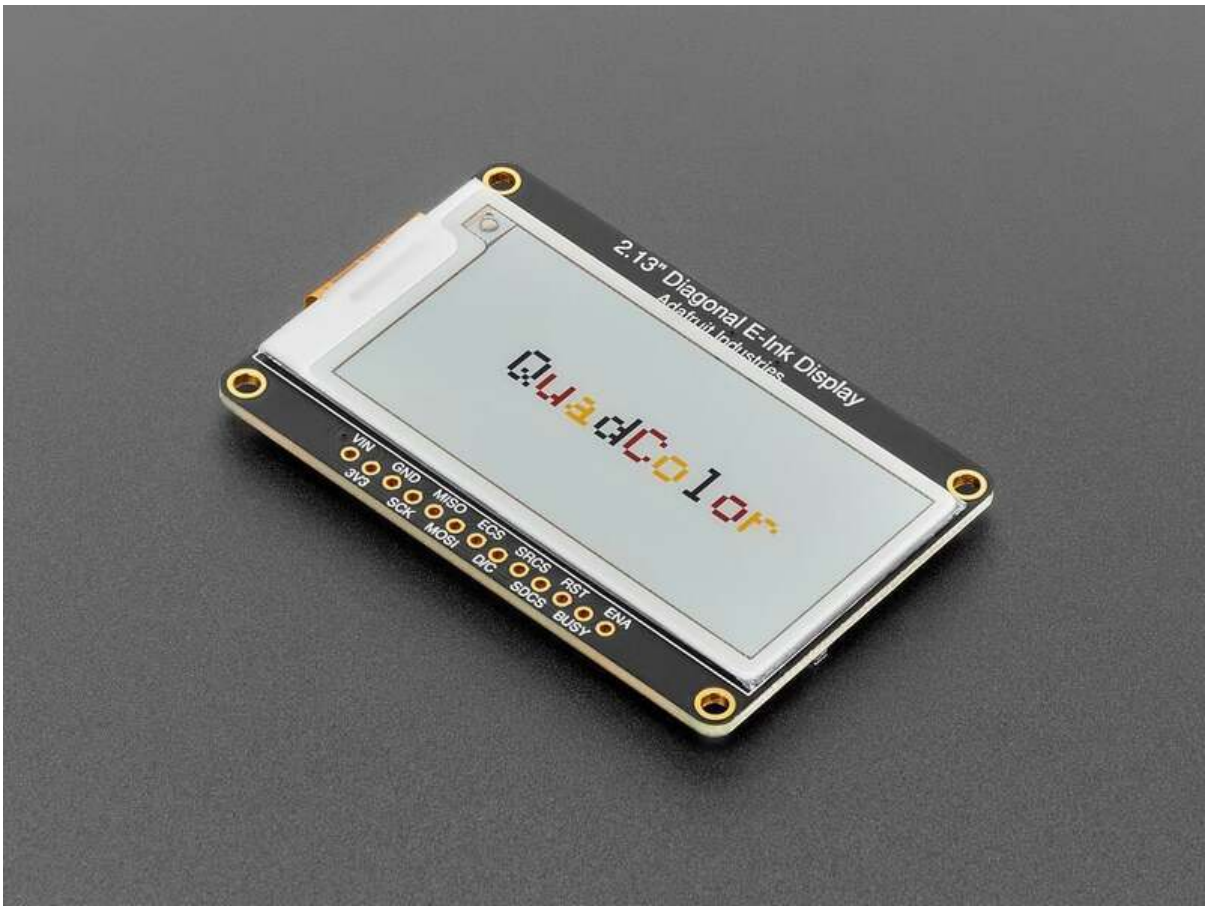
Adafruit 2.13" 250x122 Quad-Color eInk

Created by Liz Clark

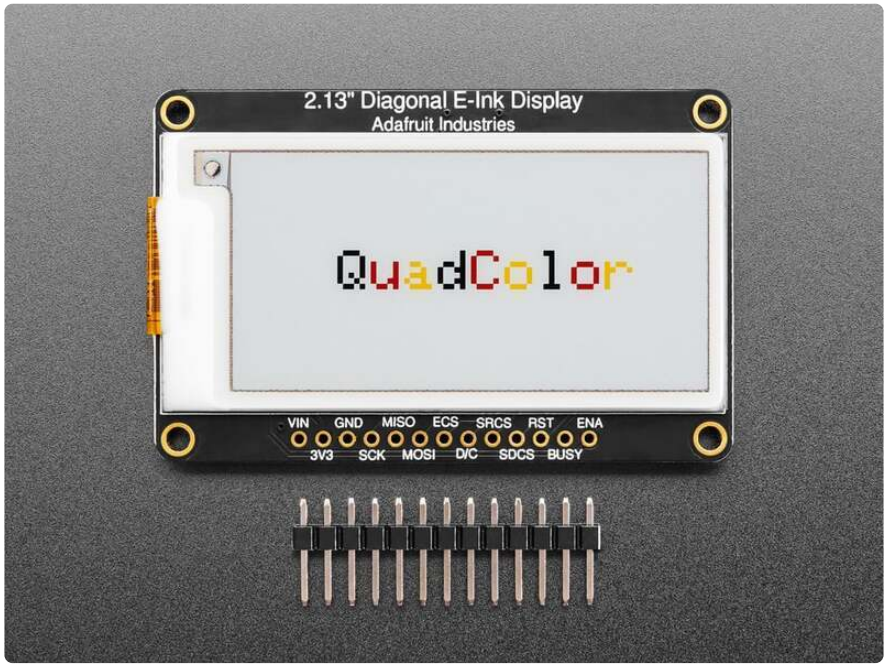


https://learn.adafruit.com/adafruit-2-13-250x122-quad-color-eink

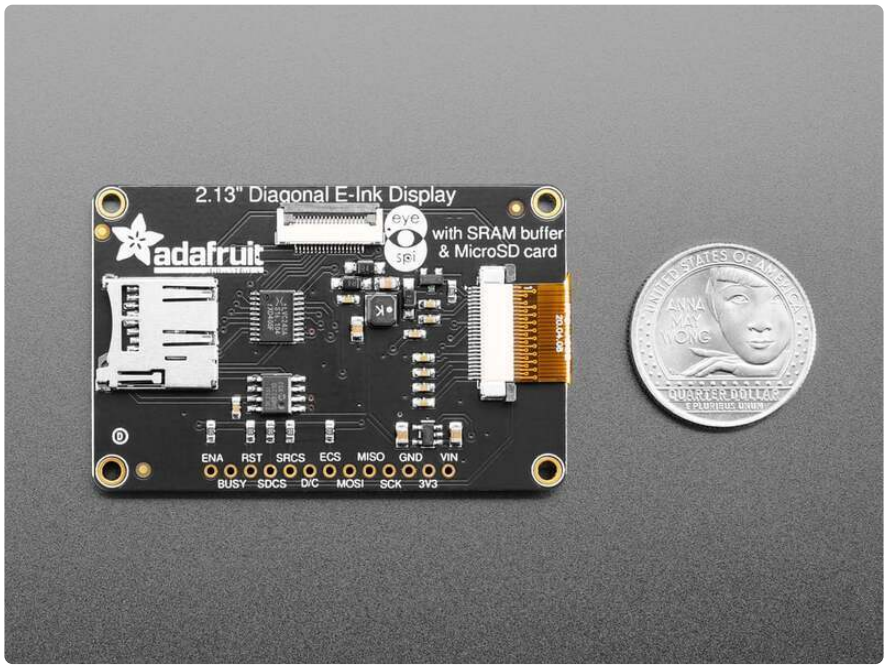
Last updated on 2025-11-11 10:20:29 PM EST

Table of Contents

Overview

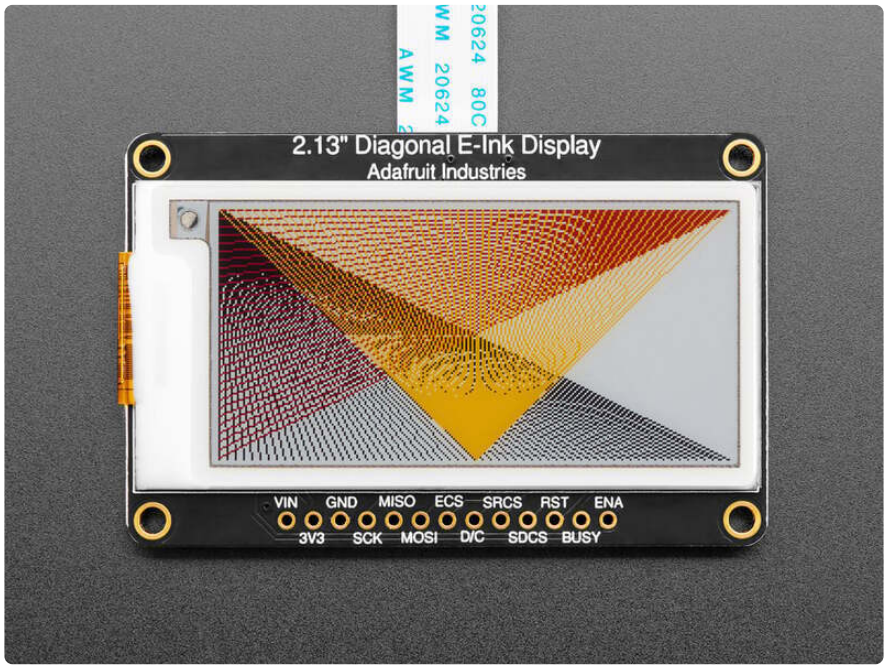


Easy e-paper finally comes to microcontrollers with this breakout that's designed to make it a breeze to add a quad-color eInk display. Chances are you've seen one of those new-fangled 'e-readers' like the Kindle or Nook. They have gigantic electronic paper 'static' displays - that means the image stays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

