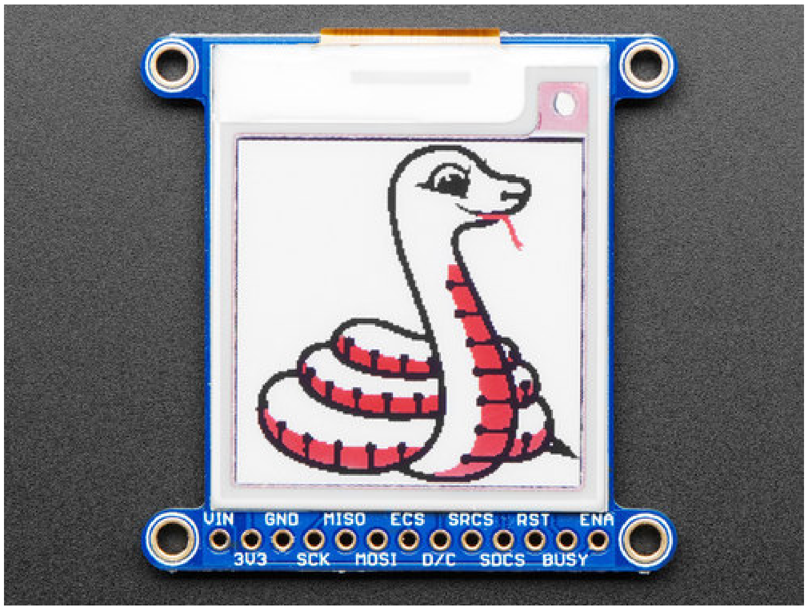
adafruit learning system

Adafruit eInk Display Breakouts

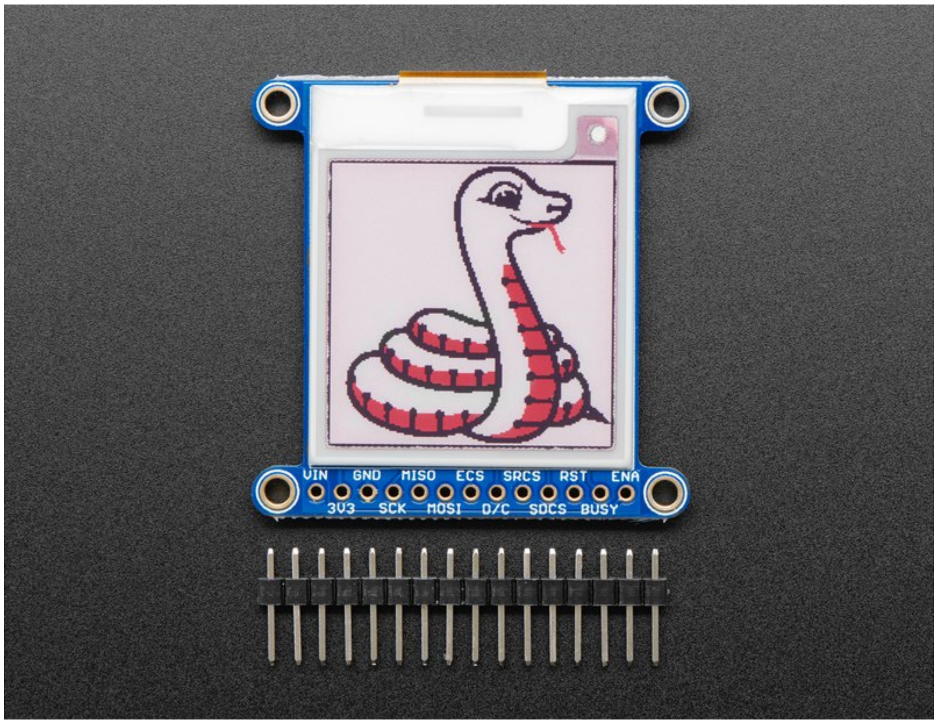
Created by lady ada



Last updated on 2019-02-06 10:03:43 PM UTC

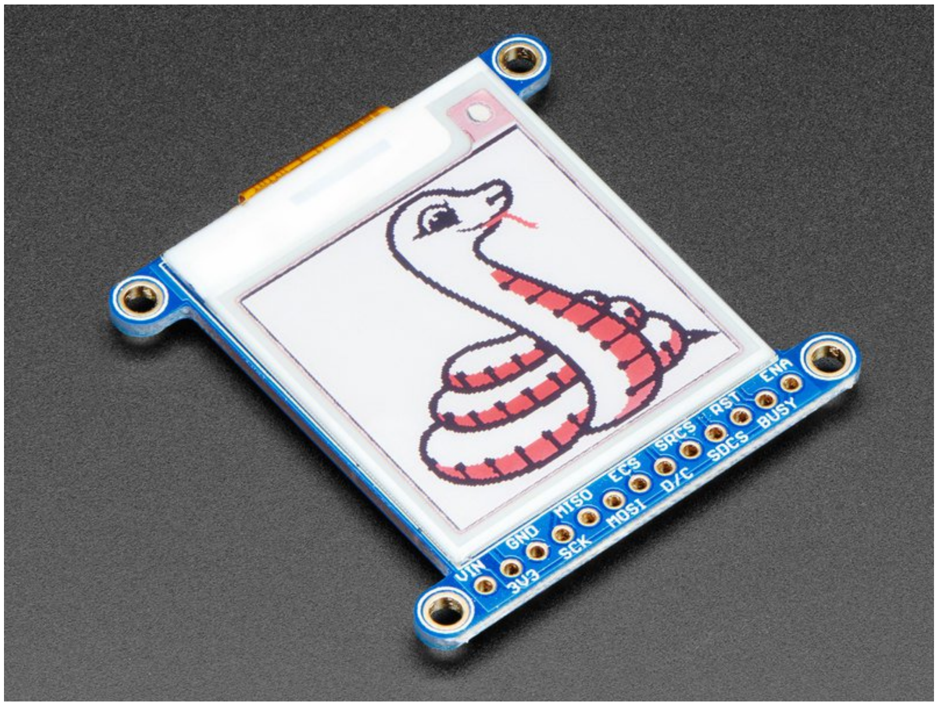
Guide Contents

Overview



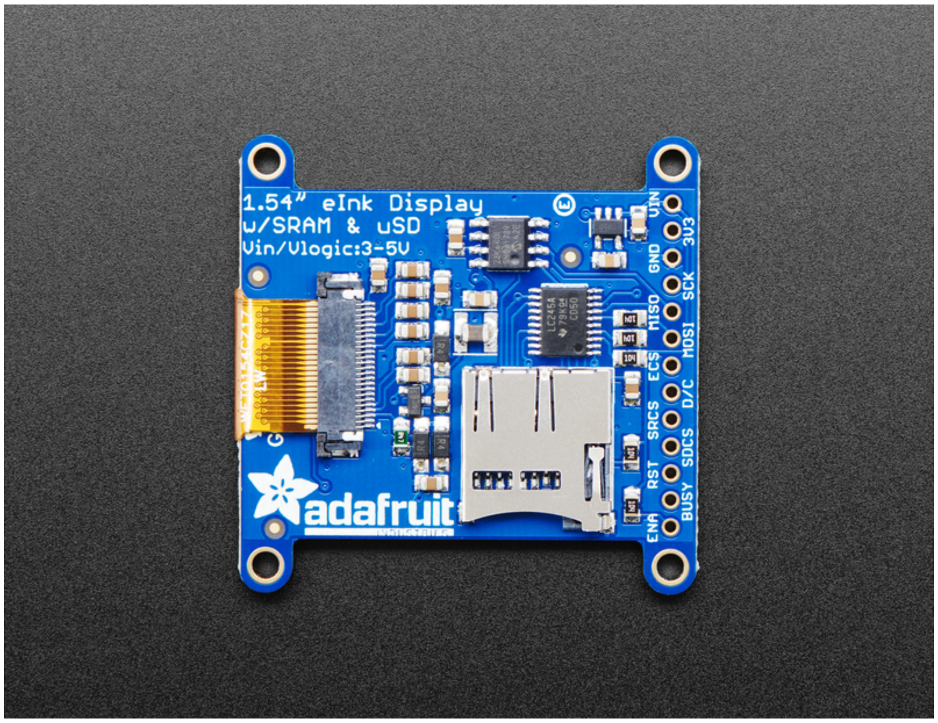
Easy e-paper finally comes to microcontrollers, with this breakout that's designed to make it a breeze to add a tri-color eInk display. Chances are you've seen one of those new-fangled 'e-readers' like the Kindle or Nook. They have gigantic electronic paper 'static' displays - that means the image stays 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 breakouts 