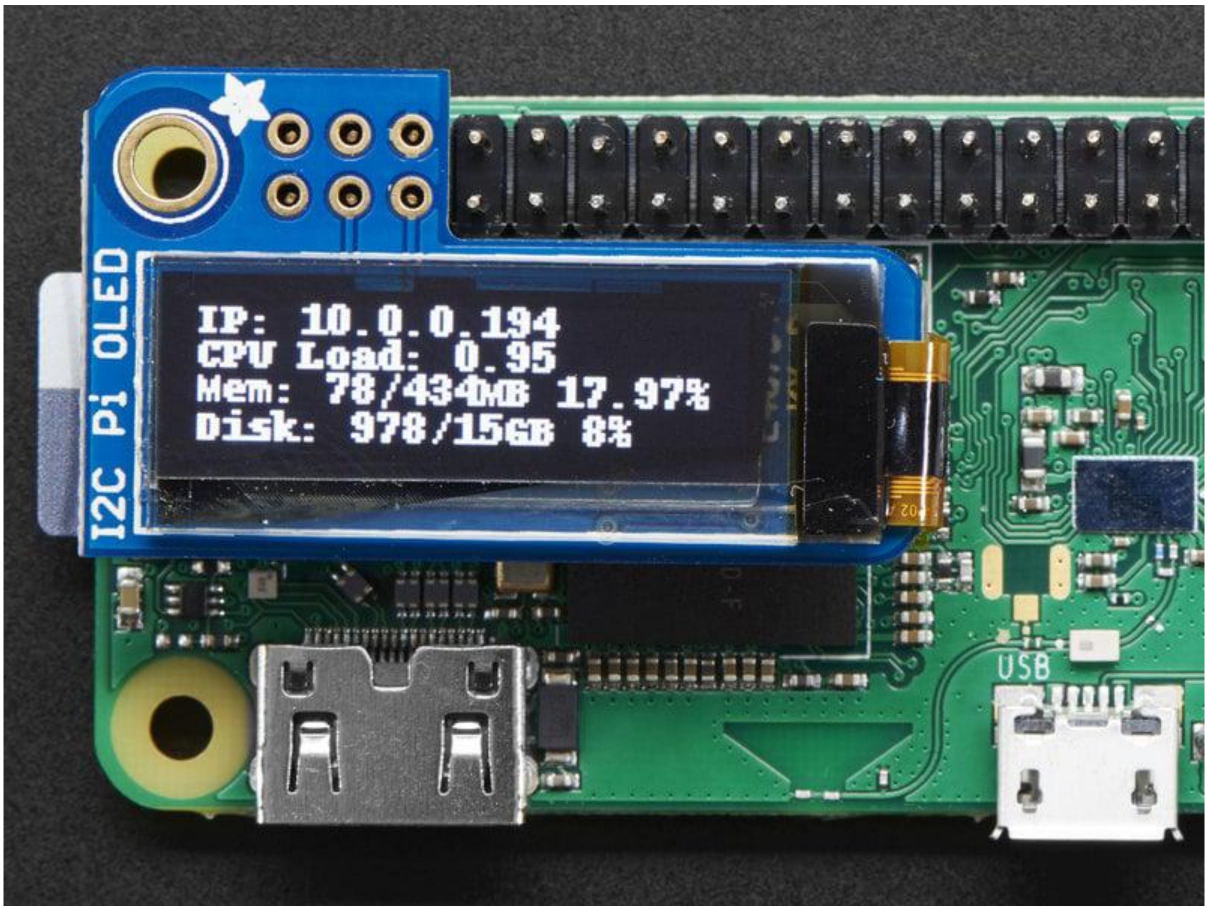
Adafruit PiOLED - 128x32 Mini OLED for Raspberry Pi

