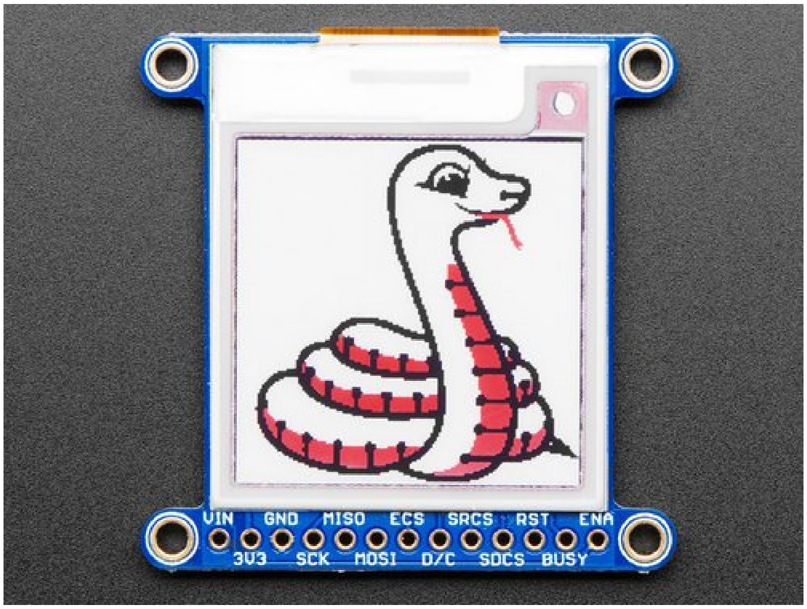
adafruit learning system

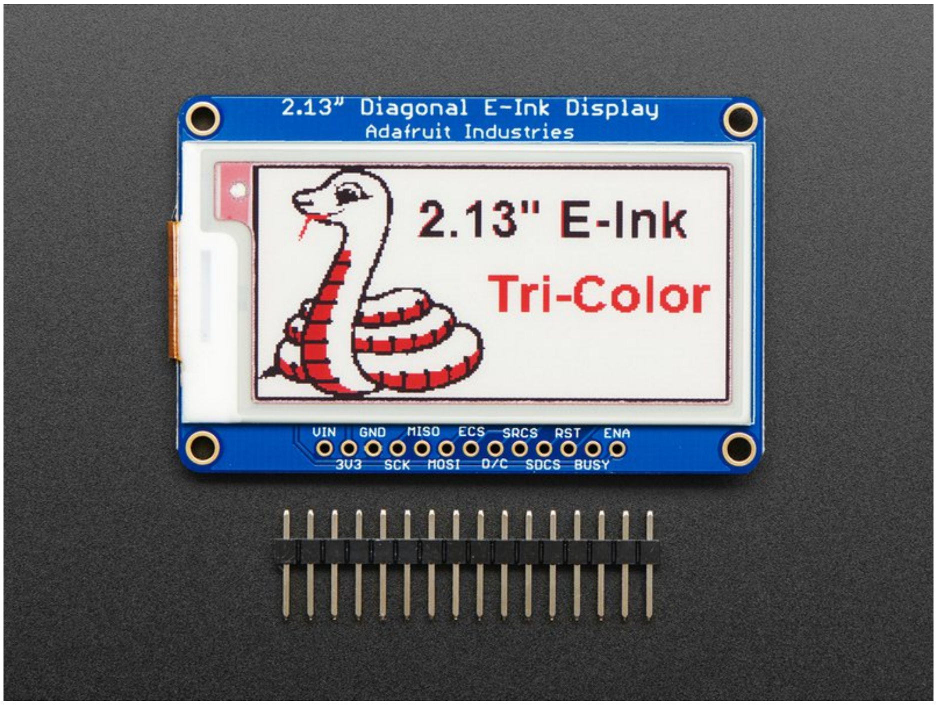
Adafruit eInk Display Breakouts

Created by lady ada



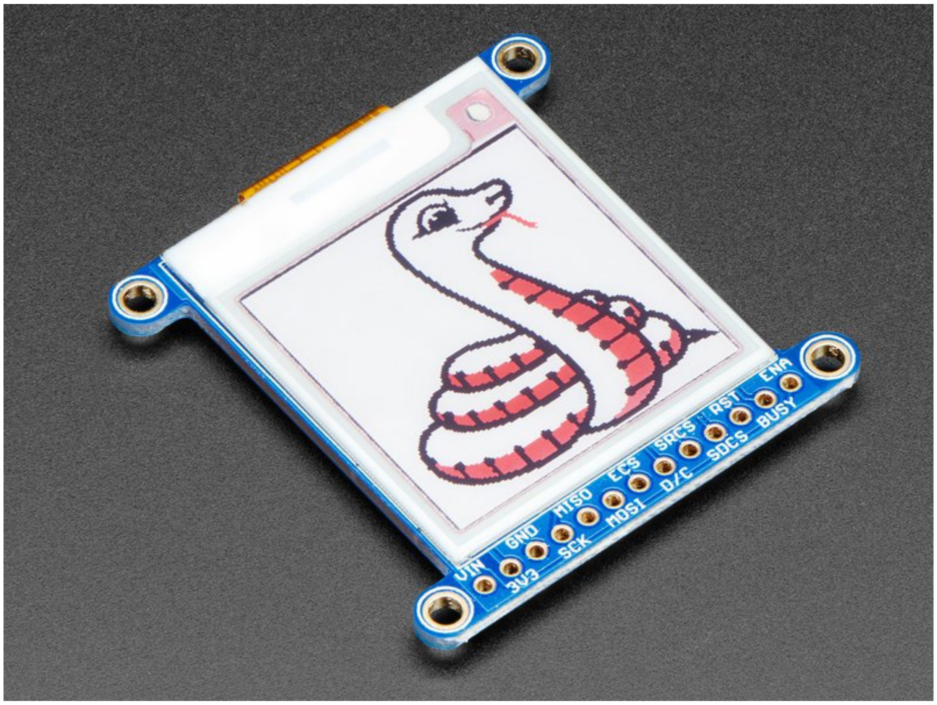
Last updated on 2019-04-23 06:49:22 PM UTC

Overview



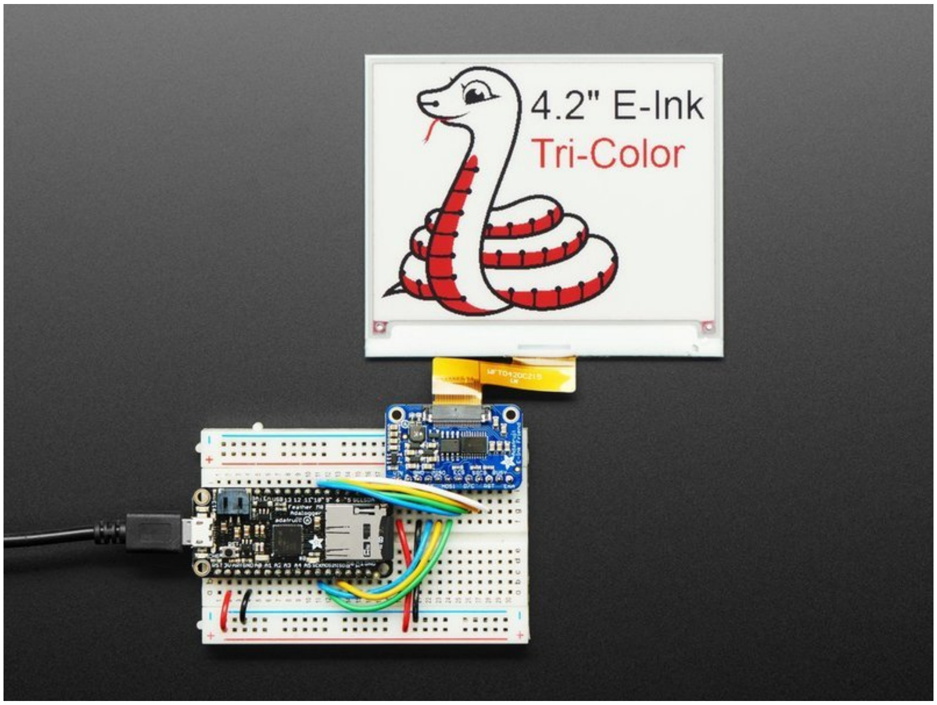
Easy e-paper finally comes to microcontrollers, with these breakouts, shields and friends that are designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those new-fangled 'e-readers' like the Kindle or Nook. They have gigantic electronic paper 'static' displays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