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 the 1.54": 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?

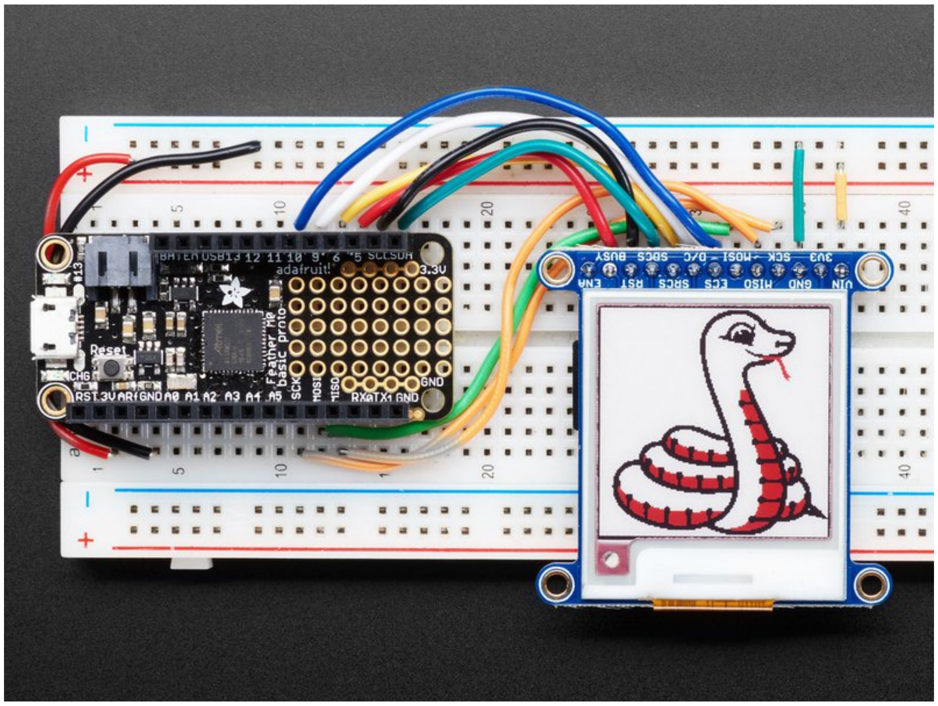
So we did you a favor and tossed a small SRAM chip on the back. This chip shares the SPI port the eInk display uses, so you only need one extra pin. And, no more frame-buffering! **You can use the SRAM to set up whatever you want to display, then shuffle data from SRAM to eInk when you're ready. The 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).



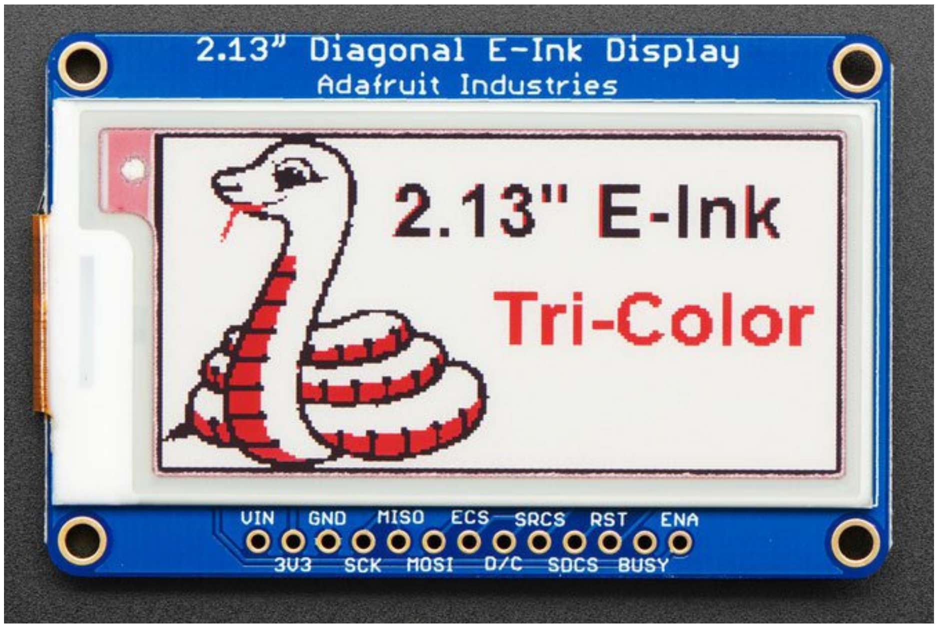
For ultra-low power usages, the onboard 3.3V regulator has the Enable pin brought out so you can shut down the power to the SRAM, MicroSD and display.

