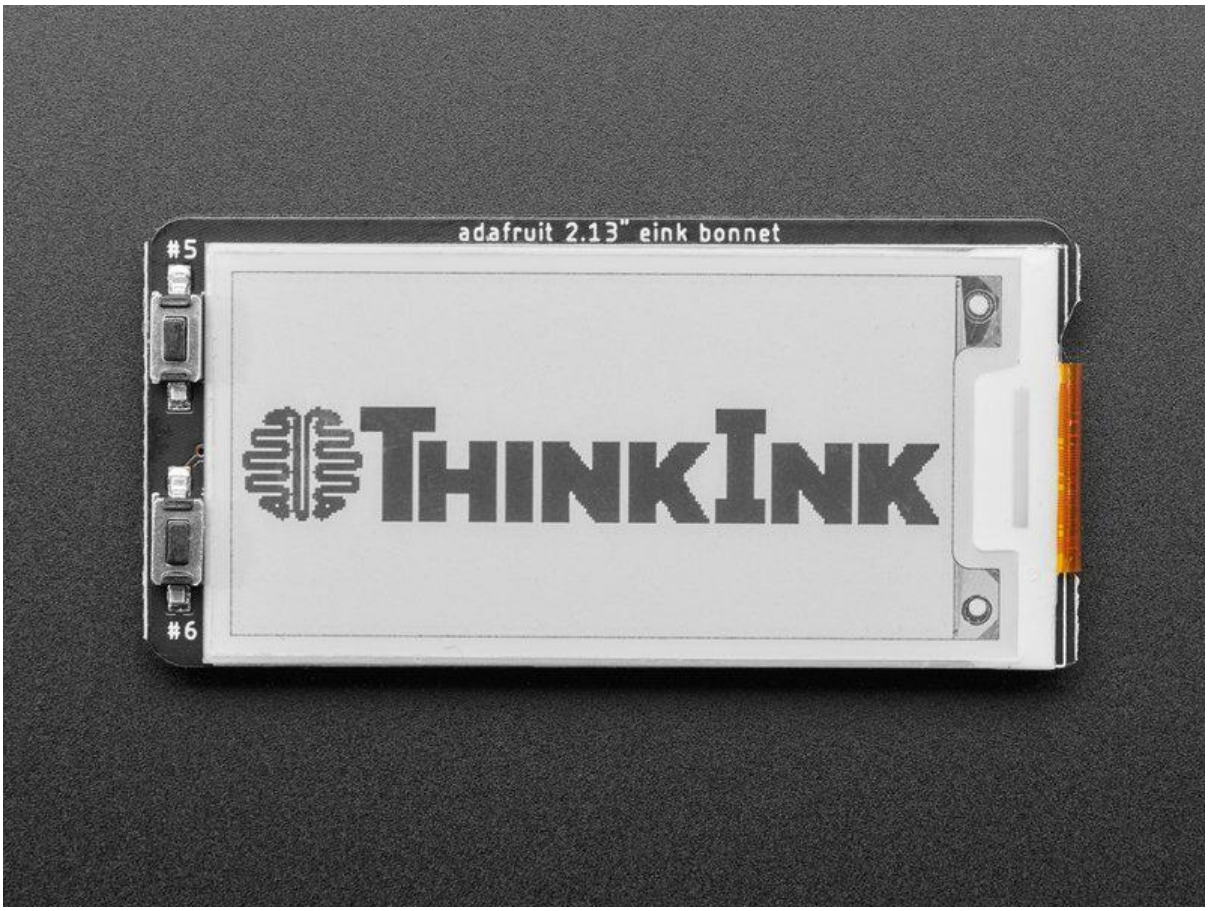
Adafruit 2.13" Monochrome E-Ink Bonnet for Raspberry Pi

