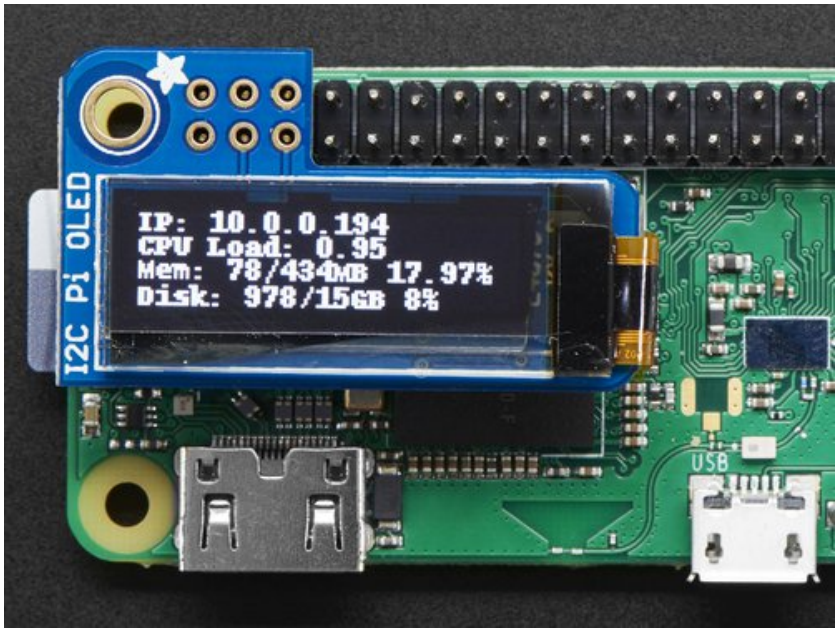
adafruit learning system

Adafruit PiOLED - 128x32 Mini OLED for Raspberry Pi

Created by lady ada

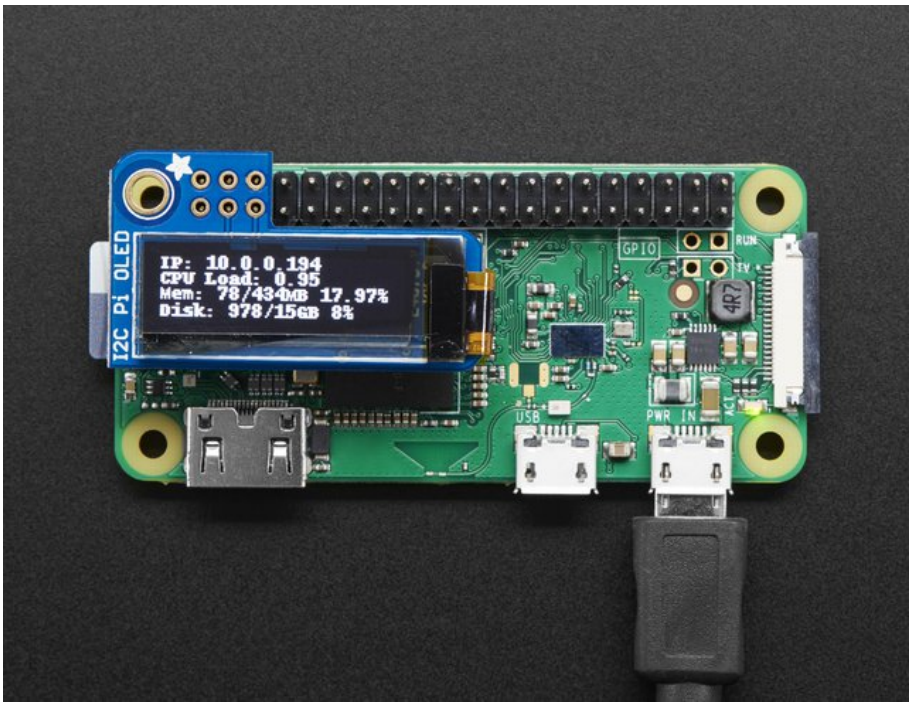


Last updated on 2017-06-02 04:28:36 AM UTC

Guide Contents

Guide Contents 2
Overview 3
Usage 5
Step 1. Dependencies 5
Step 2. Enable i2c 5
Step 3. Verify I2C Device 6
Running Stats on Boot 6
Library Usage 7
More Demos & Examples 9
Speeding Up the Display 9
Downloads 11
Files 11
Schematic & Fabrication Print 11