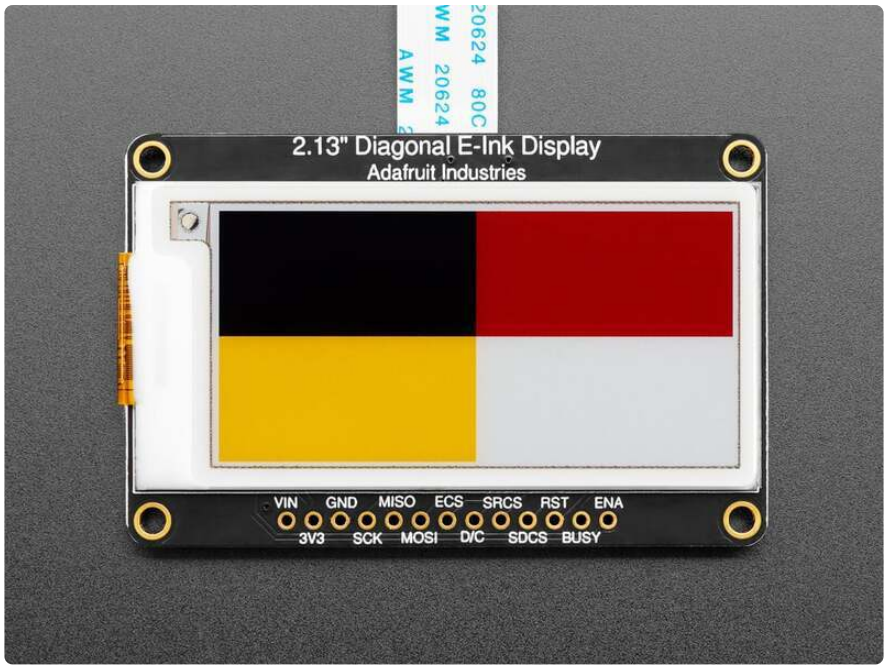


We've liked these displays for a long time, but breakouts were never designed for makers to use. Finally, we decided to make our own!

This is a **2.13" quad-color (red, black, yellow and white) display**. It has 250x122 black, red and yellow ink pixels and a white-ish background. It uses the JD79661 chipset, so make sure whatever firmware code you are planning to use has support for it. Using our CircuitPython or Arduino libraries, you can create a 'frame buffer' with what pixels you want to have activated and then write that out to the display. Most simple breakouts leave it at that. But if you do the math, 250 x 122 pixels x 2 bits = 7.5 KBytes. Which won't fit into many microcontroller memories. Heck, even if you do have 32KB of RAM, why waste 8KB?

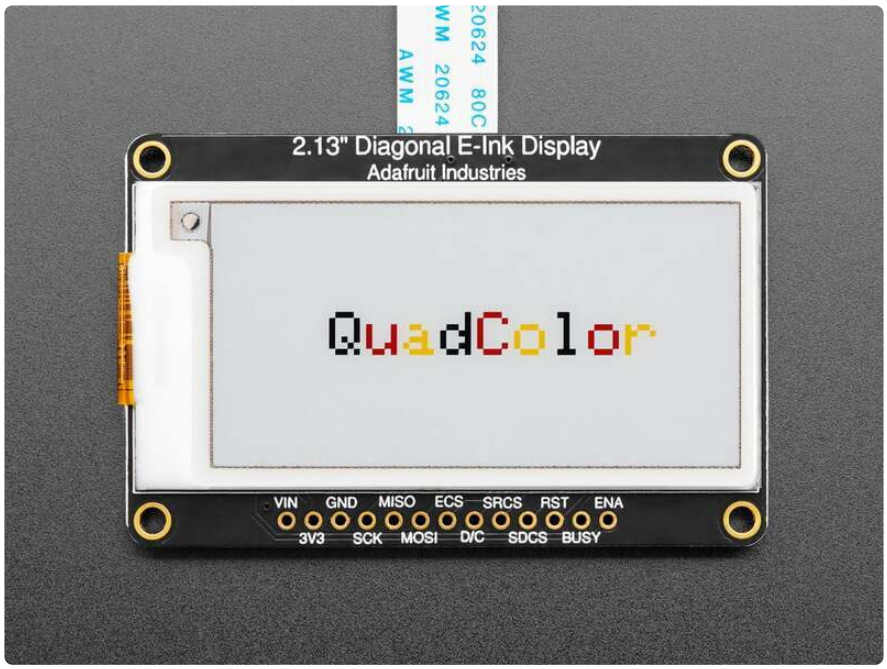


So we did you a favor and tossed a small SRAM chip on the back. This chip shares the SPI port the eInk display uses, so you only need one extra pin. And, no more frame-buffering! **You can use the SRAM to set up whatever you want to display, then shuffle data from SRAM to eInk when you're ready.** [The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



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

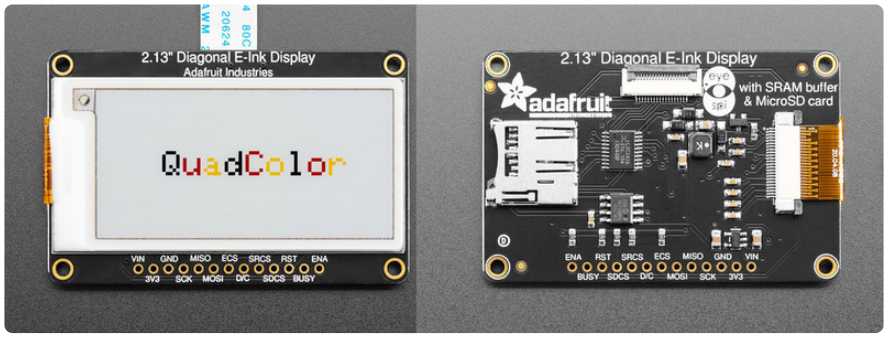
For ultra-low power usages, the onboard 3.3V regulator has the Enable pin brought out so you can shut down the power to the SRAM, MicroSD, and display.



This display breakout also features an 18-pin "EYE SPI" standard FPC connector with flip-top connector. [You can use an 18-pin 0.5mm pitch FPC cable](http://adafru.it/5239) (http://adafru.it/5239)

to connect to all the GPIO pins, for when you want to skip the soldering. Comes assembled and tested, with some header. You'll need a soldering iron to attach the header for breadboarding or installing it into your project.

Pinouts



Power Pins

- **VIN** - this is the power pin, connect to 3-5VDC - it has reverse polarity protection but try to wire it right!
- **3V3** 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** - 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

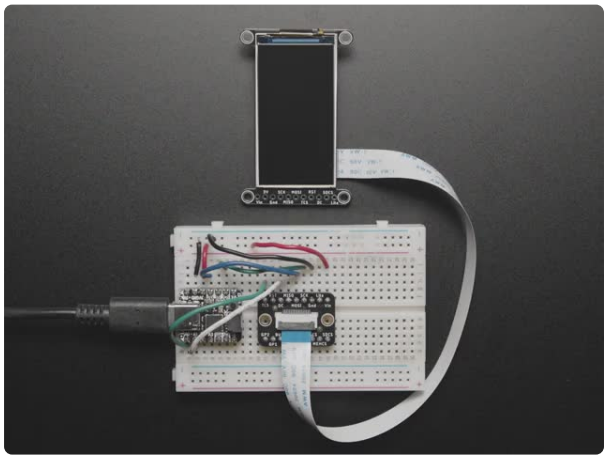
Data Control Pins

- **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)

EYESPI Display Connector

This connector, located on the back of the board, allows you to connect the display using an EYESPI cable to an EYESPI breakout with no soldering or jumper wires needed.



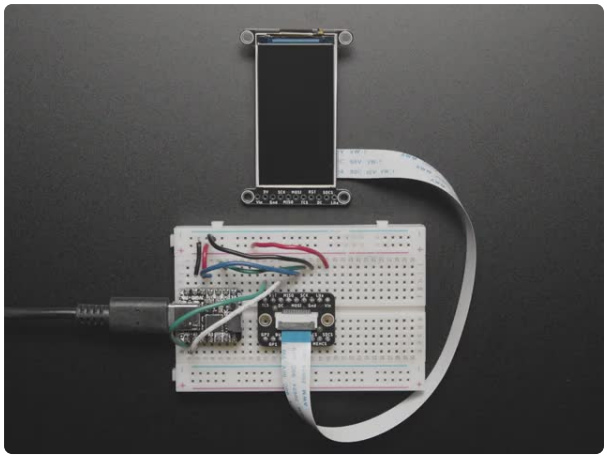
Adafruit EYESPI Breakout Board - 18 Pin FPC Connector
Our most recent display breakouts have come with a new feature: an 18-pin "EYE SPI" standard FPC...
https://www.adafruit.com/product/5613

micro SD Card Slot

On the back of the board, on the left side, is the micro SD card slot. You can use any micro SD card that supports SPI mode with one CS pin.

Arduino

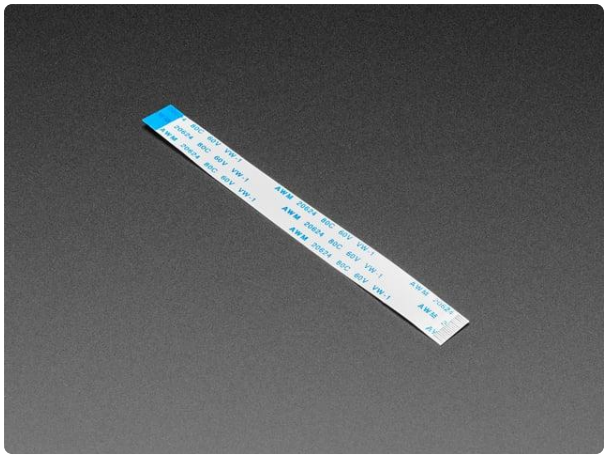
Using the **Adafruit 2.13" 250x122 Quad-Color eInk** with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller, installing the Adafruit_EPD (https://adafru.it/BRK) library, and running the provided example code.



