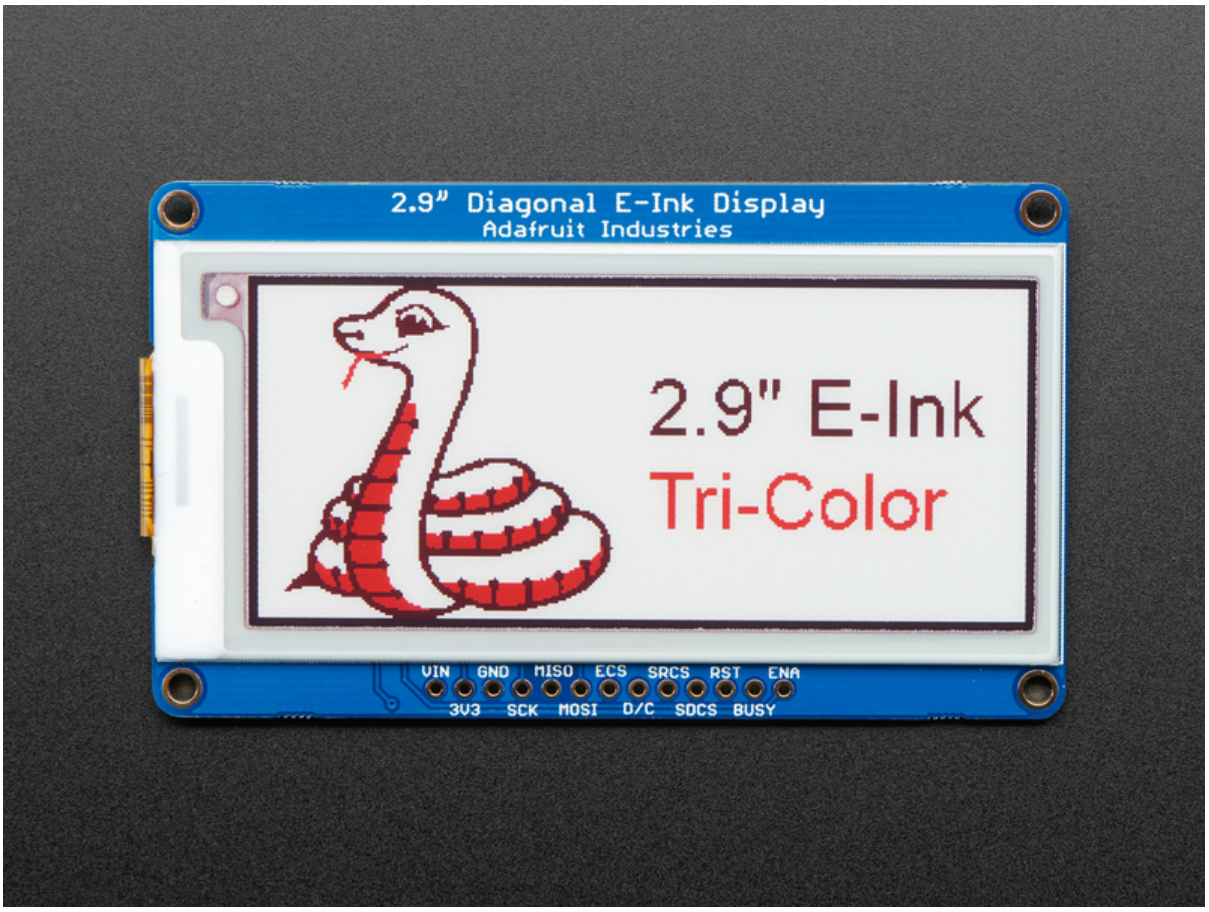
Adafruit 2.9" eInk Display Breakouts and FeatherWings

Created by Melissa LeBlanc-Williams

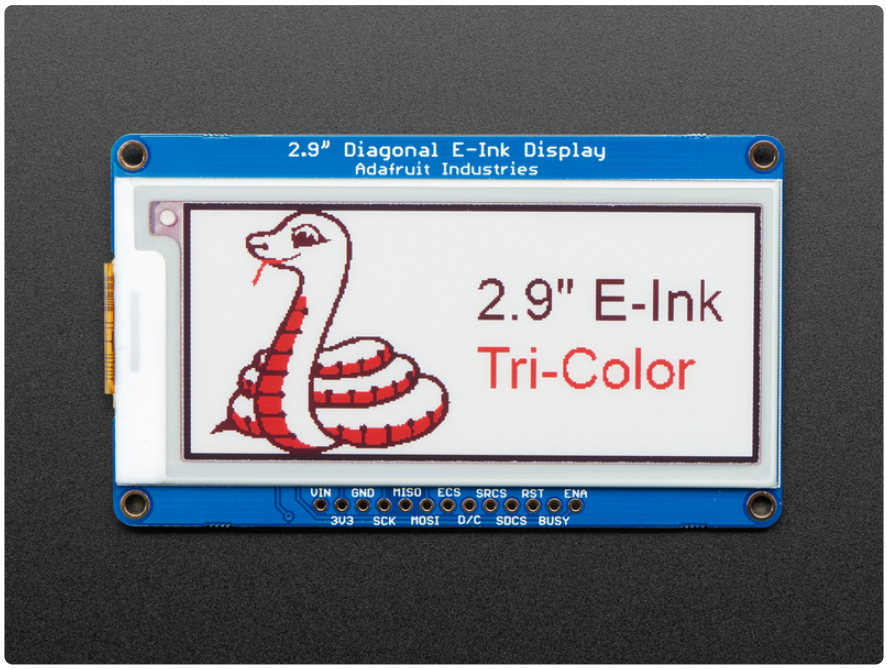


https://learn.adafruit.com/adafruit-2-9-eink-display-breakouts-and-featherwings

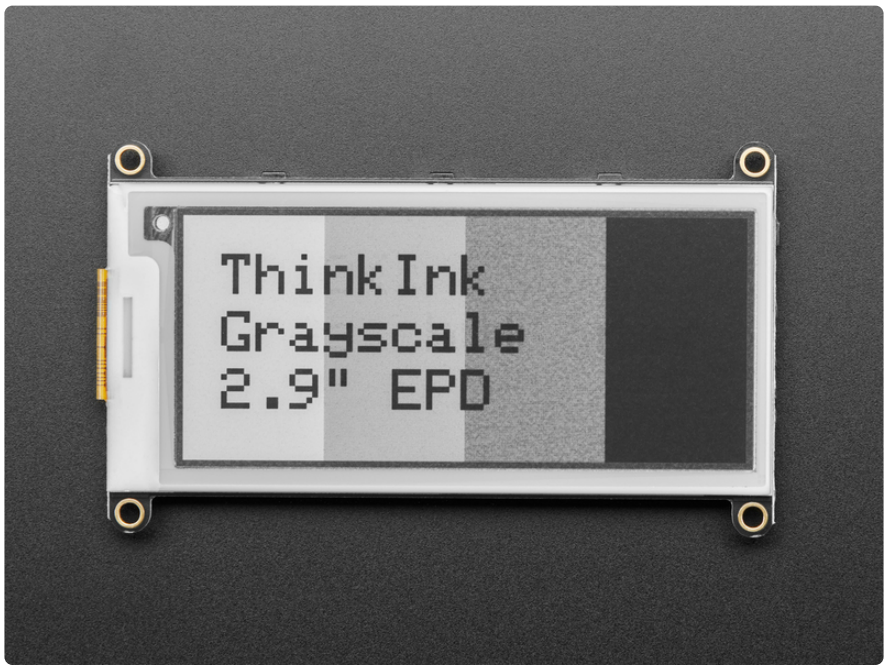
Last updated on 2024-03-08 03:05:52 PM EST

Table of Contents

Overview



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

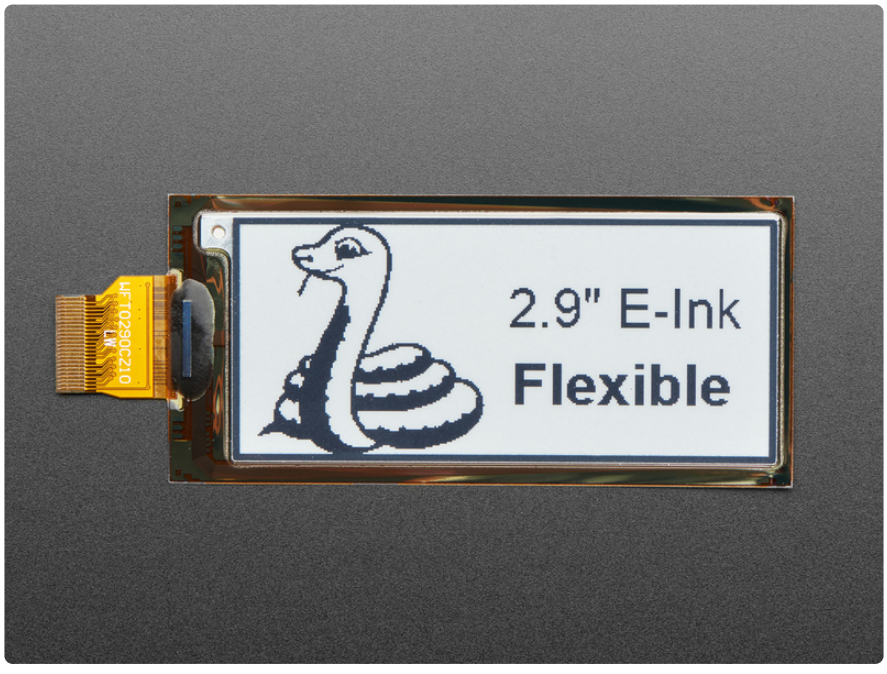
We have multiple 2.9" EPD displays:

Breakouts and Flexibles

- We have a newer 296x128 Tri-Color display (http://adafru.it/1028) with the UC8151D chipset, which has 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) 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) or e-Ink Feather Friend (http://adafru.it/4446). There is an extension cable (http://adafru.it/4230) available for this type of connection also.

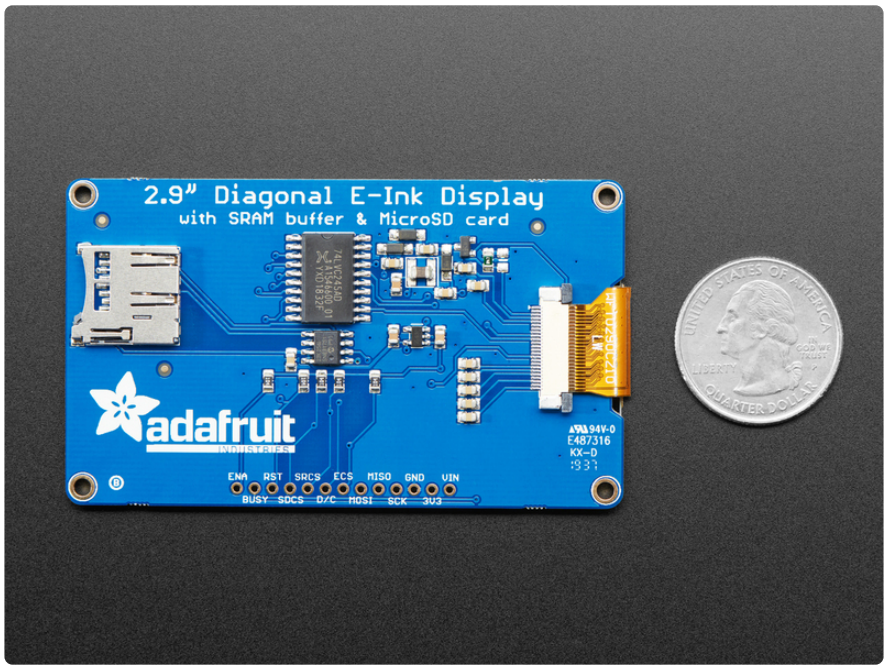
FeatherWings

- The newer 296x128 Tri-Color FeatherWing (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).



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 x 128 pixels x 2-bits-per-pixel = 9.5 KBytes. Which won't fit into many microcontroller memories. Heck, even if you do have 32KB of RAM, why waste 9KB?