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



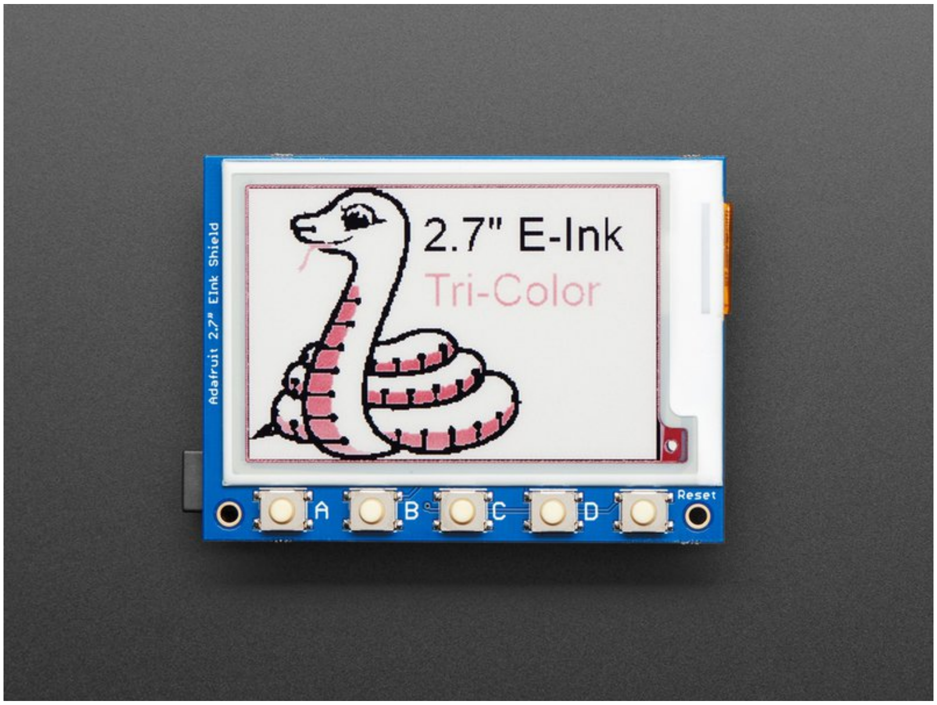
We have multiple tri-color displays. They have black and red ink pixels and a white-ish background. Using our Arduino library, you can create a 'frame buffer' with what pixels you want to have activated and then write that out to the display. Most simple breakouts leave it at that. But if you do the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

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

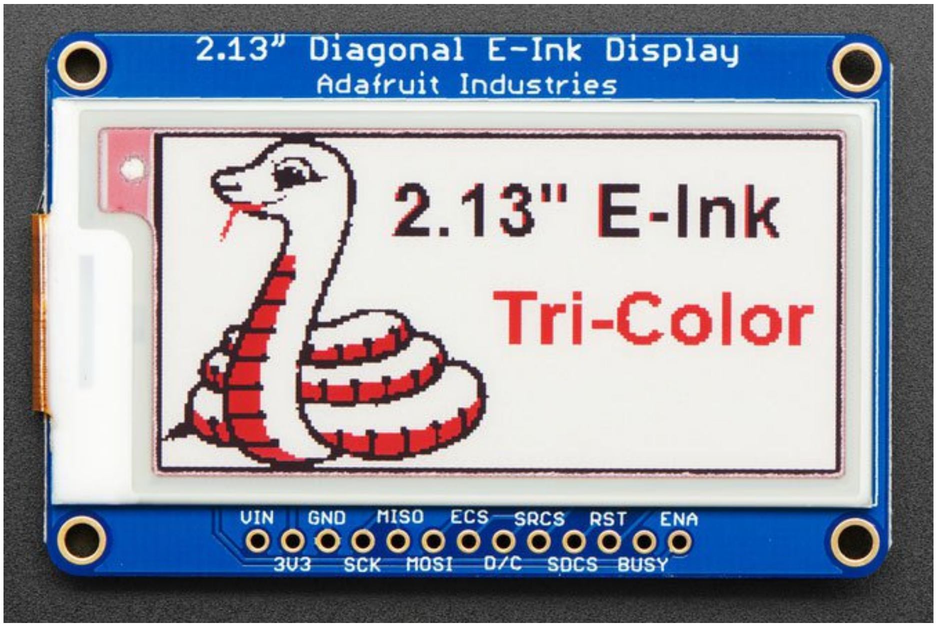


On the EInk Friends and Breakouts, for ultra-low power usages, the onboard 3.3V regulator has the Enable pin brought out so you can shut down the power to the SRAM, MicroSD and display.

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



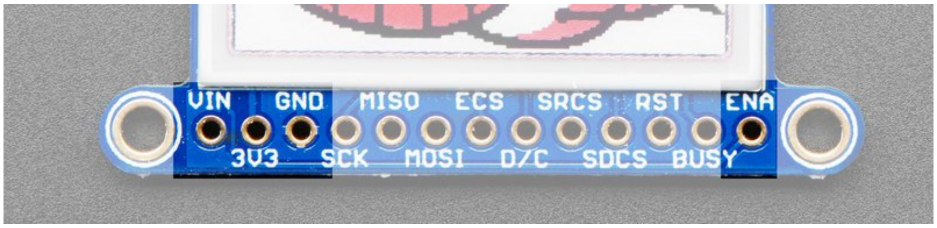
Pinouts


2.13" Diagonal E-Ink Display
Adafruit Industries
2.13" E-Ink
Tri-Color
VIN GND MISO ECS SRCS RST ENA
3V3 SCK MOSI D/C SDCS BUSY

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

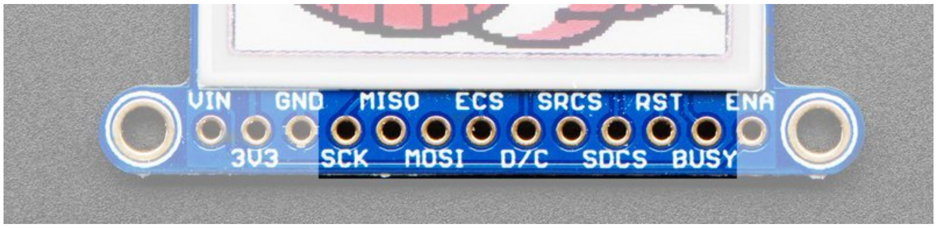
The pin outs are identical for the 1.54", 2.13" and 2.7" E-Ink display!

Power Pins



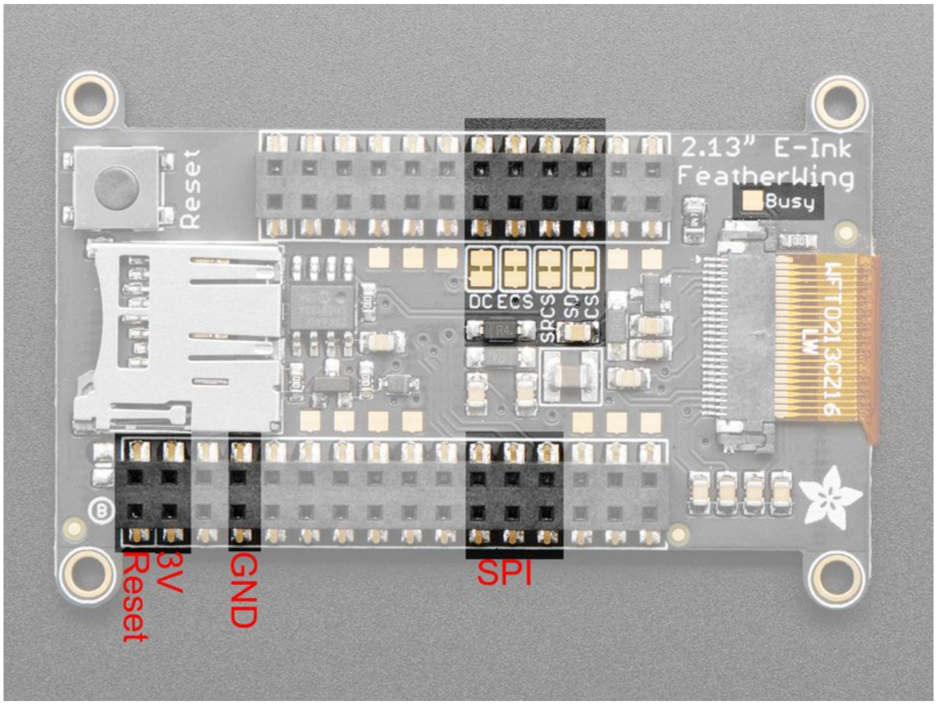
- **3-5V** / **Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3.3V** out - this is the 3.3V output from the onboard regulator, you can 'borrow' about 100mA if you need to power some other 3.3V logic devices
- **GND** - this is the power 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 Master In Slave 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 Master Out Slave 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 version is a little more compact but has just about the same pins as the breakout

