Create an E-Paper Display for Your Raspberry Pi With Python

Difficulty: Intermediate. Step-by-step instructions on how to connect an e-ink display to your Raspberry Pi using Python 3.

Jake Krajewski
Sep 3, 2020 · 11 min read ★

A popular use for e-paper is for ebooks. They have low energy requirements, and are good for eye health. Photo by bady abbas on Unsplash

Why E-ink (or E-paper)?

People like e-ink displays for the same reasons they like regular paper. It has a wide viewing angle, doesn't beam rays of blinding light into your eye sockets, and doesn't consume any electricity to keep a picture on screen. In fact, it only uses electricity when changing the display, meaning images can remain on screen for days on end without consuming any power at all. Since E-Paper is still a developing technology, manufacturers and researchers are continually raising the bar on resolutions, refresh-rates and color capabilities — and as a result, they are only gaining in popularity.

The most widely known example is probably the Kindle e-reader, but in Summer 2020, the Hisense A5 Pro e-ink smartphone was released with **full color video**, which is seriously impressive given all of the limitations. On top of that, there's Norwegian based reMarkable offering their 2nd generation, super responsive e-paper tablet that seems just right for people who prefer the simulated feel of scribbling on dead tree over the "*clackity clack clack"* of a keyboard (or the *"bumpity bump bump"* on a glass screen). If you want to mess around with your own e-ink creations, then read-on. For this tutorial, you'll need:

1. An e-Paper display with a wire harness (try Amazon)
2. A Raspberry Pi (Arduino, and Jetson Nano are supported as well, see the wiki)
3. Python 3 (C is supported as well)

Part I: Find a screen.

Many brands sell screens online, but many of them are for commercial customers only. If you're a maker or tinkerer, Shenzhen based Waveshare has a wide selection of screens from black and white to full color. They start off fairly inexpensive for those smaller than 7 inches, but start to climb in price for every inch of screen real-estate above that. They have both ESP32 and ESP8266 WiFi modules that allow you to take your displays wireless, as well as a variety of NFC-updatable screens, which make your screens not only wireless, **but powerless as well** (the power to update the screen is supplied via the NFC transmitter).

Although the prices are lowest on www.waveshare.com, your order will take longer to arrive if you live in the United States. If you'd like your hardware sooner (for a premium), try another marketplace. This tutorial uses the 2.9" Black and white screen with an 8pin harness. If you choose a differently sized screen, this tutorial still applies.

Part II: Set up your Raspberry Pi

1. Enable the pinout interface.

You need to enable your output pins on your pi-board if you haven't already. This is fairly easy to do from the pi command line, type:

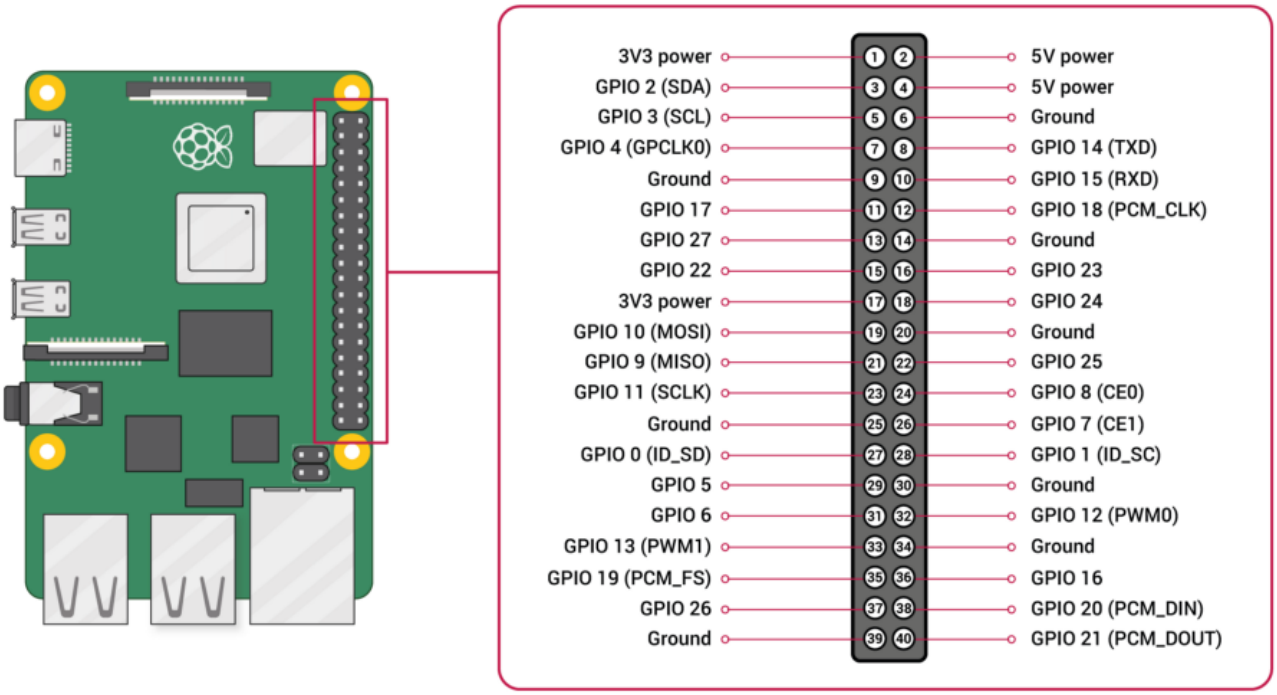
```
sudo raspi-config
```

You'll see a menu appear on the screen. Choose **Interfacing Options** > **SPI** > **Yes** to enable SPI interface and then reboot with `sudo reboot`

2. Connect your pins.

Now you'll be able to connect your screen to your pi. Let's go over the pinout really quickly because it's not intuitive. You basically have 40 pins on your pi that are not universally identified as 1–40. There are many different ways of naming them. For example, GPIO 0 is at pin 27, GPIO 1 is at pin 28, GPIO 2 is at pin 3… And so on and so forth. I'm sure it makes sense to some experts, but not to us common-folk. But we deal.

From RaspberryPi.org:

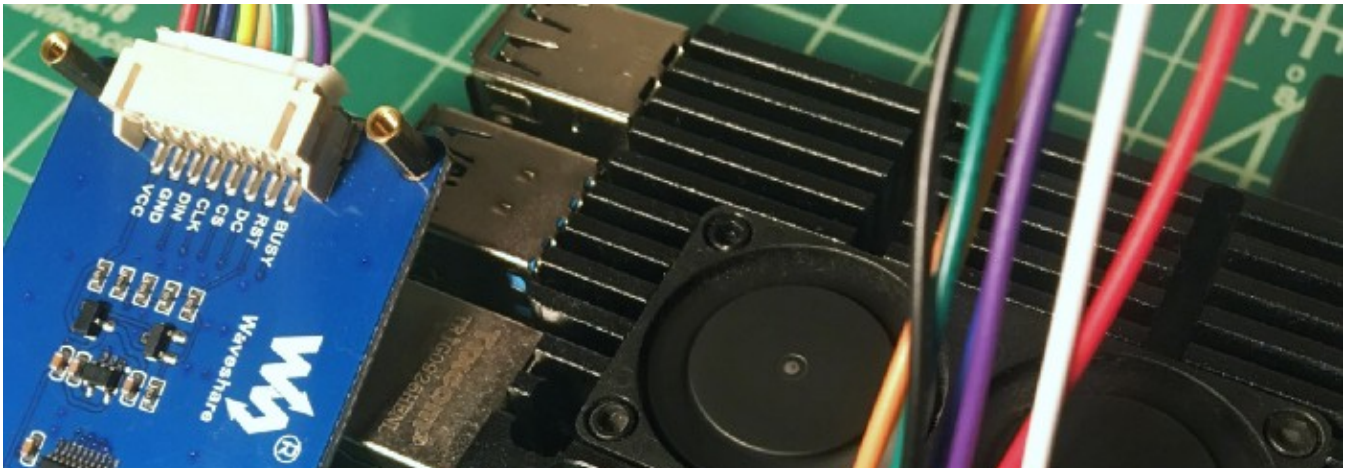


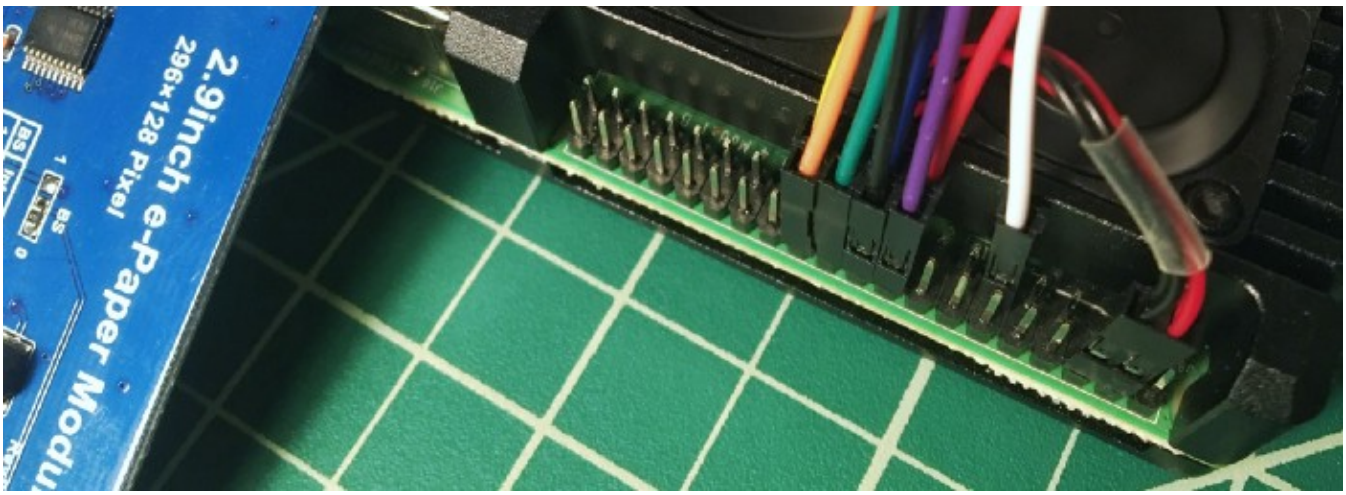
Raspberry pi pinout. https://www.raspberrypi.org/documentation/usage/gpio/

Your screen should have 8 wires that connect it to the Raspberry Pi. Those wires provide power to the screen, etc. For this tutorial, you basically don't need to worry about what they are or what they do, but if you want to know more, they're all described in the manual starting on page 9. What you *do* need to care about is connecting them to the right pins. Here is how they should be hooked up, use either the pin number in bold, or the GPIO labels, but don't mix, especially when we require both *GPIO 24* and **pin 24** which are completely different pins.

> (Board on the left to the Raspberry pi on the right)
>
> 1. VCC *(3.3V/5V power)*: → **pin 17** *(3.3V pin)*
> 2. GND *(Ground)*: → **pin 20** *(Ground pin)*
> 3. DIN *(Data in)*: → **pin 19** *(GPIO 10 / MOSI)*
> 4. CLK *(Clock)*: → **pin 23** *(GPIO 11 / SCLK)*
> 5. CS *(Chip selection)*: → **pin 24** *(GPIO 8 / CE0)*
> 6. DC *(Data/Command selection)*: → **pin 22** *(GPIO 25)*
> 7. RST *(Reset)*: → **pin 11** *(GPIO 17)*
> 8. BUSY: → **pin 18** *(GPIO24)*

Just connect the wires according to the list above. If you did everything correctly, you'll have a cluster of pins from 17–24 on your Pi, with the white reset wire outlying on pin 11. It should look like this (ignore the fan wires to the far right):





Here you can see the back-side of the e-paper display, along with the names of each wire across the upper edge. You might be able to match the wires by color, if you can see them all. Also take note, the black and red wires to the right are for the dual fan system and are hooked up to a 5V and ground pin. They are not part of this project, but they came with the cool case. Link to case on Amazon.