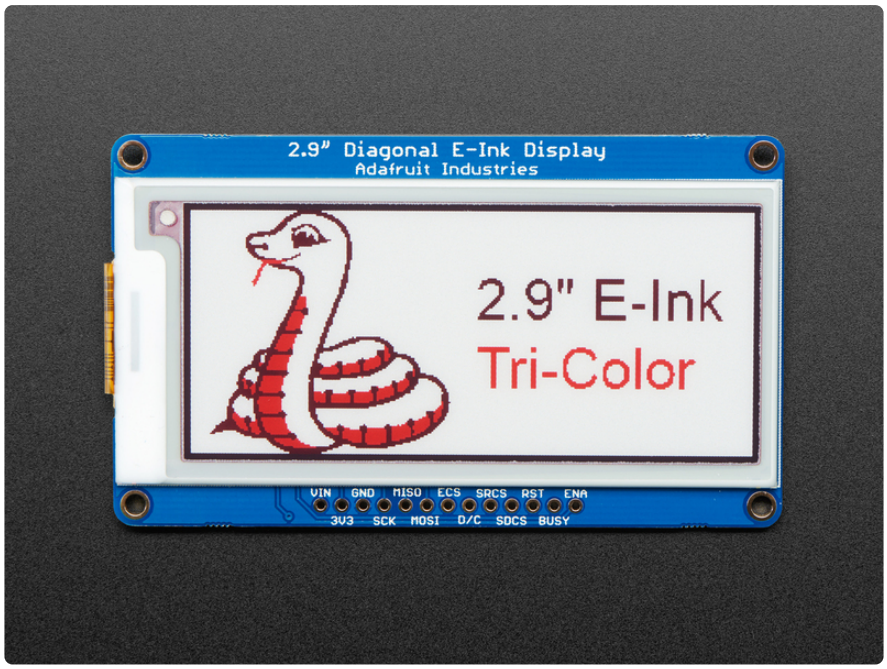
So we did you a favor and tossed a small SRAM chip on the back. This chip shares the SPI port the eInk display uses, so you only need one extra pin. And, no more frame-buffering! **You can use the SRAM to set up whatever you want to display, then shuffle data from SRAM to eInk when you're ready.** The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



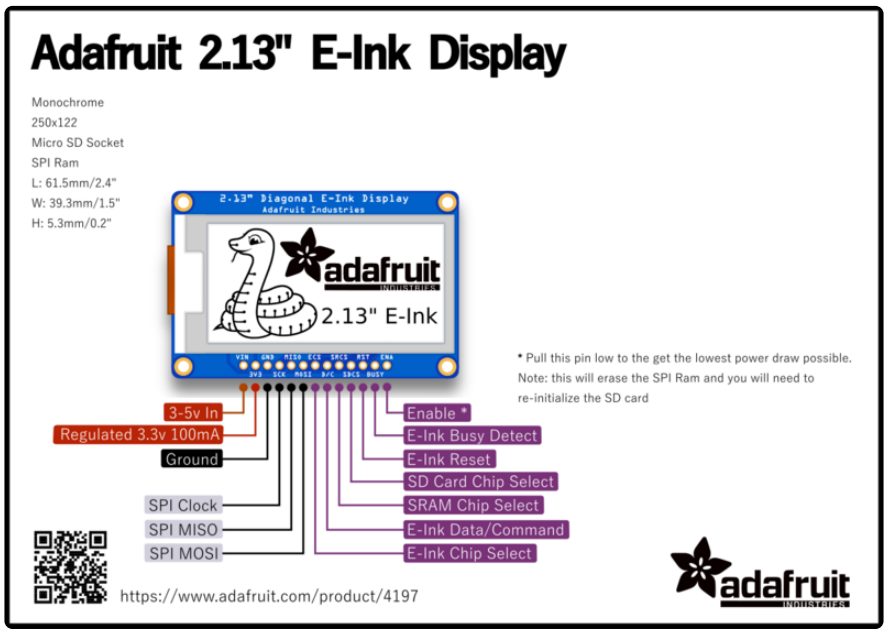
For ultra-low power usages, the onboard 3.3V regulator has the Enable pin brought out so you can shut down the power to the SRAM, MicroSD and display.

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.

Pinouts



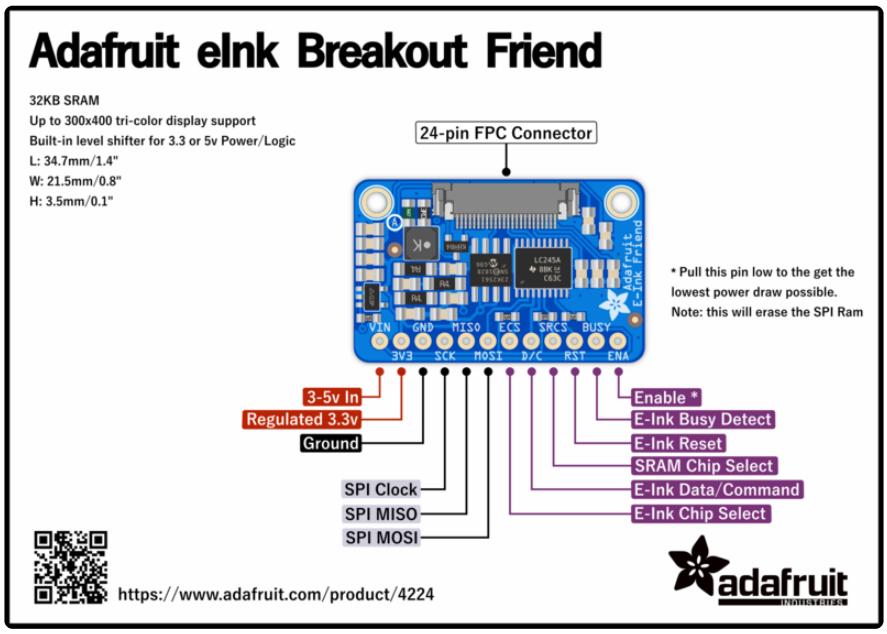
This e-Paper display uses SPI to receive image data. Since the display is SPI, it was easy to add two more SPI devices to share the bus - an SPI SRAM chip and SPI-driven SD card holder. There's quite a few pins and a variety of possible combinations for control depending on your needs



The 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!



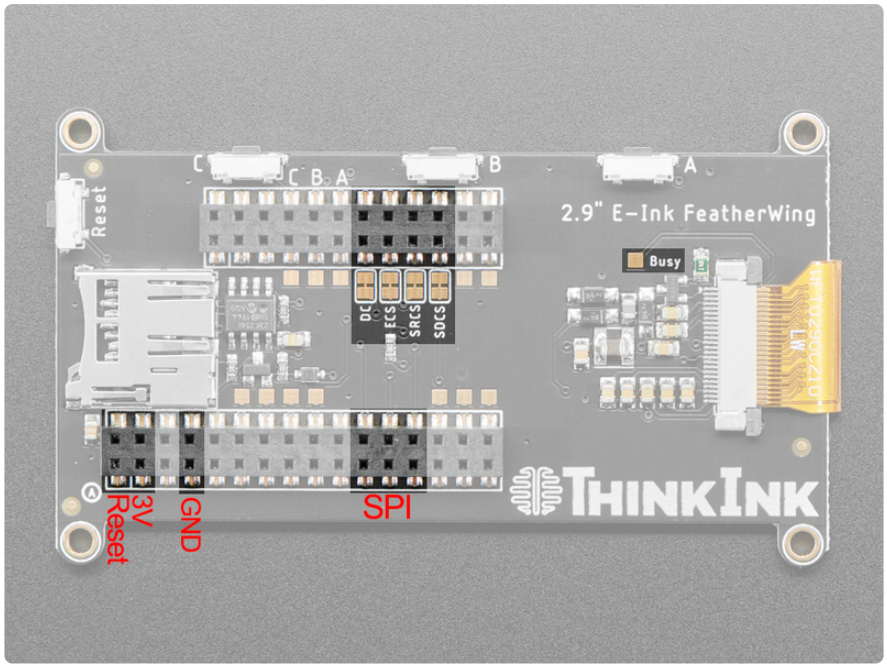
Power Pins

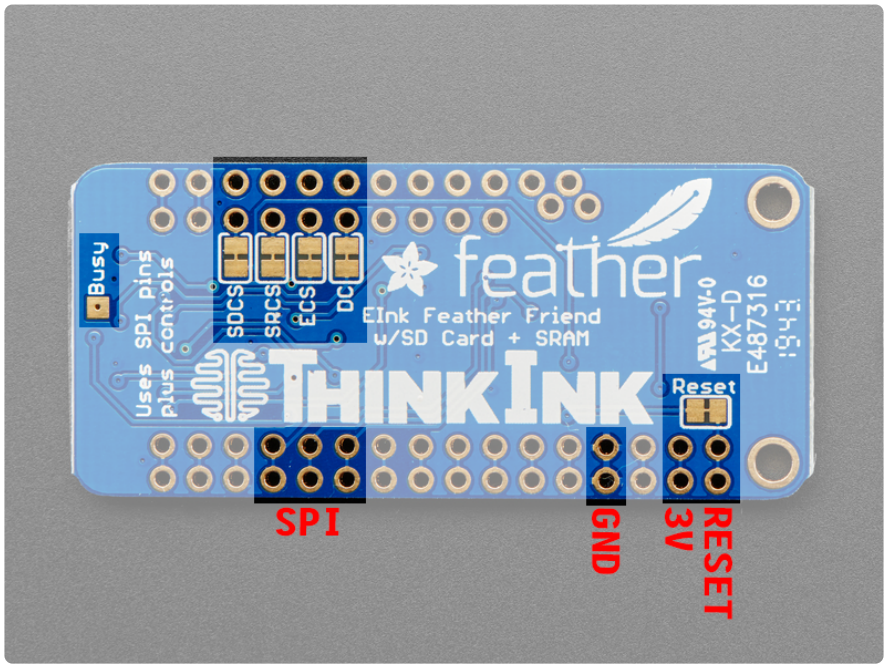
- **3-5V / Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3.3V** out - this is the 3.3V output from the onboard regulator, you can 'borrow' about 100mA if you need to power some other 3.3V logic devices
- **GND** - this is the power and signal ground pin
- **ENA**ble - 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 **SR**AM **C**hip **S**elect, required for communicating with the onboard RAM chip.
- **SDCS** - this is the **SD** card **C**hip **S**elect, 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 **R**e**S**e**T** pin, you may be able to share this with your microcontroller reset pin but if you can, connect it to a digital pin.
- **BUSY** - this is the e-Ink busy detect pin, and is optional if you don't want to connect the pin (in which case the code will just wait an approximate number of seconds)

FeatherWing Connections





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

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

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

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

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

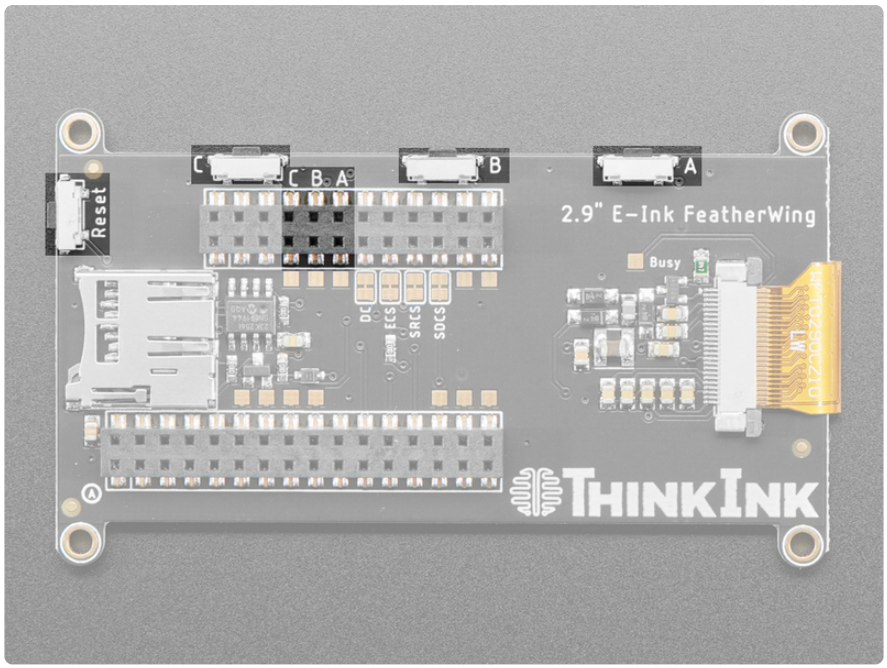
The **Reset** pin for the E-Ink display is connected to an auto-reset circuit and also to the Feather Reset pin, so it will reset when you press the reset button.