Adafruit EYESPI Breakout Board - 18 Pin FPC Connector

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

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



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

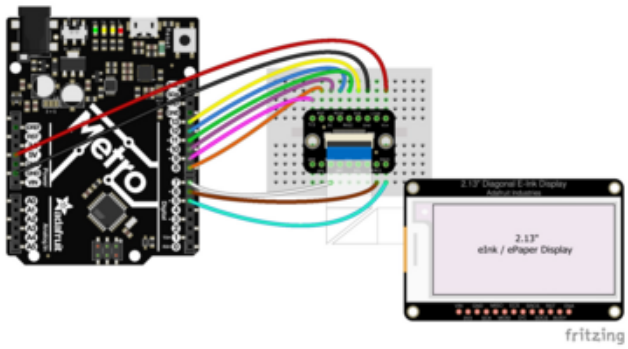
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...

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

Wiring

Wire as shown for a **5V** board like an Uno. If you are using a **3V** board, like an Adafruit Feather, wire the board's 3V pin to the breakout VIN.

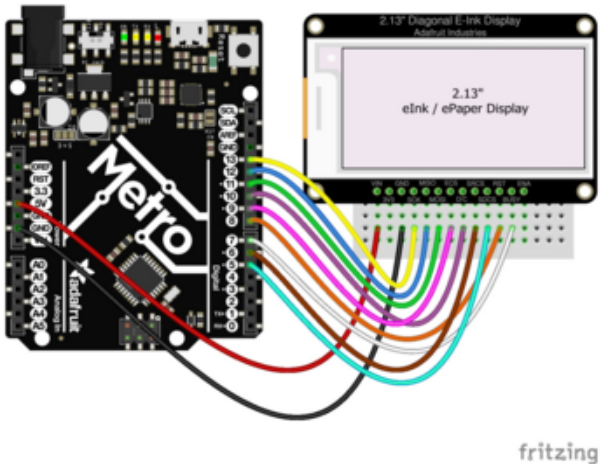
Here is an Adafruit Metro wired up to the display using the EYESPI breakout:



Metro 5V to **breakout Vin (red wire)**
Metro GND to **breakout Gnd (black wire)**
Metro SCK/D13 to **breakout SCK (yellow wire)**
Metro MISO/D12 to **breakout MISO (blue wire)**
Metro MOSI/D11 to **breakout MOSI (green wire)**
Metro D5 to **breakout SDCS (cyan wire)**
Metro D6 to breakout MEMCS (brown wire)
Metro D7 to breakout BUSY (white wire)
Metro D8 to **breakout RST (orange wire)**
Metro D9 to **breakout TCS (pink wire)**
Metro D10 to **breakout DC (purple wire)**

Attach the **eInk display to the EYESPI breakout** with an **EYESPI cable** as described on the [Plugging in an EYESPI Cable (https://adafru.it/18eU)](https://adafru.it/18eU) page.

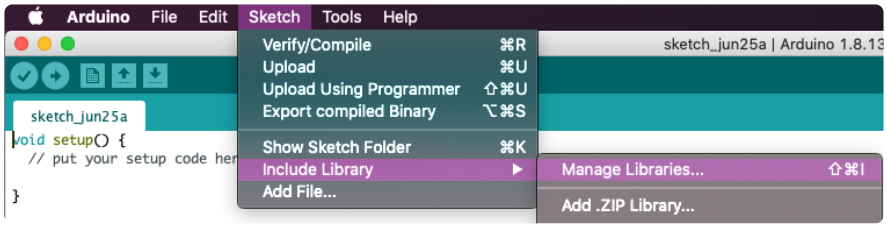
Here is an Adafruit Metro wired up using a solderless breadboard:



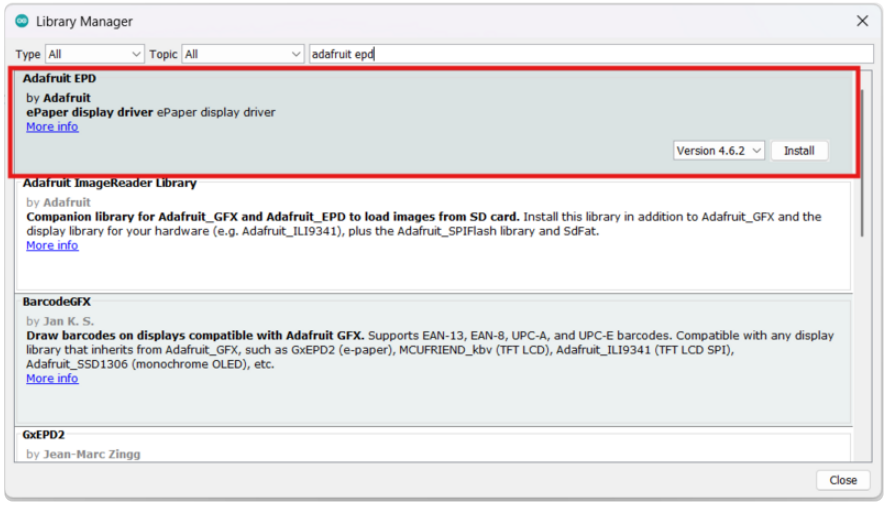
Metro 5V to **breakout Vin (red wire)**
Metro GND to **breakout Gnd (black wire)**
Metro SCK/D13 to **breakout SCK (yellow wire)**
Metro MISO/D12 to **breakout MISO (blue wire)**
Metro MOSI/D11 to **breakout MOSI (green wire)**
Metro D5 to **breakout SDCS (cyan wire)**
Metro D6 to **breakout SRCS (brown wire)**
Metro D7 to **breakout BUSY (white wire)**
Metro D8 to **breakout RST (orange wire)**
Metro D9 to **breakout ECS (pink wire)**
Metro D10 to **breakout D/C (purple wire)**

Library Installation

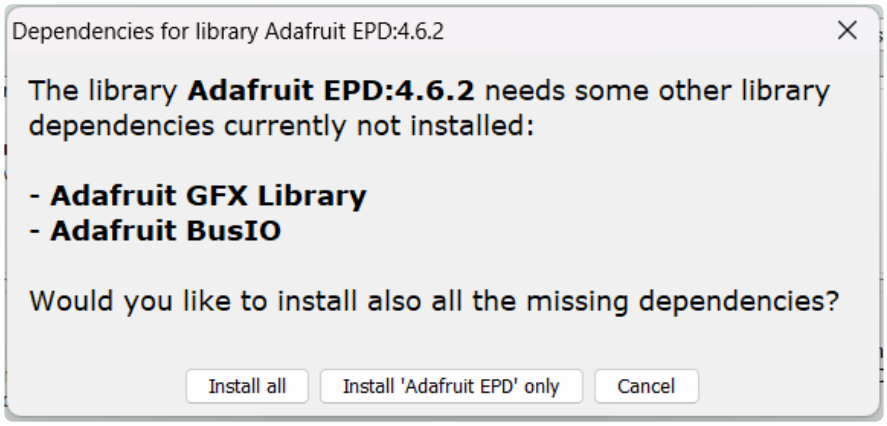
You can install the **Adafruit_EPD** library for Arduino using the Library Manager in the Arduino IDE.



Click the **Manage Libraries ...** menu item, search for **Adafruit_EPD**, and select the **Adafruit EPD** library:



If asked about dependencies, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

> e dependencies are already installed, you must make sure you update
> ı through the Arduino Library Manager before loading the example!