Created by lady ada

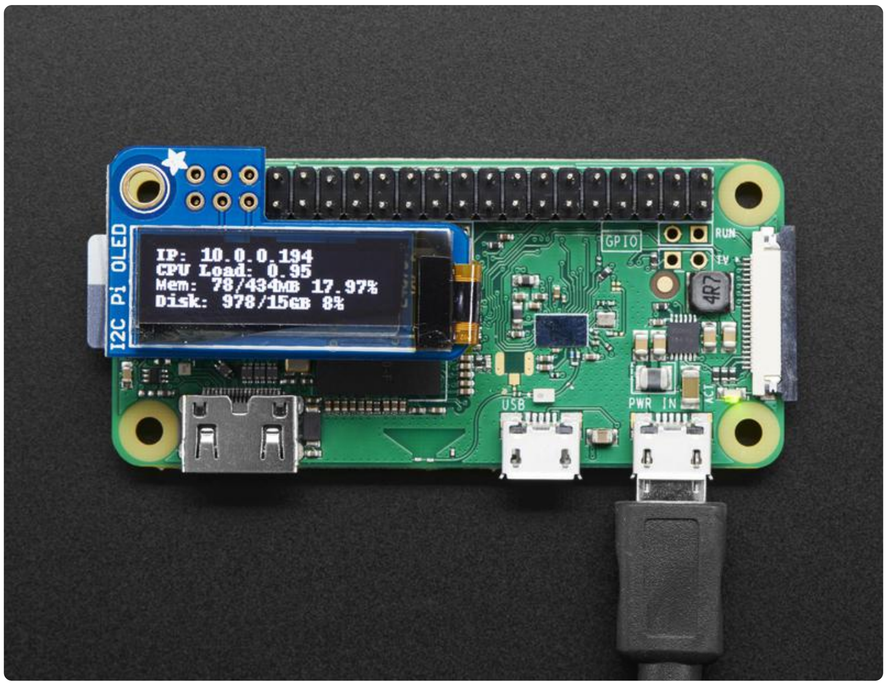


https://learn.adafruit.com/adafruit-pioled-128x32-mini-oled-for-raspberry-pi

Last updated on 2022-12-01 02:56:37 PM EST

Table of Contents

Overview

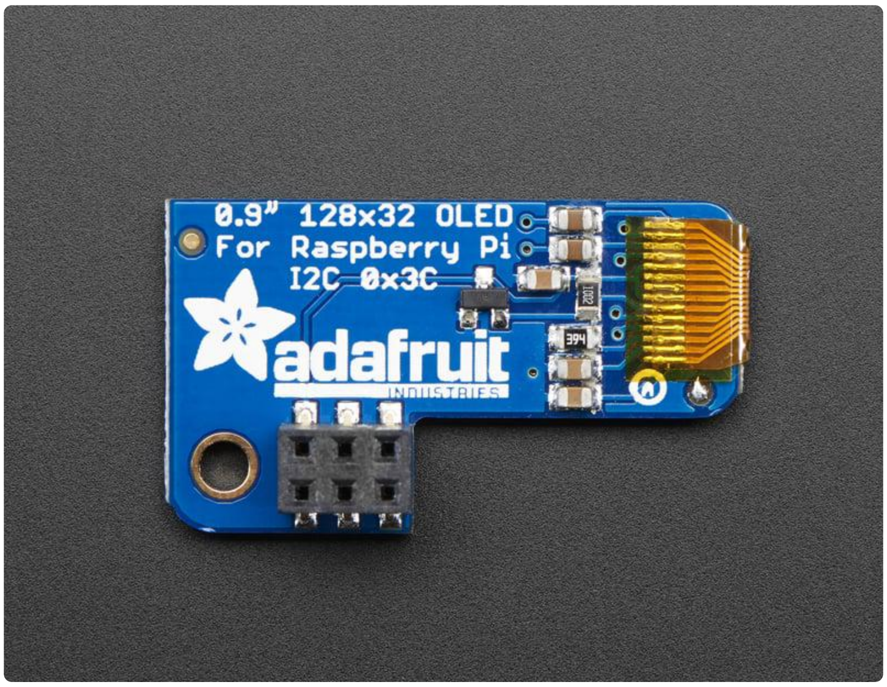


If you're looking for the most compact lil' display for a Raspberry Pi () (most likely a Pi Zero ()) project, this might be just the thing you need!