3. Install drivers on your Pi.

We need to install a couple of things to make this work. This is from the wiki, I'm not exactly sure why they chose to install some of these this way, but it will also work if you use pip to install numpy or Pillow, etc.

```
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install python3-pil
sudo apt-get install python3-numpy
sudo pip3 install RPi.GPIO
sudo pip3 install spidev
```

> ***Troubleshooting:*** *If you have any trouble at all after this point, I recommend jumping to the manual on page 15.*
>
> *I had this issue described there on my fresh install of Pi OS (formerly Raspbian): "If you get the error while installing Pillow:ImportError:* ***libopenjp2.so.7: cannot open shared object file: No such file or directory.Please install libopenjp2–7-devwith the command:*** `sudo apt-get install libopenjp2-7-dev` *then try again."*
>
> *If you get this or something similar, it won't hurt to take a look.*

Part III: Run the demo

Assuming everything worked well (that's a big assumption), you'll want to jump into the demo corresponding with your screen size and color ability.

1. Go ahead and create a directory to clone the repository in. I created a folder called `~/Projects/e-Paper/` and made it into an empty repository with `git init`.

2. Next, clone the demo repository from github with:

```
sudo git clone https://github.com/waveshare/e-Paper
# This url is accurate as of 09/2020.
# check wiki if this is not working (link provided above).

cd e-Paper/RaspberryPi\&JetsonNano/
```

3. You'll see both `c` and `python` directories. For this tutorial, type `cd python` to enter the python directory.

4. There's a `readme_rpi_EN.txt` you can take a look at. It covers some of the basics covered in this tutorial and any relevant notes. Type `sudo nano readme_rpi_EN.txt` to view it if you like. Otherwise continue to step 5.

5. Enter the examples folder and view contents with:

```
cd examples
ls
```

6. There you'll see a large list of Python demo files. They're prefixed with "`epd_`" (which stands for e-paper display) followed by the size of the screen you acquired. If you used the same screen used in this tutorial, then run the following:

```
python3 epd_2in9_test.py
```

Notice that there are `epd_2in9b_test.py`, `epd_2in9bc_test.py`, `epd_2in9d_test.py` etc. There are several variations, where letters correspond to the different types of screens. For instance a black and white 2.9 inch board would be just `"2in9_test"`, but a black,

white, and red, or a black, white, and gold board would be `"2in9b_test"` or `"2in9bc_test"` respectively.

If your demo worked, it should cycle through a couple of screens, showing text, drawings and images. Feel free to dig through the demo file to figure out how it works.

Part IV: Dig into the code

1. Setup

If you made it this far, let's crack open our own custom file and call it `helloworld.py` . This should be fairly easy.

1. First, from the demo directory, copy the `lib/waveshare_epd` folder into whatever directory your `helloworld.py` file is located. This includes the necessary drivers to update the display.
2. Also, create a `pic/` folder to store any pics you'd like to load. For this demo, we'll put a custom font there too.
3. Import the `epd<your display>` module here, as well as Python Image Library, or PIL to include some basic image and font rendering tools. Additionally, import os for basic path tools, and time for some sleep timers.

```
import os
import time

from lib.waveshare_epd import epd2in9

from PIL import Image, ImageDraw, ImageFont

pic_dir = 'pic' # Points to pic directory .
```

2. Initialize the display

Initialize the display with a try, except. Refreshing your screen can be done in full mode or partial mode. The `lut_full_update` argument fully refreshes all the pixels on the screen during each refresh. The alternative `lut_partial_update` only updates parts

of the screen that are changing. You'll notice a big difference: on full update, the entire screen flashes, while on partial update, only the new pixels change. The only downside to partial update is that it leaves artifacts on the disply after a while, and a full refresh will always eventually be needed to keep the display clear and crisp.