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

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

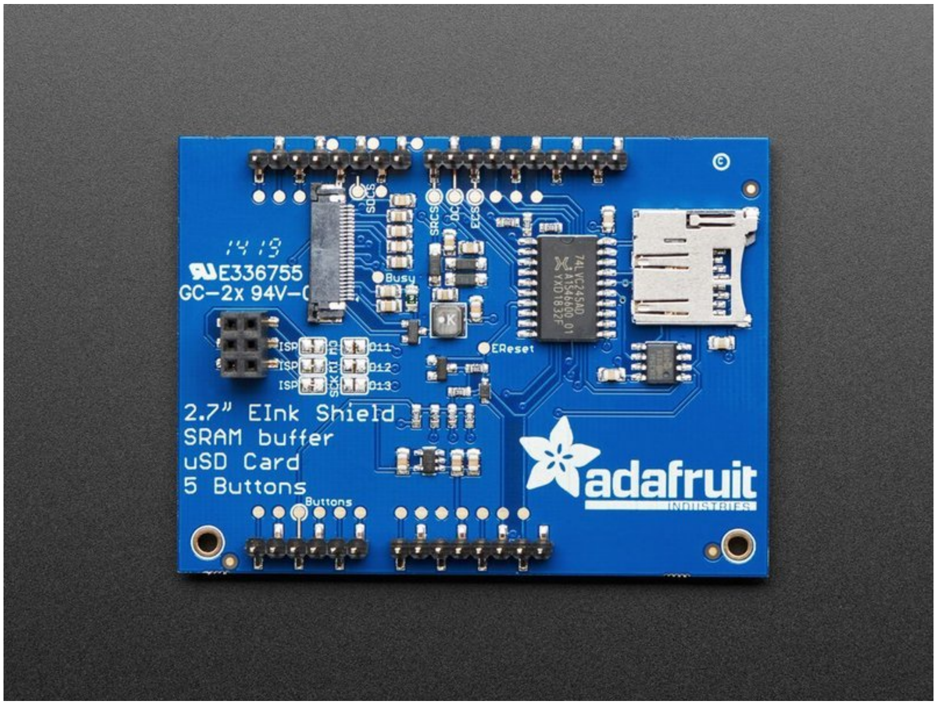
- **SD CS** to Pin **D5**
- **SRAM CS** to Pin **D6**
- **EINK CS** to Pin **D9**
- **EINK DC** to Pin **D10**

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

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

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

Shield Pinouts



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

Power Pins

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

Data Pins

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

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

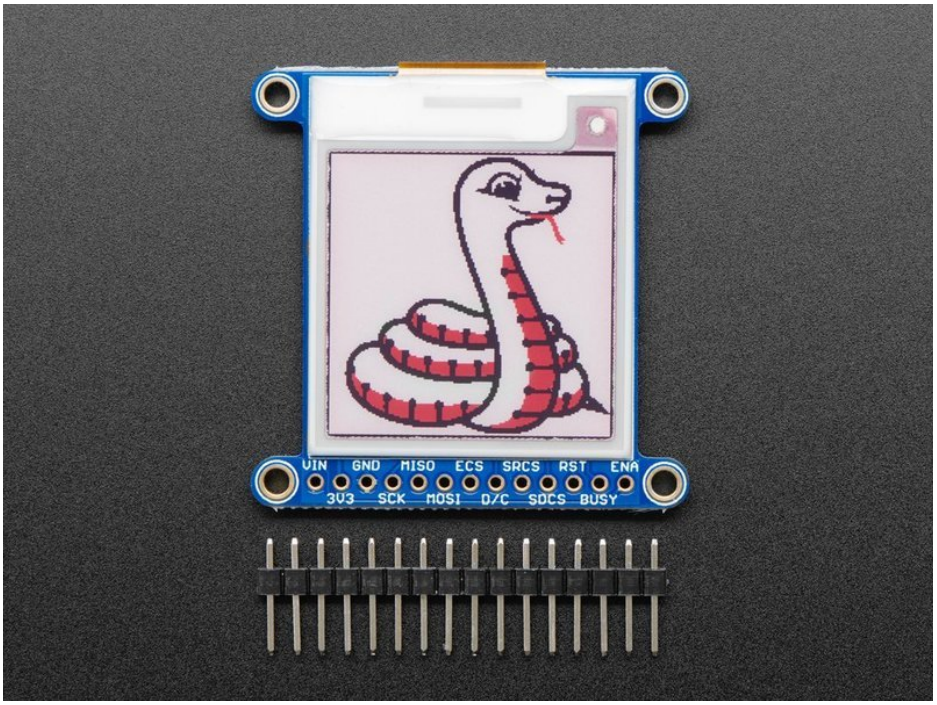
The **RESET** pin is connected to the microcontroller reset pin, but is available on a pad labeled **EReset** if you want to toggle it yourself!

Buttons

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

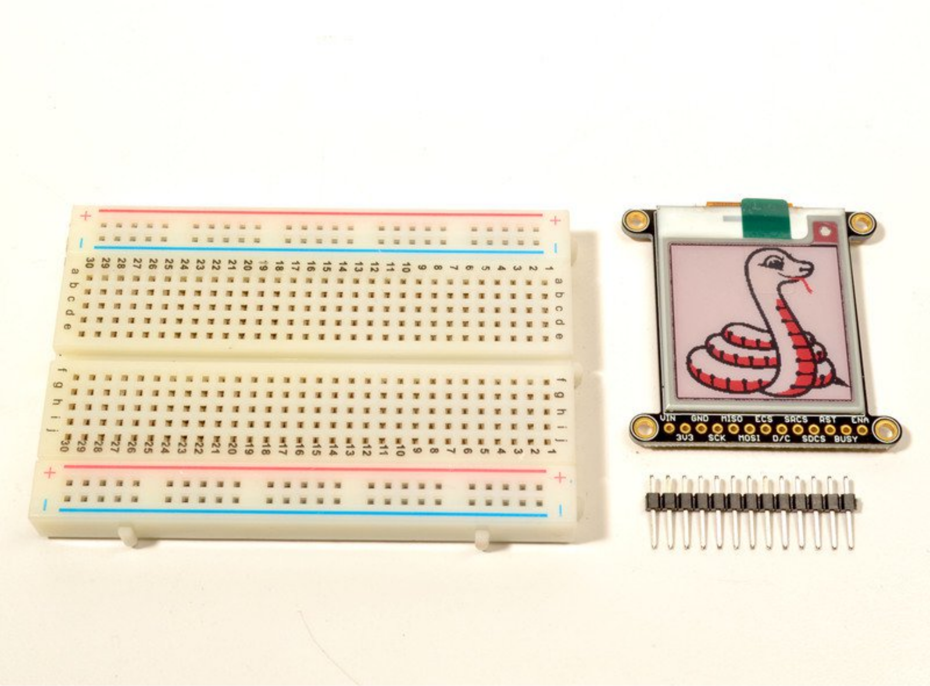
```
int8_t readButtons(void) {
  uint16_t reading = analogRead(A3);
  //Serial.println(reading);

  if (reading > 600) {
    return 0; // no buttons pressed
  }
  if (reading > 400) {
    return 4; // button D pressed
  }
  if (reading > 250) {
    return 3; // button C pressed
  }
  if (reading > 125) {
    return 2; // button B pressed
  }
  return 1; // Button A pressed
}
```

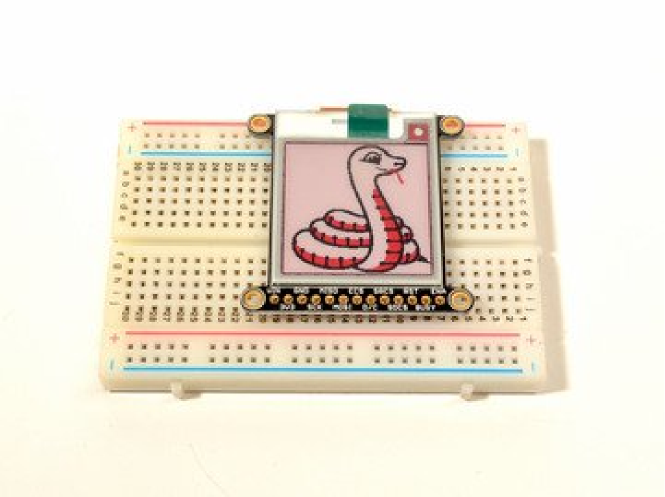
Assembly



Assembly

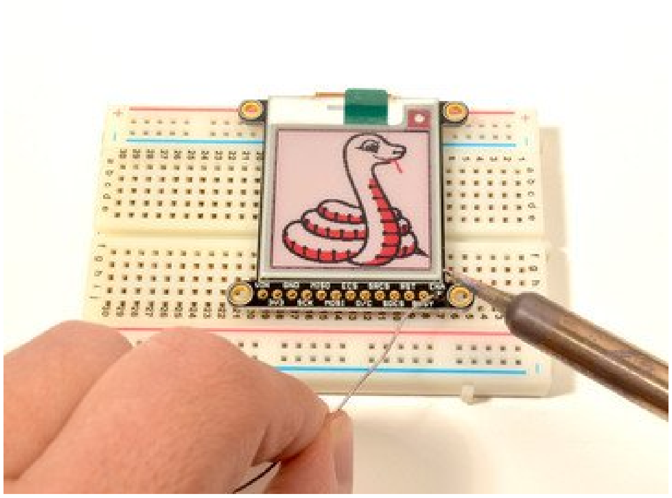
Cut the header down to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