Created by Kattni Rembor

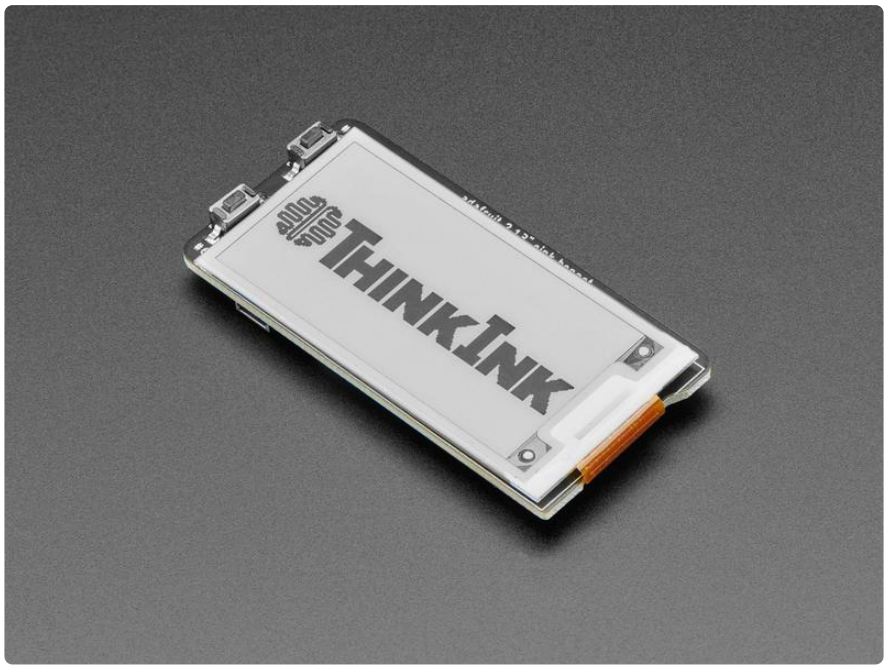


https://learn.adafruit.com/2-13-in-e-ink-bonnet

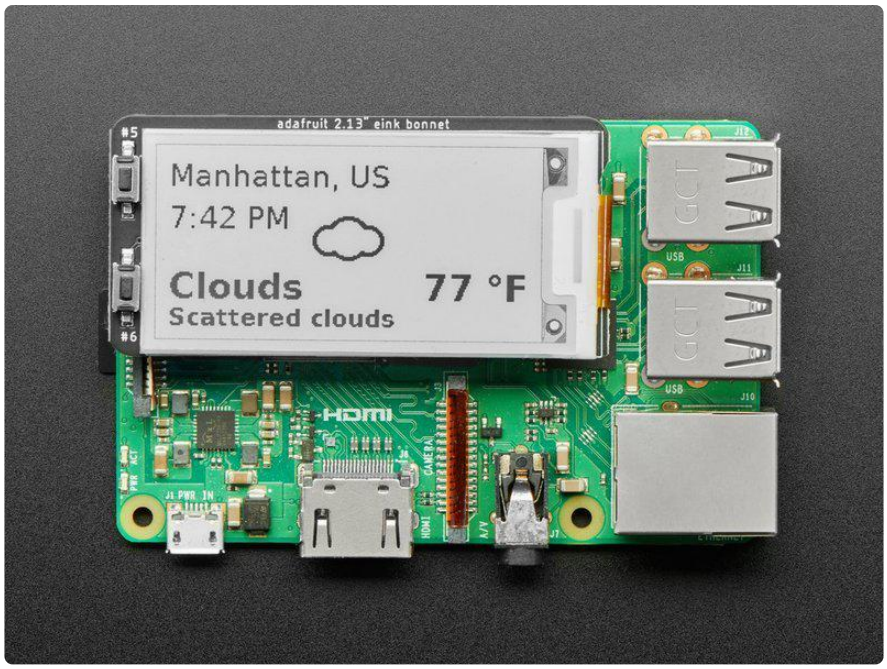
Last updated on 2023-08-29 04:30:22 PM EDT