The Adafruit PiOLED is your little OLED pal, ready to snap onto any and all Raspberry Pi computers, to give you a little display. The PiOLED comes with a monochrome 128x32 OLED, with sharp white pixels. The OLED uses only the I2C pins so you have plenty of GPIO connections available for buttons, LEDs, sensors, etc. It's also nice and compact so it will fit into any case.

These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. This screen is made of 128x32 individual white OLED pixels and because the display makes its own light, no backlight is required. This reduces the power required to run the OLED and is why the display has such high contrast; we really like this miniature display for its crispness!

Using the display is very easy, we have a Python library for the SSD1306 chipset. Our example code allows you to draw images, text, whatever you like, using the Python imaging library. Our tests showed 30 FPS update rates so you can do animations or simple video.



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, including the original Pi 1, B+, Pi 2, Pi 3, Pi 4, and Pi Zero.

Usage

Install CircuitPython

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

CircuitPython Installation Guide

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

```
sudo pip3 install adafruit-circuitpython-ssd1306
```

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

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

We also need PIL to allow 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
```

Enable I2C

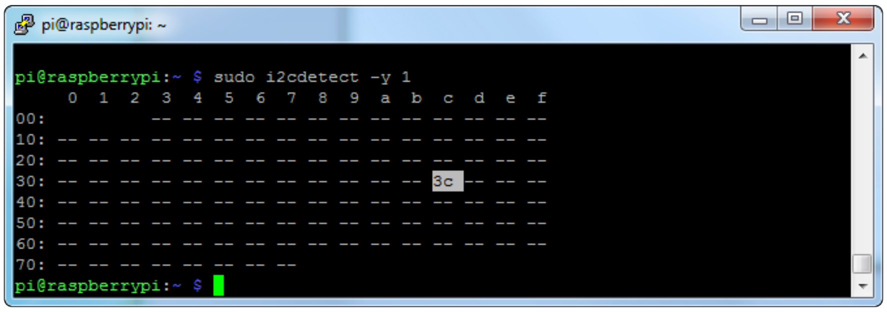
To enable i2c, you can follow our detailed guide on configuring the Pi with I2C support here. ()

After you've enabled I2C you will need to shutdown with `sudo shutdown -h now`

Once the Pi has halted, plug in the PiOLED. Now you can power the Pi back up, and log back in. Run the following command from a terminal prompt to scan/detect the I2C devices

`sudo i2cdetect -y 1`

You should see the following, indicating that address 0x3c (the OLED display) was found



```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

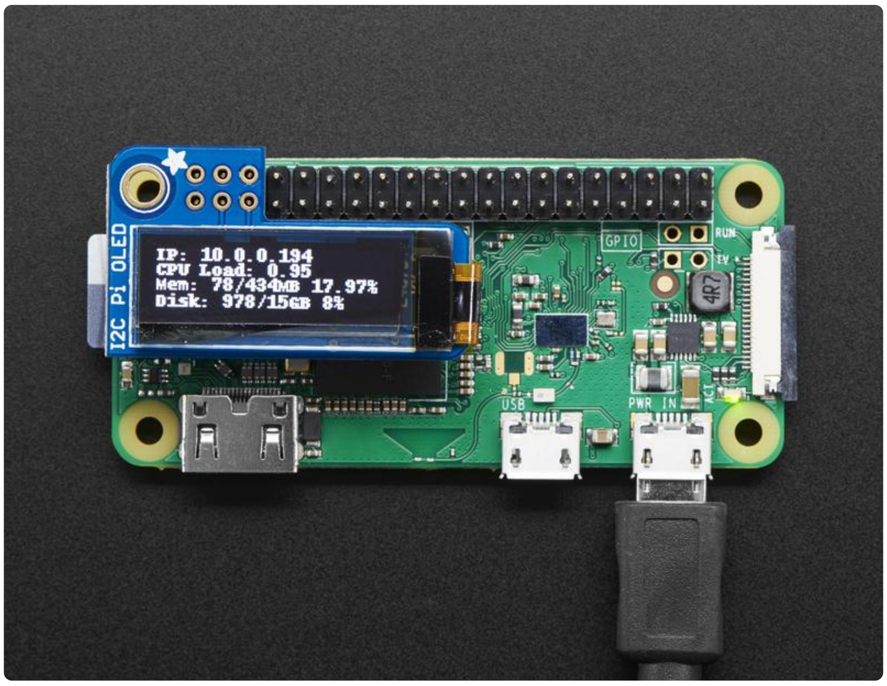
Verify I2C Device

You can run our stats example, which will query the Pi for details on CPU load, disk space, etc. and print it on the OLED.