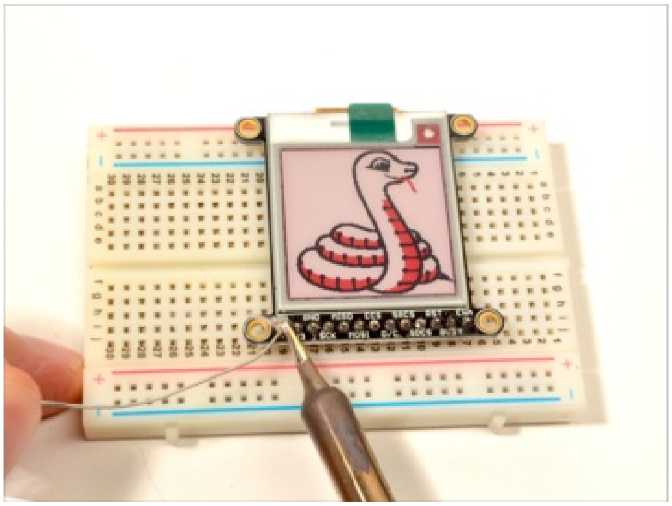


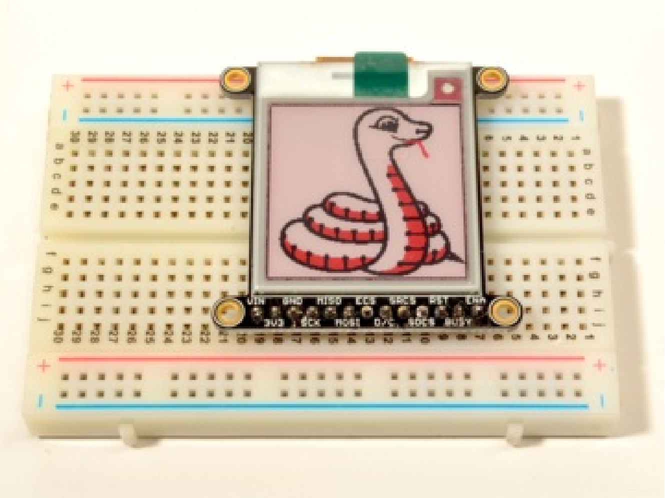


Add the E-Ink Display

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





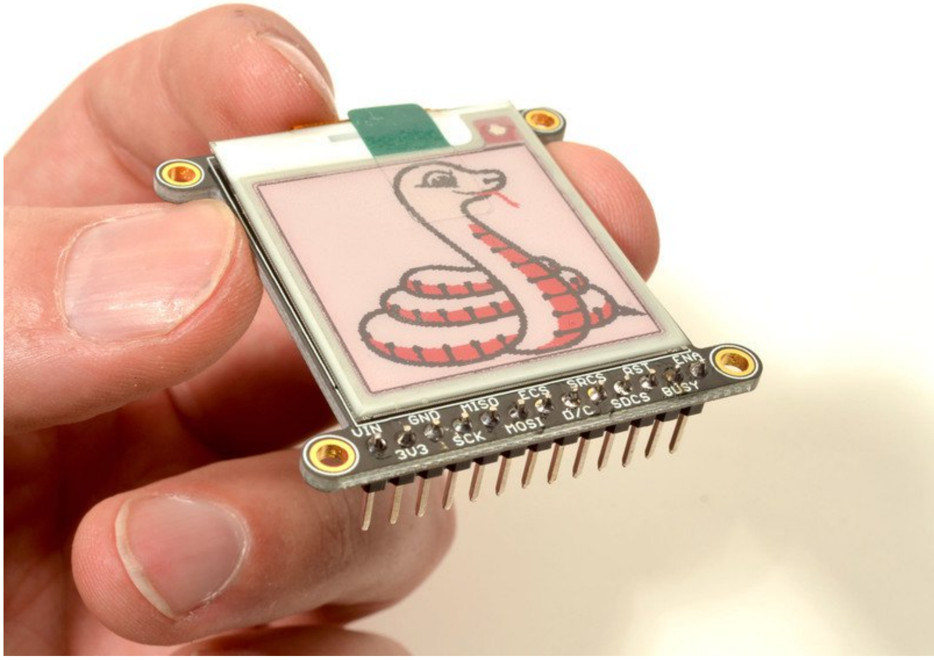


And Solder!

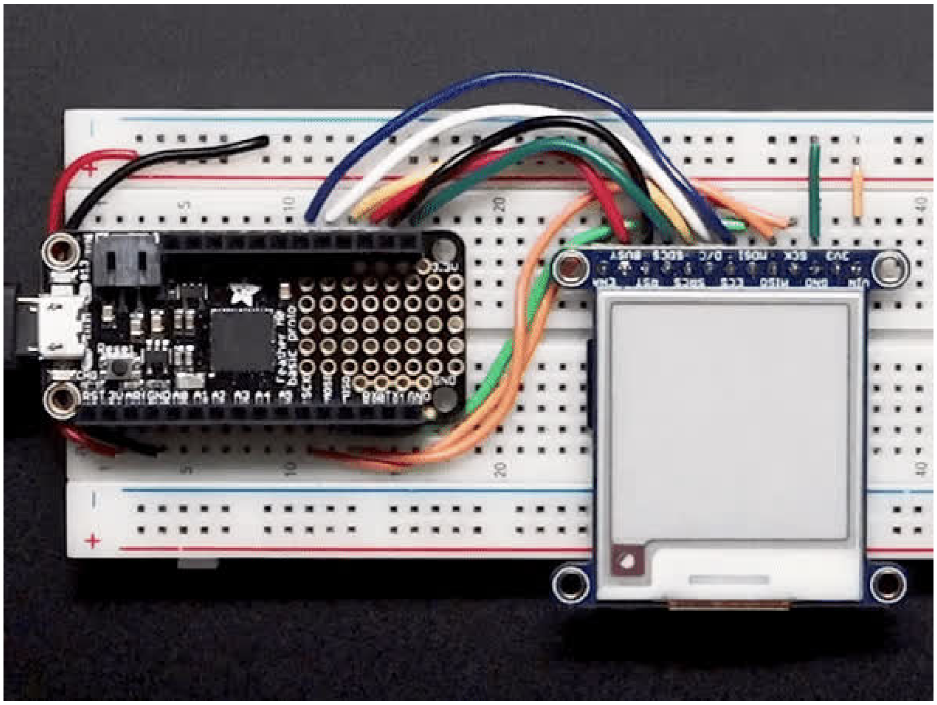
Be sure to solder all pins for reliable electrical contact.

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

OK, you're done!