Overview

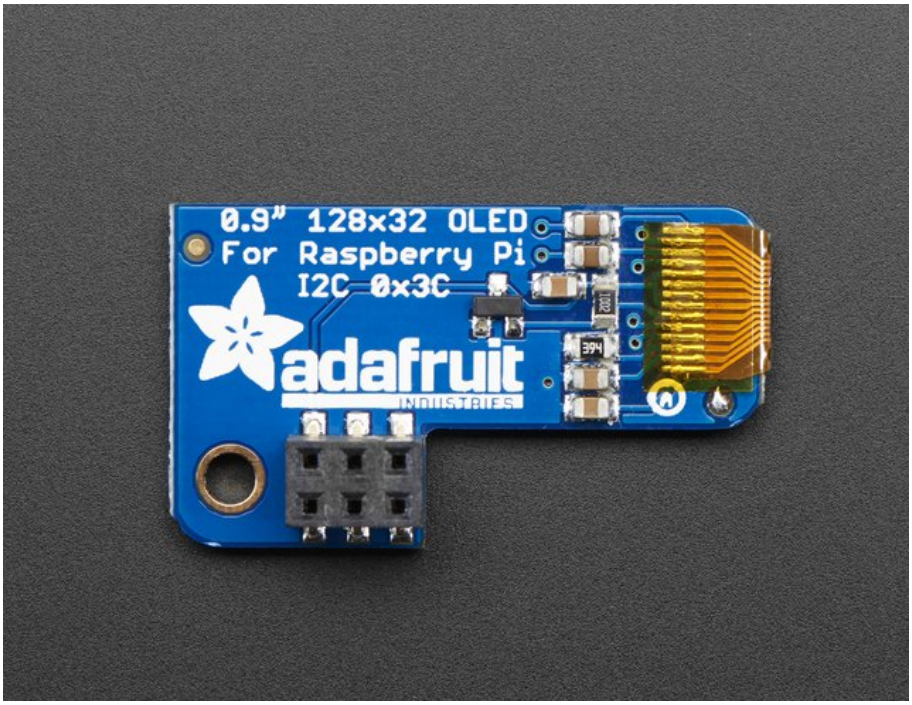


If you're looking for the most compact lil' display for a Raspberry Pi (http://adafru.it/wF8) (most likely a Pi Zero (http://adafru.it/vIa)) project, this might be just the thing you need!

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 and Pi Zero.

Usage

We'll be using Python to control the display. In theory you can use any language you like that gives you access to the computer's I2C ports, but our library is for Python only!

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

Step 1. Dependencies

Before using the library you will need to make sure you have a few dependencies installed.Connect to your Pi using SSH (http://adafru.it/vbC) and follow the steps below.

Install the RPi.GPIO library by running the following at the command line:

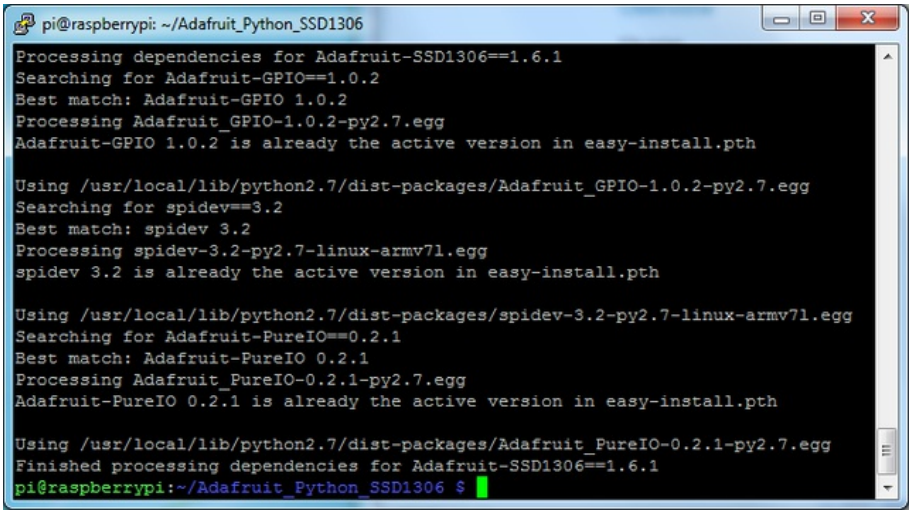
sudo apt-get update
sudo apt-get install build-essential python-dev python-pip
sudo pip install RPi.GPIO

Finally, install the Python Imaging Library (http://adafru.it/dvB) and smbus library by executing:

sudo apt-get install python-imaging python-smbus

Now to download and install the latest Adafruit SSD1306 python library code and examples, execute the following commands:

sudo apt-get install git
git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git (http://adafru.it/dEH)
cd Adafruit_Python_SSD1306
sudo python setup.py install



Step 2. Enable i2c

To enable i2c, you can follow our detailed guide on configuring the Pi with I2C support here(http://adafru.it/dEO)

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

Step 3. Verify I2C Device

While in the **Adafruit_Python_SSD1306** folder, 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.

