# Python Radio 35: You’ve got mail…

[Simon Quellen Field](https://medium.com/@simon.field_37276?source=post_page---byline--4006e3b05a0a---------------------------------------)

[Simon Quellen Field](https://medium.com/@simon.field_37276?source=post_page---byline--4006e3b05a0a---------------------------------------)

Follow

11 min read

·

Mar 18, 2025

Listen

Share

More

Get a Wi-Fi notification of snail-mail

Press enter or click to view image in full size

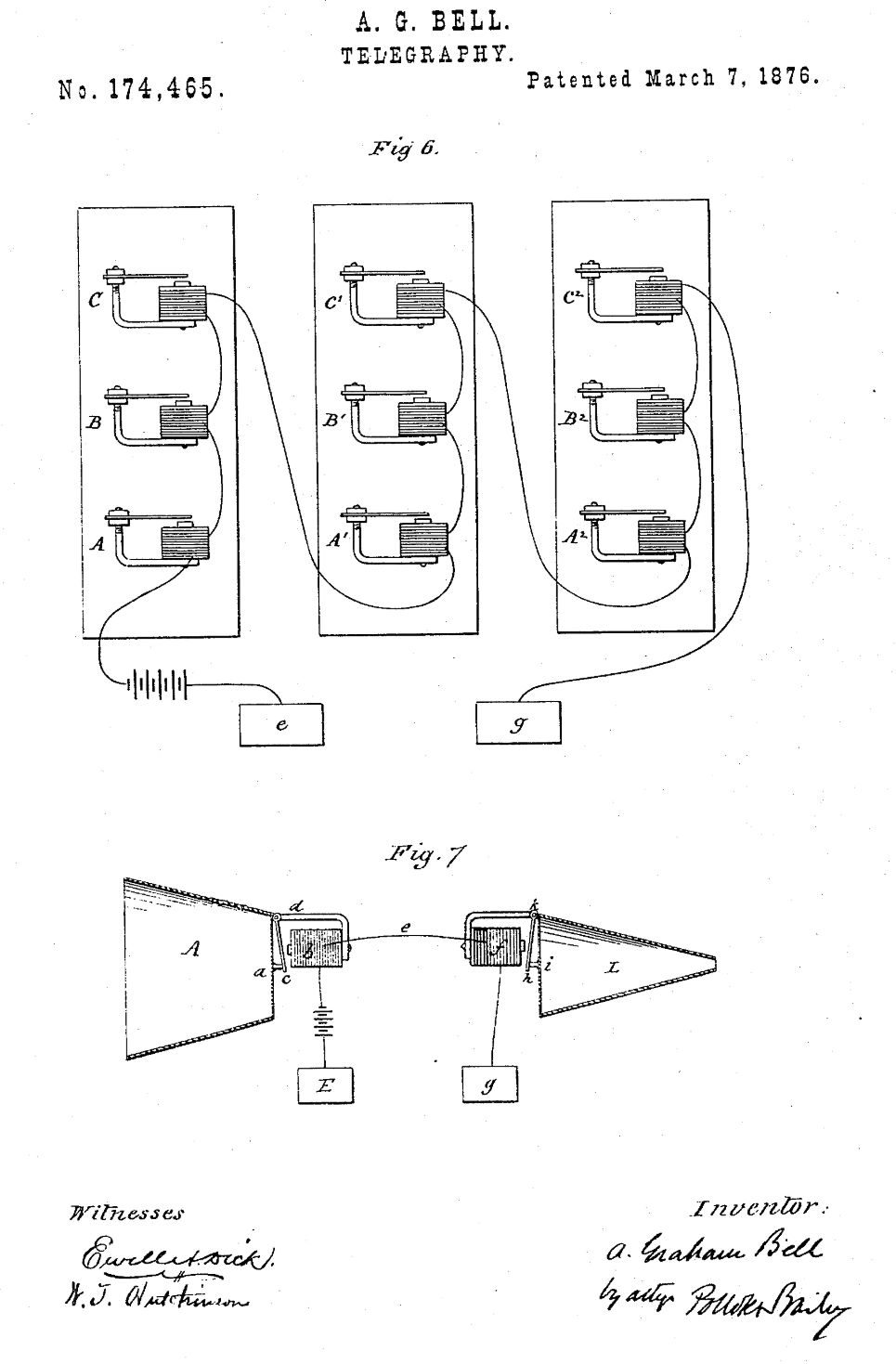


Image of A.G. Bell’s telephone patent

In 1876, Alexander Graham Bell patented his system for sending voice information over a wire.