VIN
GND
MISO
ECS
SRCS
RST
ENA
3V3
SCK
MOSI
D/C
SDCS
BUSY

Usage & Expectations



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

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

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

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

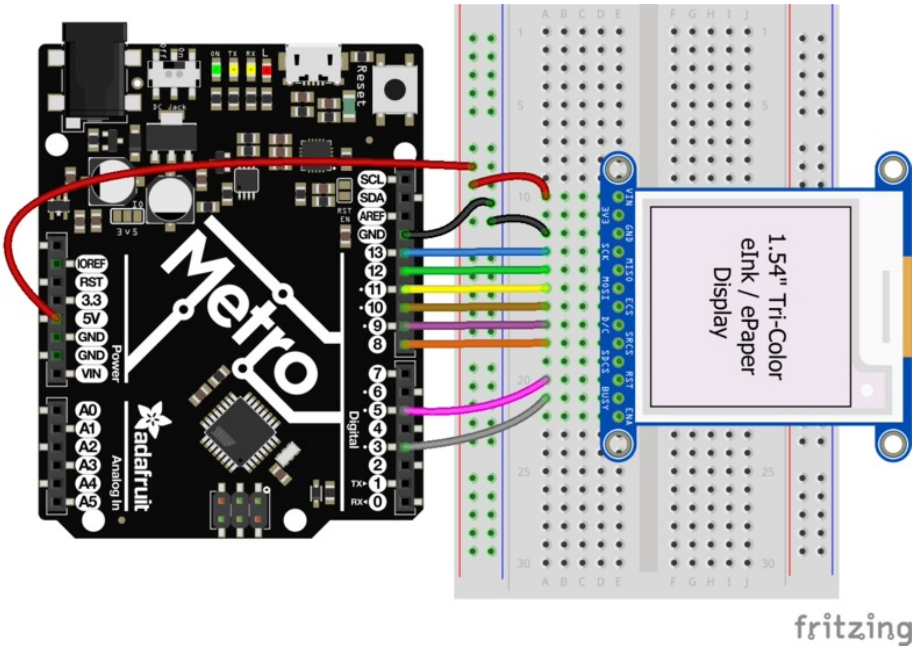
Arduino Code

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

Wiring

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

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



Start by connecting the power pins

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

Required SPI Pins

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

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

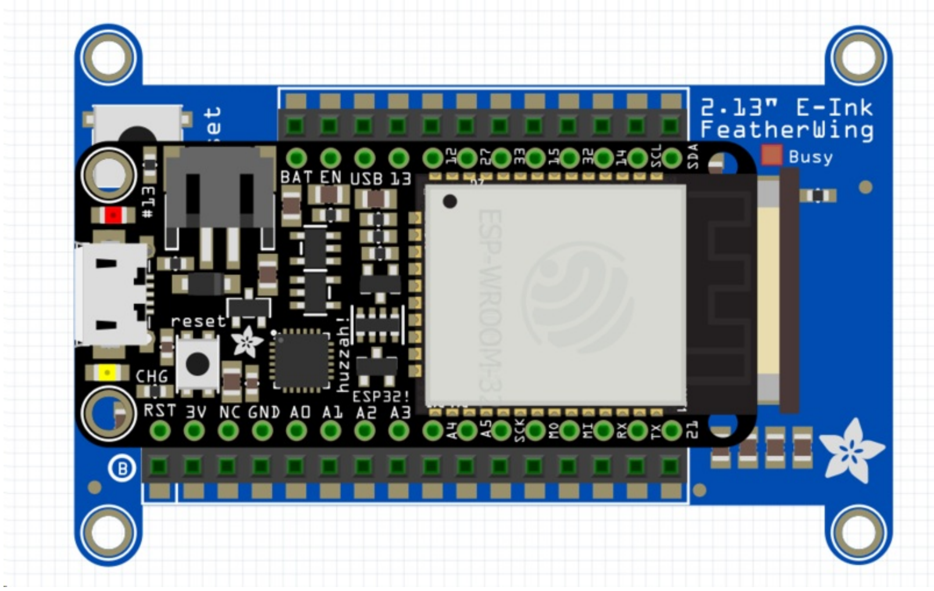
Other Digital I/O Pins

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

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

FeatherWing Connection

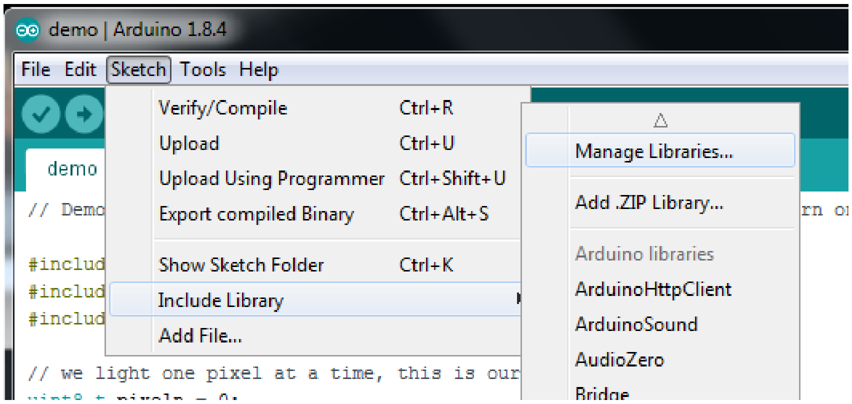
FeatherWing usage is easy, simply plug your Feather into the Wing



Install Adafruit_EPD & GFX libraries

To begin reading sensor data, 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.

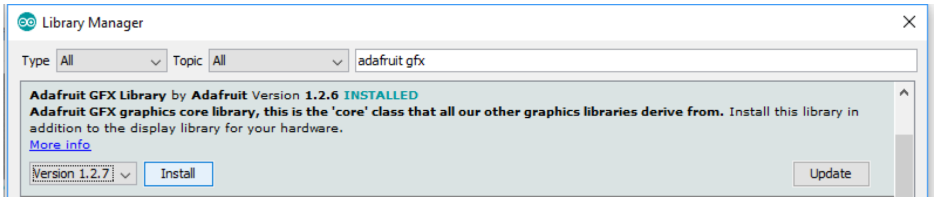
From the IDE open up the library manager...



```
// Demo
#includ
#includ
#includ
// we light one pixel at a time, this is our
```

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

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

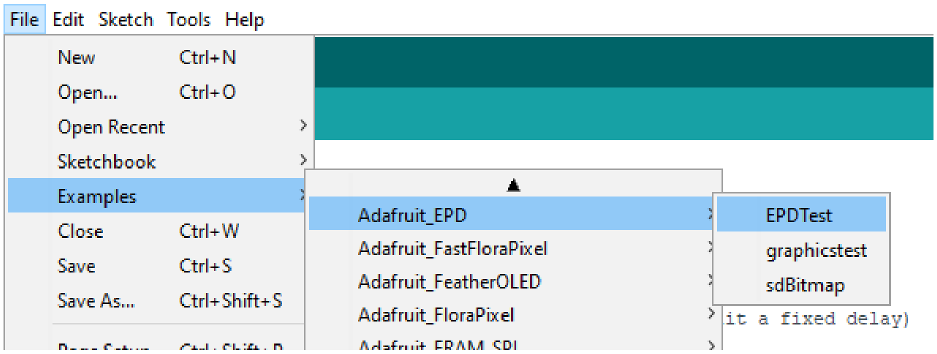


Adafruit GFX Library by **Adafruit** Version **1.2.6** **INSTALLED**
Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from. Install this library in addition to the display library for your hardware.
More info

Load First Demo

Open up **File->Examples->Adafruit_EPD->EPDtest**

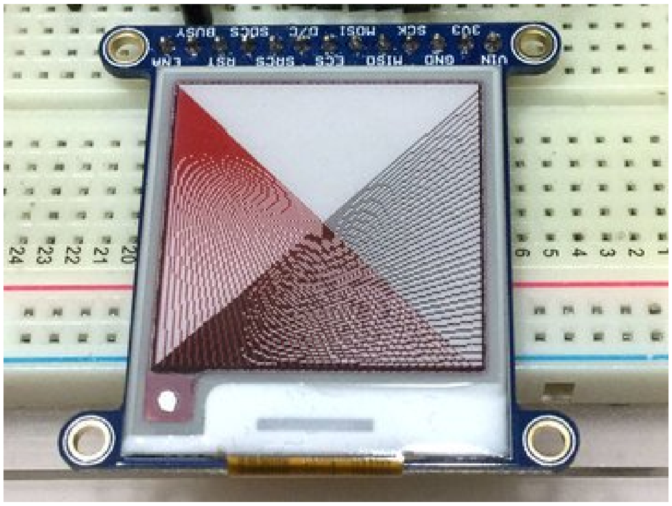
If you're using a FeatherWing, try **File->Examples->Adafruit_EPD->FeatherWingTest**



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

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

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



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

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

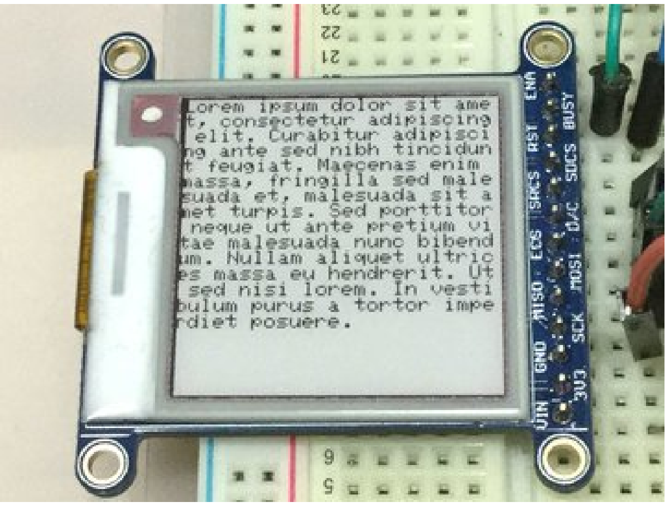
Load Graphics Test Demo

Open up **File->Examples->Adafruit_EPD->graphicstest** and upload to your microcontroller wired up to the display

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

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

and copy those into the top of the graphics test sketch



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

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

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

Unnecessary Pins

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

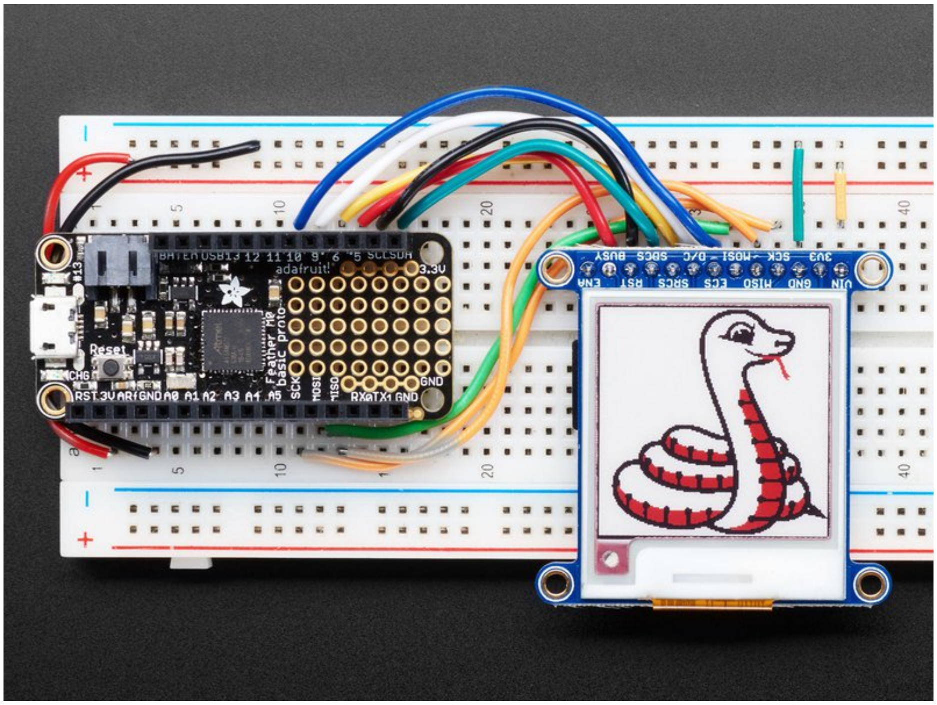
You can set the code as below to remove control of those pins from the Adafruit_EPD library:

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

Thus saving you two pins!