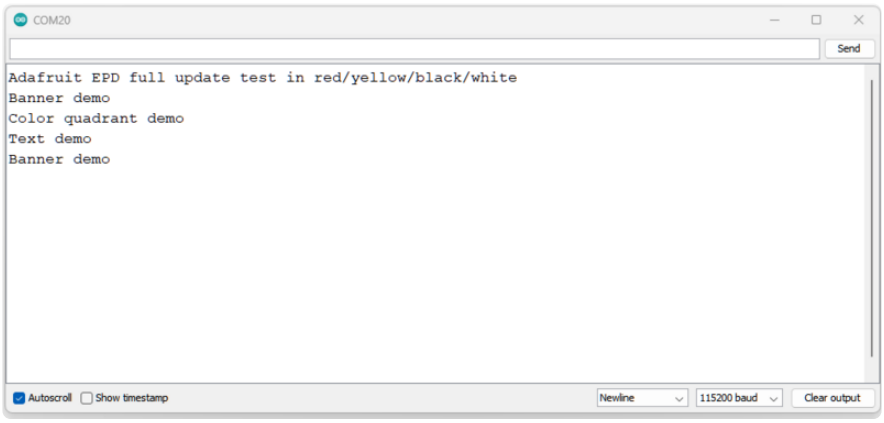
Example Code

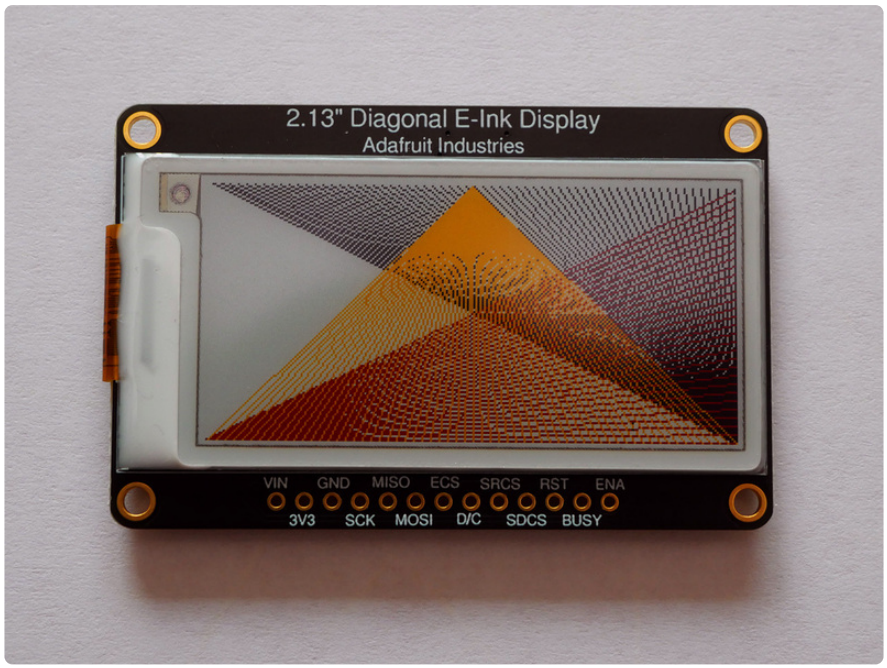
```
// SPDX-FileCopyrightText: 2025 Liz Clark for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/***************************************************
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/

#include "Adafruit_ThinkInk.h"

#ifdef ARDUINO_ADAFRUIT_FEATHER_RP2040_THINKINK // detects if compiling for
                                                // Feather RP2040 ThinkInk
#define EPD_DC PIN_EPD_DC       // ThinkInk 24-pin connector DC
#define EPD_CS PIN_EPD_CS       // ThinkInk 24-pin connector CS
#define EPD_BUSY PIN_EPD_BUSY   // ThinkInk 24-pin connector Busy
#define SRAM_CS -1              // use onboard RAM
#define EPD_RESET PIN_EPD_RESET // ThinkInk 24-pin connector Reset
#define EPD_SPI &SPI1           // secondary SPI for ThinkInk
#else
#define EPD_DC 10
#define EPD_CS 9
#define EPD_BUSY 7 // can set to -1 to not use a pin (will wait a fixed delay)
#define SRAM_CS 6
#define EPD_RESET 8  // can set to -1 and share with microcontroller Reset!
#define EPD_SPI &SPI // primary SPI
#endif

// 2.13" Quadcolor EPD with JD79661 chipset
ThinkInk_213_Quadcolor_AJHE5 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY,
                                     EPD_SPI);

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

void loop() {
  Serial.println("Banner demo");
  display.clearBuffer();
```

```
  display.setTextSize(3);
  display.setCursor((display.width() - 144) / 2, (display.height() - 24) / 2);
  String text = "QuadColor";
  uint16_t colors[] = {EPD_BLACK, EPD_RED, EPD_YELLOW};

  for (int i = 0; i < text.length(); i++) {
    // Change color for every character (0: BLACK, 1: RED, 2: YELLOW, 3: BLACK,
etc.)
    display.setTextColor(colors[i % 3]);
    display.print(text.charAt(i));
  }
  display.display();

  delay(15000);

  Serial.println("Color quadrant demo");
  display.clearBuffer();
  // Top-left quadrant - EPD_BLACK
  display.fillRect(0, 0, display.width() / 2, display.height() / 2, EPD_BLACK);
  // Top-right quadrant - EPD_RED
  display.fillRect(display.width() / 2, 0, display.width() / 2, display.height() /
2, EPD_RED);
  // Bottom-left quadrant - EPD_YELLOW
  display.fillRect(0, display.height() / 2, display.width() / 2, display.height() /
2, EPD_YELLOW);
  // Bottom-right quadrant - assume you have a 4th color like EPD_WHITE or another
color
  display.fillRect(display.width() / 2, display.height() / 2, display.width() / 2,
display.height() / 2, EPD_WHITE);

  display.display();

  delay(15000);

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

  delay(15000);

  display.clearBuffer();
  for (int16_t i = 0; i < display.width(); i += 4) {
    display.drawLine(0, 0, i, display.height() - 1, EPD_BLACK);
  }
  for (int16_t i = 0; i < display.height(); i += 4) {
    display.drawLine(display.width() - 1, 0, 0, i, EPD_RED);
  }
  for (int16_t i = 0; i < display.width(); i += 4) {
     display.drawLine(display.width()/2, display.height()-1, i, 0,
                      EPD_YELLOW);
  }

  display.display();

  delay(15000);
}

void testdrawtext(const char *text, uint16_t color) {
  display.setCursor(0, 0);
```

```
  display.setTextColor(color);
  display.setTextWrap(true);
  display.print(text);
}
```

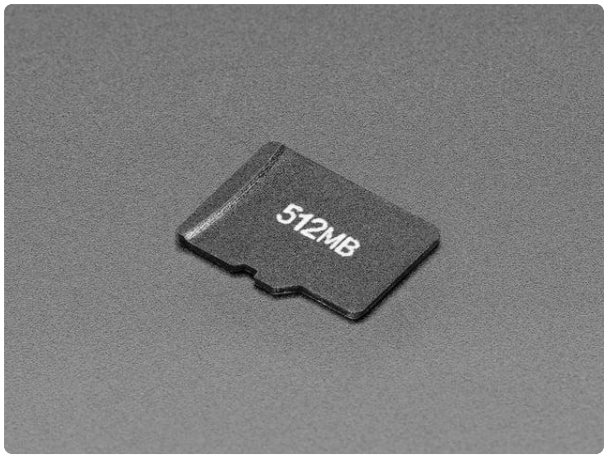
Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. You'll see the display recognized over SPI. Then, as different demos are sent to the display, you'll see the demo name print to the Serial Monitor.





Displaying Images

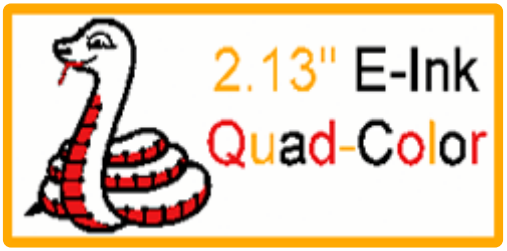
You can also load images from a microSD card to show on the display using the Adafruit_ImageReader (https://adafru.it/1aoJ) library.



512MB micro SD Memory Card
Add storage in a jiffy using this 512MB microSD card. Preformatted to FAT32, so it works out of the packaging with our projects. Works great with any device in the...
https://www.adafruit.com/product/5252

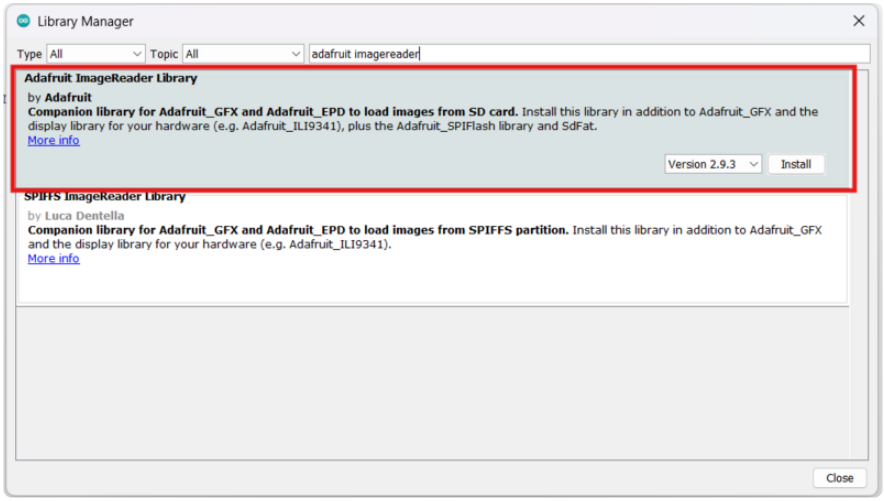
Image File

Download the **.BMP** file below and place it into the base directory of a microSD card and insert it into the microSD socket on the back of the display.

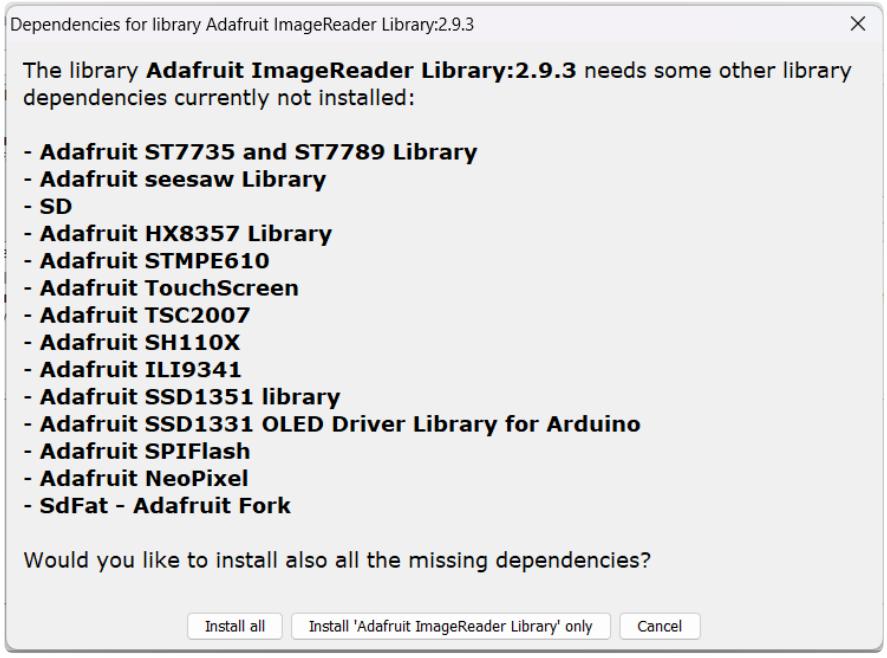


Library Installation

Click the **Manage Libraries ...** menu item, search for **Adafruit_ImageReader**, and select the **Adafruit ImageReader** library:



If asked about dependencies, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

e dependencies are already installed, you must make sure you update
n through the Arduino Library Manager before loading the example!