Create a new file with nano ~pi/stats.py and paste this code below in! Then save it.

```
# SPDX-FileCopyrightText: 2017 Tony DiCola for Adafruit Industries
# SPDX-FileCopyrightText: 2017 James DeVito for Adafruit Industries
# SPDX-License-Identifier: MIT

# This example is for use on (Linux) computers that are using CPython with
# Adafruit Blinka to support CircuitPython libraries. CircuitPython does
# not support PIL/pillow (python imaging library)!

import time
import subprocess

from board import SCL, SDA
import busio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306


# Create the I2C interface.
i2c = busio.I2C(SCL, SDA)

# Create the SSD1306 OLED class.
# The first two parameters are the pixel width and pixel height.  Change these
# to the right size for your display!
disp = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)

# Clear display.
disp.fill(0)
disp.show()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp.width
height = disp.height
image = Image.new("1", (width, height))

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

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=0)

# Draw some shapes.
# First define some constants to allow easy resizing of shapes.
padding = -2
top = padding
bottom = height - padding
# Move left to right keeping track of the current x position for drawing shapes.
x = 0


# Load default font.
font = ImageFont.load_default()

# Alternatively load a TTF font.  Make sure the .ttf font file is in the
# same directory as the python script!
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
# font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', 9)

while True:

    # Draw a black filled box to clear the image.
    draw.rectangle((0, 0, width, height), outline=0, fill=0)

    # Shell scripts for system monitoring from here:
    # https://unix.stackexchange.com/questions/119126/command-to-display-memory-
usage-disk-usage-and-cpu-load
    cmd = "hostname -I | cut -d' ' -f1"
    IP = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = 'cut -f 1 -d " " /proc/loadavg'
    CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\",
$3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = 'df -h | awk \'$NF=="/"{printf "Disk: %d/%d GB  %s", $3,$2,$5}\''
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.

    draw.text((x, top + 0), "IP: " + IP, font=font, fill=255)
    draw.text((x, top + 8), "CPU load: " + CPU, font=font, fill=255)
    draw.text((x, top + 16), MemUsage, font=font, fill=255)
    draw.text((x, top + 25), Disk, font=font, fill=255)

    # Display image.
    disp.image(image)
    disp.show()
    time.sleep(0.1)
```

Run `sudo python3 stats.py` and you should see something like the following image:



```
IP: 10.0.0.194
CPU Load: 0.95
Mem: 78/434MB 17.97%
Disk: 978/15GB 8%
```

Running Stats on Boot

You can pretty easily make it so this handy program runs every time you boot your Pi.

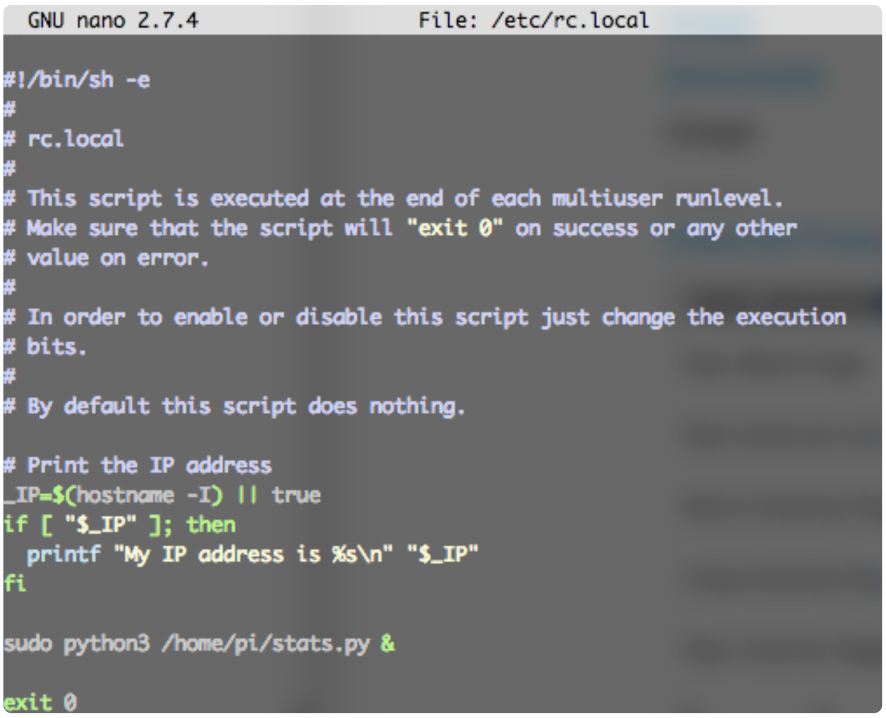
The fastest/easiest way is to put it in /etc/rc.local