Here's the initialization code:

```
try:
    # Display init, clear
    display = epd2in9.EPD()
    display.init(display.lut_full_update)
    display.Clear(0) # 0: Black, 255: White

    w = display.height
    h = display.width
    print('width:', w)
    print('height:', h)

    ### ... IMAGE CODE ... ###

except IOError as e:
    print(e)
```

> ***Note**: I reversed w and h intentionally. This is because by default, they have it arranged as short side is width, and long side is height. I wanted to not have to deal with that nuance so I simply changed it up front.*

3. Defining text styles

If you want to write anything, define the font you would like to use. I copied *AvenirNext.ttc* into my folder for this demo, but use any font you like. There are a ton of free fonts here. Use the `truetype` method of `ImageFont`, and just point it to your font file that you placed inside `pic_dir`.

```
body = ImageFont.truetype(os.path.join(pic_dir, 'AvenirNext.ttc'),
18, index=5)
```

Specify the desired font size in pixels (I used `size=18`), and then specify a variation of the font with `index=5` (this is optional). For example, if your font contains more than one font weight, like regular, bold, italic, etc., `index=` lets you choose one based on the order it is listed in the font file itself. See below:

Variations of Avenir Next

```
"""Index of Avenir Next font:
    0 Medium
    1 Regular
    2 Ultra Light
    3 Italic
    4 Medium Italic
    5 Ultra Light Italic
    6 Bold
    7 Demi Bold
    8 Heavy
    9 Bold Italic
    10 Demi Bold Italic
    11 Heavy Italic"""
```

4. Create new image to push to the screen.

In order to display anything on the screen, create a new image with `Image.new`. Place this code in place of `### … IMAGE CODE … ###` in the block above.

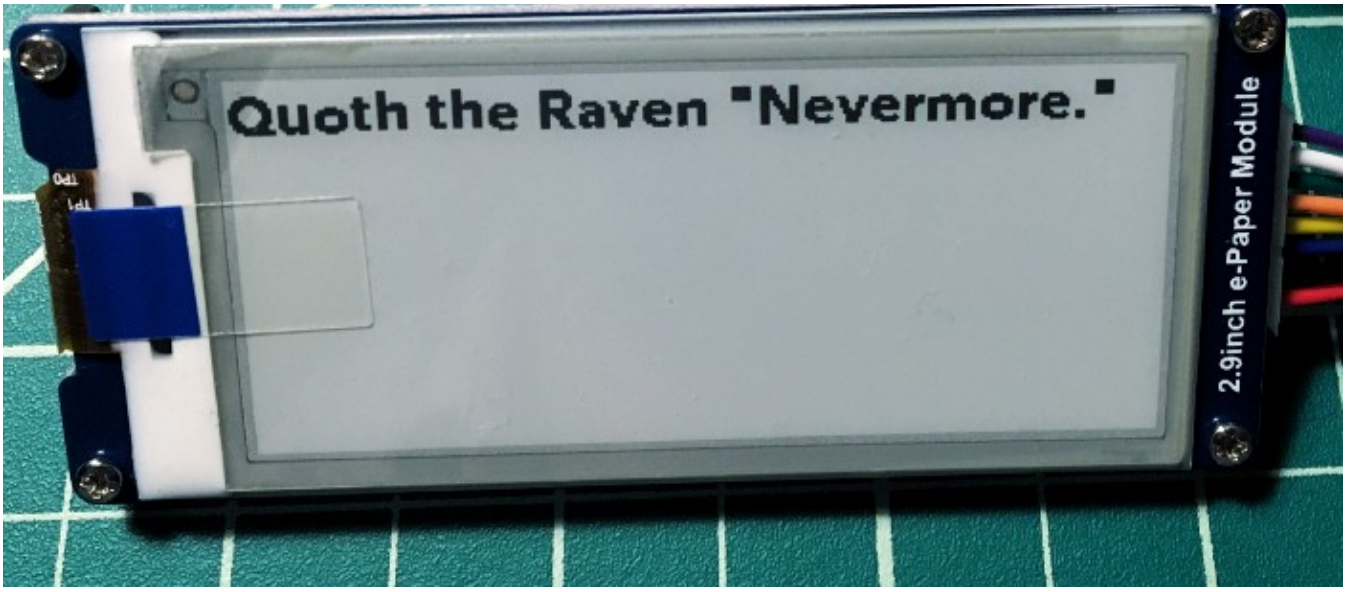
```
image = Image.new(mode='1', size=(w, h), color=255)
draw = ImageDraw.Draw(image)

draw.text((0, 0), 'Quoth the Raven "Nevermore."',
          font=body, fill=0, align='left')

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

`mode=` refers to the the display mode. In this case 1 refers to "1-bit pixels, black and white, stored with one pixel per byte". The width and height dimensions are placed next, they may be reversed to use the display in a portrait orientation, and 255 is white (0 is black).

The text is added with `draw.text` at position (0, 0) placing it in the top left corner of the image, the text 'Quoth the Raven "Nevermore"' is added to the image buffer at our chosen font, size and style, and then with `display.display(display.getbuffer(image))` the image is printed to the screen.



This is our version of Hello World.

Using these tools, you can place any text you want in any position you want, in any font you want, in any style you want. Pretty powerful isn't it? What about images though…

5. Add images

Use a lossless image file (uncompressed). A BMP or PNG should be good. I'll use these two:

296 × 128 px sr71.png

296 × 128 px sunset.png

Put them in your pic folder. and Add the following lines after your text:

```
# Show SR-71 image.
time.sleep(3) # Pause for 3 seconds.
```

```
blackbird = Image.open('pic/sr71.png')

image.paste(blackbird, (0, 0))
display.display(display.getbuffer(image)) # Update display

# Show sunset image.
time.sleep(3)
sunset = Image.open('pic/sunset.png')

image.paste(sunset, (0, 0))
display.display(display.getbuffer(image)) # Update display
```

After the text is printed, the screen will sit for 3 seconds. The image is added to the buffer, and by calling `display.display(display.getbuffer(image))` again, the entire screen is once again refreshed. And it will repeat for the sunset image.

1-bit sunset anyone? the PIL library automatically handles shading for us, and the result is pretty cool.

If you want to leave your image like that, it will stay permanently. Although the manual recommends to refresh at least once every 10 days or damage *might* occur. I'm not exactly sure what could happen, maybe it would result in some image burn? I've never tried it out. But now you can display whatever you want on your e-ink displays. Whether it's a really tiny novel, or a pricetag, or a stock label, I've given you the information you need to get cracking…

That's it really!

I've shown you how to

- enable SPI on your Raspberry Pi
- connect the wires correctly
- install the necessary libraries
- initialize your screen
- full refresh and clear screen
- define a basic font, size, and style
- type a basic message
- add an image
- add a delay

Further challenges

As an added challenge, try making a clock that updates every second. Change `lut_full_update` to `lut_partial_update` and see how it works. It should only update the part that is different, and you'll avoid a full screen flash on update. You'll see why it's imperfect though after you run it a few revolutions. The artifacts will start to show, so use at your own discretion. I'll often partial update for a few repetitions, and then do a full update every 10th rotation just to clear it out and start fresh.

If you're interested, I'm currently working on a 7" display connected to a wifi board, as well as an NFC updated powerless screen. Let me know if you'd like to know more about those projects. Until then —

Happy making!

Disclaimer: I am not affiliated with any of the companies mentioned in this article. They just happen to be the products I bought and used myself.

Raspberry Pi Makers Hardware Technology Python