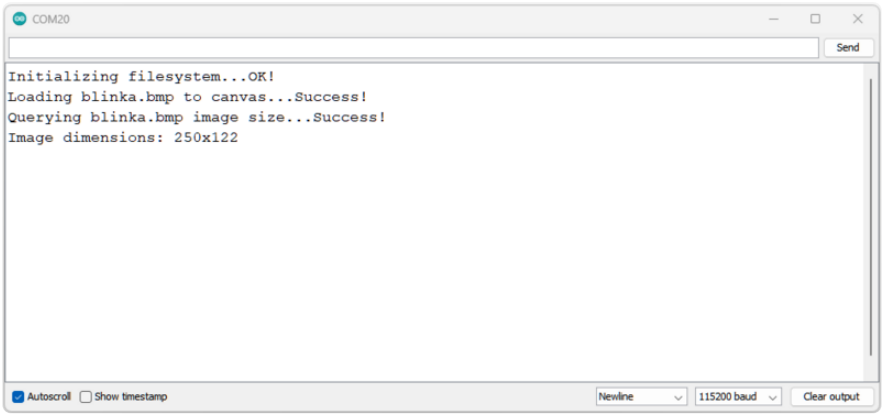
ImageReader Example

```
// SPDX-FileCopyrightText: 2025 Liz Clark for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Adafruit_ImageReader test for Adafruit E-Ink Breakouts.
// Demonstrates loading images from SD card or flash memory to the screen,
// to RAM, and how to query image file dimensions.
// Requires BMP file in root directory of QSPI Flash:
// blinka.bmp.

#include <Adafruit_GFX.h>         // Core graphics library
#include "Adafruit_ThinkInk.h"
#include <SdFat_Adafruit_Fork.h>  // SD card & FAT filesystem library
#include <Adafruit_SPIFlash.h>    // SPI / QSPI flash library
#include <Adafruit_ImageReader_EPD.h> // Image-reading functions

// Comment out the next line to load from SPI/QSPI flash instead of SD card:
```

```
#define USE_SD_CARD

#define EPD_DC 10
#define EPD_CS 9
#define EPD_BUSY 7 // can set to -1 to not use a pin (will wait a fixed delay)
#define SRAM_CS 6
#define EPD_RESET 8  // can set to -1 and share with microcontroller Reset!
#define EPD_SPI &SPI // primary SPI
#define SD_CS       5  // SD card chip select

// 2.13" Quadcolor EPD with JD79661 chipset
ThinkInk_213_Quadcolor_AJHE5 display(EPD_DC, EPD_RESET, EPD_CS, SRAM_CS, EPD_BUSY,
                                     EPD_SPI);

#if defined(USE_SD_CARD)
  SdFat                    SD;         // SD card filesystem
  Adafruit_ImageReader_EPD reader(SD); // Image-reader object, pass in SD filesys
#else

// SPI or QSPI flash filesystem (i.e. CIRCUITPY drive)
  #if defined(__SAMD51__) || defined(NRF52840_XXAA)
    Adafruit_FlashTransport_QSPI flashTransport(PIN_QSPI_SCK, PIN_QSPI_CS,
      PIN_QSPI_IO0, PIN_QSPI_IO1, PIN_QSPI_IO2, PIN_QSPI_IO3);
  #else
    #if (SPI_INTERFACES_COUNT == 1 || defined(ADAFRUIT_CIRCUITPLAYGROUND_M0))
      Adafruit_FlashTransport_SPI flashTransport(SS, &SPI);
    #else
      Adafruit_FlashTransport_SPI flashTransport(SS1, &SPI1);
    #endif
  #endif
  Adafruit_SPIFlash         flash(&flashTransport);
  FatVolume             filesys;
  Adafruit_ImageReader_EPD  reader(filesys); // Image-reader, pass in flash filesys
#endif

Adafruit_Image_EPD   img;        // An image loaded into RAM
int32_t              width  = 0, // BMP image dimensions
                     height = 0;

void setup(void) {

  ImageReturnCode stat; // Status from image-reading functions

  Serial.begin(115200);
  while(!Serial);           // Wait for Serial Monitor before continuing

  display.begin();
  display.setRotation(3);

  // The Adafruit_ImageReader constructor call (above, before setup())
  // accepts an uninitialized SdFat or FatVolume object. This MUST
  // BE INITIALIZED before using any of the image reader functions!
  Serial.print(F("Initializing filesystem..."));
  // SPI or QSPI flash requires two steps, one to access the bare flash
  // memory itself, then the second to access the filesystem within...
#if defined(USE_SD_CARD)
  // SD card is pretty straightforward, a single call...
  if(!SD.begin(SD_CS, SD_SCK_MHZ(10))) { // Breakouts require 10 MHz limit due to
longer wires
    Serial.println(F("SD begin() failed"));
    for(;;); // Fatal error, do not continue
  }
#else
  // SPI or QSPI flash requires two steps, one to access the bare flash
  // memory itself, then the second to access the filesystem within...
  if(!flash.begin()) {
    Serial.println(F("flash begin() failed"));
    for(;;);
  }
```

```
  if(!filesys.begin(&flash)) {
    Serial.println(F("filesys begin() failed"));
    for(;;);
  }
#endif
  Serial.println(F("OK!"));

  // Load full-screen BMP file 'blinka.bmp' at position (0,0) (top left).
  // Notice the 'reader' object performs this, with 'epd' as an argument.
  Serial.print(F("Loading blinka.bmp to canvas..."));
  stat = reader.drawBMP((char *)"/blinka.bmp", display, 0, 0);
  reader.printStatus(stat); // How'd we do?
  display.display();

  // Query the dimensions of image 'blinka.bmp' WITHOUT loading to screen:
  Serial.print(F("Querying blinka.bmp image size..."));
  stat = reader.bmpDimensions("blinka.bmp", &width, &height);
  reader.printStatus(stat);   // How'd we do?
  if(stat == IMAGE_SUCCESS) { // If it worked, print image size...
    Serial.print(F("Image dimensions: "));
    Serial.print(width);
    Serial.write('x');
    Serial.println(height);
  }

  delay(30 * 1000); // Pause 30 seconds before continuing because it's eInk

}

void loop() {

}
```

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. You'll see the microSD card recognized over SPI. Then, you'll see the image show on your display.



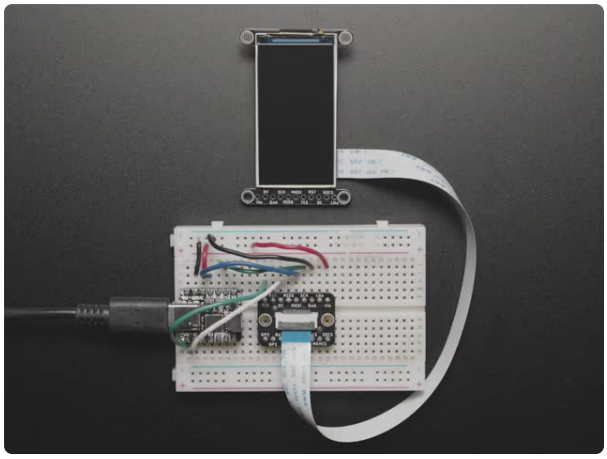
Arduino Docs

Arduino Docs (https://adafru.it/BST)

CircuitPython

ll need to use CircuitPython 10.0.0-beta.2 or later with this display.

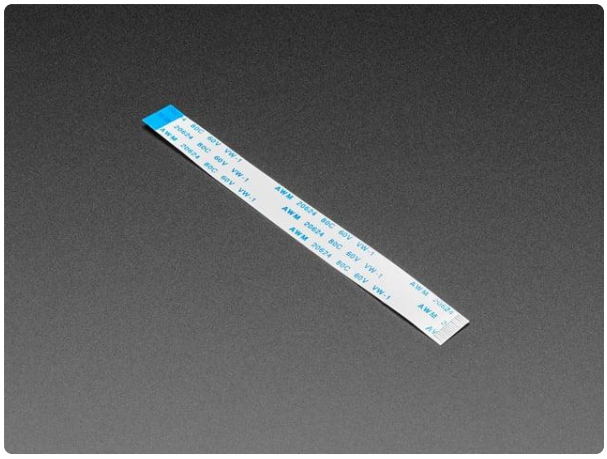
This page goes over how to use the **Adafruit 2.13" 250x122 Quad-Color eInk** displays with CircuitPython.