Run sudo nano /etc/rc.local and add the line

```
sudo python3 /home/pi/stats.py &
```

on its own line right before exit 0

Then save and exit. Reboot to verify that the screen comes up on boot!



```
  GNU nano 2.7.4                    File: /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

sudo python3 /home/pi/stats.py &

exit 0
```

For more advanced usage, check out our linux system services guide ()

Library Usage

In the examples subdirectory of the Adafruit_CircuitPython_SSD1306 repository (), you'll find examples which demonstrate the usage of the library.

To help you get started, I'll walk you through the stats.py code you ran earlier, that way you can use this file as a basis of a future project.

```
import time
import subprocess

from board import SCL, SDA
import busio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306
```

First, a few Python modules are imported. This includes the `adafruit_ssd1306` module which contains the OLED driver classes.

To draw images, shapes, and text/fonts, the code imports some of the Python Imaging Library's modules like **`Image`**, **`ImageDraw`**, and **`ImageFont`**.

```
# Create the I2C interface.
i2c = busio.I2C(SCL, SDA)

# Create the SSD1306 OLED class.
# The first two parameters are the pixel width and pixel height.  Change these
# to the right size for your display!
disp = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
```

Since the PiOLED is a 128x32 I2C only device, the screen size and the i2c interface are passed to the `adafruit_ssd.SSD1306_I2C` class.

```
# Clear display.
disp.fill(0)
disp.show()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp.width
height = disp.height
image = Image.new('1', (width, height))

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

# Draw a black filled box to clear the image.
draw.rectangle((0, 0, width, height), outline=0, fill=0)
```

The next chunk of code clears the display by inverting its fill with `fill(0)` and then writing to the display with `show()`.

Then it will configure a PIL drawing class to prepare for drawing graphics. Notice that the image buffer is created in 1-bit mode with the `'1'` parameter, this is important because the display only supports black and white colors.

We then re-draw a large black rectangle to clear the screen. In theory we don't have to clear the screen again, but its a good example of how to draw a shape!

```
# Draw some shapes.
# First define some constants to allow easy resizing of shapes.
padding = -2
top = padding
bottom = height-padding
# Move left to right keeping track of the current x position for drawing shapes.
x = 0


# Load default font.
font = ImageFont.load_default()

# Alternatively load a TTF font.  Make sure the .ttf font file is in the
# same directory as the python script!
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
#font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf', 9)
```

Once the display is initialized and a drawing object is prepared, you can draw shapes, text and graphics using PIL's drawing commands (). We'll define some constraints based on the height and width of the display to allow for easy resizing of shapes.

Then, the code loads the `default` font, which works fine, but there's other fonts you can load.