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

Comes assembled and tested, with some header. You'll need a soldering iron to attach the header for breadboarding or installing into your project.



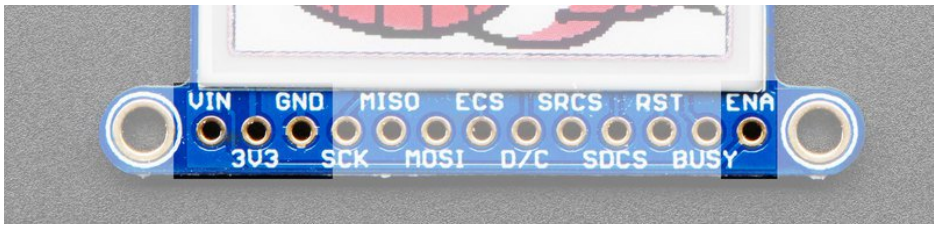
Pinouts


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

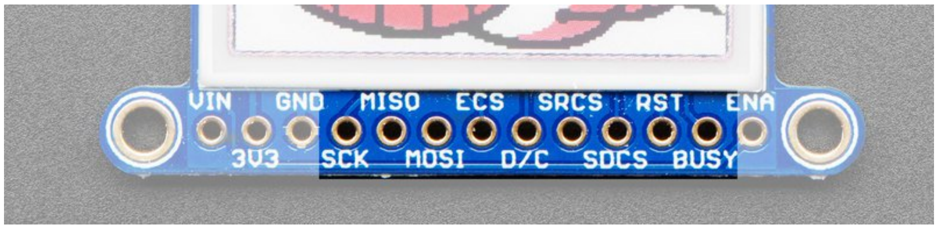
The pin outs are identical for the 1.54" and 2.13" E-Ink display!

Power Pins



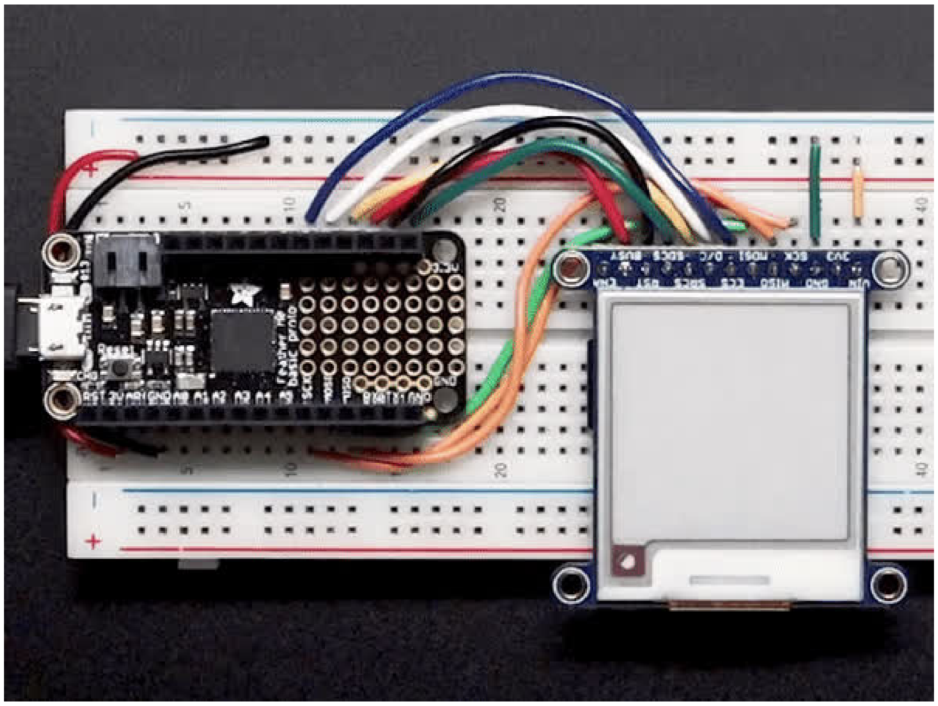
- **3-5V** / **Vin** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3.3V** out - this is the 3.3V output from the onboard regulator, you can 'borrow' about 100mA if you need to power some other 3.3V logic devices
- **GND** - this is the power 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)

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**

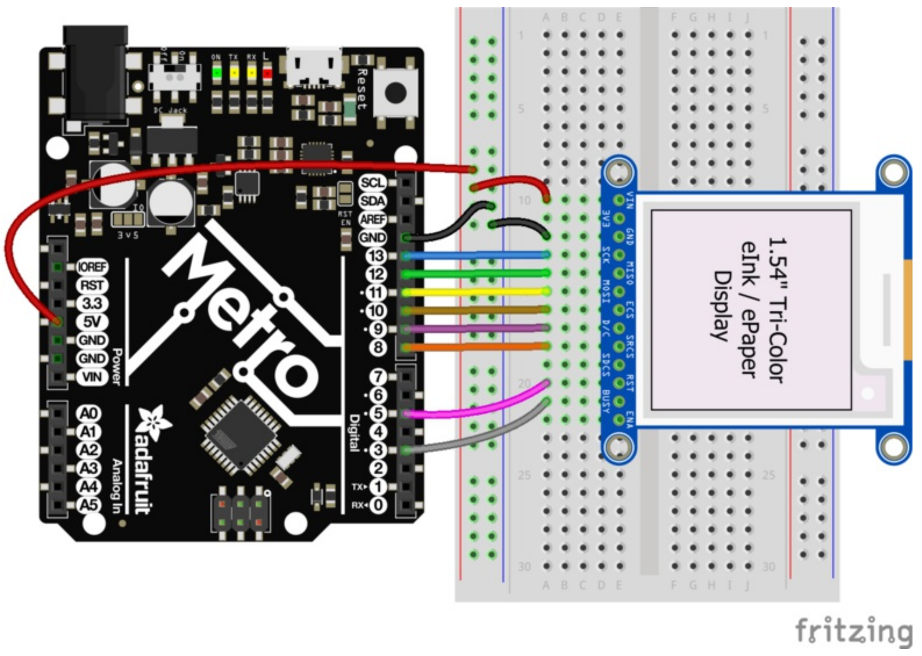
Do not update more than once every 180 seconds or you may permanently damage the display

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.