One wire. The “return path” was the earth.

The earth acts like a great big bucket of electrons. It can both source and “sink” any amount of current. So while the current in the wire carried the signal, the earth at each end acted like a spring, able to push or pull electrons in either direction.

The transmitter and receiver used the same design. An electromagnet sat next to a diaphragm that had a bit of metal attached. As the metal moved back and forth in the magnetic field, it induced a signal in the coil of the electromagnet.

When this signal reached the receiver, the magnetic field in the coil varied with the sound, moving the metal and thus the diaphragm, producing sound.

These days, we have speakers that operate on a similar principle. A thin coil of wire is attached to a diaphragm. As the coil moves back and forth next to a permanent magnet, an electric signal is produced.

If we connect two speakers together with a pair of long wires, one person can talk into one speaker, and another person can hear the voice in the second speaker. Just like the tin-can telephones we made as kids.

### Using a speaker as a microphone

We can use this trick to make our tiny computers sensitive to sound.

We connect one speaker terminal to the ground pin of the microcontroller.

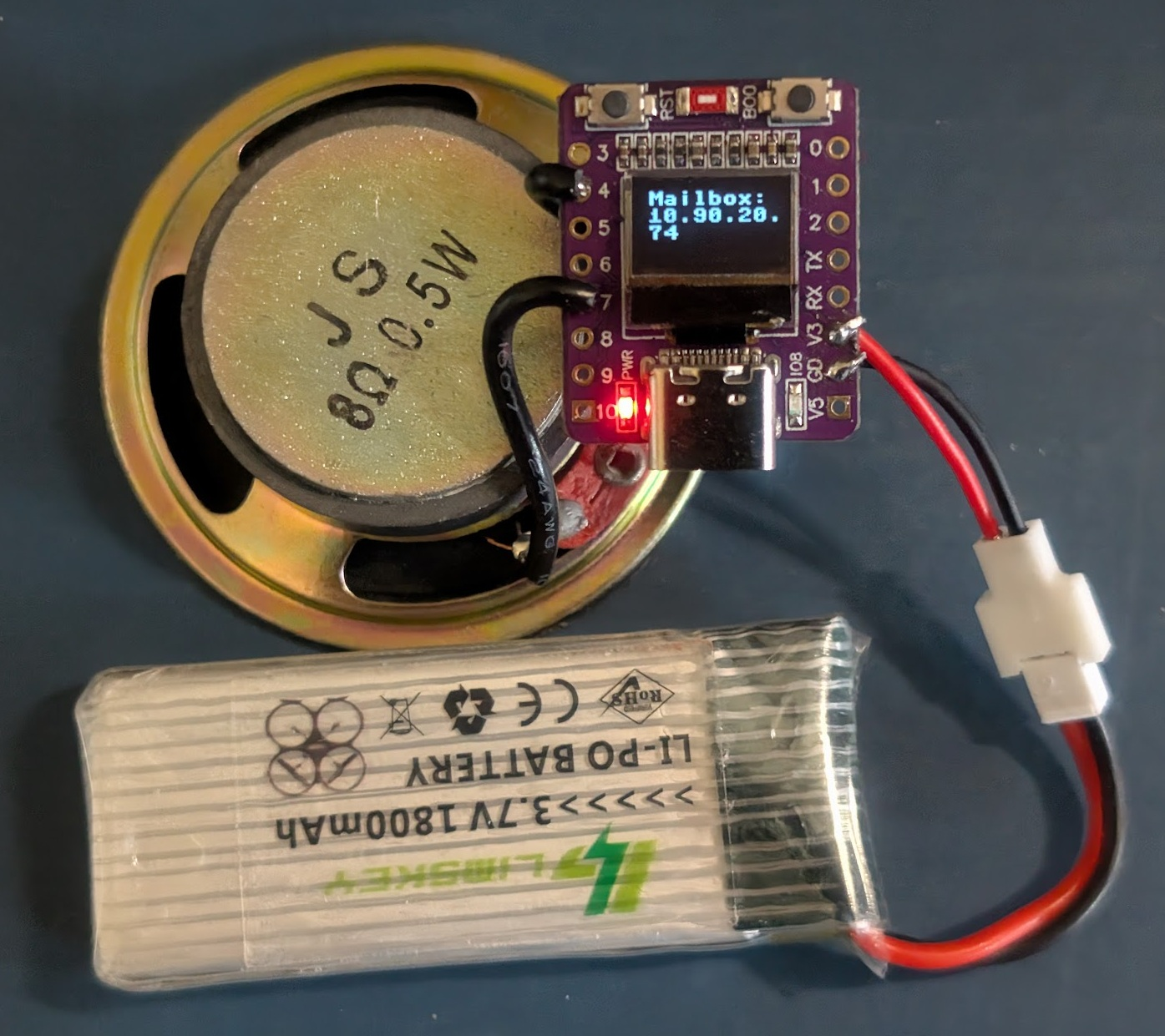
We connect the other pin to an analog input pin.