Adafruit EYESPI Breakout Board - 18 Pin FPC Connector

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

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



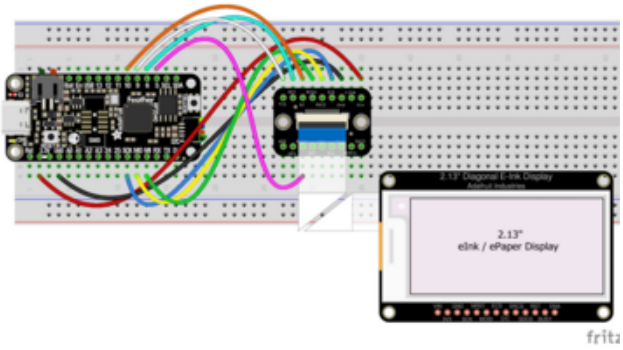
EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type

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

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

CircuitPython Microcontroller Wiring

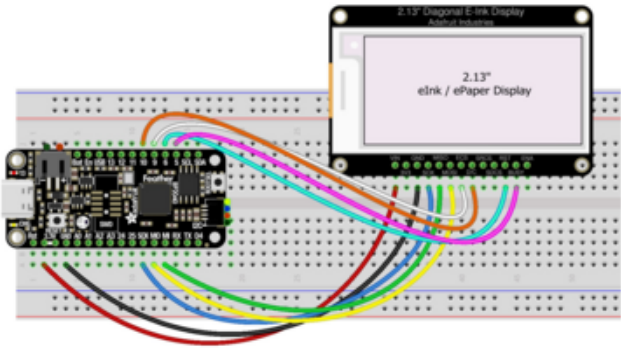
First wire up the display to your board exactly as follows. The following is the display connected to a Feather RP2040 using the EYESPI connector:



Feather 3.3V to **breakout Vin (red wire)**
Feather GND to **breakout Gnd (black wire)**
Feather SCK to **breakout SCK (blue wire)**
Feather MO to **breakout MOSI (yellow wire)**
Feather MI to **breakout MISO (green wire)**
Feather D10 to **breakout DC (orange wire)**
Feather D9 to **breakout TCS (white wire)**
Feather D6 to **breakout RST (cyan wire)**
Feather D5 to **breakout BUSY (pink wire)**

Attach the **TFT screen to the EYESPI breakout** with an **EYESPI cable** as described on the Plugging in an EYESPI Cable (https://adafru.it/18eU) page.

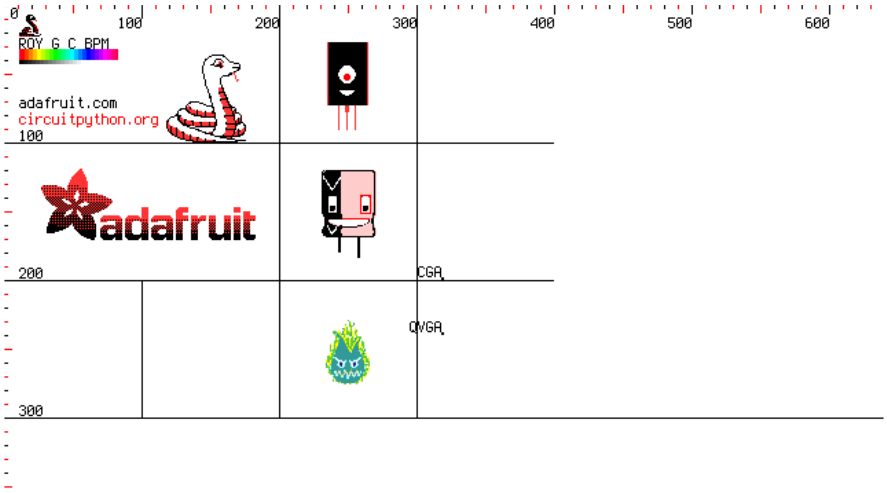
The following is the breakout wired to a Feather RP2040 using a solderless breadboard:



Feather 3.3V to **breakout VIN (red wire)**
Feather GND to **breakout GND (black wire)**
Feather SCK to **breakout SCK (blue wire)**
Feather MO to **breakout MOSI (yellow wire)**
Feather MI to **breakout MISO (green wire)**
Feather D10 to **breakout D/C (orange wire)**
Feather D9 to **breakout ECS (white wire)**
Feather D6 to **breakout RST (cyan wire)**
Feather D5 to **breakout BUSY (pink wire)**

Image File

The example below uses a bitmap image. You'll need the **display-ruler-640x360.bmp** bitmap file on your CIRCUITPY drive.



CircuitPython Usage

To use with CircuitPython, you need to first install the **Adafruit_CircuitPython_JD79661** library, and its dependencies, into the **lib** folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

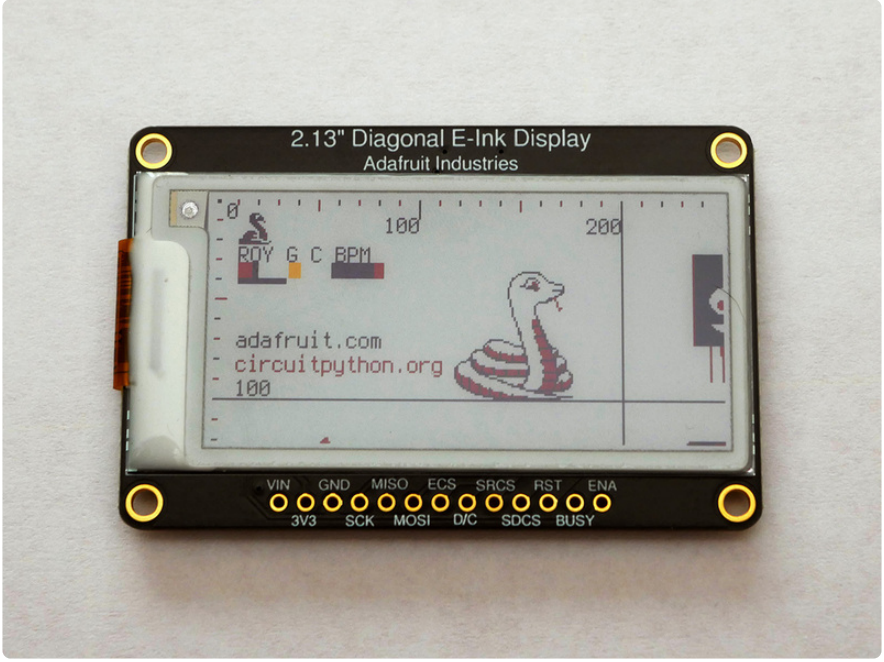
Your **CIRCUITPY/lib** folder should contain the following folders and file:

- **adafruit_jd79661.mpy**

Example Code

```
# SPDX-FileCopyrightText: 2025 Scott Shawcroft, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense

"""Simple test script for 2.13" Quad Color Display"""

import time

import board
import displayio
from fourwire import FourWire

import adafruit_jd79661

displayio.release_displays()

spi = board.SPI()
epd_cs = board.D9
epd_dc = board.D10
epd_reset = board.D6
epd_busy = board.D5

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

display = adafruit_jd79661.JD79661(
    display_bus,
    width=250,
    height=122,
    busy_pin=epd_busy,
    rotation=270,
    colstart=0,
    highlight_color=0x00FF00,
    highlight_color2=0xFF0000,
)

g = displayio.Group()

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

display.root_group = g

display.refresh()

print("refreshed")

time.sleep(display.time_to_refresh + 5)
# Always refresh a little longer. It's not a problem to refresh
# a few seconds more, but it's terrible to refresh too early
# (the display will throw an exception when if the refresh
# is too soon)
print("waited correct time")


# Keep the display the same
while True:
    time.sleep(10)
```

Once everything is saved to the **CIRCUITPY** drive, the code will begin running. Your display will look like this:



CircuitPython Docs

CircuitPython Docs (https://adafru.it/1aoH)

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).

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

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

Setup Virtual Environment

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

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

To activate the virtual environment:

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

Install Adafruit_Blinka

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

Python Installation of EPD Library

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

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

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

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

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

Download font5x8.bin

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

https://adafru.it/Xbr

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

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

DejaVu TTF Font

Raspberry Pi usually comes with the DejaVu font already installed, but in case it didn't, you can run the following to install it:

```
sudo apt-get install fonts-dejavu
```

This package was previously calls **ttf-dejavu**, so if you are running an older version of Raspberry Pi OS, it may be called that.

Pillow Library

Some of the examples also use PIL, the Python Imaging Library, to allow graphics and using text with custom fonts. There are several system libraries that PIL relies on, so installing via a package manager is the easiest way to bring in everything:

```
sudo apt-get install python3-pil
```

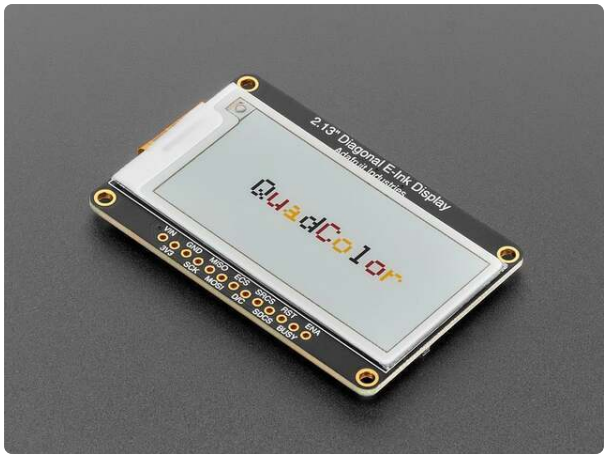
Chip Enable Lines

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

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

Python Use

After you've followed the steps on the Python Setup page (https://adafru.it/1aoK), you can get started with using the **2.13" 250x122 Quad-Color eInk** display with the Adafruit CircuitPython EPD library and a Raspberry Pi single-board computer.