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

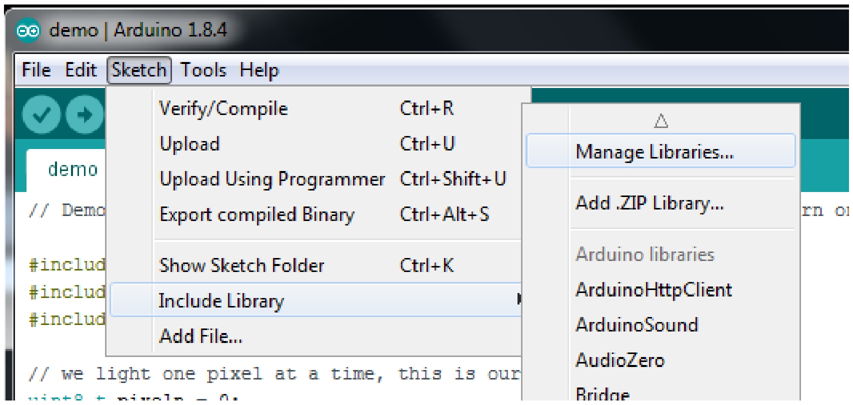
too.

- **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.

Install Adafruit_EPD & GFX libraries

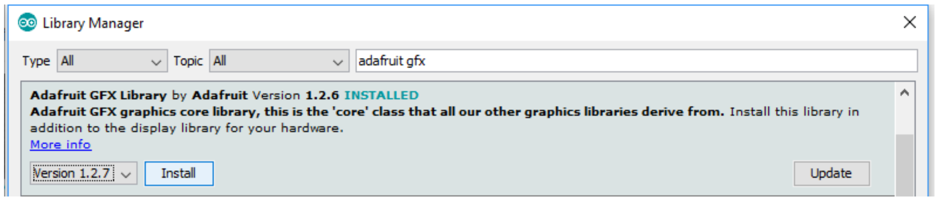
To begin reading sensor data, you will need to install the Adafruit_EPD library (code on our github repository) (https://adafru.it/BRK). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...



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

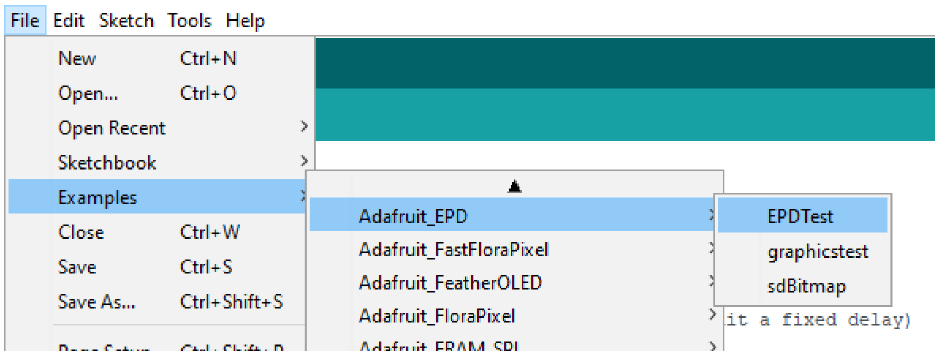
Do the same to install the latest **adafruit GFX** library, click **Install**

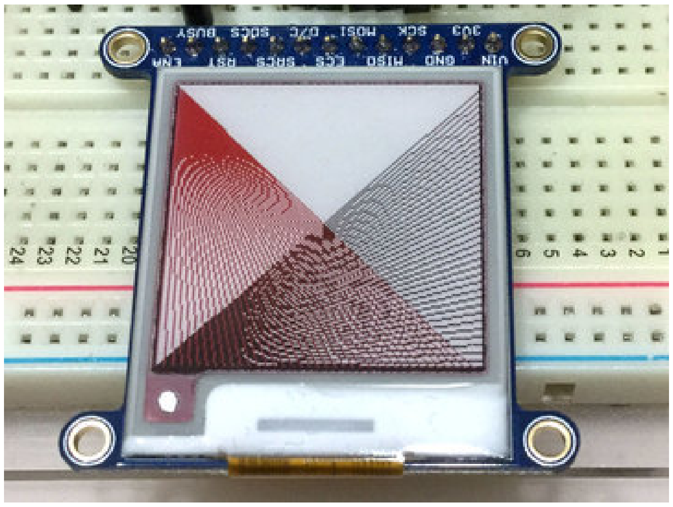


Adafruit GFX Library by **Adafruit** Version **1.2.6** INSTALLED
Adafruit GFX graphics core library, this is the 'core' class that all our other graphics libraries derive from. Install this library in addition to the display library for your hardware.
More info

Load First Demo

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



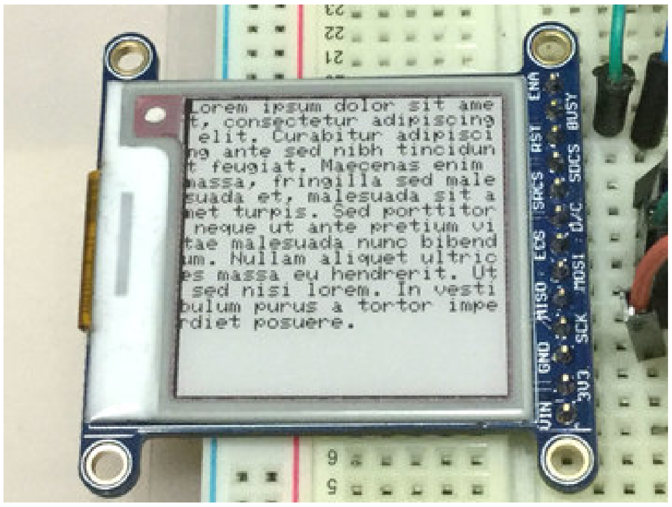


You will see the display flash a bunch and then a set of black and red lines will appear like shown on the left.