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

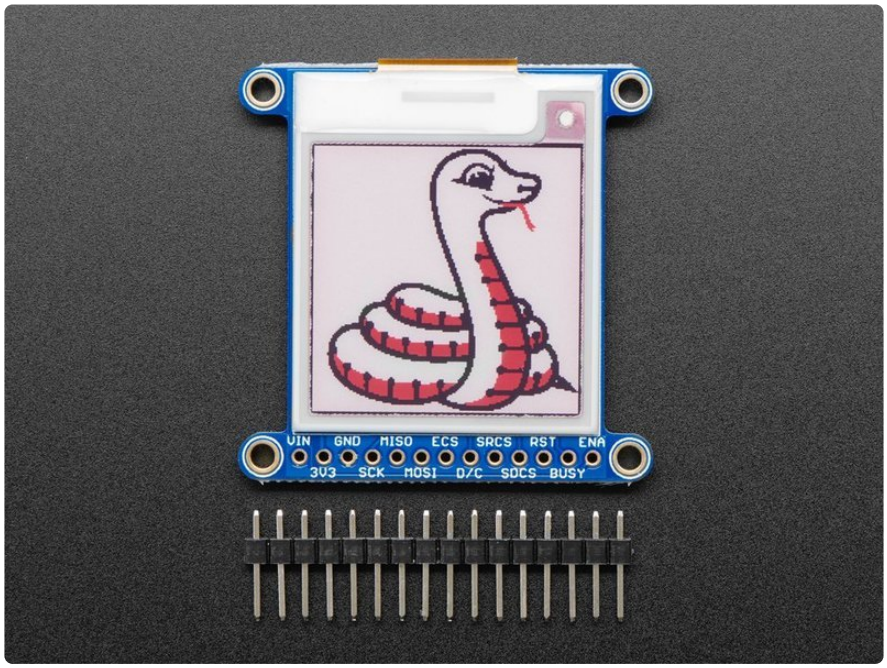
FeatherWing Buttons

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.

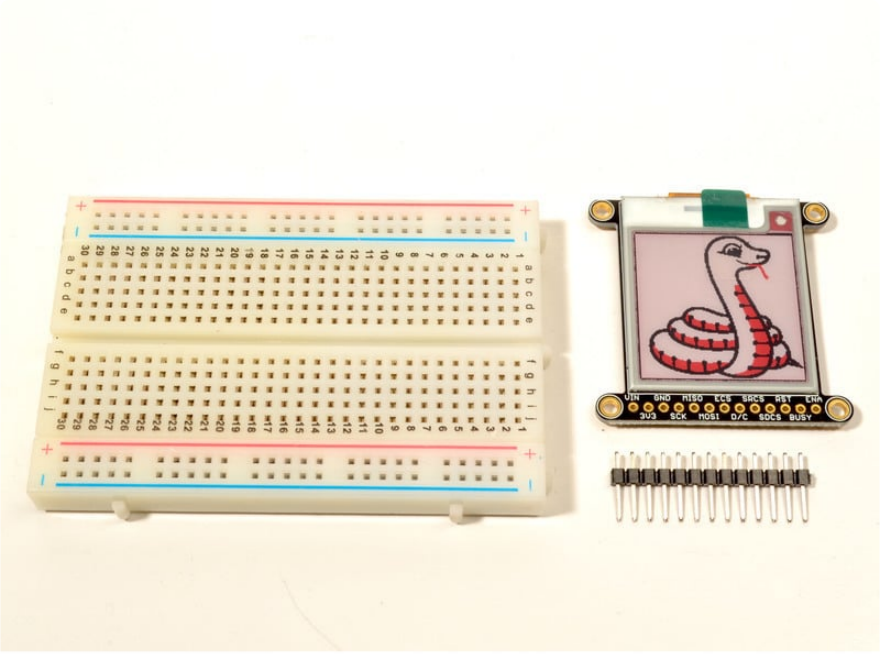


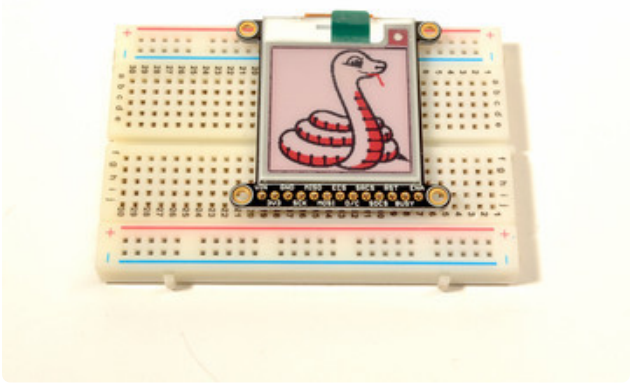
Assembly



Assembly

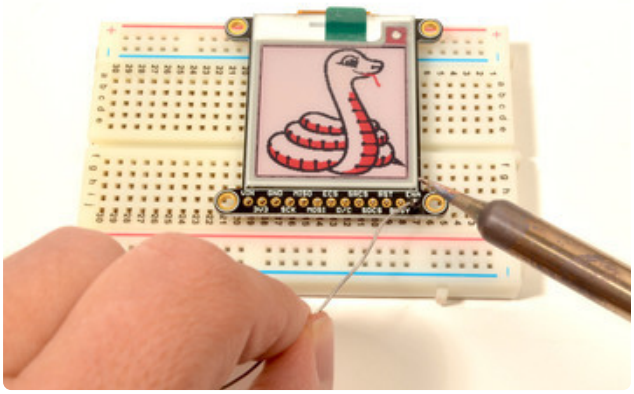
Cut the header down to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

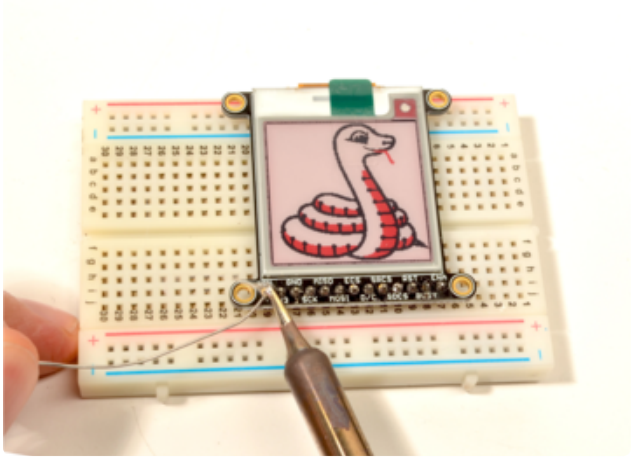




Add the E-Ink Display

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

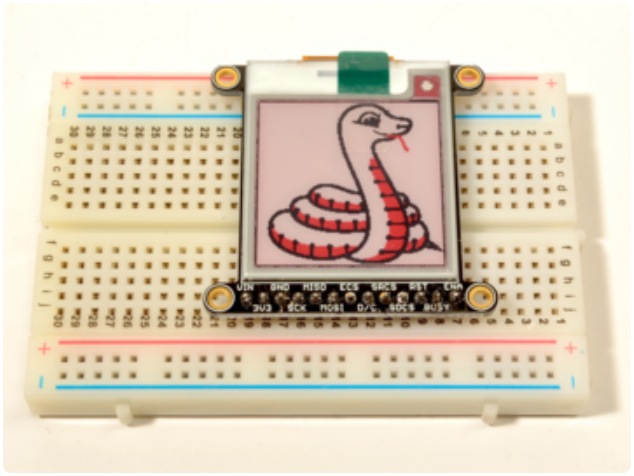




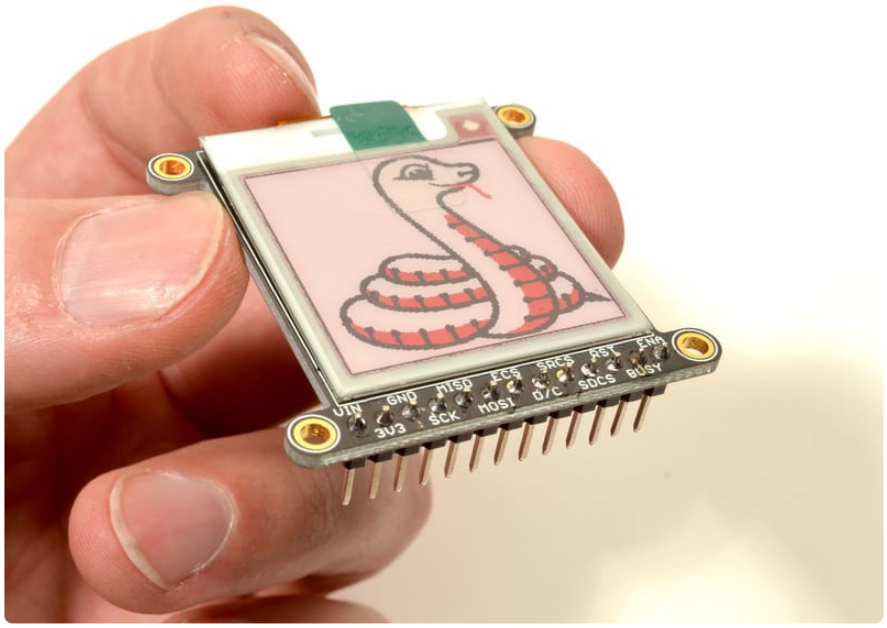
And Solder!

Be sure to solder all pins for reliable electrical contact.

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



OK, you're done!

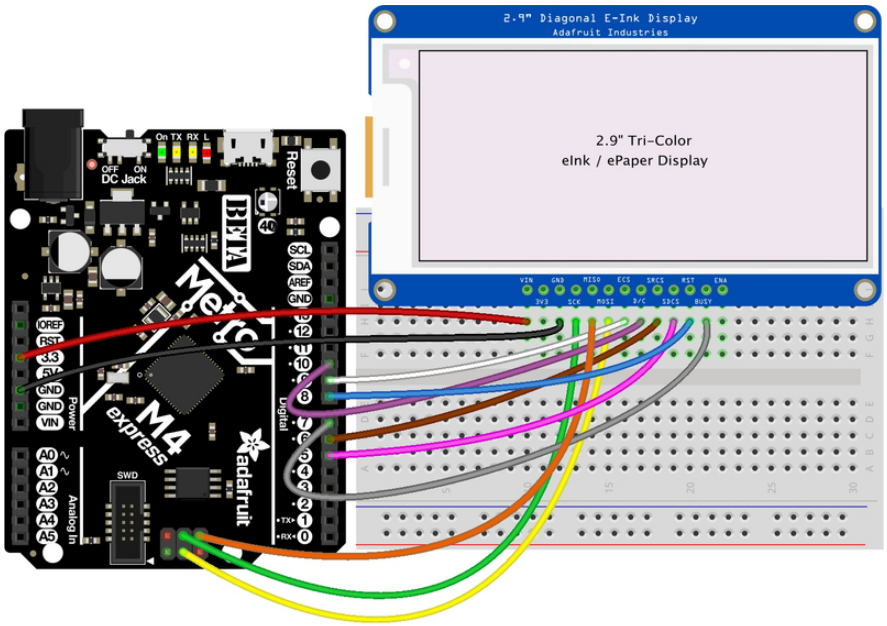


Wiring

Breakout Wiring

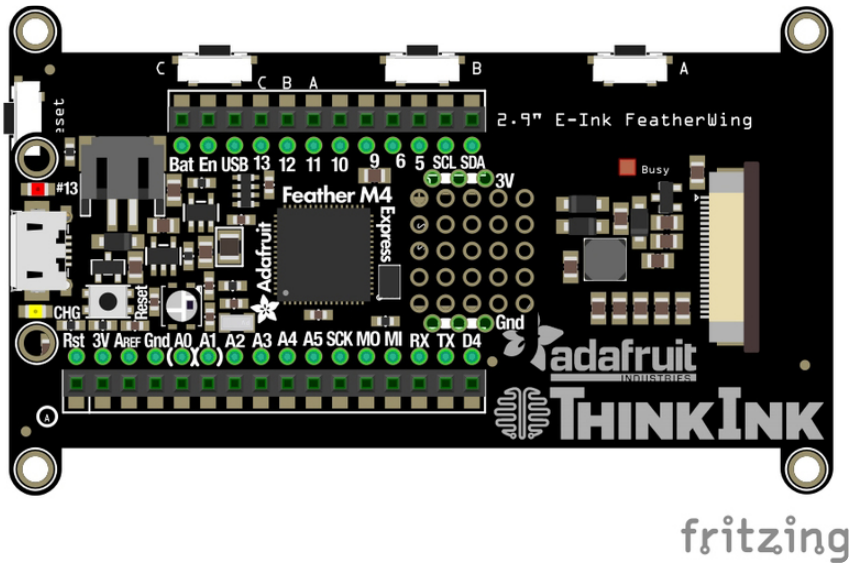
Though it looks like a lot of connections, wiring up an eInk breakout is pretty straightforward! Below shows using hardware SPI to connect it to an Adafruit Metro M4.

- **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**.