Table of Contents

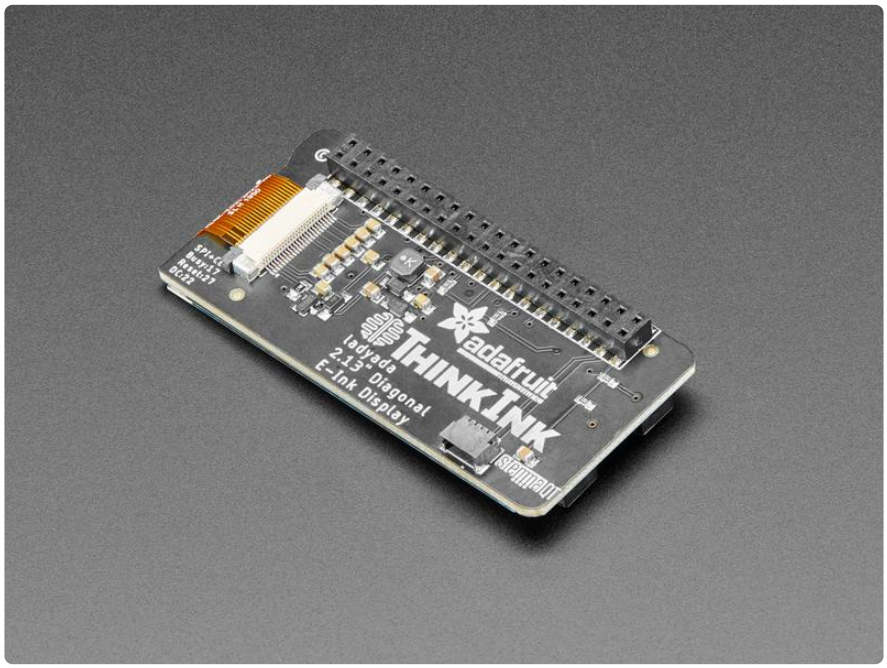
Overview



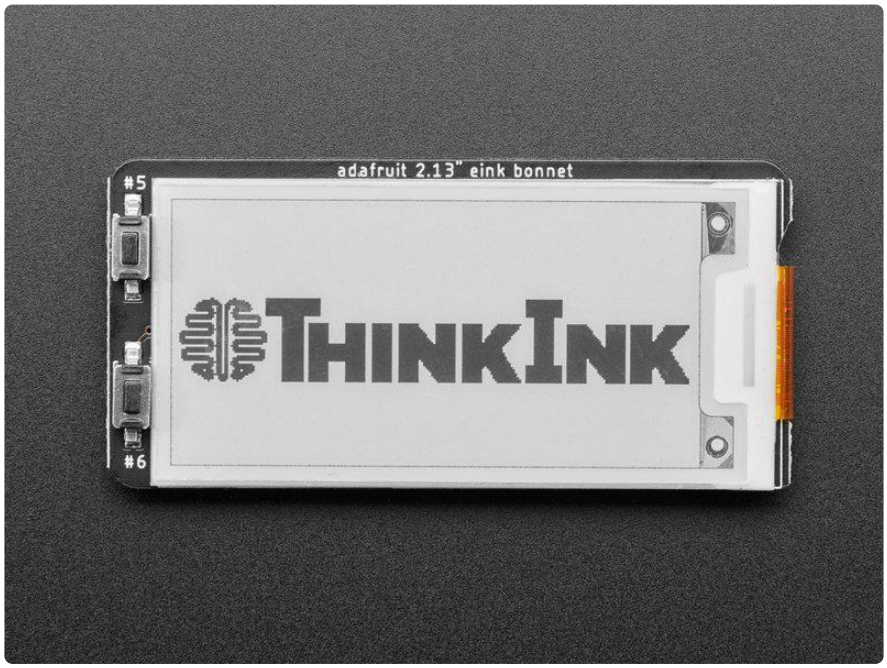
Easy e-paper finally comes to Raspberry Pi, with this bonnet that's designed to make it a breeze to add a 2.13" 250x122 crisp monochromic eInk display. Chances are you've seen one of those new-fangled 'e-readers' like the Kindle or Nook. They have gigantic electronic paper 'static' displays - that means 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!



The Adafruit 2.13" Monochrome E-Ink Bonnet for Raspberry Pi snaps onto any modern Raspberry Pi and provides a Python-programmable display with two buttons that can be used to select programs or scroll through options.

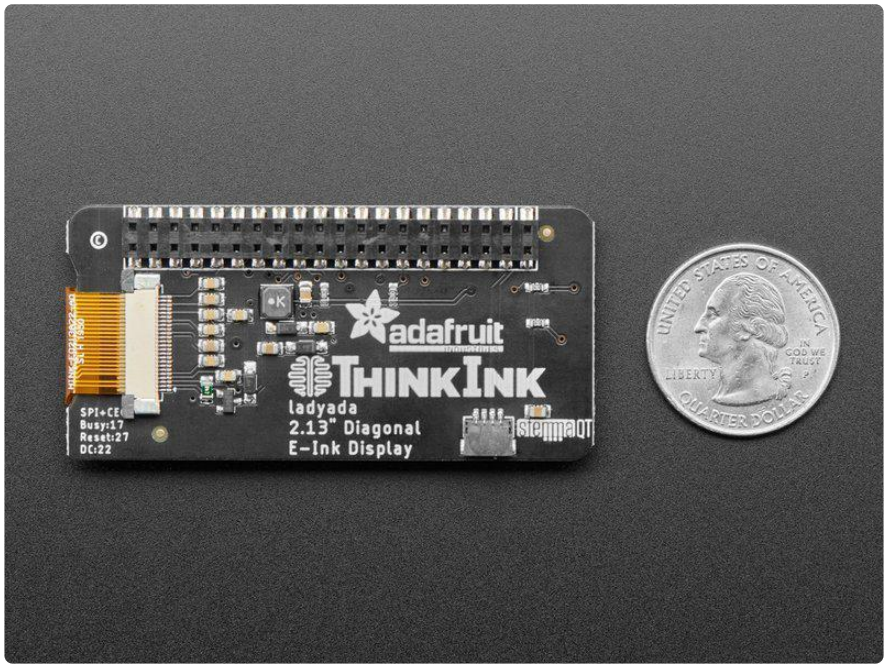


We have two fun starter guides to use with this bonnet, an Open Weather display () and an event calendar that auto-syncs with a Google Calendar () to show you what your next meeting or event is. We also have examples in our E-Ink Python library you can use to craft your own project ().