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


Lorem ipsum dolor sit ame
t, consectetur adipiscing
elit. Curabitur adipisci
ng ante sed nibh tincidun
t feugiat. Maecenas enim
massa, fringilla sed male
suada et, malesuada sit a
met turpis. Sed porttitor
neque ut ante pretium vi
tae malesuada nunc bibend
um. Nullam aliquet ultric
es massa eu hendrerit. Ut
sed nisi lorem. In vesti
bulum purus a tortor impe
rdiet posuere.
ENA
RST
SRCS
ECS
MISO
GND
VIN
BUSY
SDCS
D/C
MOSI
SCK
3V3

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!

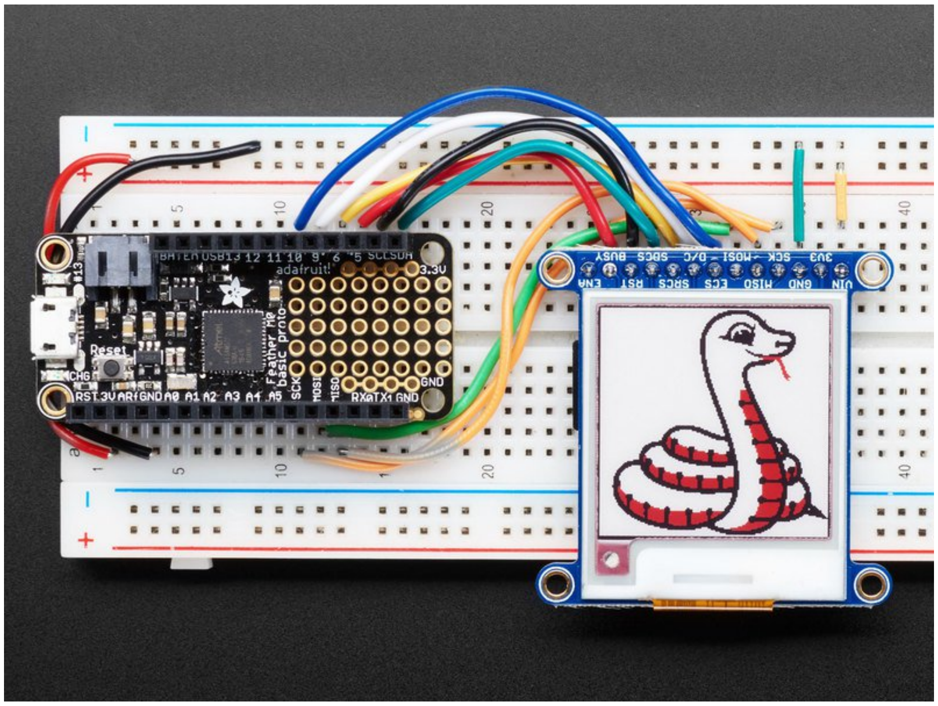
Adafruit GFX Library

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

Arduino Library Documentation

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

Drawing Bitmaps



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

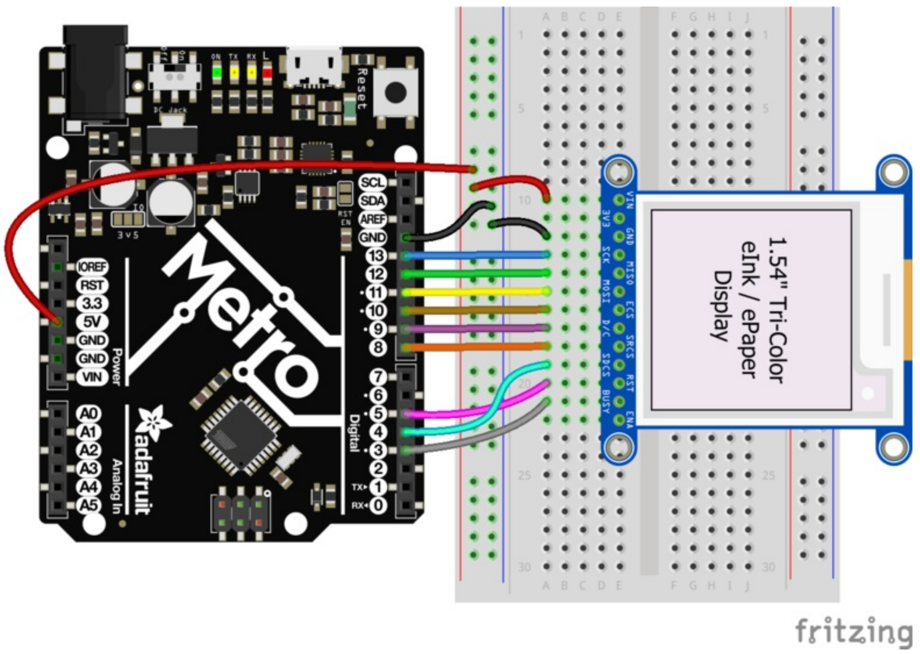
The 1.54" display can show a max of 152x152 pixels. Lets use this Blinka bitmap as our demo:

https://adafru.it/BTa

https://adafru.it/BTa

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.