FeatherWing Connection

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

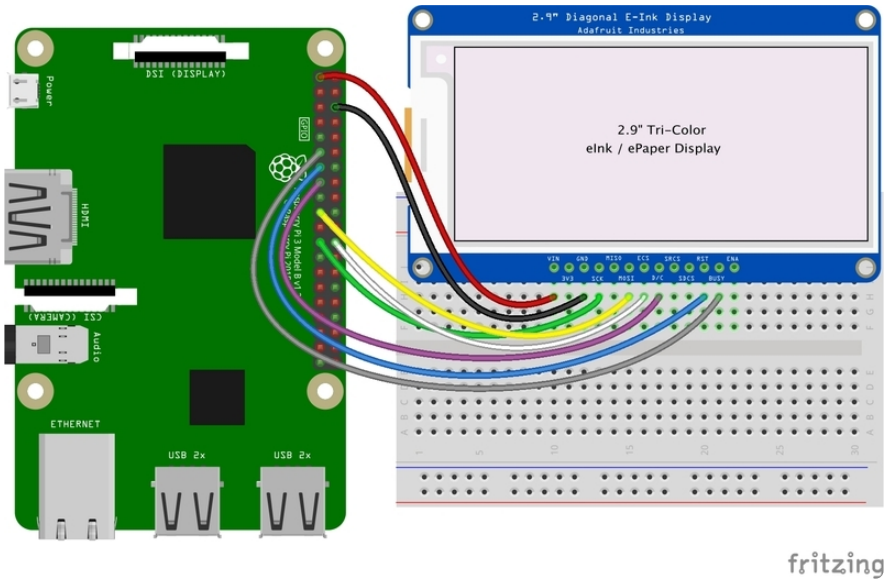


Python Wiring

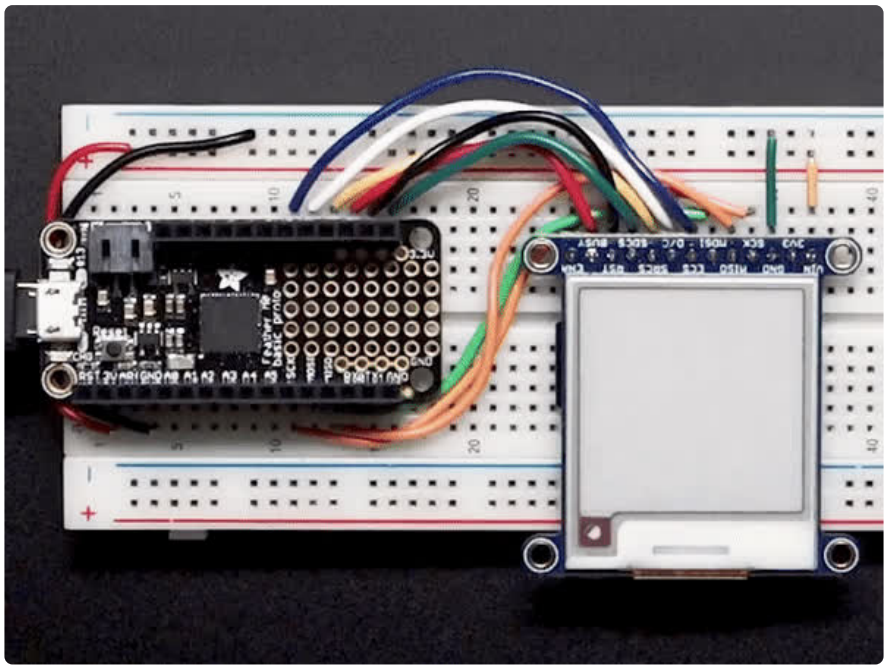
Wiring for the eInk friend breakout is the same as the Tri-Color breakout.

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**



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 may seconds to fully erase and replace an image**

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

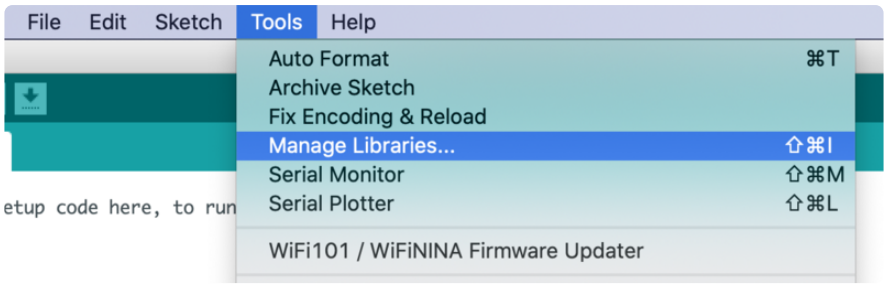
You don't have to refresh often, but with tri-color displays, the larger red ink dots will slowly rise, turning the display pinkish instead of white background. **To keep the background color clear and pale, refresh once a day**

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

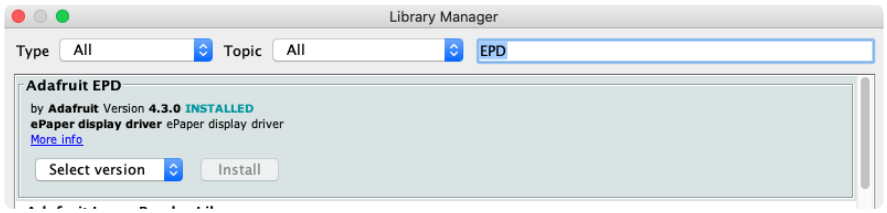
Arduino Setup

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

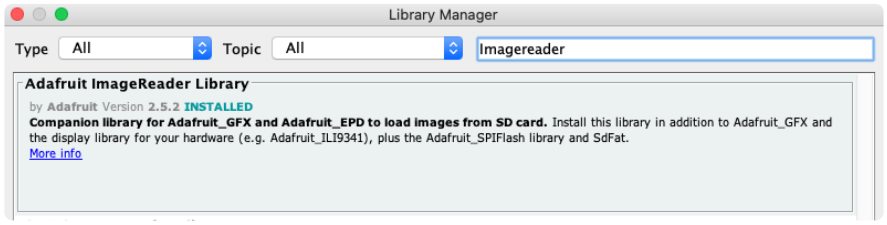
From the IDE open up the library manager...



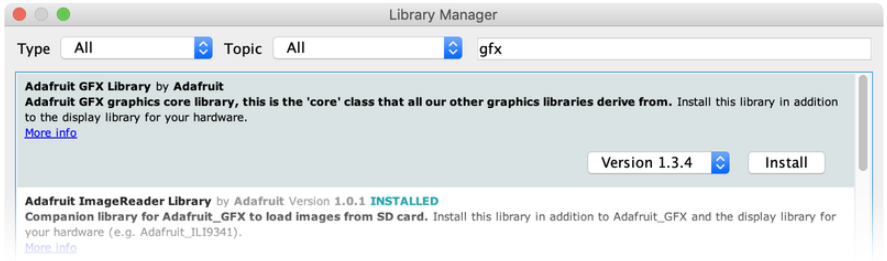
And type in **adafruit EPD** to locate the library. Click **Install**



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



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



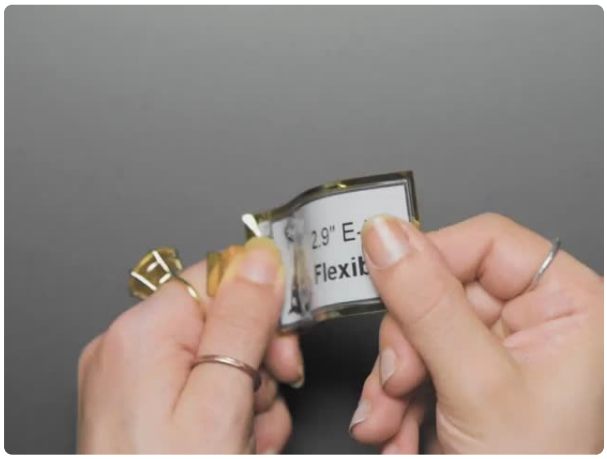
If using an earlier version of the Arduino IDE (pre-1.8.10), locate and install **Adafruit_BusIO** (newer versions handle this prerequisite automatically).

Arduino 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 ThinkInk type for your display or you won't see anything happen on the EPD (or the image may be really weird looking).

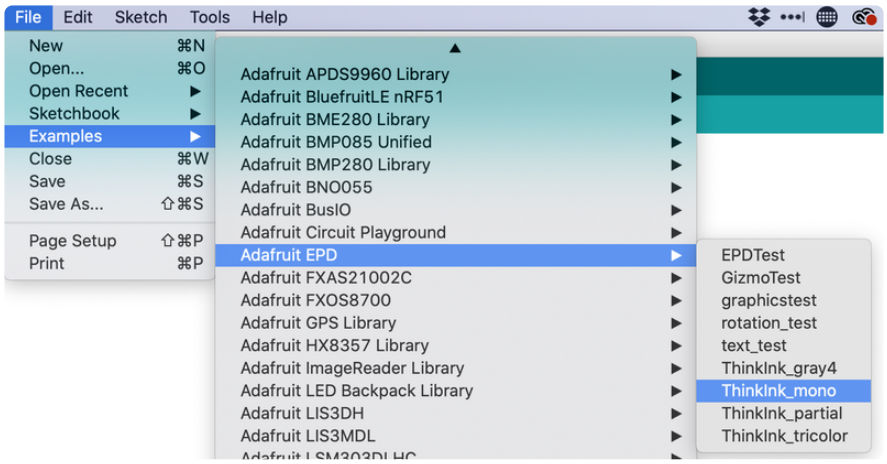
2.9" Monochrome 296x128 Pixel Display

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



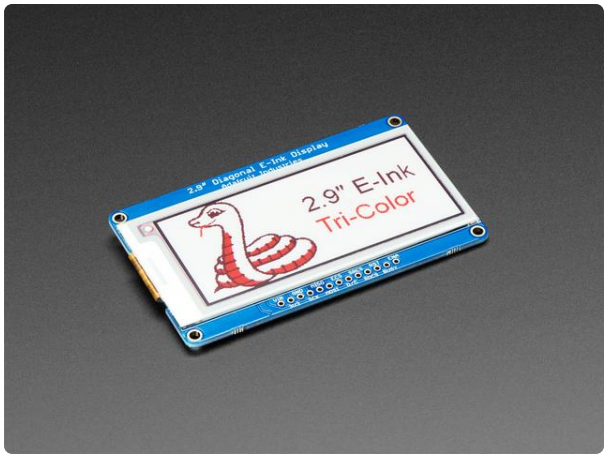
[2.9" Flexible 296x128 Monochrome eInk / ePaper Display](https://www.adafruit.com/product/4262)
Woah, the cyber-future is here! Flexible E-Ink has been demo'd at high-tech events for years but now you can actually get your paws on it. This display is true E-Ink / E-Paper,...
https://www.adafruit.com/product/4262

Open up **File→Examples→Adafruit_EPD→ThinkInk_mono**

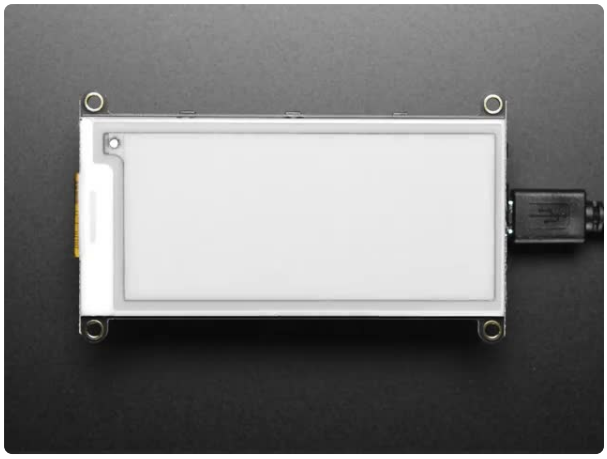


2.9" Tri-Color 296x128 Pixel Display

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



[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



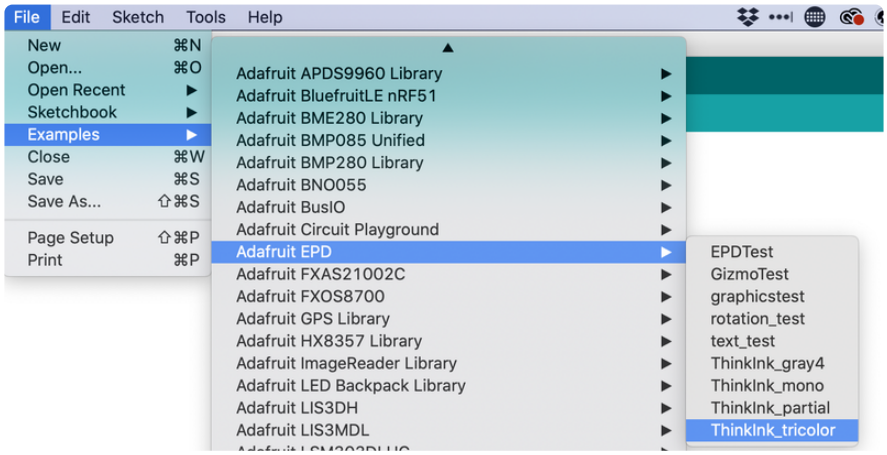
Adafruit 2.9" Tri-Color eInk / ePaper Display FeatherWing