Next the code loads a built-in default font and draws a few lines of text. You can also load your own TrueType font and use it to render fancy text in any style you like

```
while True:

    # Draw a black filled box to clear the image.
    draw.rectangle((0, 0, width, height), outline=0, fill=0)

    # Shell scripts for system monitoring from here:
    # https://unix.stackexchange.com/questions/119126/command-to-display-memory-
usage-disk-usage-and-cpu-load
    cmd = "hostname -I | cut -d\' \' -f1"
    IP = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%s MB  %.2f%%\",
$3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell=True).decode("utf-8")
    cmd = "df -h | awk '$NF==\"/\"{printf \"Disk: %d/%d GB  %s\", $3,$2,$5}'"
    Disk = subprocess.check_output(cmd, shell=True).decode("utf-8")

    # Write four lines of text.

    draw.text((x, top+0), "IP: "+IP, font=font, fill=255)
    draw.text((x, top+8), CPU, font=font, fill=255)
    draw.text((x, top+16), MemUsage, font=font, fill=255)
    draw.text((x, top+25), Disk, font=font, fill=255)

    # Display image.
    disp.image(image)
    disp.show()
    time.sleep(.1)
```

Using the subprocess class, python can utilize linux commands to access the Pi's system information. This loop updates the screen at 10 times a second.

That's all there is to the stats.py code!

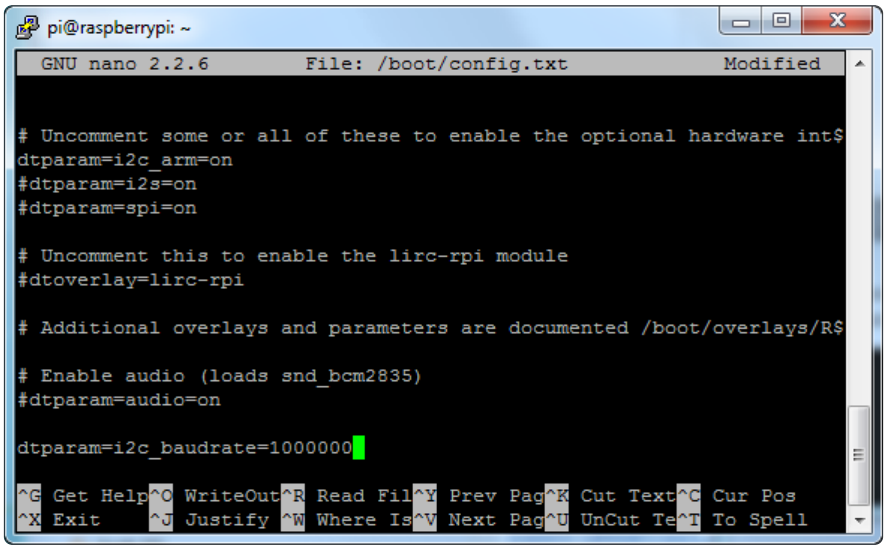
Speeding up the Display

For the best performance, especially if you are doing fast animations, you'll want to tweak the I2C core to run at 1MHz. By default it may be 100KHz or 400KHz

To do this edit the config with `sudo nano /boot/config.txt`

and add to the end of the file

`dtparam=i2c_baudrate=1000000`



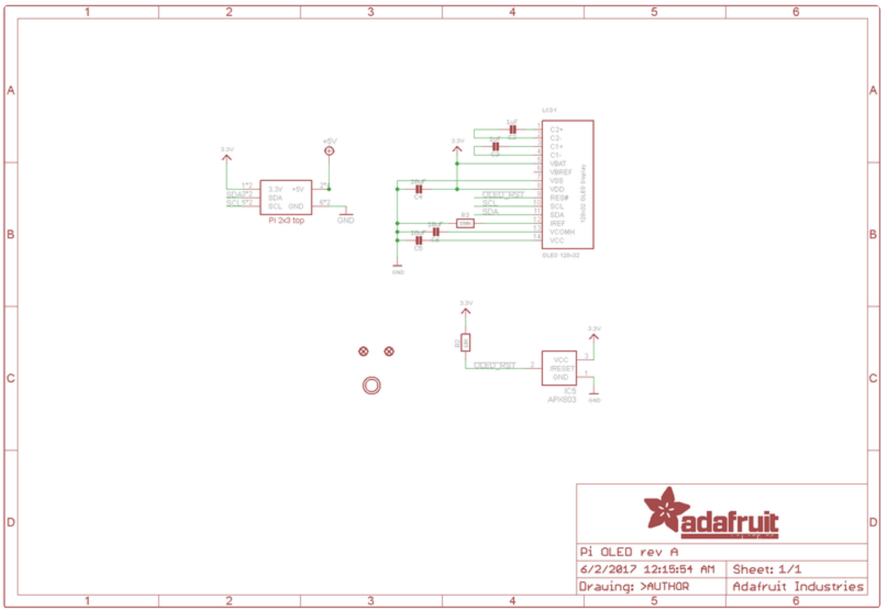
```
  GNU nano 2.2.6          File: /boot/config.txt                    Modified


# Uncomment some or all of these to enable the optional hardware int$
dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable the lirc-rpi module
#dtoverlay=lirc-rpi

# Additional overlays and parameters are documented /boot/overlays/R$

# Enable audio (loads snd_bcm2835)
#dtparam=audio=on

dtparam=i2c_baudrate=1000000

^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos
^X Exit    ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell
```