That’s it. Except for a bit of code.

### The microcontroller.

For this project, I chose the ESP32-C3 Supermini. Mainly because it is so darn cute.

Press enter or click to view image in full size



The ESP32-C3 Supermini

The little board is tiny. The size of my thumbnail. You can see the USB-C connector on it for a size reference. Those holes for the pins are 1/10th of an inch apart. The speaker is 2 inches across (50 millimeters).

That tiny screen is 0.42 inches diagonally (10.66 mm). It can display 9 characters across four lines.

### Why on earth did you do that?

I live on a 20-acre hobby farm called The Lakeview Birdfarm. We have a lot of birds. Including an emu.

But all that space means that my mailbox is 600 feet from the main house. That’s 2 American football fields. Almost two if we’re talking about the football the rest of the world thinks of.

Out here in the mountains, the mail delivery has no reliable schedule. I thought it would be nice to know when my latest parts for my next project have arrived, so I can dive right in and solder something up.

The mailbox is a big steel box with a door that slams itself shut, making a nice loud noise. So a computer that can radio home when it hears a loud noise would be a nice thing to park on top of the mailbox.

### The code in the Supermini

from machine import Pin, I2C, ADC  
from SH1106 import SH1106\_I2C  
import urequests as requests  
from network import WLAN, STA\_IF  
from time import sleep  
  
class Display:  
  
 def \_\_init\_\_(self):  
 self.i2c\_display = I2C(0, sda=Pin(5), scl=Pin(6), freq=400\_000)  
 self.display = SH1106\_I2C(128, 64, self.i2c\_display, rotate=180)  
 self.display.contrast(255)  
  
 BufferWidth, BufferHeight = 128, 64  
 ScreenWidth, ScreenHeight = 72, 40  
 self.xOffset, self.yOffset = (BufferWidth - ScreenWidth) // 2, (BufferHeight - ScreenHeight) // 2  
  
 def oled(self, s, line, column):  
 self.display.fill\_rect(self.xOffset, self.yOffset + 2 + line \* 9, 128-self.xOffset, 9, 0)  
 self.display.text(s, self.xOffset + 2 + column \* 8, self.yOffset + 2 + line \* 9, 1)  
 self.display.show()  
  
d = Display()  
  
sensor = ADC(4)  
gnd = Pin(7, Pin.OUT)  
led = Pin(8, Pin.OUT)  
  
gnd.value(0)  
led.value(0)  
  
station = WLAN(STA\_IF)  
  
station.active(True)  
station.connect("BirdfarmOffice2", "")  
  