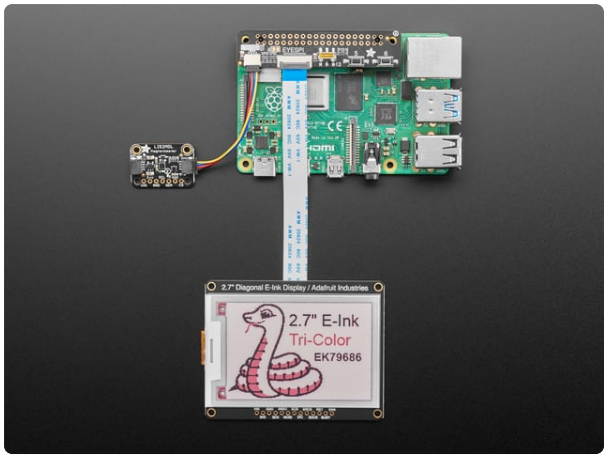


Adafruit 2.13" 250x122 Quad-Color eInk
By Liz Clark
Python Setup

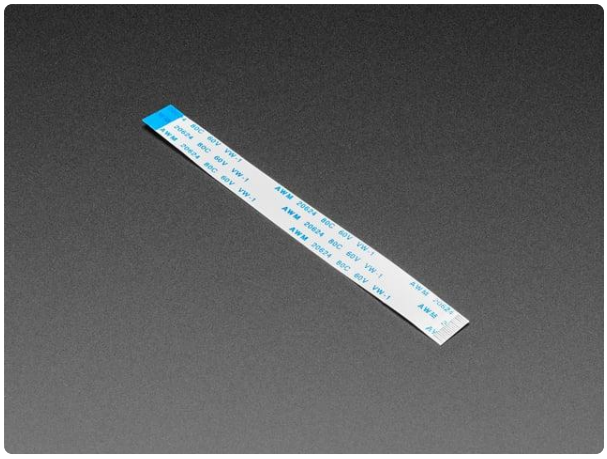
https://learn.adafruit.com/adafruit-2-13-250x122-quad-color-eink/python-setup

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

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



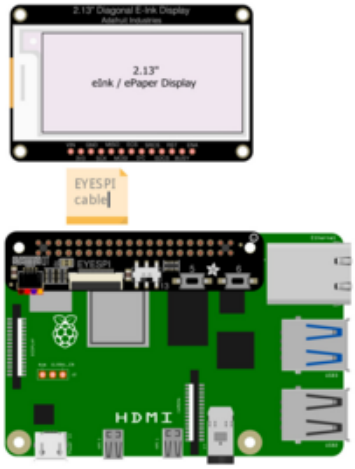
[Adafruit EYESPI Pi Beret - Buttons, EYESPI and STEMMA QT]
Raspberry Pi's make for handy lil computers, but they're really wonderful when you can connect all sorts of nifty hardware to them: color TFT or E-Ink displays, and sensors are...
https://www.adafruit.com/product/5783



[EYESPI Cable - 18 Pin 100mm long Flex PCB (FPC) A-B type]
Connect this to that when a 18-pin FPC connector is needed. This 25 cm long cable is made of a flexible PCB. It's A-B style which means that pin one on one side will match...
https://www.adafruit.com/product/5239

Python Wiring

The following is the display connected to a Raspberry Pi with an EYESPI Pi Beret using the EYESPI connector:

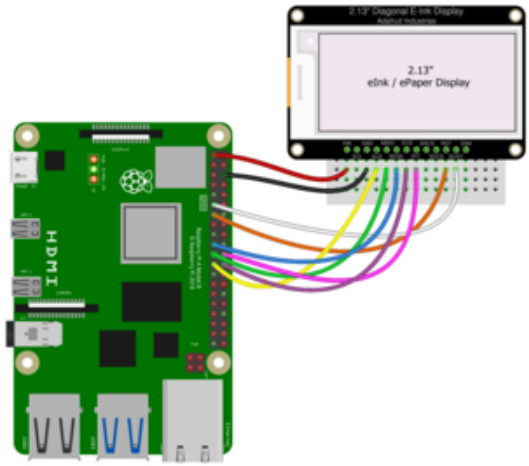


Plug the **EYESPI Pi Beret** into the **Raspberry Pi 2x20 header**. It has the following pin connections:

Pi 3.3V to **breakout Vin**
Pi GND to **breakout Gnd**
Pi SCK to **breakout SCK**
Pi MOSI to **breakout MOSI**
Pi MISO to **breakout MISO**
Pi GPIO 27 to **breakout RST**
Pi GPIO 25 to **breakout DC**
Pi GPIO CE0 to **breakout CS**
Pi GPIO 17 to **breakout BUSY**

Attach the **eInk display to the EYESPI breakout** with an **EYESPI cable** as described on the Plugging in an EYESPI Cable (https://adafru.it/18eU) page.

The following is the breakout wired to a Raspberry Pi using a solderless breadboard:



Pi 3.3V to **breakout Vin (red wire)**
Pi GND to **breakout Gnd (black wire)**
Pi SCK to **breakout SCK (yellow wire)**
Pi MOSI to **breakout MOSI (blue wire)**
Pi MISO to **breakout MISO (green wire)**
Pi GPIO 27 to **breakout RST (orange wire)**
Pi GPIO 25 to **breakout DC (pink wire)**
Pi GPIO CE0 to **breakout CS (purple wire)**
Pi GPIO 17 to **breakout BUSY (white wire)**

Pillow Graphics Demo

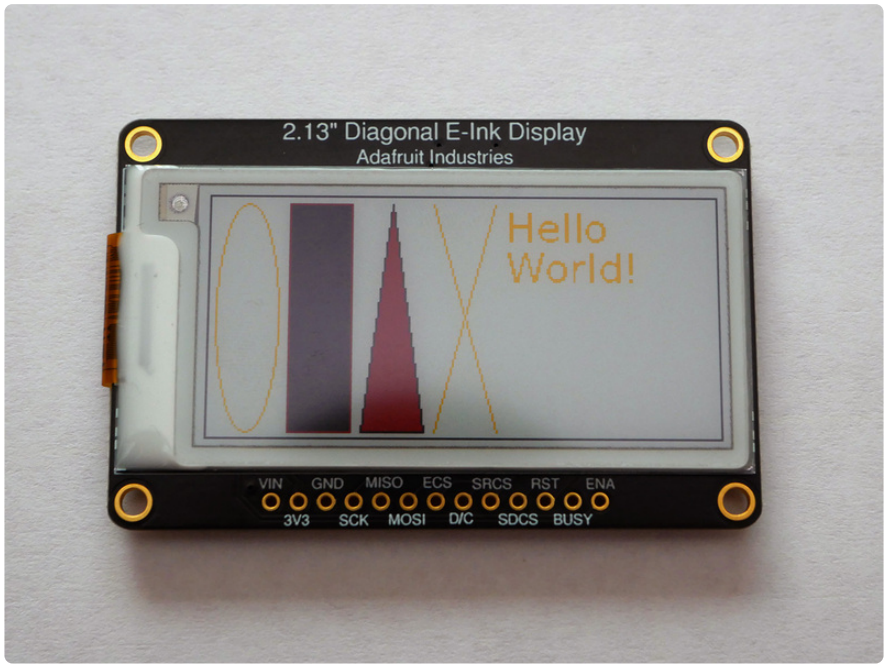
The great part about using a display on a Raspberry Pi is that you can use Pillow graphics alongside the CircuitPython driver.

The following example uses Pillow for text and drawing shapes. Copy or download the following example to your computer, and run the following, replacing **code.py** with whatever you named the file:

`python3 code.py`

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
"""Blinka EPD Demo for the Quad Color eInk"""
import board
import digitalio
from PIL import Image, ImageDraw, ImageFont

from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.jd79661 import Adafruit_JD79661

# create the spi device and pins we will need
spi = board.SPI()
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D25)
srcs = None
rst = digitalio.DigitalInOut(board.D27)  # can be None to not use this pin
busy = digitalio.DigitalInOut(board.D17)  # can be None to not use this pin

display = Adafruit_JD79661(122, 250, spi,
                           cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                           rst_pin=rst, busy_pin=busy)

display.rotation = 3
width = display.width
height = display.height
image = Image.new("RGB", (width, height))

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

# clear the buffer
display.fill(Adafruit_EPD.WHITE)

# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)
# empty it
draw.rectangle((0, 0, width, height), fill=WHITE)

# Draw an outline box
draw.rectangle((1, 1, width - 2, height - 2), outline=BLACK, fill=WHITE)

# Draw some shapes.
# First define some constants to allow easy resizing of shapes.
padding = 5
shape_width = 30
top = padding
bottom = height - padding
# Move left to right keeping track of the current x position for drawing shapes.
x = padding
# Draw an ellipse.
draw.ellipse((x, top, x + shape_width, bottom), outline=YELLOW, fill=WHITE)
x += shape_width + padding
# Draw a rectangle.
draw.rectangle((x, top, x + shape_width, bottom), outline=RED, fill=BLACK)
x += shape_width + padding
# Draw a triangle.
draw.polygon(
    [(x, bottom), (x + shape_width / 2, top), (x + shape_width, bottom)],
```

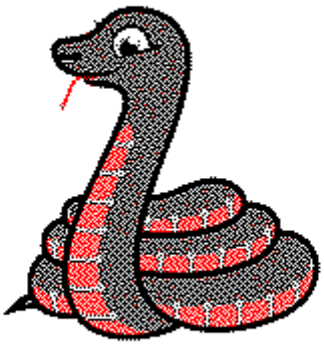
```
    outline=BLACK,
    fill=RED,
)
x += shape_width + padding
# Draw an X.
draw.line((x, bottom, x + shape_width, top), fill=YELLOW)
draw.line((x, top, x + shape_width, bottom), fill=YELLOW)
x += shape_width + padding

# Load default font.
font = ImageFont.truetype("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf", 20)

draw.text((x, top), "Hello", font=font, fill=YELLOW)
draw.text((x, top + 20), "World!", font=font, fill=YELLOW)