Downloads

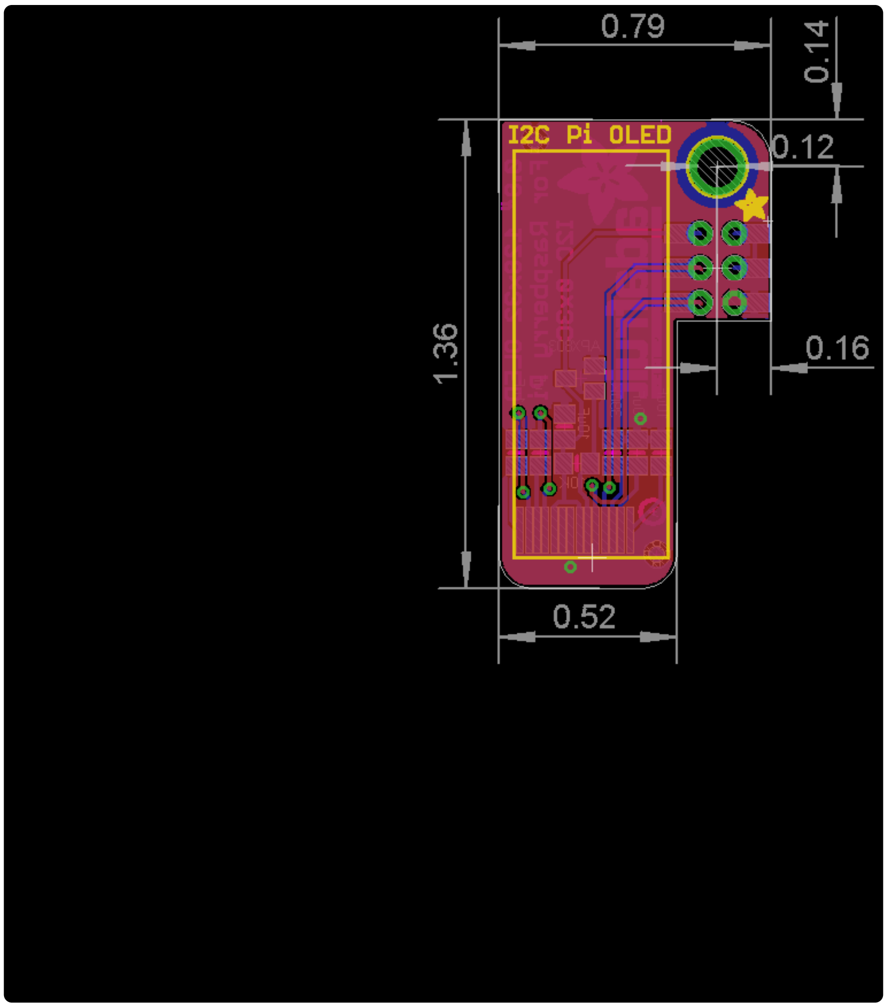
Files

- EagleCAD PCB files on GitHub ()
- UG-2832HSWEG02 () Datasheet
- SSD1306 () Datasheet
- Fritzing object in Adafruit Fritzing Library ()

Schematic & Fabrication Print



adafruit
Pi OLED rev A
6/2/2017 12:15:54 AM
Sheet: 1/1
Drawing: >AUTHOR
Adafruit Industries
Pi 2x3 top
APX803



0.79
0.14
0.12
1.36
0.16
0.52
I2C Pi OLED