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

Drawing Bitmaps



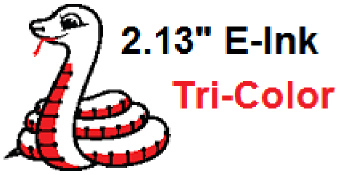
Not only can you draw shapes but you can also load images from the SD card, perfect for static images!

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

https://adafru.it/BTa

https://adafru.it/BTa

For the 2.13" display, use this image instead

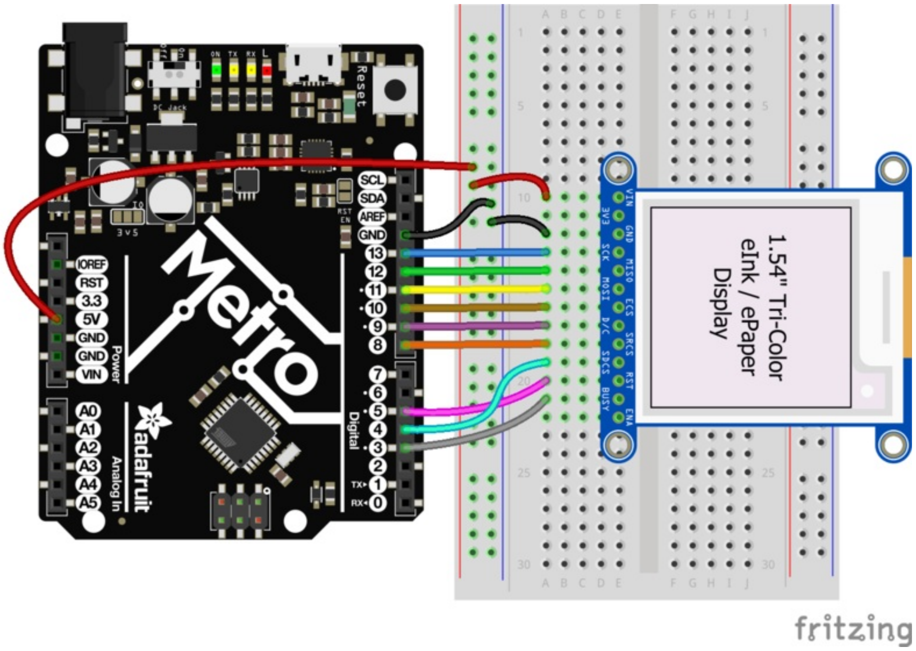


https://adafru.it/EaE

https://adafru.it/EaE

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

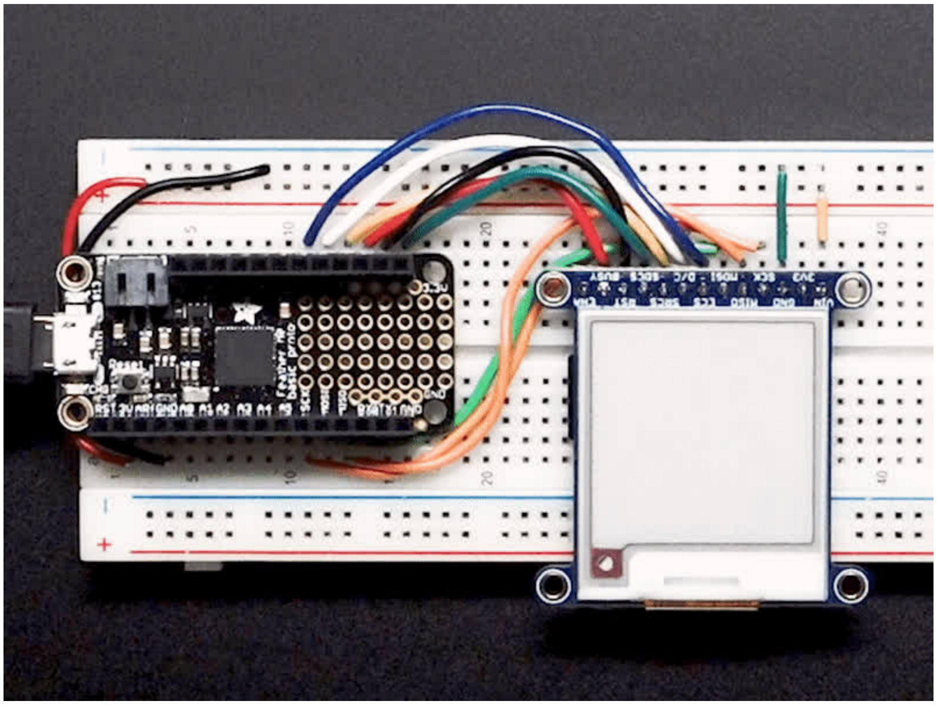
One extra wire is required, for **SDCS** which is the SD card Chip Select. We'll connect that to pin #4 but you can use any pin.



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

Open the **file->examples->Adafruit_EPD->spitftbitmap** example

Upload to the upload & you will see blinka appear!



Arduino Library Documentation

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

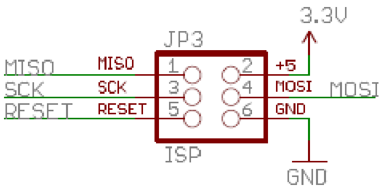
CircuitPython Code

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

CircuitPython Microcontroller Wiring

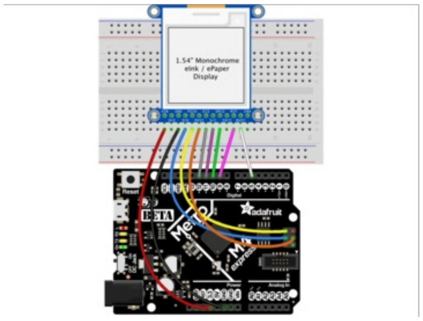
First, wire up your eInk breakout as shown below.

Note: to get to **MISO**, **MOSI**, and **SCLK** on Arduino-style boards, they are on the ISP header at the back of the board rather than on the pin headers on the sides. See the schematic pinout below.



1.54" eInk Display

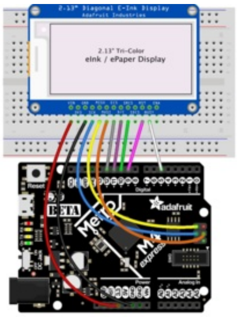
The wiring is the same for the 1.54" monochrome and tri-color displays! The following is an example of a 1.54" monochrome eInk display wired to a Metro M4:



- **Metro 3.3** to **display VIN**
- **Metro GND** to **display GND**
- **Metro SCK** to **display SCK**
- **Metro MISO** to **display MISO**
- **Metro MOSI** to **display MOSI**
- **Metro D12** to **display ECS**
- **Metro D11** to **display D/C**
- **Metro D10** to **display SRCS**
- **Metro D9** to **display RST**
- **Metro D5** to **display BUSY**

2.13" eInk display

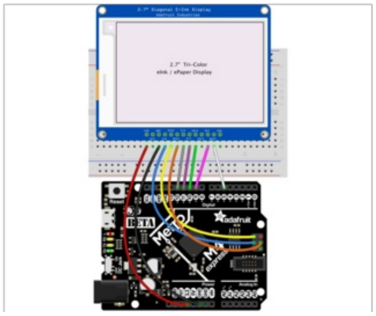
The wiring is the same for the 2.13" monochrome and tri-color displays! The following is an example of a 2.13" tri-color eInk display wired to a Metro M4:



- **Metro 3.3** to **display VIN**
- **Metro GND** to **display GND**
- **Metro SCK** to **display SCK**
- **Metro MISO** to **display MISO**
- **Metro MOSI** to **display MOSI**
- **Metro D12** to **display ECS**
- **Metro D11** to **display D/C**
- **Metro D10** to **display SRCS**
- **Metro D9** to **display RST**
- **Metro D5** to **display BUSY**

2.7" eInk display

The following is an example of a 2.7" tri-color eInk display wired to a Metro M4:



- **Metro 3.3** to **display VIN**
- **Metro GND** to **display GND**
- **Metro SCK** to **display SCK**
- **Metro MISO** to **display MISO**
- **Metro MOSI** to **display MOSI**
- **Metro D12** to **display ECS**
- **Metro D11** to **display D/C**
- **Metro D10** to **display SRCS**
- **Metro D9** to **display RST**
- **Metro D5** to **display BUSY**

CircuitPython EPD Library Installation

You'll need to install the Adafruit CircuitPython EPD (https://adafru.it/BTd) library on your CircuitPython board.