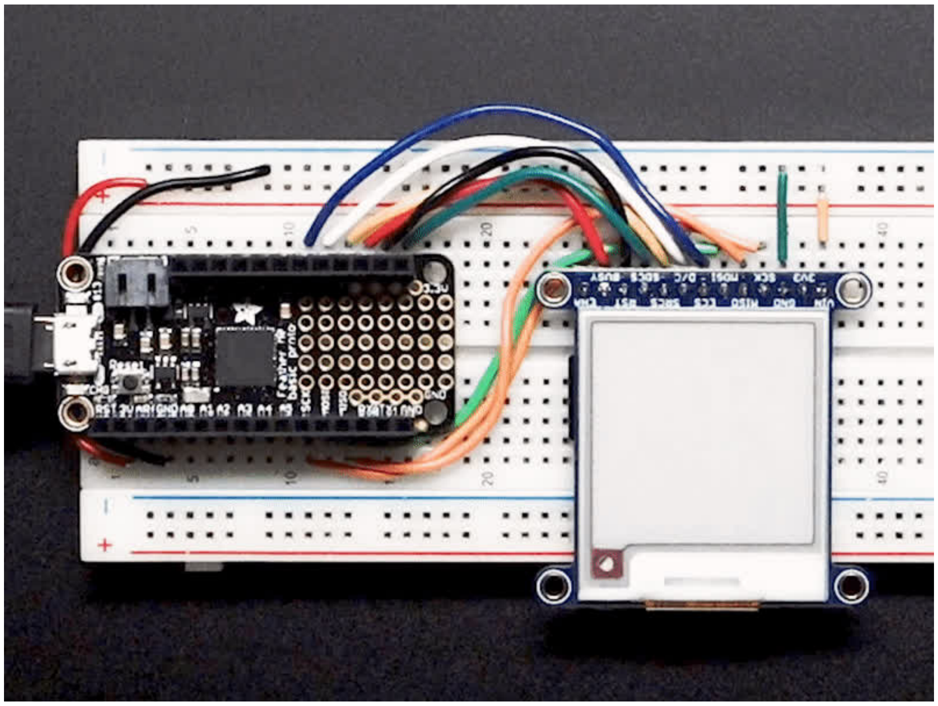
One extra wire is required, for **SDCS** which is the SD card Chip Select. We'll connect that to pin #4 but you can use any pin.



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!



CircuitPython Code

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

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.

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

- **adafruit_epd**

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

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.

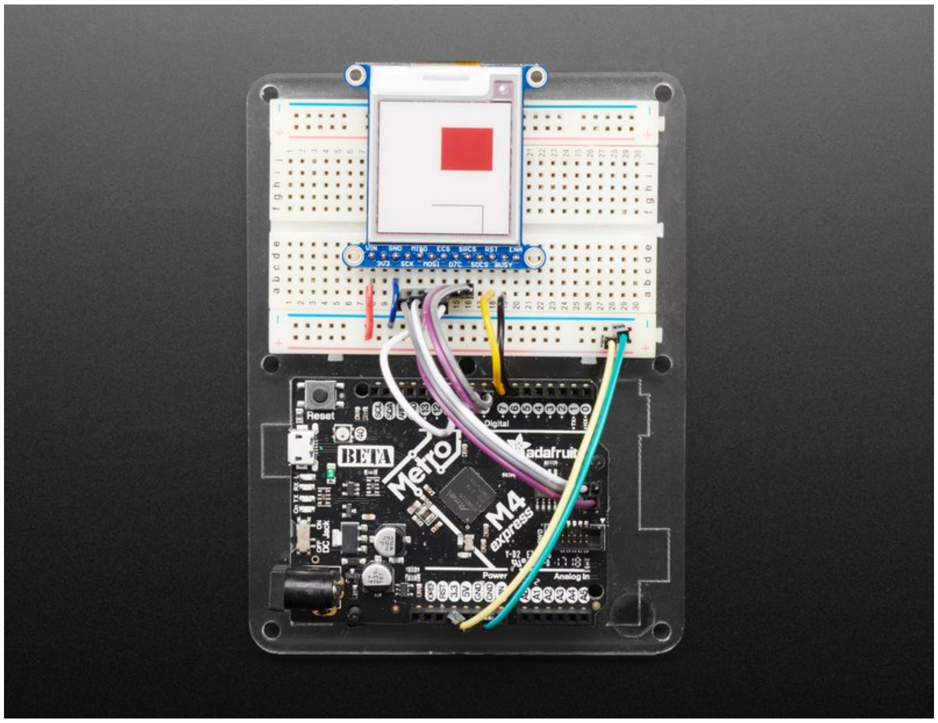
Run the following code to import the necessary modules and initialize the display:

```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373

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.D8)
rst = digitalio.DigitalInOut(board.D7)
busy = digitalio.DigitalInOut(board.D6)

display = Adafruit_IL0373(152, 152, rst, dc, busy, srcs, ecs, spi)
```

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.clear_buffer()

display.fill_rect(30, 20, 50, 60, Adafruit_EPD.RED)
display.hline(120, 30, 60, Adafruit_EPD.BLACK)
display.vline(120, 30, 60, Adafruit_EPD.BLACK)

display.display()
```

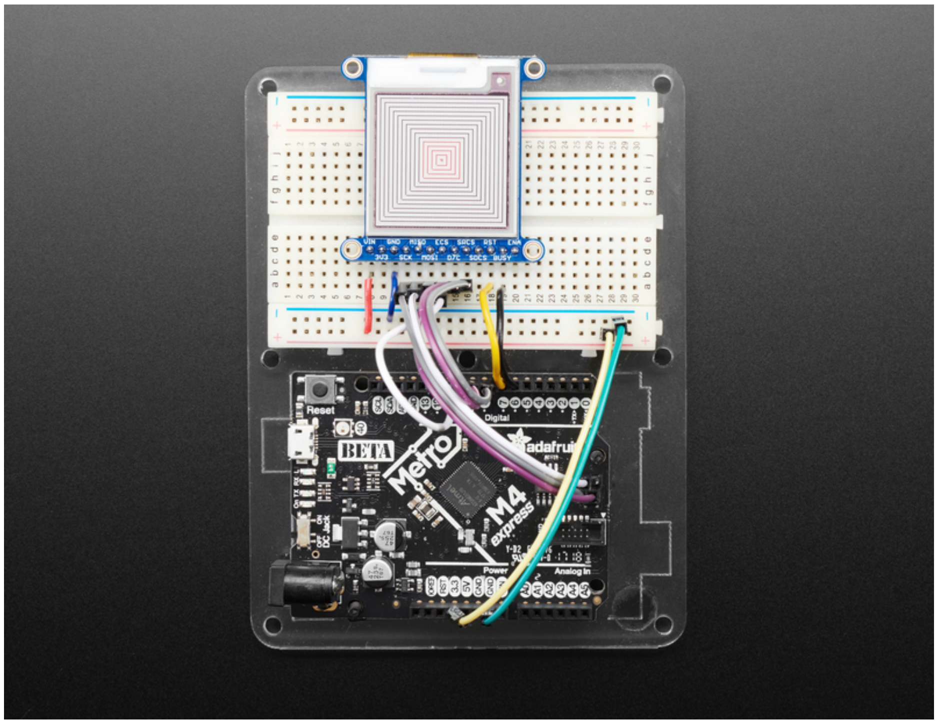
Your display will look like this:



Here's a complete example that shows drawing of a rectangle pattern. Save this as a **main.py** on your board (note it assumes a **hardware SPI connection** to the sensor):