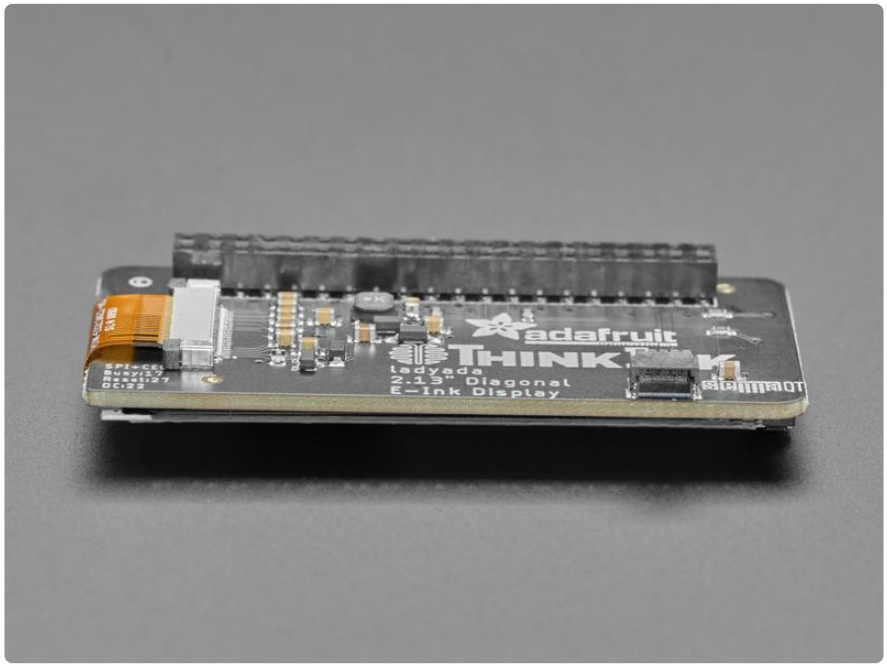


Comes completely pre-assembled and tested so you don't need to do anything but plug it in and install our Python code! Works with any Raspberry Pi computer that has

a 2x20 connector, such as the Pi B+, Pi 2, Pi 3, Pi 4, and Pi Zero (and any others that have a 2x20 connector!)



On the bottom, we have a Qwiic/STEMMA QT connector for I2C sensors and devices so you can plug and play any of our STEMMA QT devices ().



Display Versions

As of May 1, 2021 - we are now selling this bonnet with an E-Ink display that uses the SSD1680 chipset as the SSD1675 is discontinued/unavailable. The resolution and size

are the same, but firmware/python code will need to be updated to use the SSD1680-driver in our library rather than SSD1675!

As of October 2022, we've updated this PCB with Adafruit Pinguin () to make a lovely and legible silkscreen - you may get the new PCB or the older version with vector fonts - both are identical other than the fancy silkscreen. Additionally, this bonnet may come with a black or tan STEMMA QT connector. They work the same!

Because of the different chipsets, you will need to be able to choose the correct driver for the display you will be using it with. It's easy to tell the difference if you know what to look for.

On the eInk display itself, the SSD1675 version has two tabs on the righthand side:

The SSD1680 version has only a single tab on the righthand side:

Usage

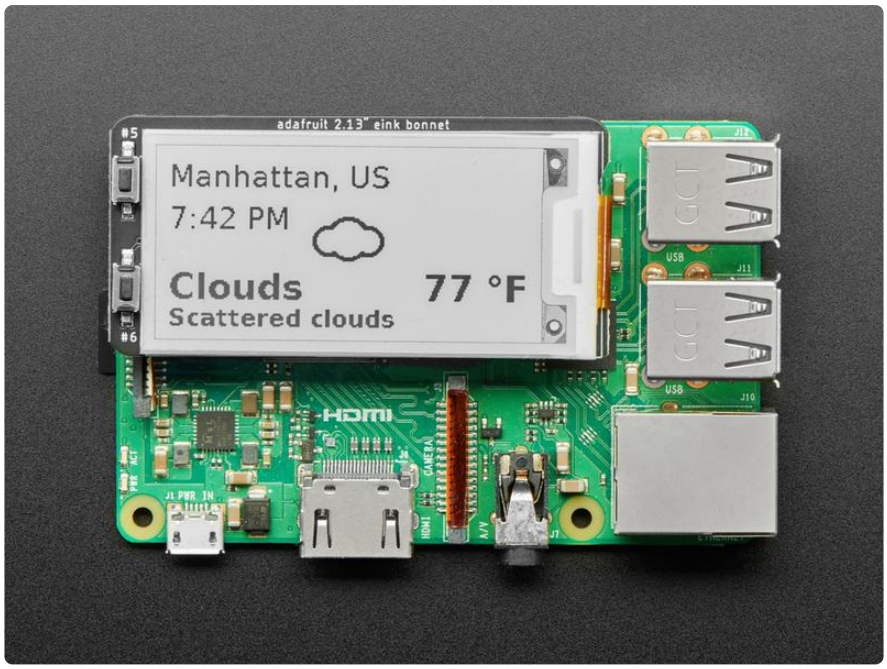
This guide assumes you have your Raspberry Pi all set up with an operating system, network connectivity, SSH, and SPI enabled!