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

The demo updates every 15 seconds which is fine for demonstrating the functionality for a short time, but we recommend not updating more often than 180 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

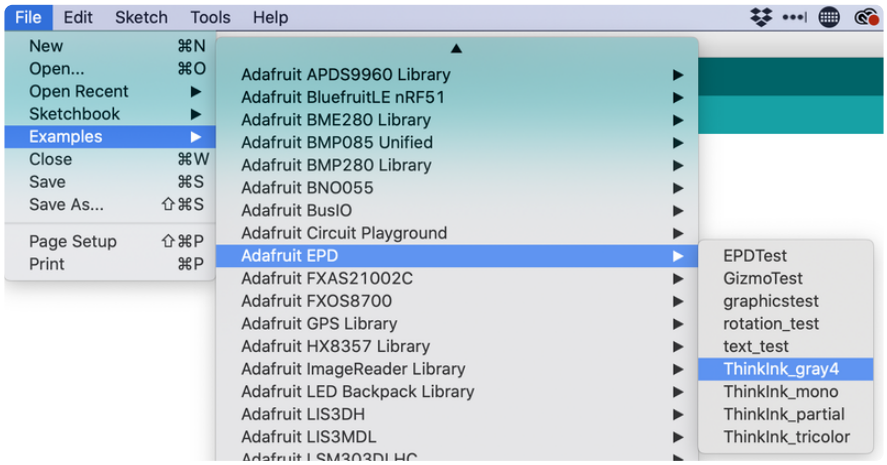
For the 296x128 Grayscale display, below is the grayscale demo.

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

For the FeatherWing you must be sure both EPD_RESET and EPD_BUSY are set 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 eInk 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), you will use the `ThinkInk_290_Mono_M06` display initializer.

For the newer **2.9" 296x128 Tri-Color breakout** (http://adafru.it/1028), you will use the `ThinkInk_290_Tricolor_Z13` display initializer.

For the older **2.9" 296x128 Tri-Color breakout** (http://adafru.it/1028) or **2.9" 296x128 Tri-Color FeatherWing** (http://adafru.it/4778), you will use the `ThinkInk_290_Tricolor_Z10` display initializer.

For the **2.9" 296x128Grayscale breakout** (http://adafru.it/4086) , you will use the `ThinkInk_290_Grayscale4_T5` display initializer.

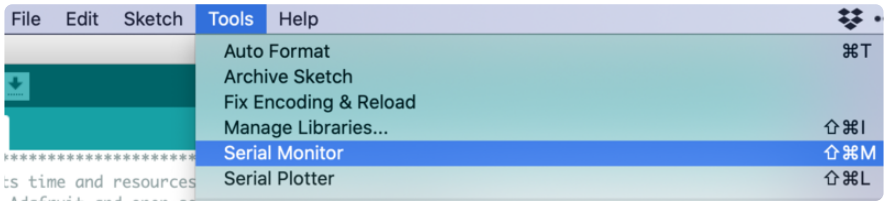
For example, for the mono 296x128 flexible display, uncomment this line, and comment any other line that is creating a ThinkInk display object

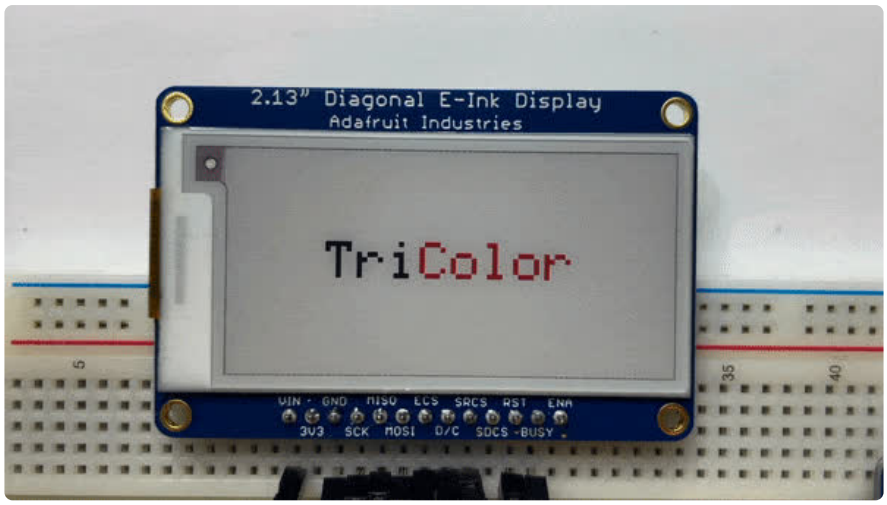
```
// 2.9" Monochrome displays with 296x128 pixels and UC8151D chipset
//ThinkInk_290_Mono_M06 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY);
```

Upload Sketch

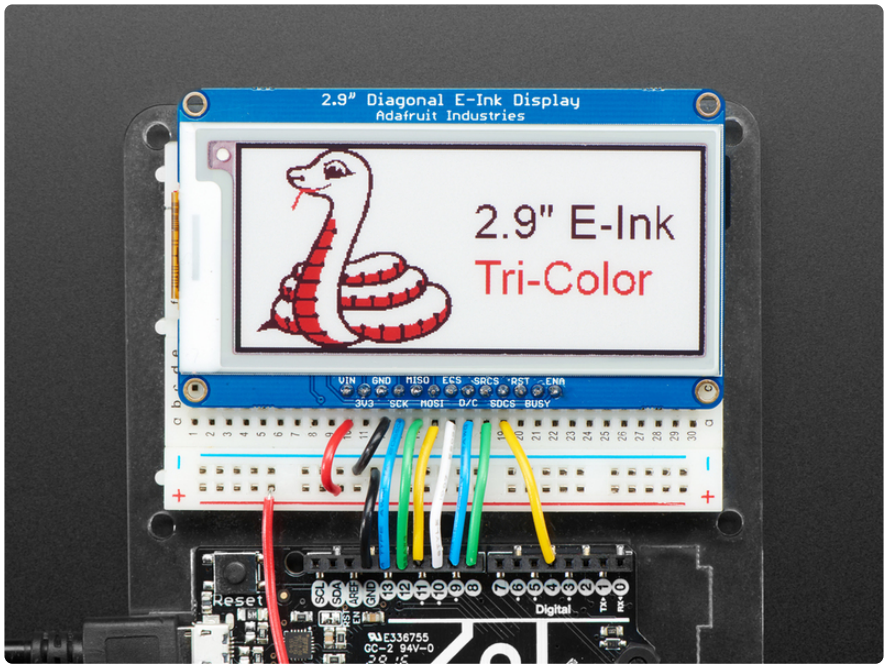
After checking the pinouts and the display type one more time, go ahead and upload the sketch to your board. Once it is done uploading, open the **Serial Monitor**.

The display should start running a series of display tests





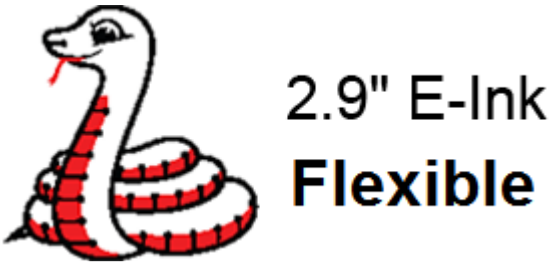
Arduino Bitmaps



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:



Download Flexible blinka.bmp

https://adafru.it/19f2

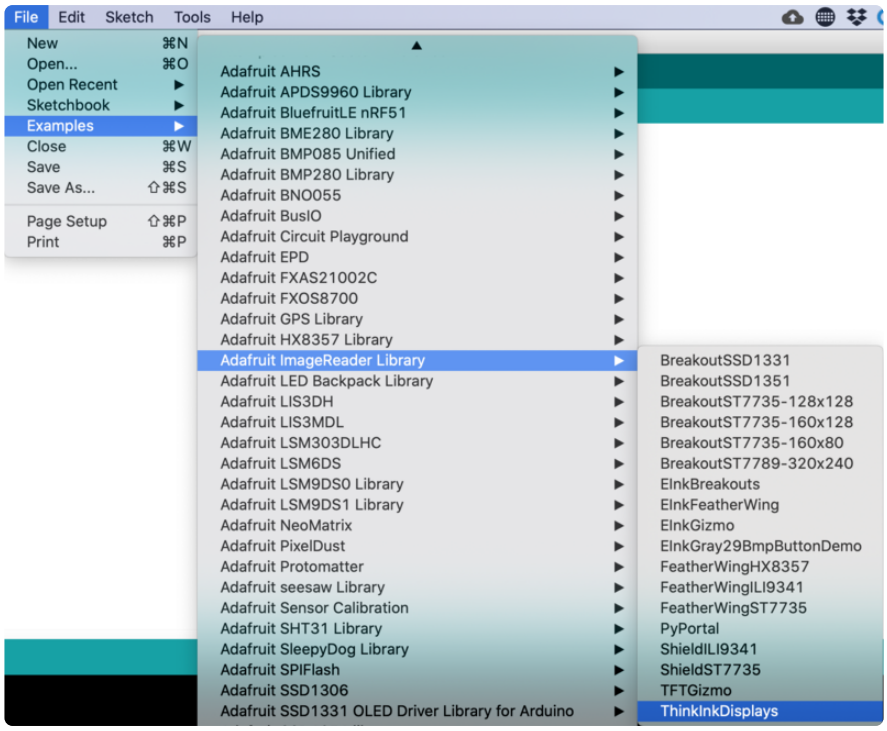
Download Tri-Color blinka.bmp

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

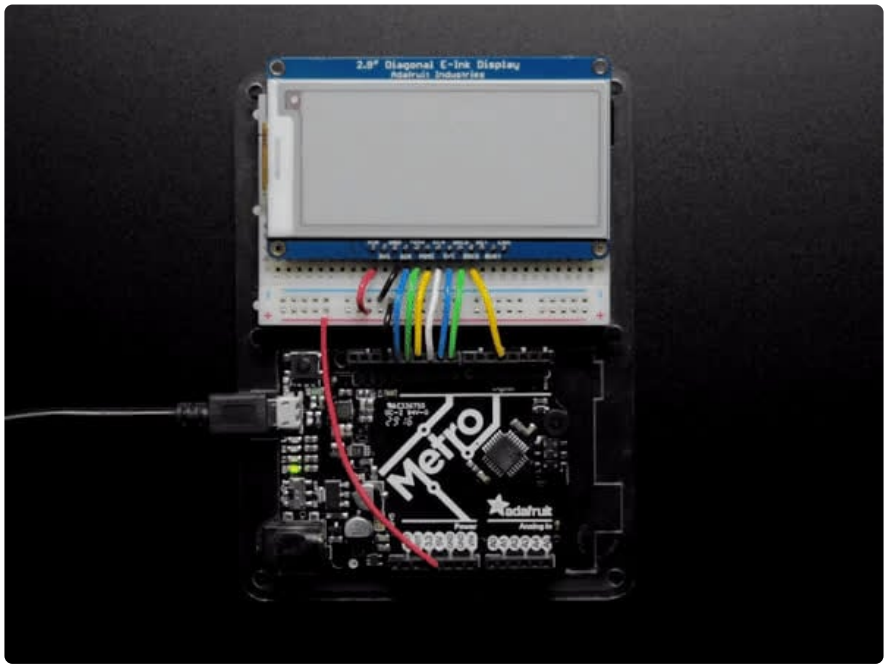


There are just a couple of things you may need to change in this file.

If you are using the FeatherWing, set `EPD_RESET` and `EPD_BUSY` to **-1** otherwise it will wait indefinitely.

If you are using the Flexible Monochrome Display, you may need to change your initializer to use `ThinkInk_290_Mono_M06`.