# Display image.
display.image(image)

display.display()
```



Pillow Image Demo

Pillow is really useful for displaying image files as well.

The following example uses Pillow to display a **.PNG** file. The image is available to download here or via the Project Bundle with the **code.py** file.

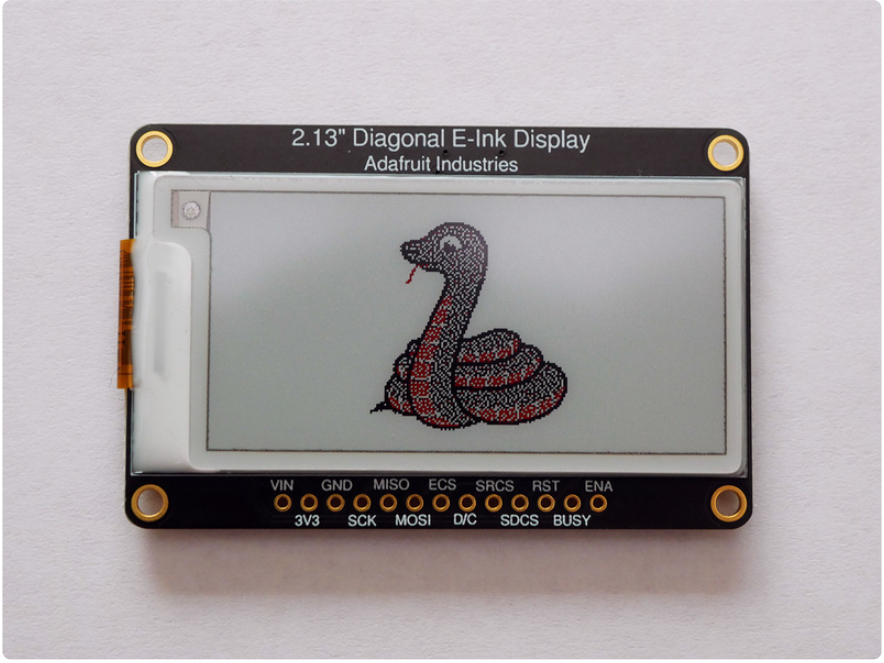


Copy or download the following example to your computer, and run the following, replacing **code.py** with whatever you named the file:

`python3 code.py`

```
# SPDX-FileCopyrightText: 2019 Melissa LeBlanc-Williams for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
Image resizing and drawing using the Pillow Library for Quad Color eInk
"""
import board
import digitalio
from PIL import Image, ImageEnhance
from adafruit_epd.jd79661 import Adafruit_JD79661

# create the spi device and pins we will need
spi = board.SPI()
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D25)
srcs = None
rst = digitalio.DigitalInOut(board.D27)  # can be None to not use this pin
busy = digitalio.DigitalInOut(board.D17)  # can be None to not use this pin

# give them all to our driver
display = Adafruit_JD79661(122, 250,        # 2.13" Quad-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)
display.rotation = 3

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")

quad_colors = [
    (0, 0, 0),        # Black
    (255, 255, 255),  # White
    (255, 0, 0),      # Red
    (255, 255, 0),    # Yellow
]
palette_image = Image.new('P', (1, 1))

# Create palette data - PIL expects 768 values (256 colors * 3 channels)
palette_data = []
for color in quad_colors:
    palette_data.extend(color)
# Fill remaining palette entries with black
for i in range(4, 256):
    palette_data.extend([0, 0, 0])

palette_image.putpalette(palette_data)

enhancer = ImageEnhance.Color(image)
image = enhancer.enhance(1.5)

temp_image = image.quantize(palette=palette_image,
dither=Image.Dither.FLOYDSTEINBERG)

pixels = temp_image.load()
width, height = temp_image.size

final_palette = Image.new('P', (1, 1))
final_palette.putpalette(palette_data)
final_image = Image.new('P', (width, height))
final_pixels = final_image.load()

# Copy pixels, ensuring they use indices 0-3
for y in range(height):
    for x in range(width):
        # Clamp pixel values to 0-3 range
        final_pixels[x, y] = min(pixels[x, y], 3)

final_image.putpalette(palette_data)

# Convert back to RGB for display
image = final_image.convert('RGB')

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



Python Docs

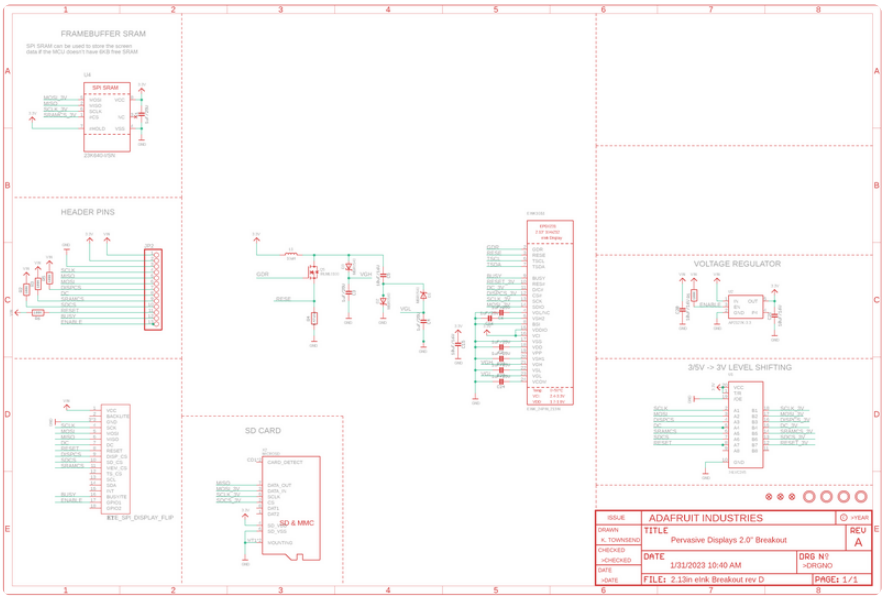
Python Docs (https://adafru.it/1aoI)

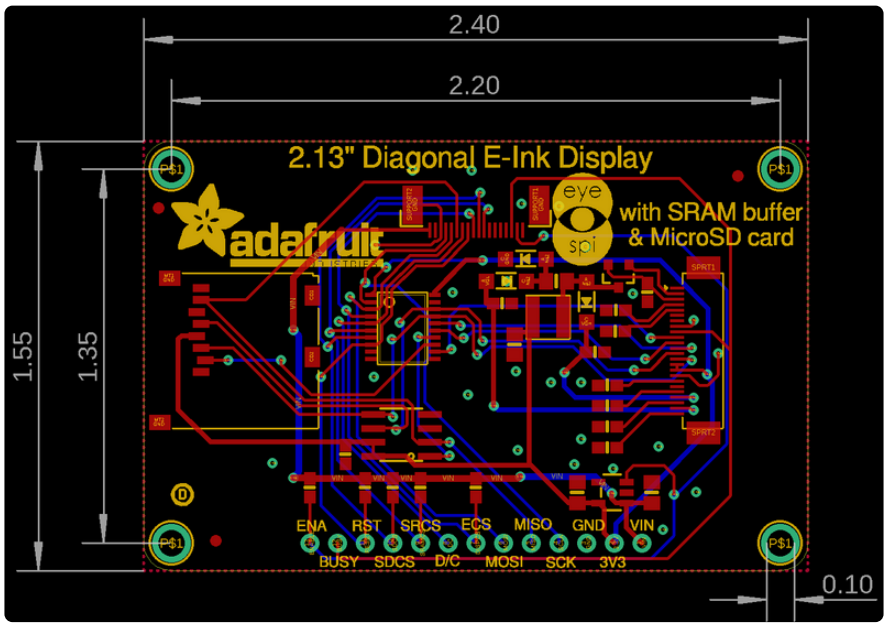
Downloads

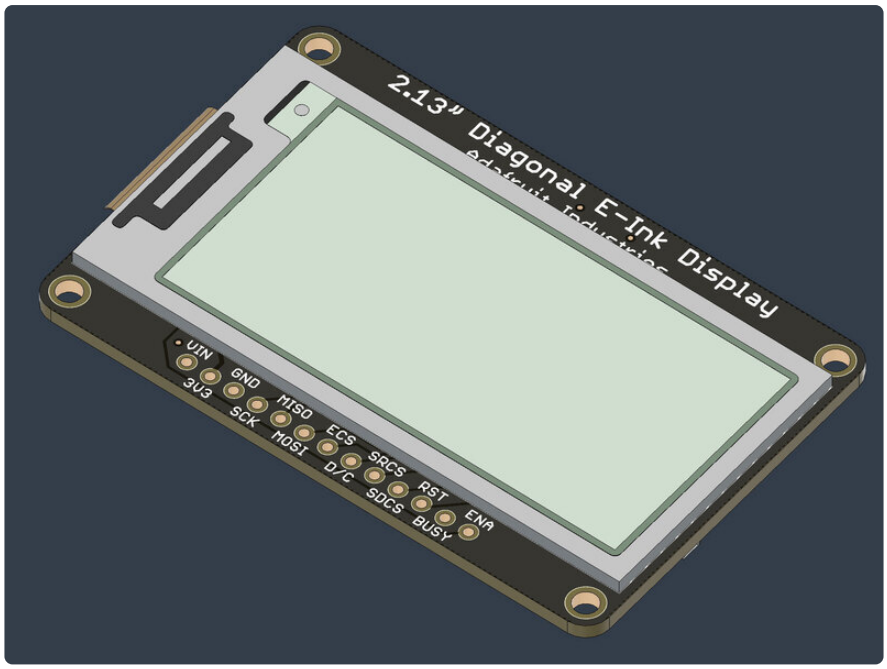
Files

- JD79661 Chipset Datasheet (https://adafru.it/1aoL)
- EagleCAD PCB files on GitHub (https://adafru.it/BRX)
- 3D models on GitHub (https://adafru.it/1aoM)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/1aoN)

Schematic and Fab Print






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