Attaching

Since the eInk Bonnet comes preassembled, all you need to do is place it onto the GPIO pins.

Since there's 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 ().

Connect the display as shown below to your Raspberry Pi.



Install CircuitPython

This guide assumes that you've gotten your Raspberry Pi up and running, and have Blinka installed. If not, check out the guide:

CircuitPython Installation Guide

To install the EPD library for the Pi (), enter the following into the terminal:

- `pip3 install adafruit-circuitpython-epd`

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. If you don't have it, you can easily get it by running the following command:

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

DejaVu TTF Font

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

- `sudo apt-get install fonts-dejavu`

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

Pillow Library

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

- `sudo apt-get install python3-pil`

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

You'll want to refrain from updating your E-Ink display too often. Once every 3 minutes is generally a safe wait time.

EPD Library Usage

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

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

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

spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
srcs = None
```

Run only one the following code sections to initialize the display.

If you have an SSD1675-chipset display (original chipset) use this code:

```
from adafruit_epd.ssd1675 import Adafruit_SSD1675
display = Adafruit_SSD1675(122, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                           rst_pin=rst, busy_pin=busy)
```

OR if you have an SSD1680-chipset display use this code:

```
from adafruit_epd.ssd1680 import Adafruit_SSD1680
display = Adafruit_SSD1680(122, 250, spi, cs_pin=ecs, dc_pin=dc, sramcs_pin=srcs,
                           rst_pin=rst, busy_pin=busy)
```

Note that the chip name is different, SSD1675 vs SSD1680. If one doesn't work to display, simply try the other!

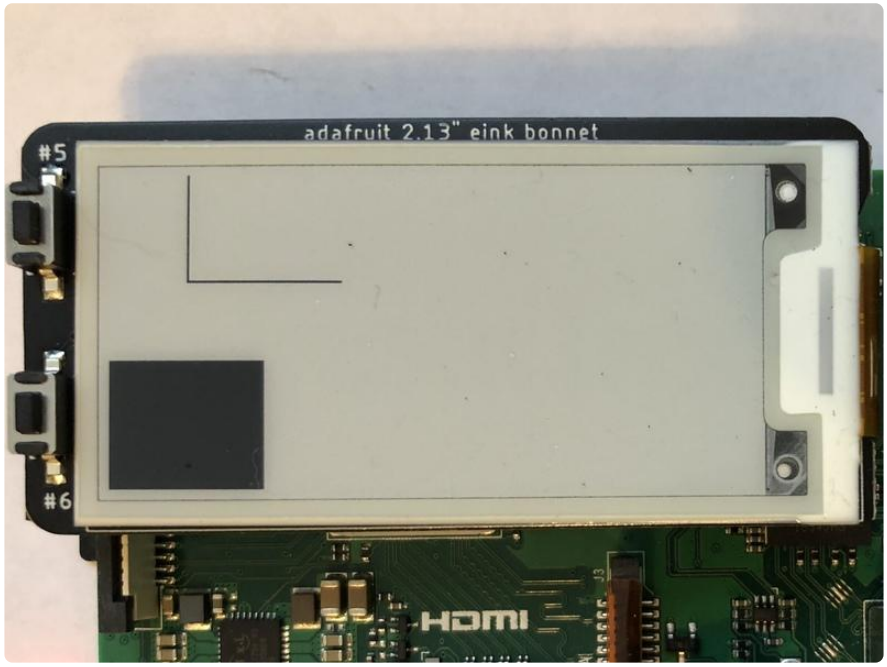
Now you can clear the screens buffer and draw some shapes. Once you're done drawing, you need to tell the screen to update using the `display()` method.

```
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 display will look something like this:



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

Button Usage

To use the buttons, you just need to use digitalio. Then it's a matter of setting up the buttons as digital inputs:

```
import digitalio

up_button = digitalio.DigitalInOut(board.D5)
up_button.switch_to_input()
down_button = digitalio.DigitalInOut(board.D6)
down_button.switch_to_input()
```

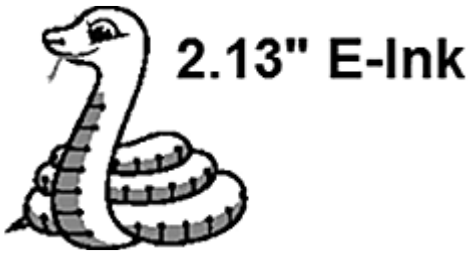
One thing to be aware of is since the buttons are pulled low when they are pushed, they will return `False` when pressed and `True` when they aren't:

```
if not up_button.value:
    print("Up Button Pushed")

if not down_button.value:
    print("Down Button Pushed")
```

Image Drawing with Pillow

In this image, you will use Pillow to resize and crop the image automatically and draw it the the ePaper Display. Pillow is really powerful and with it you can open and render additional file formats such as PNG or JPG. Let's start with downloading a PNG of blinka. We are using PNG for this because it is a lossless format and won't introduce unexpected colors in.