def noise():  
 print("Hit!")  
 while True:  
 try:  
 requests.get(url="http://10.90.20.10:8143/")  
 break  
 except Exception as e:  
 print(e)  
  
 led.value(0)  
 sleep(.2)  
 led.value(1)  
 sleep(.2)  
 led.value(0)  
 sleep(.2)  
 led.value(1)  
 sleep(.2)  
  
def main():  
 while station.isconnected() == False:  
 pass  
  
 d.oled("Mailbox:", 0, 0)  
 ip = station.ifconfig()[0]  
 gateway = station.ifconfig()[2]  
 print(station.ifconfig())  
 last = ip[9:]  
 d.oled(ip, 1, 0)  
 d.oled(last, 2, 0)  
 print(ip)  
  
 old\_microvolts = 0  
 while True:  
 microvolts = sensor.read\_uv()  
 if microvolts > 50000:  
 print(microvolts)  
 noise()  
 old\_microvolts = microvolts  
  
main()

The display on the Supermini uses the SH1106 driver. I’ll add that later without discussion, but it is available on GitHub.

Our Display class handles all the peculiarities of the display. And it is definitely peculiar. The visible part of the display is a rectangle inside the frame buffer of the driver. The oled() method handles all the arithmetic to deal with that. It takes a string, a line, and a column as arguments.

We connect the speaker between pins 4 and 7. Pin 7 we will set at ground. Pin 4 is the ADC pin that will listen to the speaker. I didn’t need pin 7 for anything, and it was more convenient than soldering to the GND pin (which I planned to use for the battery).

Next, we set up the Wi-Fi connection. We have excellent Wi-Fi on the farm, so the mailbox has good coverage. The SSID is BirdfarmOffice2, and there is no password. We let anyone passing by use our 5 gigabit Internet connection. Our neighbours are a long way away.

The server in my office has the IP address 10.90.20.10. We will run a Python server on that machine listening on port 8143.

When the Supermini hears a noise, it prints Hit! (although nobody will be reading the serial port). Then it connects to the server using requests.get().

We don’t need to send any data, we just want to connect. The server will know that a connection means there has been a noise.

We also flash the blue LED, so I can tell if the noise was loud enough without running 600 feet back to my office to look at the server output.

The main routine waits for a Wi-Fi connection, then prints Mailbox on the screen, along with the IP address it has been given. The address is longer than 9 characters, so I had to split it into two lines.

Then main() listens to the speaker. The read\_uv() method returns microvolts, even though the last three digits are always zero, so it could have been millivolts.

Here is where you tune the volume we need to hear to decide to send an alert. By dropping the whole project on the desktop from about half an inch up, I chose 50,000 microvolts. I may change that if I am getting to many or too few alerts.

### The Server

The server side is a little simpler because the Microdot module does all the heavy lifting. You install it using “pip install microdot”.

from microdot import Microdot  
from datetime import datetime  
  
app = Microdot()  
  
@app.route('/')  
async def hit(r):  
 print(datetime.now(), "Device at", r.client\_addr[0], "heard a loud noise. You may have mail.")  
 return "OK"  
  
app.run(port=8143)

If a connection comes in on port 8143 it prints a message with a timestamp.

The timestamp lets us decide if the alarm is the mail or some other event that made a loud noise. We could add some checks to see if the alert is in the time window that makes sense for the mail, but I think it would be fun to know how many times a loud noise happened out there.

The output looks like this:

C:\simon\mailbox\_alarm>server.py  
2025-03-16 13:35:06.311245 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 13:35:12.346001 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 13:35:15.622433 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 13:42:19.284935 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 15:23:26.683053 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 15:50:36.045345 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:23:06.027225 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:23:09.308156 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:23:34.948135 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:23:37.562645 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:23:44.837076 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:23:46.984213 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-16 17:26:10.905210 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 09:24:51.373658 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 13:08:23.083649 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:25:10.423883 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:54:01.483339 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:54:39.730194 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:55:16.245313 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:55:39.365604 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:55:59.530717 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 16:56:01.336591 Device at 10.90.20.74 heard a loud noise. You may have mail.  
2025-03-17 17:10:04.631543 Device at 10.90.20.74 heard a loud noise. You may have mail.