Run sudo python examples/stats.py to run the demo, you should see something like the below:

Running Stats on Boot

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

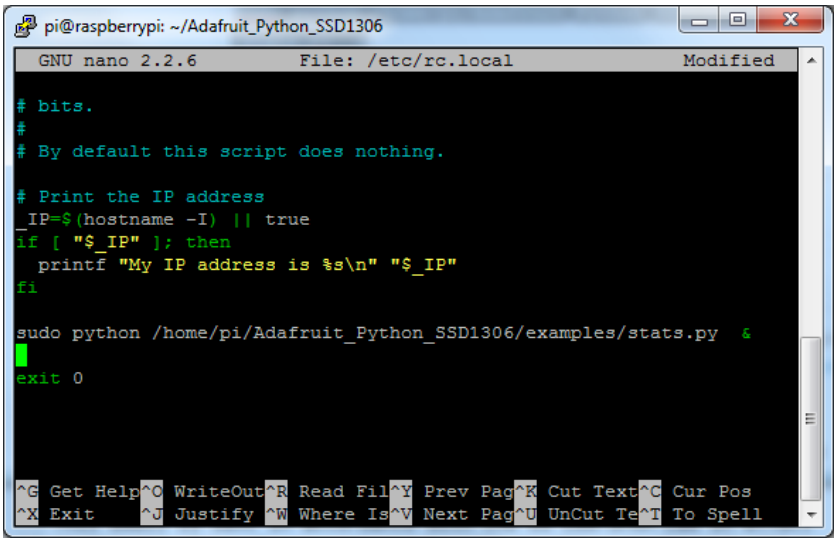
The fastest/easiest way is to put it in **/etc/rc.local**

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

```
sudo python /home/pi/Adafruit_Python_SSD1306/examples/stats.py  &
```

on its own line right before **exit 0**

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



For more advanced usage, check out our linux system services guide(http://adafru.it/wFR)

Library Usage

Inside the examples subdirectory you'll find python scripts which demonstrate the usage of the library.These are covered in more detail in our OLED guide here, so do check them out. (http://adafru.it/wF9)

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

```
import time

import Adafruit_GPIO.SPI as SPI
import Adafruit_SSD1306

from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont

import subprocess
```

First a few modules are imported, including the **Adafruit_SSD1306** module which contains the OLED display driver classes. You can also see some of the **P**ython **I**maging **L**ibrary modules like **Image**, **ImageDraw**, and **ImageFont** being imported. Those are, as you can imagine, are for drawing images, shapes and text/fonts!

```
# Raspberry Pi pin configuration:
RST = None
# Note the following are only used with SPI:
DC = 23
SPI_PORT = 0
SPI_DEVICE = 0

# 128x32 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST)

# 128x64 display with hardware I2C:
# disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)

# Alternatively you can specify an explicit I2C bus number, for example
# with the 128x32 display you would use:
# disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST, i2c_bus=2)
```

```
# 128x32 display with hardware SPI:
# disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST, dc=DC, spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE, max_speed_hz=8000000))

# 128x64 display with hardware SPI:
# disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST, dc=DC, spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE, max_speed_hz=8000000))

# Alternatively you can specify a software SPI implementation by providing
# digital GPIO pin numbers for all the required display pins.  For example
# on a Raspberry Pi with the 128x32 display you might use:
# disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST, dc=DC, sclk=18, din=25, cs=22)
```

Below the configuration values is the display class setup. There are 4 variants of OLED displays, with 128x32 pixels or 128x64 pixels, and with I2C or with SPI.

However since the PiOLED is a 128x32 I2C display *only* you should only use the

```
# 128x32 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST)
```

variant for creating the display object! The rest can remain commented out.

Note that above, we initialize `RST = None` because the PiOLED does not require a reset pin.

```
# Initialize library.
disp.begin()

# Clear display.
disp.clear()
disp.display()

# 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 bit of code will initialize the display library with `begin()` and clear the display with `clear()` and `display()`.

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!

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

# Alternatively load a TTF font.
# Some other nice fonts to try: http://www.dafont.com/bitmap.php
#font = ImageFont.truetype('Minecraftia.ttf', 8)
```

Once the display is initialized and a drawing object is prepared, you can draw shapes, text and graphics using PIL's drawing commands (http://adafru.it/dfH). Here we are loading 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 )
```