You can easily download it directly to your pi using the following command:

- `wget https://raw.githubusercontent.com/adafruit/Adafruit_Learning_System_Guides/main/EInk_Bonnet/blinka.png`

Make sure you save it as blinka.png and place it in the same folder as your script.

Here's the code you'll be loading onto the Raspberry Pi. Go ahead and copy it onto your Raspberry Pi and save it as epd_pillow_image.py. You'll go over the interesting parts along with a couple changes that you will need to make.

```
# SPDX-FileCopyrightText: 2019 Melissa LeBlanc-Williams for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
Image resizing and drawing using the Pillow Library. For the image, check out the
associated Adafruit Learn guide at:
https://learn.adafruit.com/adafruit-eink-display-breakouts/python-code

"""

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


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

# give them all to our driver
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color or mono display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_EK79686(176, 264,        # 2.7" Tri-color display
# display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
# display = Adafruit_UC8151D(128, 296,        # 2.9" mono flexible display
# display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

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

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

```
display.rotation = 1

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

# Scale the image to the smaller screen dimension
image_ratio = image.width / image.height
screen_ratio = display.width / display.height
if screen_ratio < image_ratio:
    scaled_width = image.width * display.height // image.height
    scaled_height = display.height
else:
    scaled_width = display.width
    scaled_height = image.height * display.width // image.width
image = image.resize((scaled_width, scaled_height), Image.BICUBIC)

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

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

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

So the script starts with the usual imports including a couple of Pillow modules and the ePaper display drivers.

```
import digitalio
import busio
import board
from PIL import Image, ImageDraw
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
```

That is followed by initializing the SPI bus and defining a few pins here. The reason we chose these is because they allow you to use the same code with the EPD bonnets if you chose to do so.

```
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
ecs = digitalio.DigitalInOut(board.CE0)
dc = digitalio.DigitalInOut(board.D22)
srcs = None
rst = digitalio.DigitalInOut(board.D27)
busy = digitalio.DigitalInOut(board.D17)
```

We wanted to make these examples work on as many displays as possible with very few changes. The 2.13" Tri-color display is selected by default. Since this is a monochrome display, you'll want to go ahead and comment out the following lines:

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

and uncomment the line for the 2.13" HD mono display:

```
display = Adafruit_SSD1675(122, 250,
```

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

The next two lines are for flexible displays. This tells the library to change a couple of settings so that it is writing the correct colors to the correct places. Since this isn't a flexible display, you can skip past these lines.

```
# 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)
```

Next the script tells the display the rotation setting you want to use. This can be a value between `0` to `3`. For the bonnet, you'll want to stick with the default value.

```
display.rotation = 1
```

Next the script opens the Blinka image, which you've named blinka.png. The open command assumes it is in the same directory that you are running the script from. Feel free to change it if it doesn't match your configuration.

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

Here's where it starts to get interesting. You want to scale the image so that it matches either the width or height of the display, depending on which is smaller, so that you have some of the image to chop off when you crop it. So you'll start by calculating the width to height ratio of both the display and the image. If the height is the closer of the dimensions, you'll want to match the image height to the display

height and let it be a bit wider than the display. Otherwise, you'll want to do the opposite.

Once the script figures out how it's going to scale it, it passes in the new dimensions and using a Bicubic rescaling method, it reassigns the newly rescaled image back to `image`. Pillow has quite a few different methods to choose from, but Bicubic does a great job and is reasonably fast.

Nearest actually gives a little better result with the Tri-color eInks, but loses detail with displaying a color image on the monochrome display, so we decided to go with the best balance.

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

Next to figure the starting x and y points of the image to begin cropping it so that it ends up centered. That is done by using a standard centering function, which is basically requesting the difference of the center of the display and the center of the image. Just like with scaling, replace the `image` variable with the newly cropped image.

```
x = scaled_width // 2 - display.width // 2
y = scaled_height // 2 - display.height // 2
image = image.crop((x, y, x + display.width, y + display.height))
```