The gadget has been running for a couple of days on the 1800 milliampere battery. I could run it from a USB-C charger, but my long-term plan is to run it on solar power. I am just waiting for my solar battery charger boards to arrive. With a big enough solar panel, I wouldn’t need a battery, since it would be enough to run the gadget on a cloudy day, and the mail doesn’t arrive at night.

I promised the code for the SS1106 driver:

#  
# MicroPython SH1106 OLED driver, I2C and SPI interfaces  
#  
# The MIT License (MIT)  
#  
# Copyright (c) 2016 Radomir Dopieralski (@deshipu),  
# 2017-2021 Robert Hammelrath (@robert-hh)  
# 2021 Tim Weber (@scy)  
#  
# Permission is hereby granted, free of charge, to any person obtaining a copy  
# of this software and associated documentation files (the "Software"), to deal  
# in the Software without restriction, including without limitation the rights  
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
# copies of the Software, and to permit persons to whom the Software is  
# furnished to do so, subject to the following conditions:  
#  
# The above copyright notice and this permission notice shall be included in  
# all copies or substantial portions of the Software.  
#  
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,  
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE  
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN  
# THE SOFTWARE.  
#  
# Sample code sections for ESP8266 pin assignments  
# ------------ SPI ------------------  
# Pin Map SPI  
# - 3v - xxxxxx - Vcc  
# - G - xxxxxx - Gnd  
# - D7 - GPIO 13 - Din / MOSI fixed  
# - D5 - GPIO 14 - Clk / Sck fixed  
# - D8 - GPIO 4 - CS (optional, if the only connected device)  
# - D2 - GPIO 5 - D/C  
# - D1 - GPIO 2 - Res  
#  
# for CS, D/C and Res other ports may be chosen.  
#  
# from machine import Pin, SPI  
# import sh1106  
  
# spi = SPI(1, baudrate=1000000)  
# display = sh1106.SH1106\_SPI(128, 64, spi, Pin(5), Pin(2), Pin(4))  
# display.sleep(False)  
# display.fill(0)  
# display.text('Testing 1', 0, 0, 1)  
# display.show()  
#  
# --------------- I2C ------------------  
#  
# Pin Map I2C  
# - 3v - xxxxxx - Vcc  
# - G - xxxxxx - Gnd  
# - D2 - GPIO 5 - SCK / SCL  
# - D1 - GPIO 4 - DIN / SDA  
# - D0 - GPIO 16 - Res  
# - G - xxxxxx CS  
# - G - xxxxxx D/C  
#  
# Pin's for I2C can be set almost arbitrary  
#  
# from machine import Pin, I2C  
# import sh1106  
#  
# i2c = I2C(scl=Pin(5), sda=Pin(4), freq=400000)  
# display = sh1106.SH1106\_I2C(128, 64, i2c, Pin(16), 0x3c)  
# display.sleep(False)  
# display.fill(0)  
# display.text('Testing 1', 0, 0, 1)  
# display.show()  
  
from micropython import const  
import utime as time  
import framebuf  
  
  
# a few register definitions  
\_SET\_CONTRAST = const(0x81)  
\_SET\_NORM\_INV = const(0xa6)  
\_SET\_DISP = const(0xae)  
\_SET\_SCAN\_DIR = const(0xc0)  
\_SET\_SEG\_REMAP = const(0xa0)  
\_LOW\_COLUMN\_ADDRESS = const(0x00)  
\_HIGH\_COLUMN\_ADDRESS = const(0x10)  
\_SET\_PAGE\_ADDRESS = const(0xB0)  
  
  
class SH1106(framebuf.FrameBuffer):  
  
 def \_\_init\_\_(self, width, height, external\_vcc, rotate=0):  
 self.width = width  
 self.height = height  
 self.external\_vcc = external\_vcc  
 self.flip\_en = rotate == 180 or rotate == 270  
 self.rotate90 = rotate == 90 or rotate == 270  
 self.pages = self.height // 8  
 self.bufsize = self.pages \* self.width  
 self.renderbuf = bytearray(self.bufsize)  
 self.pages\_to\_update = 0  
  