```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373

# 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.D8)
rst = digitalio.DigitalInOut(board.D7)
busy = digitalio.DigitalInOut(board.D6)

# give them all to our driver
display = Adafruit_IL0373(152, 152, rst, dc, busy, srcs, ecs, spi)

# clear the buffer
display.clear_buffer()

r_width = 5
r_pos = display.height

color = Adafruit_EPD.BLACK
while r_pos > display.height/2:
    if r_pos < display.height - 50:
        color = Adafruit_EPD.RED
    display.rect(display.width - r_pos, display.height - r_pos,
                 display.width - 2*(display.width - r_pos),
                 display.height - 2*(display.height - r_pos), color)
    r_pos = r_pos - r_width

display.display()
```

Your display will look like this:



Here's another complete example of how to display a bitmap image on your display.Note that any .bmp image you want to display must be exactly 152 pixels by 152 pixels. We will be using the image below. Click the button below to download the image and save it as **blinka.bmp** on your CIRPY drive.

https://adafru.it/BTa

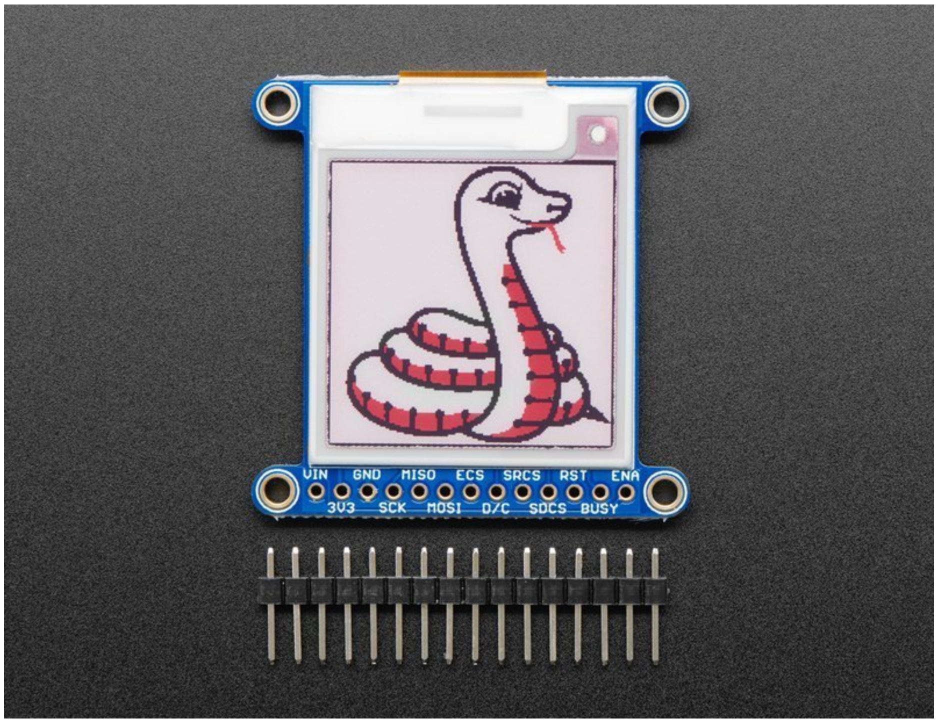
https://adafru.it/BTa

save the following code as **main.py** on your CIRPY drive:

```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373

# 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.D8)
rst = digitalio.DigitalInOut(board.D7)
busy = digitalio.DigitalInOut(board.D6)
```

```
# give them all to our driver
display = Adafruit_IL0373(152, 152, rst, dc, busy, srcs, ecs, spi)

FILENAME = "blinka.bmp"

# clear the buffer
display.clear_buffer()

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


try:
    with open("/" + FILENAME, "rb") as f:
        print("File opened")
        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)
            # print("seek to %d" % pos)
            f.seek(pos)
            for col in range(bmpWidth):
                b, g, r = bytearray(f.read(3))  # BMP files store RGB in BGR
                if r < 0x80 and g < 0x80 and b < 0x80:
                    display.draw_pixel(row, col, Adafruit_EPD.BLACK)
                elif r >= 0x80 and g >= 0x80 and b >= 0x80:
                    display.draw_pixel(row, col, Adafruit_EPD.WHITE)
                elif r >= 0x80:
                    display.draw_pixel(row, col, Adafruit_EPD.RED)

except OSError as e:
    if e.args[0] == 28:
        raise OSError("OS Error 28 0.25")
    else:
        raise OSError("OS Error 0.5")
except BMPError as e:
    print("Failed to parse BMP: " + e.args[0])