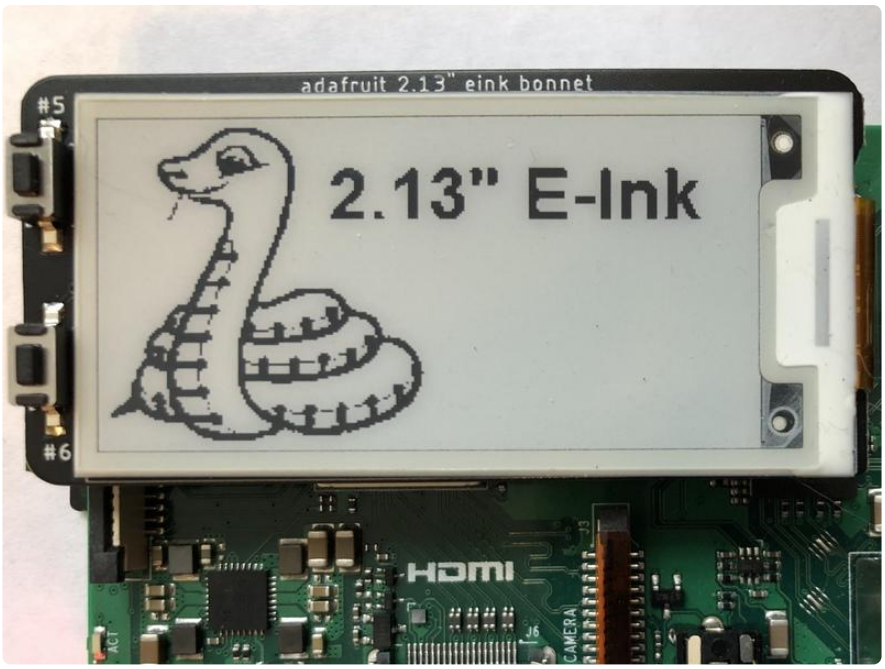
Finally, taking the `image`, draw it to the frame buffer and `display` it. At this point, the image should have the exact same dimensions at the display and fill it completely.

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

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

```
python3 epd_pillow_image.py
```

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



Adding Dithering for Monochrome displays

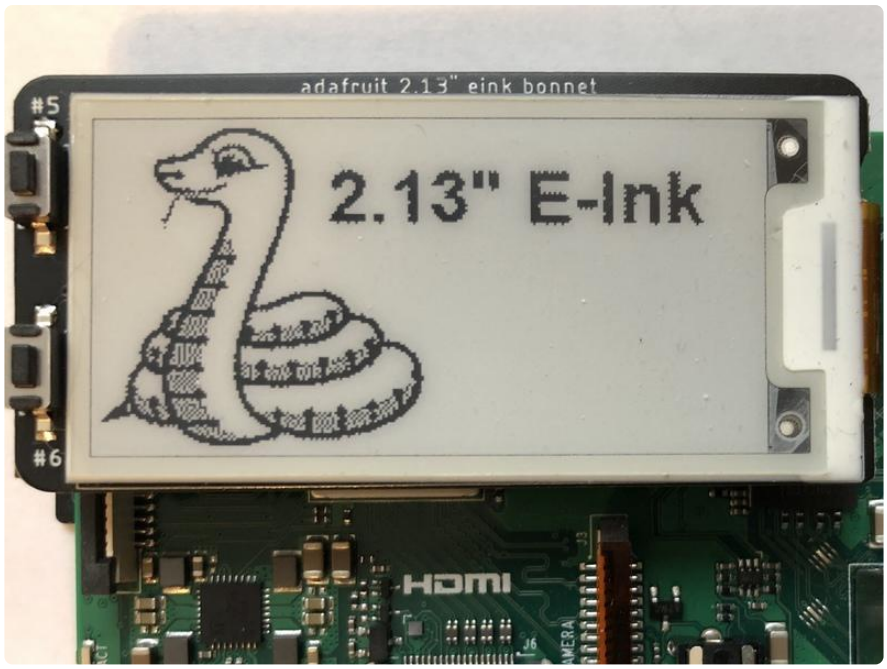
One little trick which can be done to increase the image quality on monochrome displays is to use dithering. Pillow can be forced to dither the image by converting first to 1-bit and then convert it back to either grayscale or RGB. This works great for monochrome E-Ink displays, but removes the color information for Tri-color displays.

To dither the image, add 1 line of code right before calling `display.image(image)`:

```
image = image.convert("1").convert("L")

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

When you run it with the additional code, the same image displays like this:



Drawing Shapes and Text with Pillow

In the next example, let's take a look at drawing shapes and text. This is very similar to the displayio example used on other displays, but it uses Pillow instead. Go ahead and copy it onto your Raspberry Pi and save it as epd_pillow_demo.py. Here's the code for that.

```
# SPDX-FileCopyrightText: 2019 Melissa LeBlanc-Williams for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
ePaper Display Shapes and Text demo using the Pillow Library.

"""

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

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

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

```
FOREGROUND_COLOR = WHITE
TEXT_COLOR = RED

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

# give them all to our driver
# display = Adafruit_SSD1608(200, 200,        # 1.54" HD mono display
# display = Adafruit_SSD1675(122, 250,        # 2.13" HD mono display
# display = Adafruit_SSD1680(122, 250,        # 2.13" HD Tri-color or mono display
# display = Adafruit_SSD1681(200, 200,        # 1.54" HD Tri-color display
# display = Adafruit_IL91874(176, 264,        # 2.7" Tri-color display
# display = Adafruit_EK79686(176, 264,        # 2.7" Tri-color display
# display = Adafruit_IL0373(152, 152,         # 1.54" Tri-color display
# display = Adafruit_UC8151D(128, 296,        # 2.9" mono flexible display
# display = Adafruit_IL0373(128, 296,         # 2.9" Tri-color display
# display = Adafruit_IL0398(400, 300,         # 4.2" Tri-color display
display = Adafruit_IL0373(
    104,
    212,  # 2.13" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

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

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

display.rotation = 1

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

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

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

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

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

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