 if self.rotate90:  
 self.displaybuf = bytearray(self.bufsize)  
 # HMSB is required to keep the bit order in the render buffer  
 # compatible with byte-for-byte remapping to the display buffer,  
 # which is in VLSB. Else we'd have to copy bit-by-bit!  
 super().\_\_init\_\_(self.renderbuf, self.height, self.width,  
 framebuf.MONO\_HMSB)  
 else:  
 self.displaybuf = self.renderbuf  
 super().\_\_init\_\_(self.renderbuf, self.width, self.height,  
 framebuf.MONO\_VLSB)  
  
 # flip() was called rotate() once, provide backwards compatibility.  
 self.rotate = self.flip  
 self.init\_display()  
  
 def init\_display(self):  
 self.reset()  
 self.fill(0)  
 self.show()  
 self.poweron()  
 # rotate90 requires a call to flip() for setting up.  
 self.flip(self.flip\_en)  
  
 def poweroff(self):  
 self.write\_cmd(\_SET\_DISP | 0x00)  
  
 def poweron(self):  
 self.write\_cmd(\_SET\_DISP | 0x01)  
 if self.delay:  
 time.sleep\_ms(self.delay)  
  
 def flip(self, flag=None, update=True):  
 if flag is None:  
 flag = not self.flip\_en  
 mir\_v = flag ^ self.rotate90  
 mir\_h = flag  
 self.write\_cmd(\_SET\_SEG\_REMAP | (0x01 if mir\_v else 0x00))  
 self.write\_cmd(\_SET\_SCAN\_DIR | (0x08 if mir\_h else 0x00))  
 self.flip\_en = flag  
 if update:  
 self.show(True) # full update  
  
 def sleep(self, value):  
 self.write\_cmd(\_SET\_DISP | (not value))  
  
 def contrast(self, contrast):  
 self.write\_cmd(\_SET\_CONTRAST)  
 self.write\_cmd(contrast)  
  
 def invert(self, invert):  
 self.write\_cmd(\_SET\_NORM\_INV | (invert & 1))  
  
 def show(self, full\_update = False):  
 # self.\* lookups in loops take significant time (~4fps).  
 (w, p, db, rb) = (self.width, self.pages,  
 self.displaybuf, self.renderbuf)  
 if self.rotate90:  
 for i in range(self.bufsize):  
 db[w \* (i % p) + (i // p)] = rb[i]  
 if full\_update:  
 pages\_to\_update = (1 << self.pages) - 1  
 else:  
 pages\_to\_update = self.pages\_to\_update  
 #print("Updating pages: {:08b}".format(pages\_to\_update))  
 for page in range(self.pages):  
 if (pages\_to\_update & (1 << page)):  
 self.write\_cmd(\_SET\_PAGE\_ADDRESS | page)  
 self.write\_cmd(\_LOW\_COLUMN\_ADDRESS | 2)  
 self.write\_cmd(\_HIGH\_COLUMN\_ADDRESS | 0)  
 self.write\_data(db[(w\*page):(w\*page+w)])  
 self.pages\_to\_update = 0  
  
 def pixel(self, x, y, color=None):  
 if color is None:  
 return super().pixel(x, y)  
 else:  
 super().pixel(x, y , color)  
 page = y // 8  
 self.pages\_to\_update |= 1 << page  
  
 def text(self, text, x, y, color=1):  
 super().text(text, x, y, color)  
 self.register\_updates(y, y+7)  
  
 def line(self, x0, y0, x1, y1, color):  
 super().line(x0, y0, x1, y1, color)  
 self.register\_updates(y0, y1)  
  
 def hline(self, x, y, w, color):  
 super().hline(x, y, w, color)  
 self.register\_updates(y)  
  
 def vline(self, x, y, h, color):  
 super().vline(x, y, h, color)  
 self.register\_updates(y, y+h-1)  
  