display.display()
```

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



Python Docs

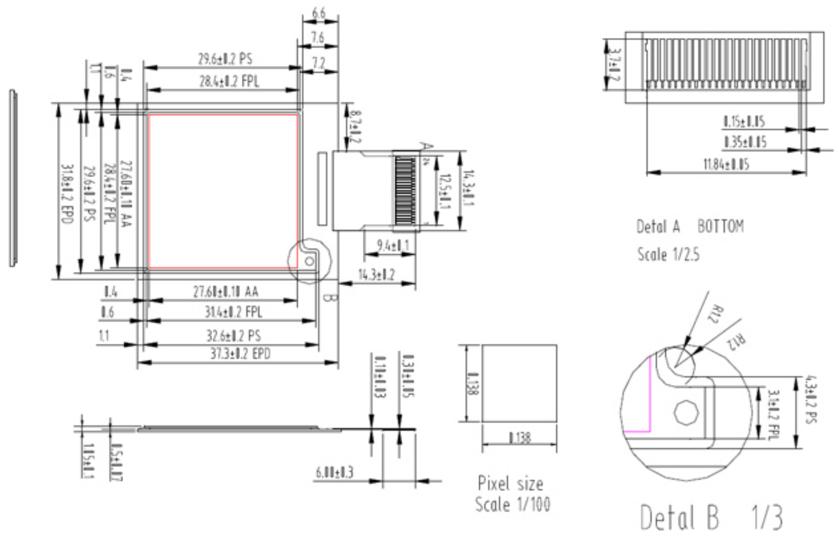
Python Docs (https://adafru.it/C4z)

Downloads

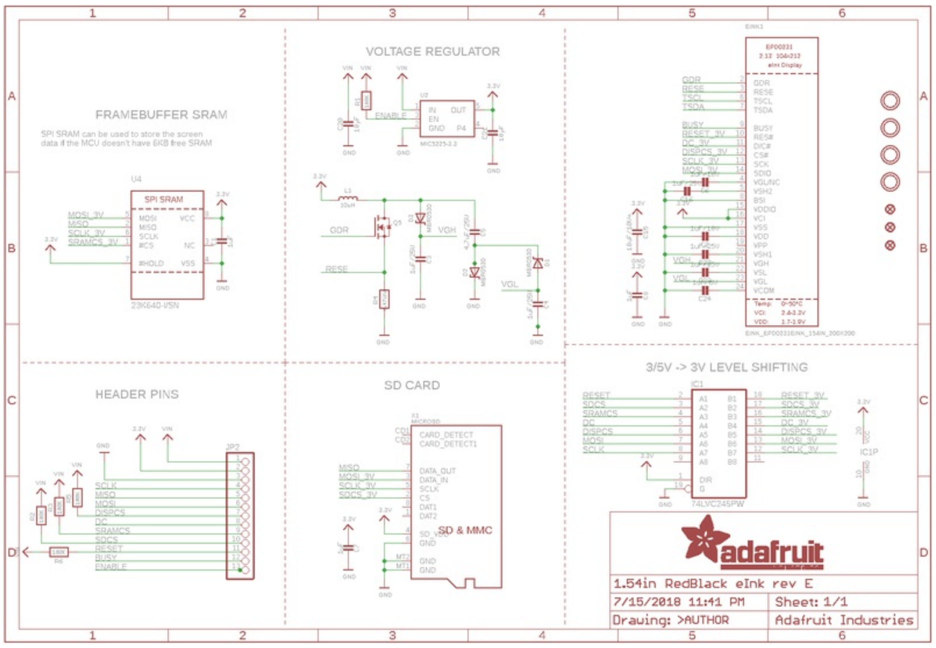
Files

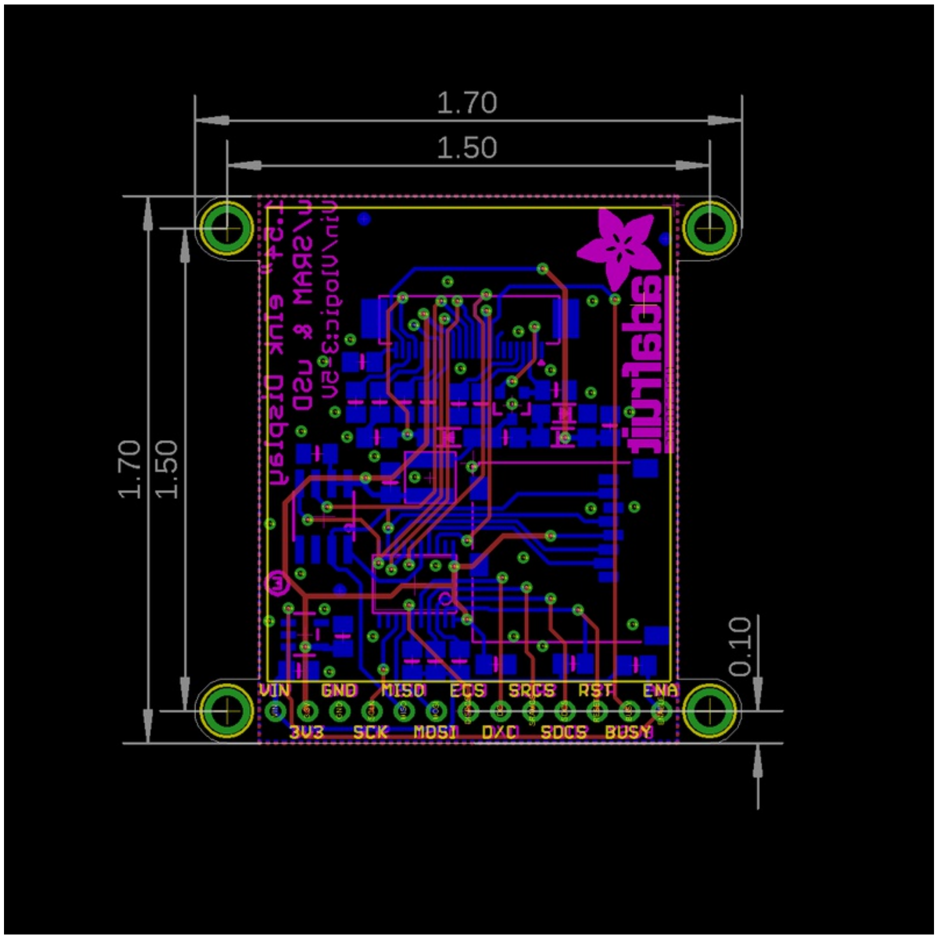
- Fritzing object in Adafruit Fritzing Library (https://adafru.it/aP3)
- IL0376F E-Ink interface chip datasheet (https://adafru.it/BRW)
- PCB Files on GitHub (https://adafru.it/BRX)

Display shape/outline:



Schematic & Fabrication Print




1.70
1.50
1.70
1.50
0.10
VIN
GND
MISO
ECS
SRCS
RST
ENA
3V3
SCK
MOSI
D/C
SDCS
BUSY