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

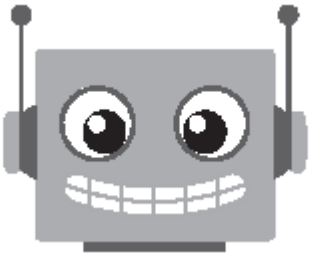


GrayScale Display Demo

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:

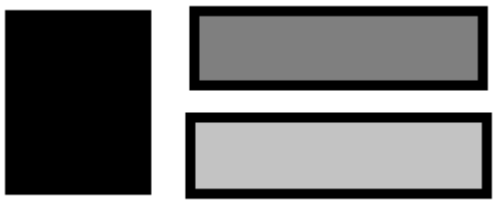
Download panda_head.bmp

https://adafru.it/OdE



Download adabot_head.bmp

https://adafru.it/OdF



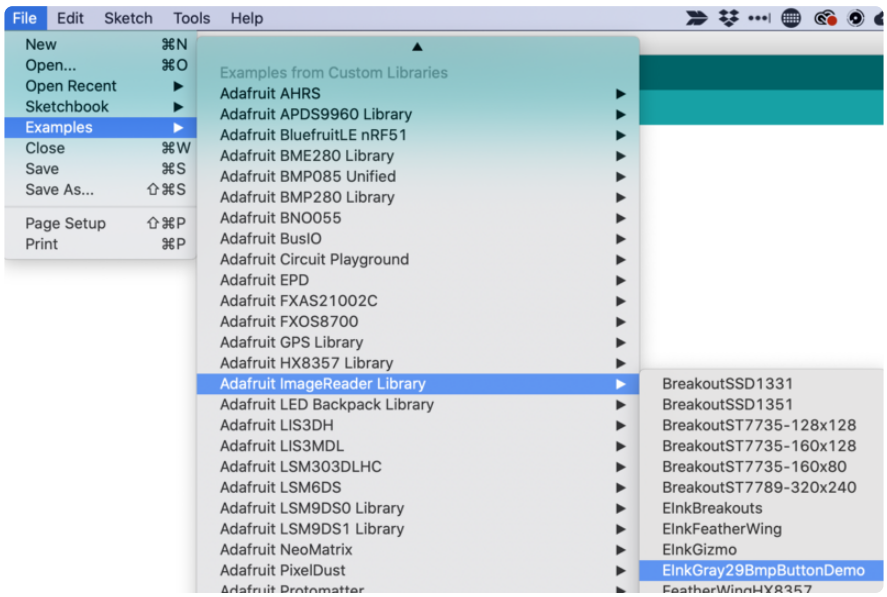
Download 29gray4.bmp

https://adafru.it/OdG

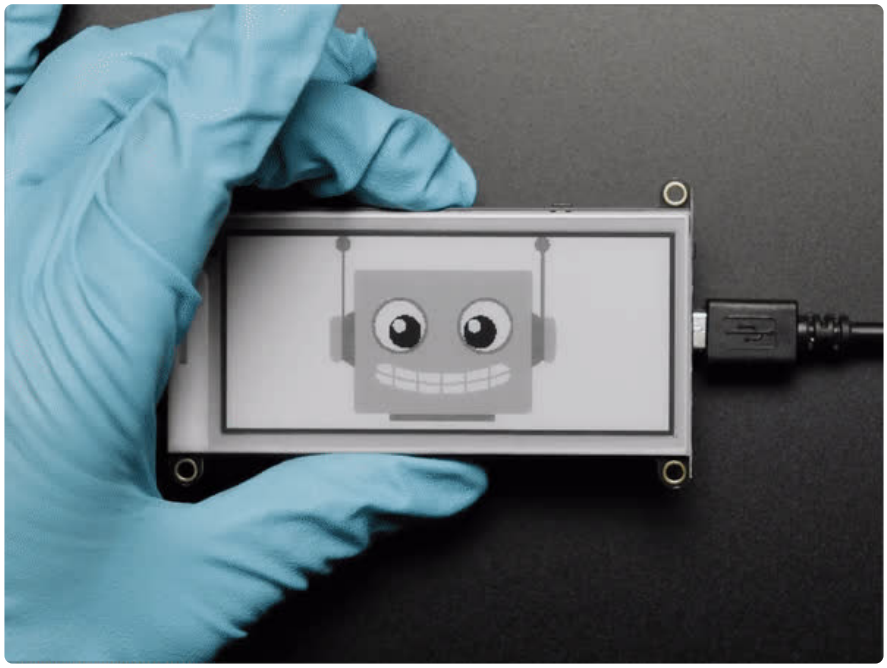
Rename the files to **panda_head.bmp**, **adabot_head.bmp**, and **29gray4.bmp** and place them into the base directory of a microSD card and insert it into the microSD socket in the breakout.

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

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



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



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 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).

CircuitPython eInk displayio Library Installation

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

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

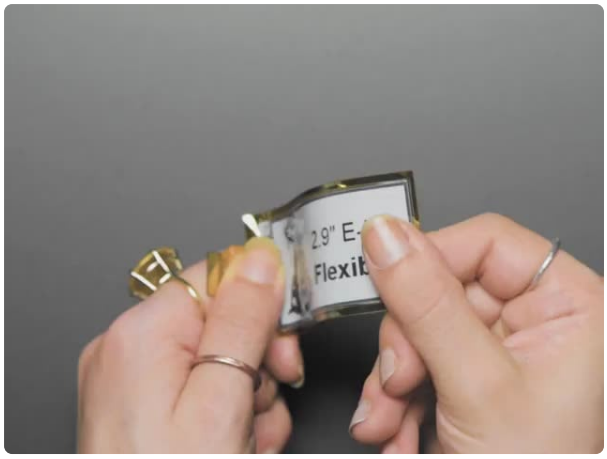
Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). The introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU)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

You will need to copy the appropriate displayio driver from the bundle **lib** folder to a **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.

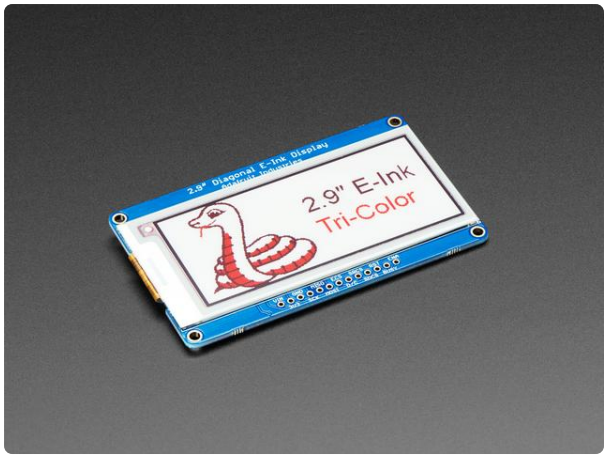
To use the eInk displays with displayio, you will need to use the latest version of CircuitPython and a board that can fit `displayio`. See the Support Matrix to determine if `displayio` is available on a given board: https://circuitpython.readthedocs.io/en/latest/shared-bindings/support_matrix.html

Adafruit_CircuitPython_UC8151D

The newer 2.9" Tri-Color and flexible displays use the **Adafruit_CircuitPython_UC8151D** library. To easily get all the required files and libraries for the flexible, you can click the **Download Project Bundle** link at the top of the monochrome 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.



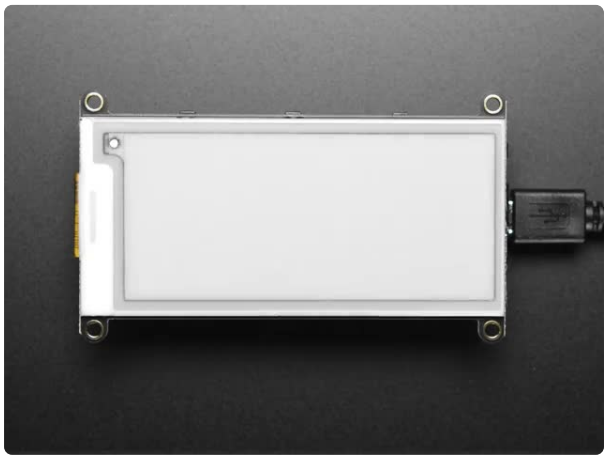
[2.9" Flexible 296x128 Monochrome eInk / ePaper Display]
Woah, the cyber-future is here! Flexible E-Ink has been demo'd at high-tech events for years but now you can actually get your paws on it. This display is true E-Ink / E-Paper,...
https://www.adafruit.com/product/4262



[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_CircuitPython_IL0373

The older 2.9" Tri-Color breakout, the 2.9" Tri-Color and GrayScale FeatherWings, and the older flexible displays 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 appropriate 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.



[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

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

Image File

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

Download display-ruler.bmp

https://adafru.it/Tsa

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

Monochrome Display Usage

2.9" 296x128 Monochrome Flexible Display

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
import adafruit_uc8151d

# For 8.x.x and 9.x.x. When 8.x.x is discontinued as a stable release, change this.
try:
    from fourwire import FourWire
```

```
except ImportError:
    from displayio import FourWire

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()

with open("/display-ruler.bmp", "rb") as f:
    pic = displayio.OnDiskBitmap(f)
    t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
    g.append(t)

    display.root_group = g

    display.refresh()

    time.sleep(120)
```

Configure and Upload

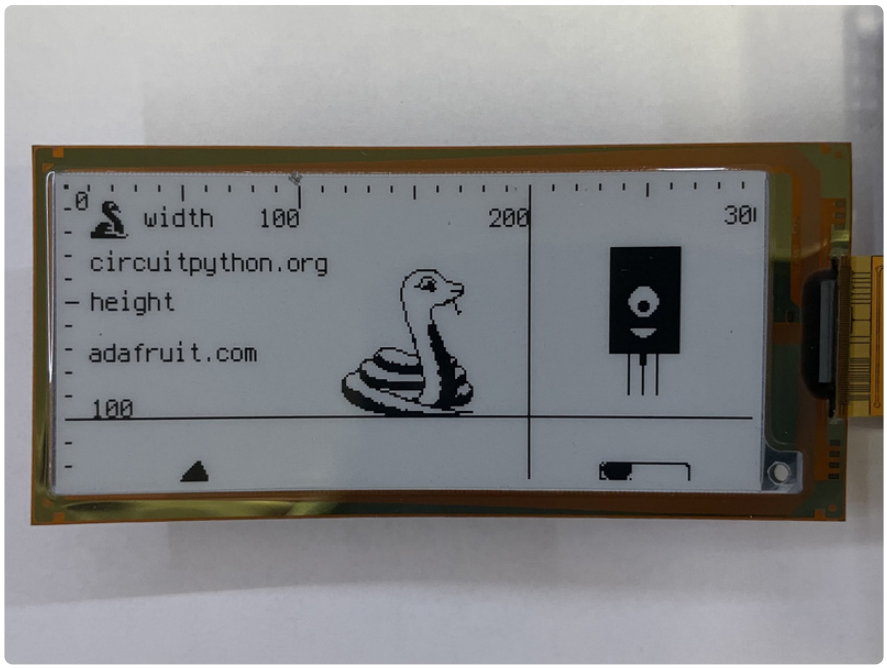
You will want to change the `epd_reset` and `epd_busy` to **the correct values**. If you are using the eInk Breakout Friend (http://adafru.it/4224) and wired it up as shown on the Wiring page, you will want to change it to these values:

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

If you are using the eInk FeatherWing Friend (http://adafru.it/4446), 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:



Tri-Color Display Usage

2.9" 296x128 HD Tri-Color Breakout

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

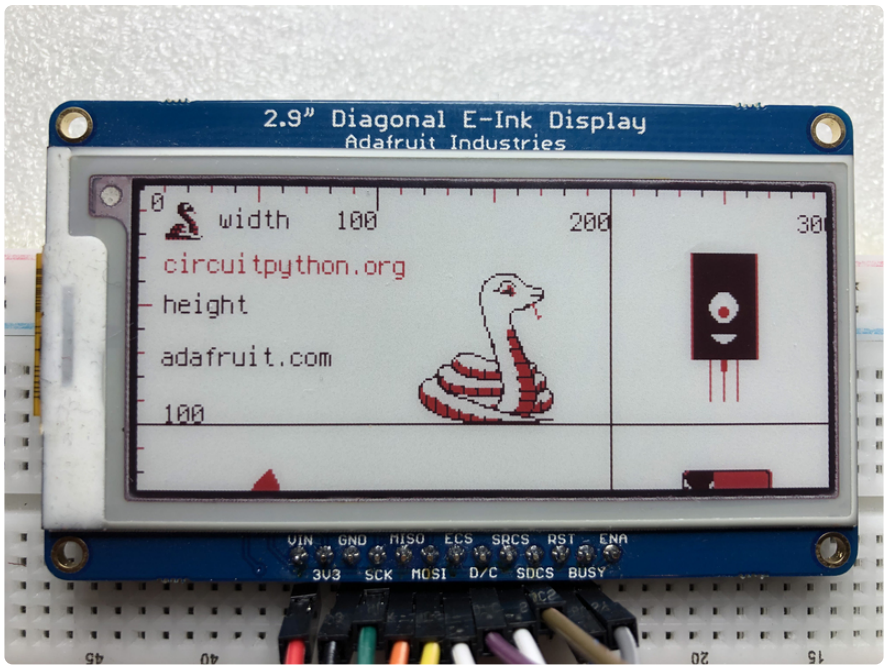
```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""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
import adafruit_uc8151d