```
)

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

Just like in the last example, the imports are done, but this time include the `ImageDraw` and `ImageFont` Pillow modules to draw some text this time.

```
import digitalio
import busio
import board
from PIL import Image, ImageDraw, ImageFont
from adafruit_epd.il0373 import Adafruit_IL0373
from adafruit_epd.il91874 import Adafruit_IL91874
from adafruit_epd.il0398 import Adafruit_IL0398
from adafruit_epd.ssd1608 import Adafruit_SSD1608
from adafruit_epd.ssd1675 import Adafruit_SSD1675
```

Next define some colors that can be used with Pillow. Since this demo can also run on a Tri-color e-Ink, there are 3 colors defined.

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

After that, the script creates some parameters that are easy to change. If you had a smaller display for instance, you could reduce the `FONTSIZE` and `BORDER` parameter s. The `BORDER` will be the size in pixels of the green border between the edge of the display and the inner purple rectangle. The `FONTSIZE` will be the size of the font in points so that you can adjust it easily for different displays. You could play around with the colors as well.

One thing to note is that on monochrome displays, the RED will show up as BLACK.

```
BORDER = 20
FONTSIZE = 24
BACKGROUND_COLOR = BLACK
FOREGROUND_COLOR = WHITE
TEXT_COLOR = RED
```

After that, the initializer and rotation sections are exactly the same as in the previous example. Go ahead and adjust the EPD initializer as explained in the previous example. After that, the script will create an `image` with the dimensions and use that to create a `draw` object. The `draw` object will have all of the drawing functions.

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

draw = ImageDraw.Draw(image)
```

Next the script clears whatever is on the screen by drawing a rectangle using the `BACKGROUND_COLOR` that takes up the full screen.

```
draw.rectangle((0, 0, display.width, display.height), fill=BACKGROUND_COLOR)
```

Next the script will draw an inner rectangle using the `FOREGROUND_COLOR`. the `BORDER` parameter is used to calculate the size and position of where to draw the rectangle.

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

Next the script will load a TTF font. The `DejaVuSans.ttf` font should come preloaded on your Pi in the location in the code. You also make use of the `FONTSIZE` parameter that we mentioned earlier.

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

Now the script will draw the text Hello World onto the center of the display. You may recognize the centering calculation was the same one we used to center crop the image in the previous example. In this example though, the script will get the font size values using the `getsize()` function of the font object.

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

Finally, just like before, the script will display the image.

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

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

```
python3 epd_pillow_demo.py
```

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

Raspberry Pi E-Ink Weather Station using Python

Raspberry Pi E-Ink Weather Station using Python ()

Raspberry Pi E-Ink Event Calendar using Python

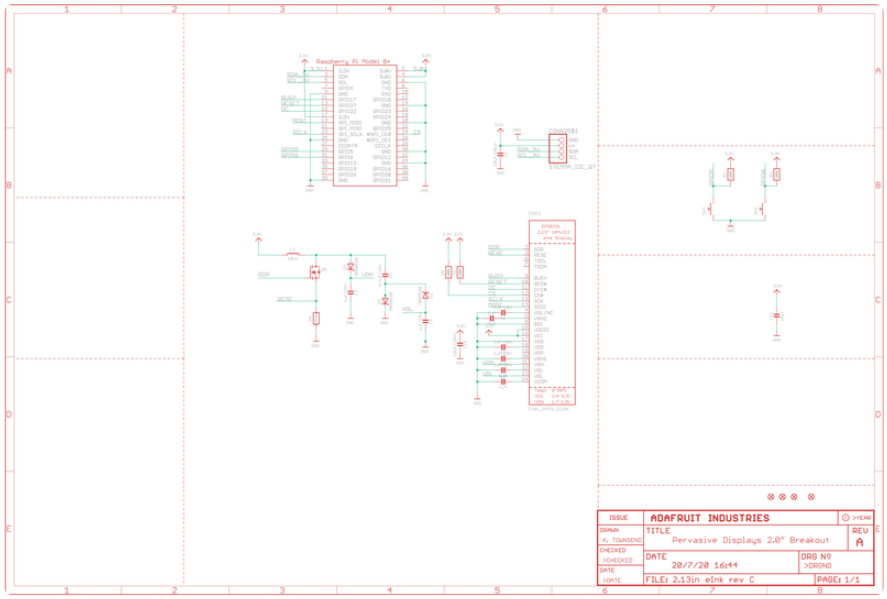
Raspberry Pi E-Ink Event Calendar using Python ()

Downloads

Files

- SSD1675 driver datasheet ()
- EagleCAD files on GitHub ()
- Fritzing object in Adafruit Fritzing Library ()

Schematic



Fab Print