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

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

Copy the following folders from the bundle **lib** folder to a **lib** folder on your **CIRCUITPY** drive:

- **adafruit_epd**
- **adafruit_bus_device**

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

This library also requires a font file to run! In the bundle folder you downloaded, you'll find an **examples** folder. Open this folder and copy the following file to your **CIRCUITPY** drive:

- **font5x8.bin**

Before continuing, make sure your **CIRCUITPY** drive contains the **font5x8.bin** file.

Next connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython >>> prompt.

Usage

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

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

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

spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.D12)
dc = digitalio.DigitalInOut(board.D11)
srcs = digitalio.DigitalInOut(board.D10)
rst = digitalio.DigitalInOut(board.D9)
busy = digitalio.DigitalInOut(board.D5)
```

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

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

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

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

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

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

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

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

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

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

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

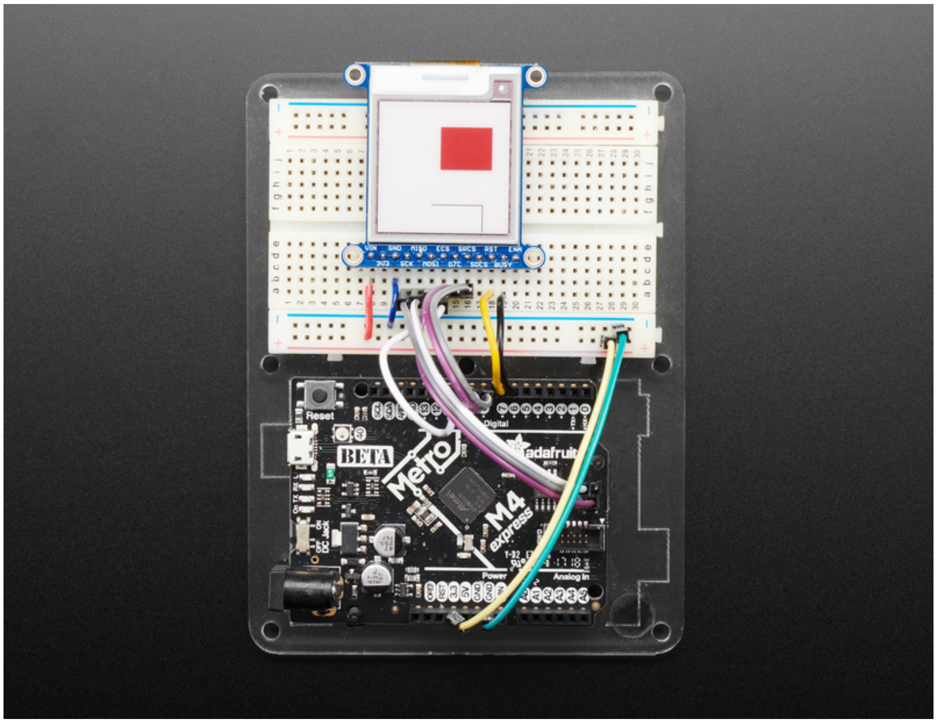
```
from adafruit_epd.ssd1608 import Adafruit_SSD1608
display = Adafruit_SSD1608(200, 200, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                          rst_pin=rst, busy_pin=busy)
```

Tri-Color Example

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

```
display.fill(Adafruit_EPD.WHITE)

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

display.display()
```

Your display will look something like this:



Monochrome Example

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

```
display.fill(Adafruit_EPD.WHITE)

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

display.display()
```

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

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

Tri-Color Bitmap Example

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

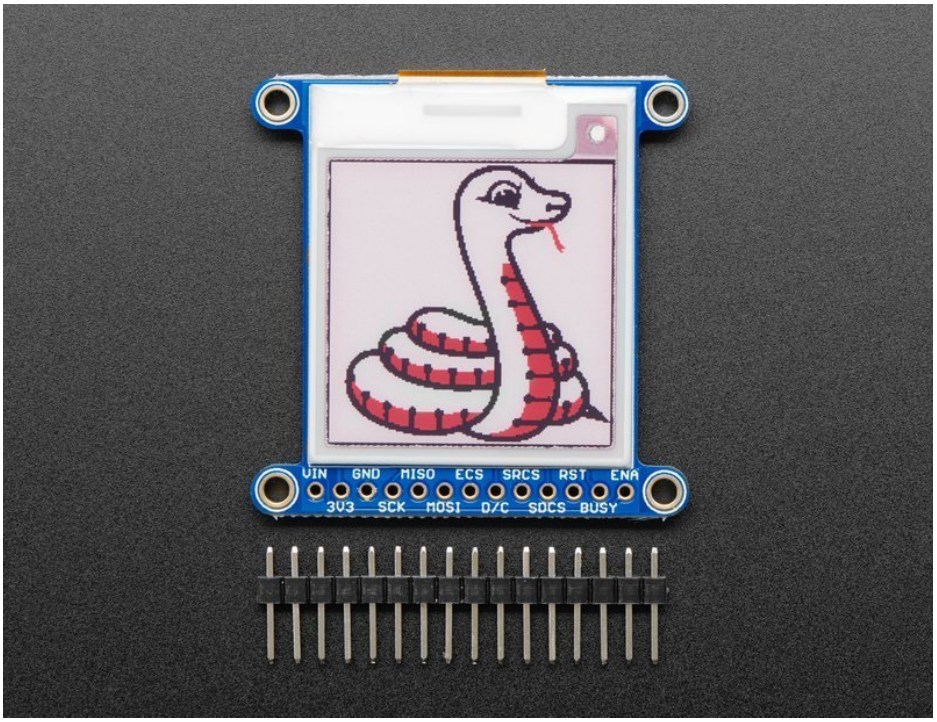
https://adafru.it/BTa

https://adafru.it/BTa

Save the following code as **code.py** on your **CIRCUITPY** drive:

```
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

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

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

display.rotation = 0

FILENAME = "blinka154mono.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")
    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
# clear the buffer
display.fill(Adafruit_EPD.WHITE)
display_bitmap(display, FILENAME)
display.display()
```

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



Full Example Code

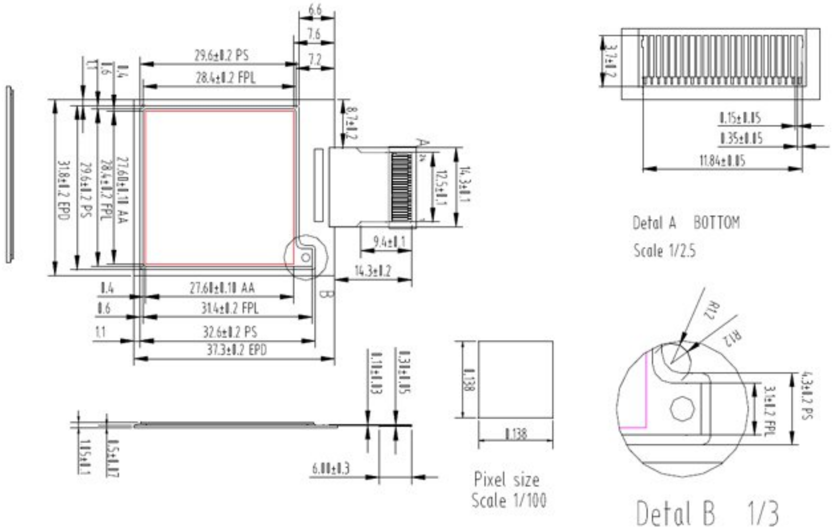
```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874  # pylint: disable=unused-import
from adafruit_epd.il0398 import Adafruit_IL0398  # pylint: disable=unused-import
from adafruit_epd.ssd1608 import Adafruit_SSD1608  # pylint: disable=unused-import
from adafruit_epd.ssd1675 import Adafruit_SSD1675  # pylint: disable=unused-import