 def fill(self, color):  
 super().fill(color)  
 self.pages\_to\_update = (1 << self.pages) - 1  
  
 def blit(self, fbuf, x, y, key=-1, palette=None):  
 super().blit(fbuf, x, y, key, palette)  
 self.register\_updates(y, y+self.height)  
  
 def scroll(self, x, y):  
 # my understanding is that scroll() does a full screen change  
 super().scroll(x, y)  
 self.pages\_to\_update = (1 << self.pages) - 1  
  
 def fill\_rect(self, x, y, w, h, color):  
 super().fill\_rect(x, y, w, h, color)  
 self.register\_updates(y, y+h-1)  
  
 def rect(self, x, y, w, h, color):  
 super().rect(x, y, w, h, color)  
 self.register\_updates(y, y+h-1)  
  
 def register\_updates(self, y0, y1=None):  
 # this function takes the top and optional bottom address of the changes made  
 # and updates the pages\_to\_change list with any changed pages  
 # that are not yet on the list  
 start\_page = max(0, y0 // 8)  
 end\_page = max(0, y1 // 8) if y1 is not None else start\_page  
 # rearrange start\_page and end\_page if coordinates were given from bottom to top  
 if start\_page > end\_page:  
 start\_page, end\_page = end\_page, start\_page  
 for page in range(start\_page, end\_page+1):  
 self.pages\_to\_update |= 1 << page  
  
 def reset(self, res):  
 if res is not None:  
 res(1)  
 time.sleep\_ms(1)  
 res(0)  
 time.sleep\_ms(20)  
 res(1)  
 time.sleep\_ms(20)  
  
  
class SH1106\_I2C(SH1106):  
 def \_\_init\_\_(self, width, height, i2c, res=None, addr=0x3c,  
 rotate=0, external\_vcc=False, delay=0):  
 self.i2c = i2c  
 self.addr = addr  
 self.res = res  
 self.temp = bytearray(2)  
 self.delay = delay  
 if res is not None:  
 res.init(res.OUT, value=1)  
 super().\_\_init\_\_(width, height, external\_vcc, rotate)  
  
 def write\_cmd(self, cmd):  
 self.temp[0] = 0x80 # Co=1, D/C#=0  
 self.temp[1] = cmd  
 self.i2c.writeto(self.addr, self.temp)  
  
 def write\_data(self, buf):  
 self.i2c.writeto(self.addr, b'\x40'+buf)  
  
 def reset(self):  
 super().reset(self.res)  
  
  
class SH1106\_SPI(SH1106):  
 def \_\_init\_\_(self, width, height, spi, dc, res=None, cs=None,  
 rotate=0, external\_vcc=False, delay=0):  
 dc.init(dc.OUT, value=0)  
 if res is not None:  
 res.init(res.OUT, value=0)  
 if cs is not None:  
 cs.init(cs.OUT, value=1)  
 self.spi = spi  
 self.dc = dc  
 self.res = res  
 self.cs = cs  
 self.delay = delay  
 super().\_\_init\_\_(width, height, external\_vcc, rotate)  
  
 def write\_cmd(self, cmd):  
 if self.cs is not None:  
 self.cs(1)  
 self.dc(0)  
 self.cs(0)  
 self.spi.write(bytearray([cmd]))  
 self.cs(1)  
 else:  
 self.dc(0)  
 self.spi.write(bytearray([cmd]))  
  
 def write\_data(self, buf):  
 if self.cs is not None:  
 self.cs(1)  
 self.dc(1)  
 self.cs(0)  
 self.spi.write(buf)  
 self.cs(1)  
 else:  
 self.dc(1)  
 self.spi.write(buf)  
  
 def reset(self):  
 super().reset(self.res)

[Python](https://medium.com/tag/python?source=post_page-----4006e3b05a0a---------------------------------------)

[Micropython](https://medium.com/tag/micropython?source=post_page-----4006e3b05a0a---------------------------------------)

[Radio](https://medium.com/tag/radio?source=post_page-----4006e3b05a0a---------------------------------------)