# For 8.x.x and 9.x.x. When 8.x.x is discontinued as a stable release, change this.
try:
    from fourwire import FourWire
except ImportError:
    from displayio import FourWire

# 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
with open("/display-ruler.bmp", "rb") as f:
    pic = displayio.OnDiskBitmap(f)
    # Create a Tilegrid with the bitmap and put in the displayio group
    # CircuitPython 6 & 7 compatible
    t = displayio.TileGrid(
        pic, pixel_shader=getattr(pic, "pixel_shader", displayio.ColorConverter())
    )
    # CircuitPython 7 compatible only
    # t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
    g.append(t)

    # 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:



2.9" 296x128 HD Tri-Color FeatherWing

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
with open("/display-ruler.bmp", "rb") as f:
    pic = displayio.OnDiskBitmap(f)
    # Create a Tilegrid with the bitmap and put in the displayio group
    # CircuitPython 6 & 7 compatible
    t = displayio.TileGrid(
        pic, pixel_shader=getattr(pic, "pixel_shader", displayio.ColorConverter())
    )
    # CircuitPython 7 compatible only
    # t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
    g.append(t)

    # 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)
```

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 should look like the Tri-Color Breakout output above.

Grayscale Display Usage

2.9" 296x128 Grayscale Display

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

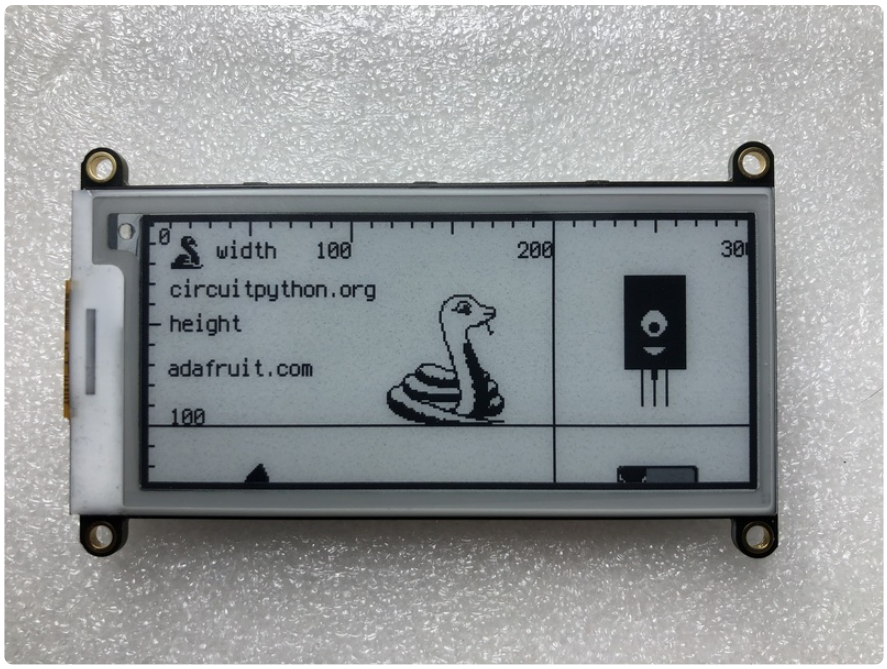
```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

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

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

```
import time
import busio
import board
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()

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

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



Python Setup

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

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://adafru.it/BSN).

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

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

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

Python Installation of EPD Library

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

```
sudo pip3 install adafruit-circuitpython-epd
```

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

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

```
sudo apt-get install python3-pip
```

Download font5x8.bin

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

Download font5x8.bin

https://adafru.it/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
```

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

Python Usage

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

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

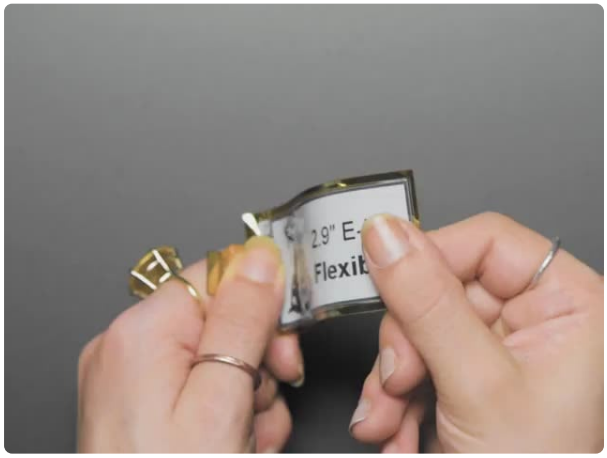
To demonstrate the usage of the display, we'll initialize it and draw some lines from the Python REPL.

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,...

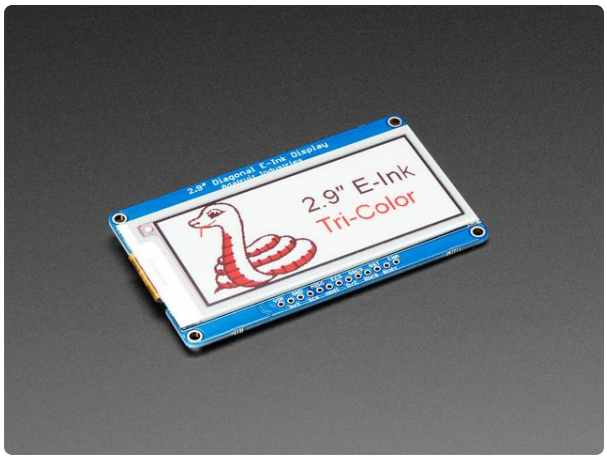
https://www.adafruit.com/product/4262

```
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=srcs,
rst_pin=rst, busy_pin=busy)
display.set_black_buffer(1, False)
display.set_color_buffer(1, False)
```

Run the following code to initialize the 2.9" 296x128 Tri-Color display:



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

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

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.

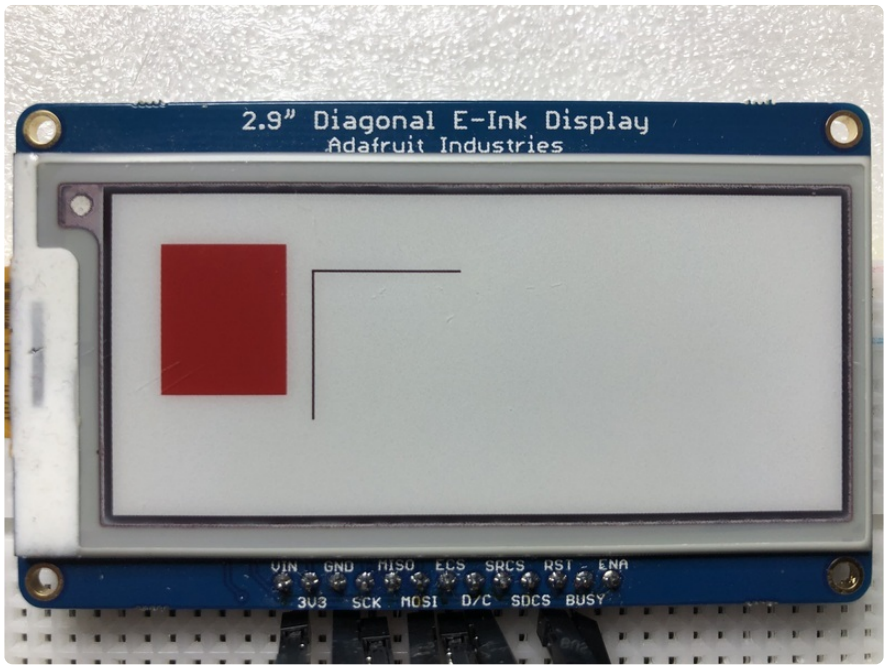
```
display.rotation = 3
display.fill(Adafruit_EPD.WHITE)

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

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

display.display()
```

Your display will look something like this:



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

Full Example Code

Here is the full example code.

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

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

import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import
from adafruit_epd.ek79686 import Adafruit_EK79686  # pylint: disable=unused-import

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

# give them all to our drivers
print("Creating display")
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_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_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

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

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

display.rotation = 1

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

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

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

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

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.

Click here to download blinka for the 2.9" Tri-Color 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 digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import
from adafruit_epd.ek79686 import Adafruit_EK79686  # pylint: disable=unused-import


# create the spi device and pins we will need
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
```

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

# give them all to our driver
print("Creating display")
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_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_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

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

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

display.rotation = 0

FILENAME = "blinka.bmp"


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


class BMPError(Exception):
    pass


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

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

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

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

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

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

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

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

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

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


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

Before running it, 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:

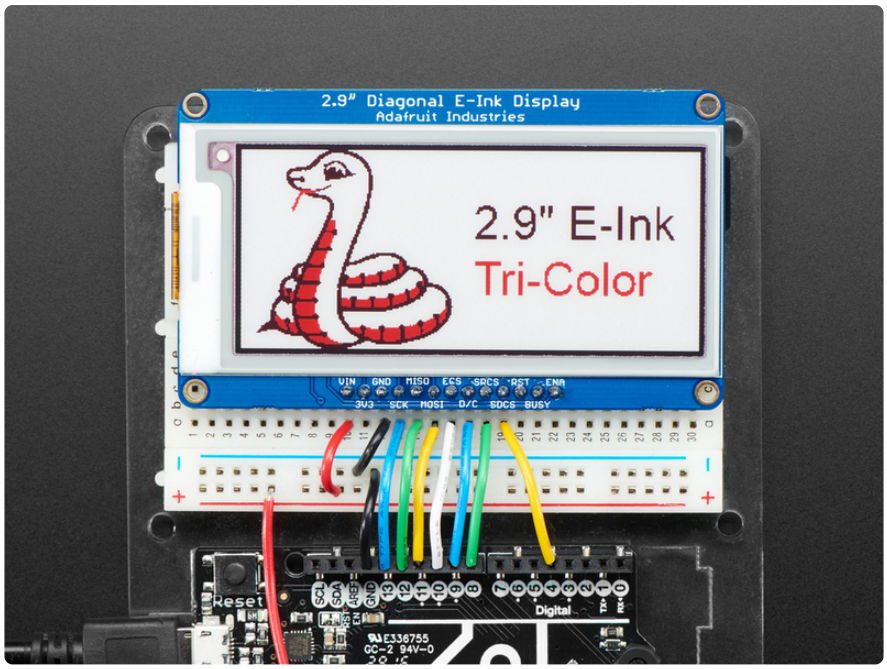
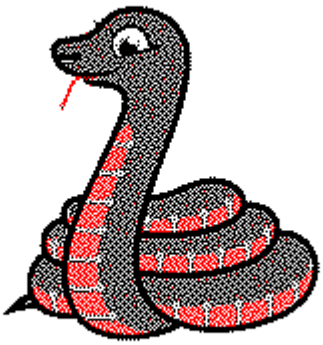


Image Drawing with Pillow

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 digitalio
import busio
import board
from PIL import Image
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import
from adafruit_epd.ek79686 import Adafruit_EK79686  # pylint: disable=unused-import


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

# give them all to our driver
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color or mono display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_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_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

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

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

display.rotation = 1

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

# Scale the image to the smaller screen dimension
```

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

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

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

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

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 Blinka image, which is named **blinka.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("blinka.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 eInks, 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 &lt; image_ratio:
    scaled_width = image.width * display.height // image.height
```

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

Next 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")
```

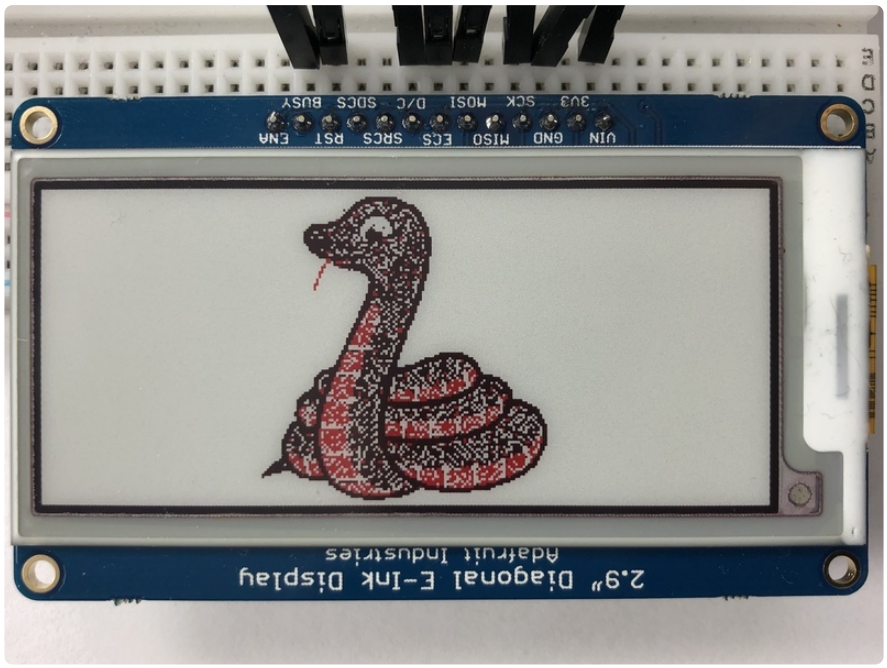
Finally, take the `image`, draw it to the frame buffer and `display` it. At this point, the image should have the exact same dimensions at the display and fill it completely.