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

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

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()
```

Downloads

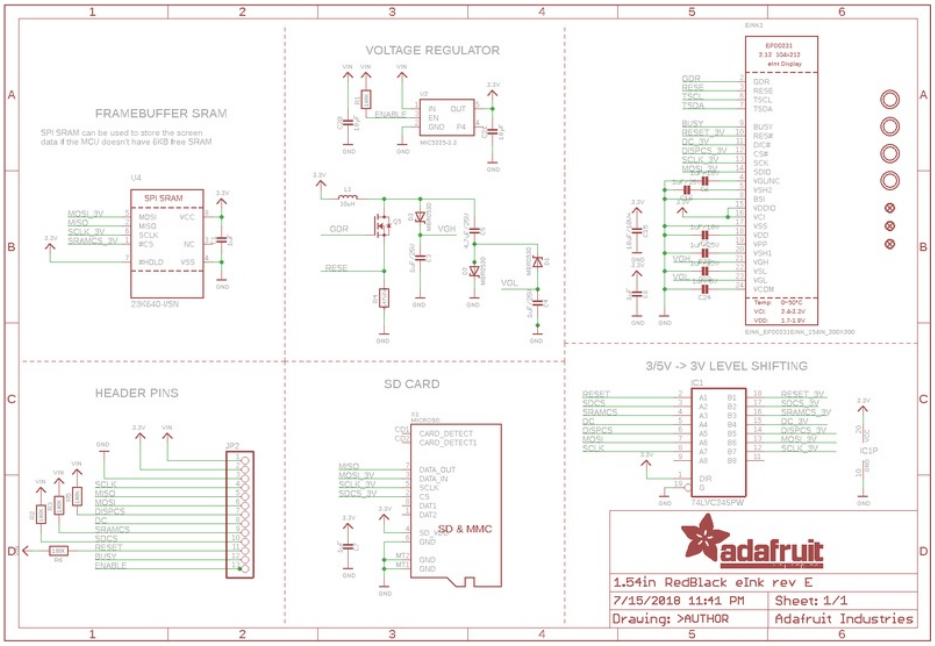
Files

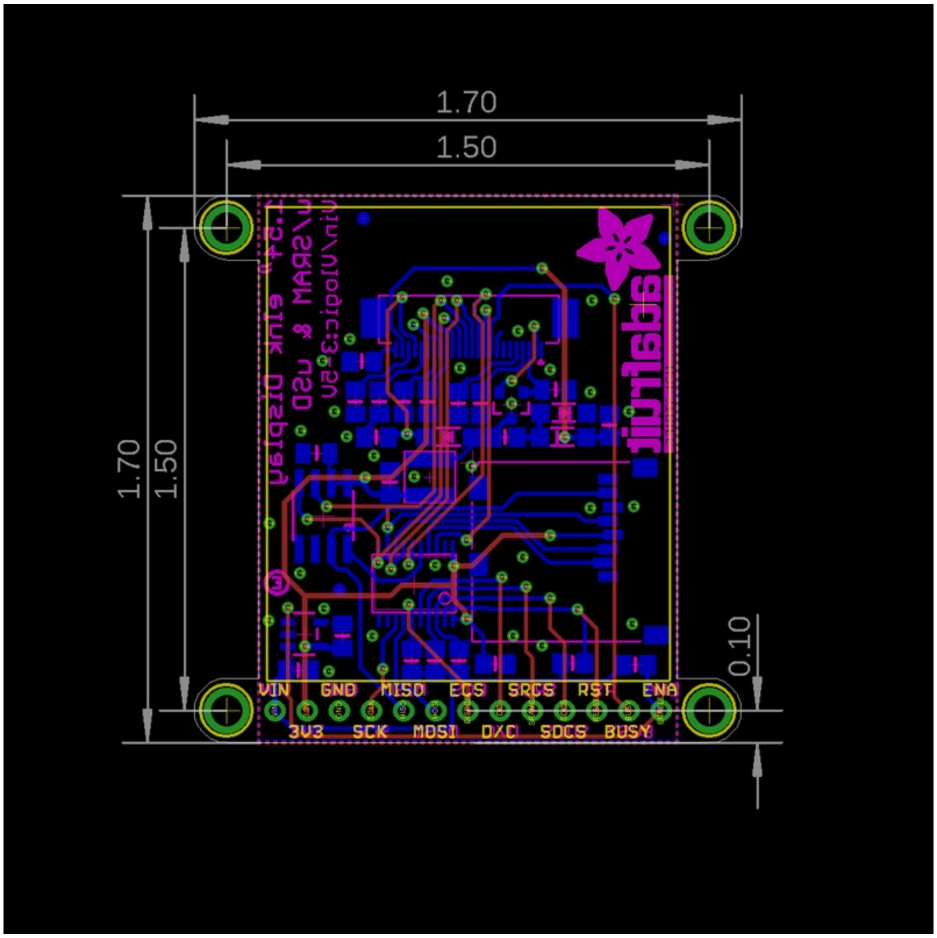
- Fritzing object in Adafruit Fritzing Library (https://adafru.it/aP3)
- IL0376F E-Ink interface chip datasheet (https://adafru.it/BRW)
- PCB Files on GitHub (https://adafru.it/BRX)

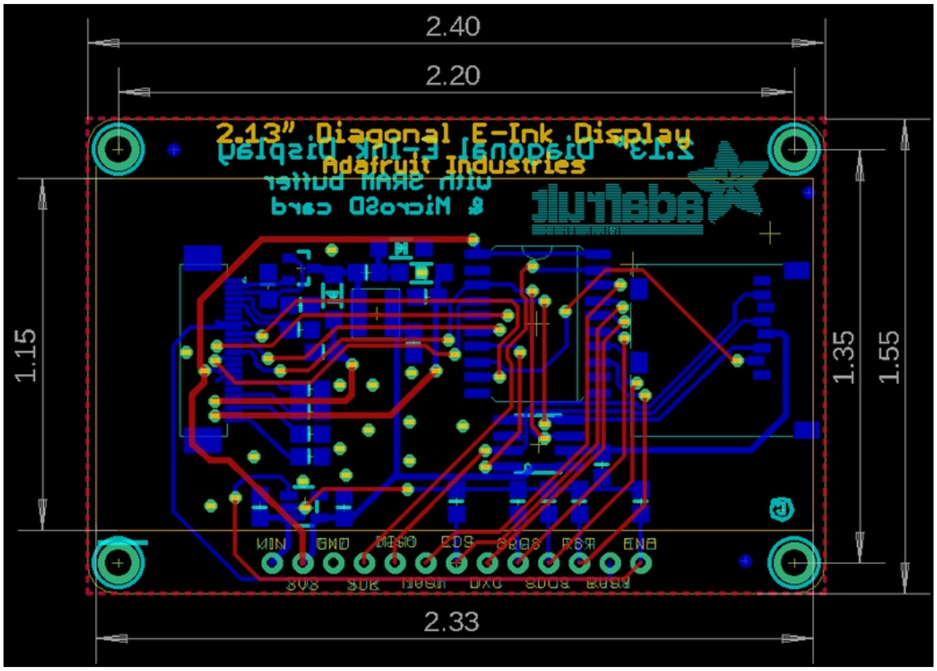
Display shape/outline:

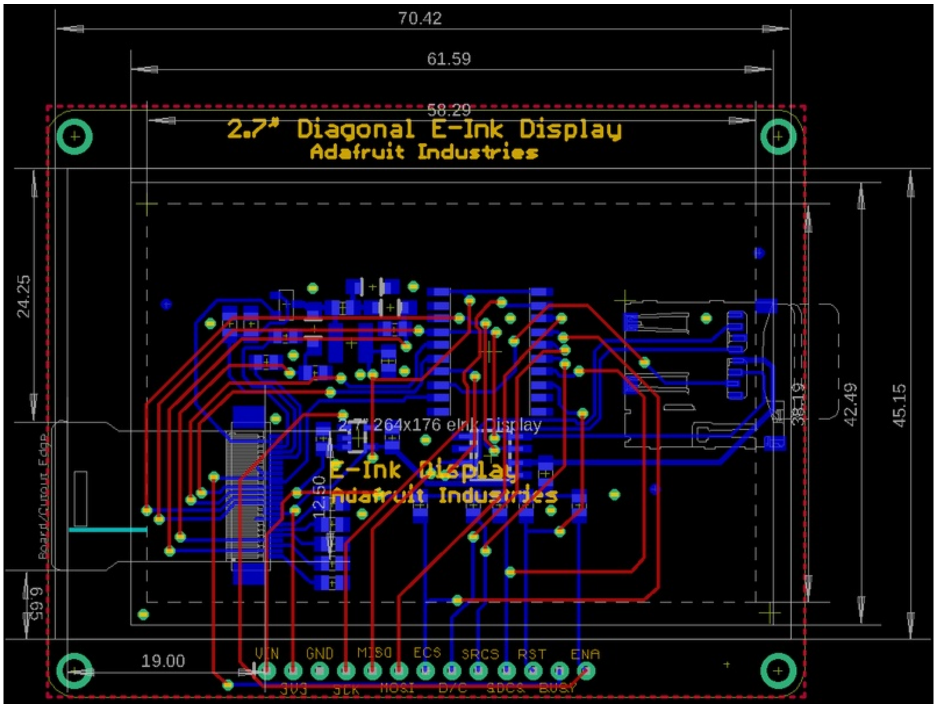


Schematic & Fabrication Prints




1.70
1.50
1.70
1.50
0.10
1.54" E-ink Display
w/SRAM & uSD
Vin/Vlogic:3-5V
adafruit
VIN
GND
MISO
ECS
SRCS
RST
ENA
3V3
SCK
MOSI
D/C
SDCS
BUSY


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


70.42
61.59
58.29
2.7" Diagonal E-Ink Display
Adafruit Industries
24.25
2.7" 264x176 eInk Display
E-Ink Display
Adafruit Industries
12.50
38.19
42.49
45.15
Board/Cutout Edge
6.65
19.00
VIN
GND
MISO
ECS
SRCS
RST
ENA
3V3
SCK
MOSI
D/C
SDCS
BUSY

Adafruit GFX Library

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

Python Docs

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