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

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

```
python3 epd_pillow_image.py
```

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



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 digitalio
import busio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import
from adafruit_epd.ssd1680 import Adafruit_SSD1680  # pylint: disable=unused-import
from adafruit_epd.ssd1681 import Adafruit_SSD1681  # pylint: disable=unused-import
from adafruit_epd.uc8151d import Adafruit_UC8151D  # pylint: disable=unused-import
from adafruit_epd.ek79686 import Adafruit_EK79686  # pylint: disable=unused-import

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

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

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

# give them all to our driver
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color or mono display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_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_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
```

```
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

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

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

display.rotation = 1

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

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

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

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

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

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

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

Just like in the last example, 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
```

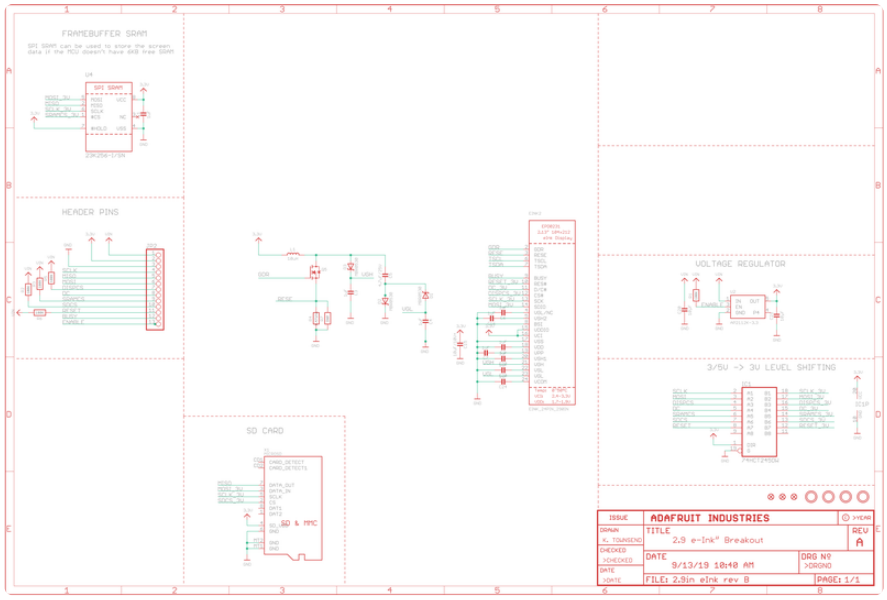
After a few seconds, your display should show this image:

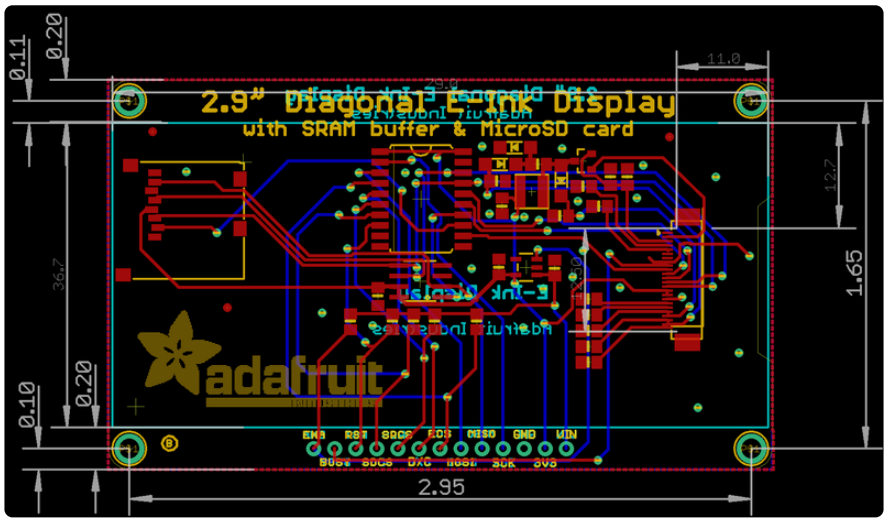
Downloads

Files

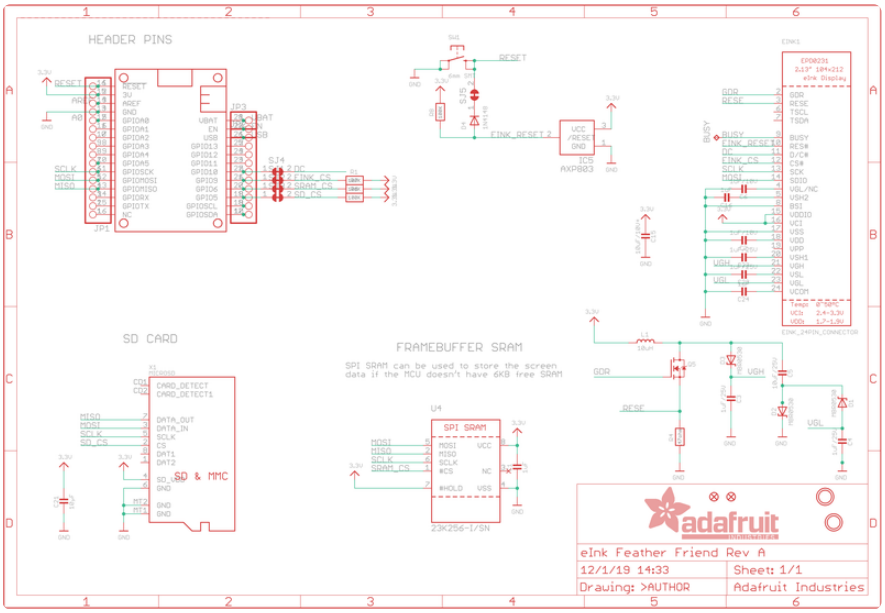
- Fritzing object in Adafruit Fritzing Library (https://adafru.it/aP3)
- IL0376F E-Ink interface chip datasheet (https://adafru.it/BRW)
- PCB Files on GitHub (https://adafru.it/BRX)

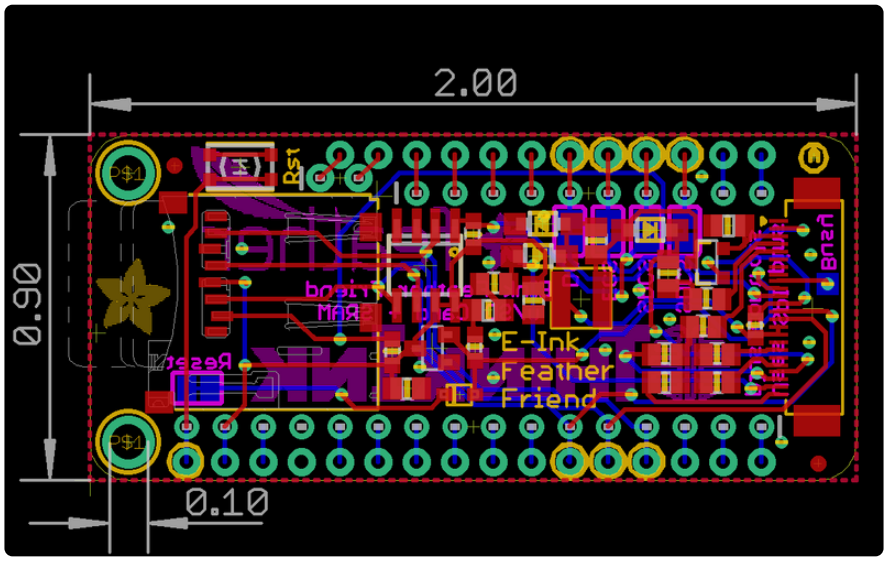
Schematic & Fabrication Prints

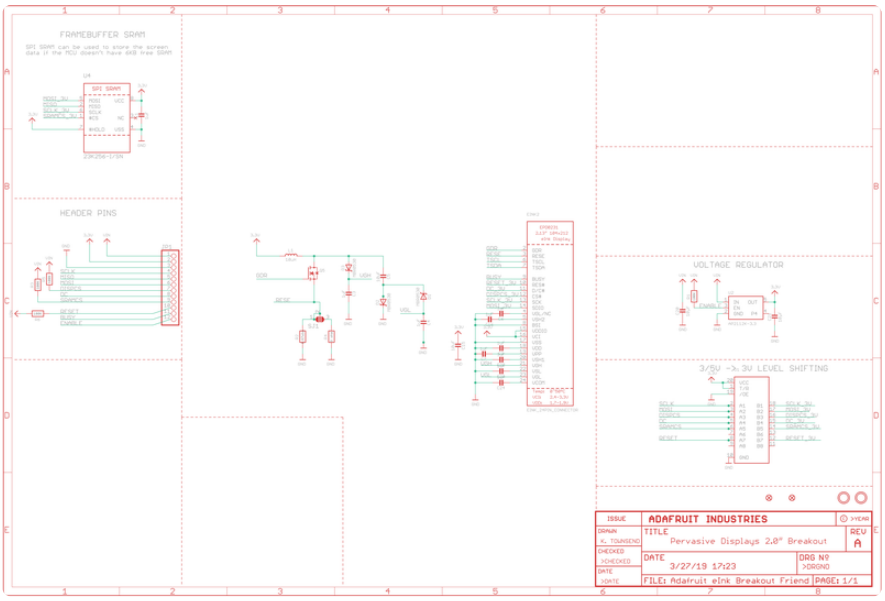


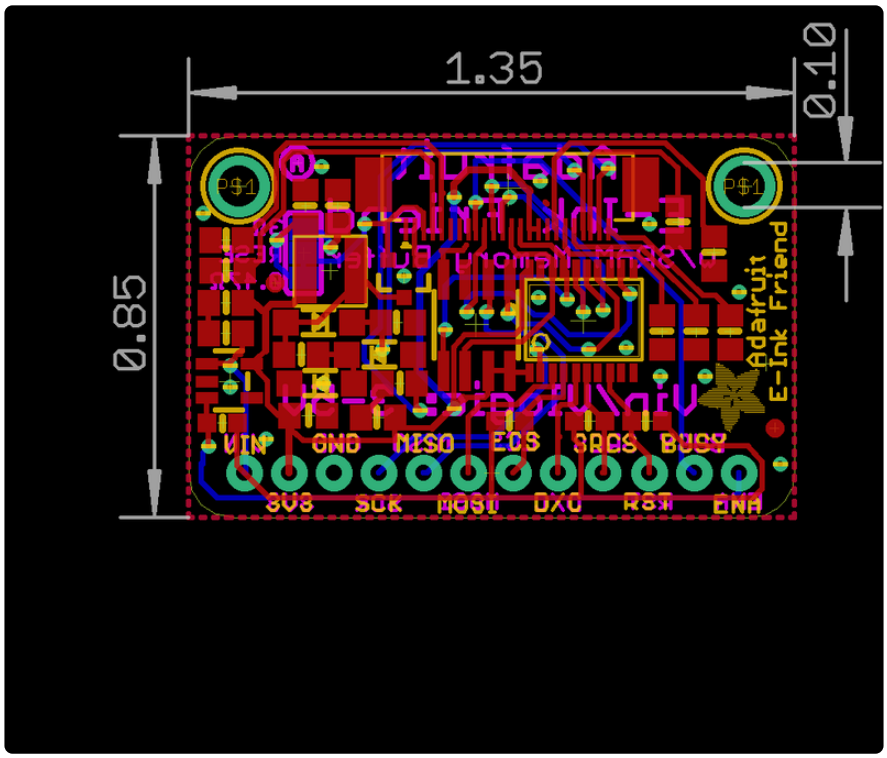


eInk Friends






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


1.35
0.10
0.85
VIN
GND
MISO
ECS
SRCS
BUSY
3V3
SCK
MOSI
DXC
RST
DNA
Adafruit
E-Ink Friend