```
    cmd = "top -bn1 | grep load | awk '{printf \"CPU Load: %.2f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell = True )
    cmd = "free -m | awk 'NR==2{printf \"Mem: %s/%sMB %.2f%%\", $3,$2,$3*100/$2 }'"
    MemUsage = subprocess.check_output(cmd, shell = True )
    cmd = "df -h | awk '$NF==\"/\"{printf \"Disk: %d/%dGB %s\", $3,$2,$5}'"
    Disk = subprocess.check_output(cmd, shell = True )

    # Write two lines of text.

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

    # Display image.
    disp.image(image)
    disp.display()
    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!

More Demos & Examples

You can check out our other examples in the example, just make sure to edit each one with **nano animate.py** for example, and find the line that says:

```
# Raspberry Pi pin configuration:
RST = 24
```

and change it to:

```
# Raspberry Pi pin configuration:
RST = None # PiOLED does not require reset pin
```

and make sure that the configuration section where you choose which type of display, looks like this

```
# 128x32 display with hardware I2C:
disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST)

# 128x64 display with hardware I2C:
# disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)

# 128x32 display with hardware SPI:
# disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST, dc=DC, spi=SPI.SpiDev(SPI_POR$

# 128x64 display with hardware SPI:
# disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST, dc=DC, spi=SPI.SpiDev(SPI_POR$
```

That is, we'll be using I2C 128x32 display!

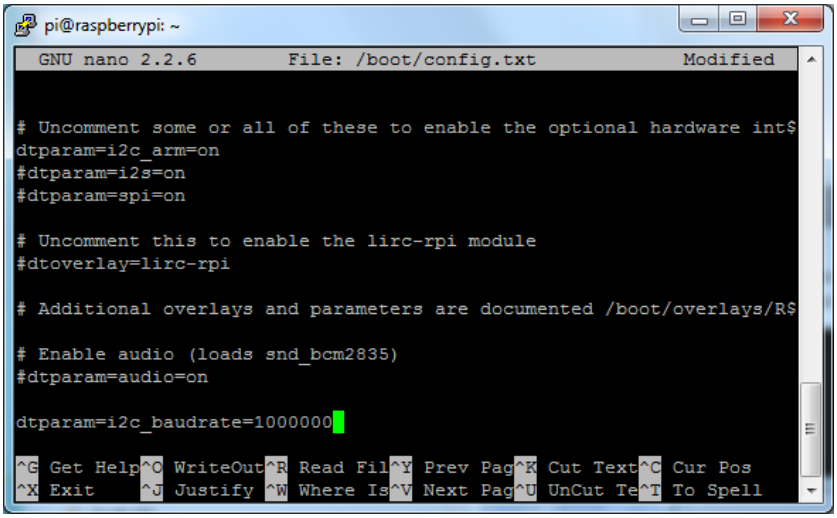
Speeding Up the Display

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



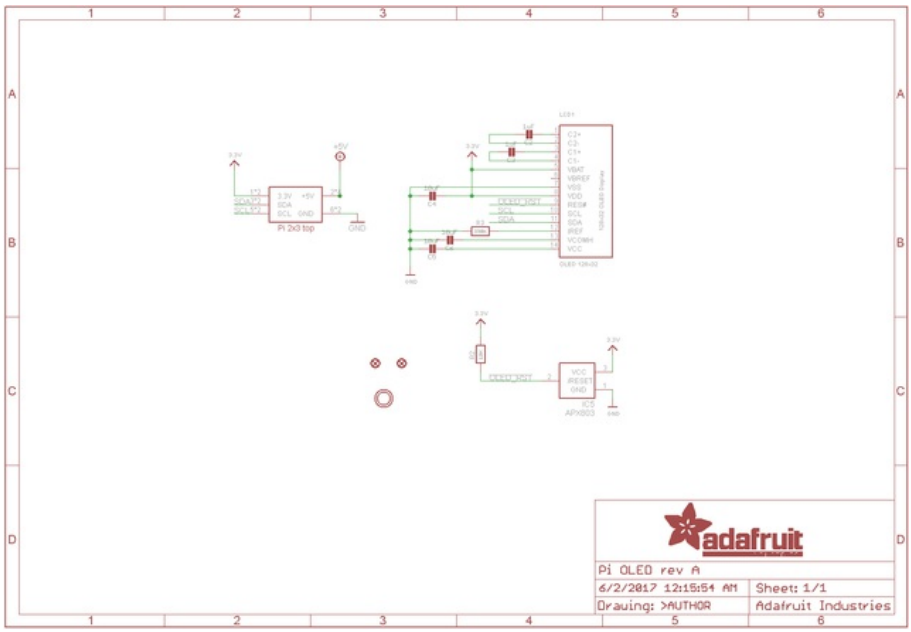
reboot to 'set' the change.

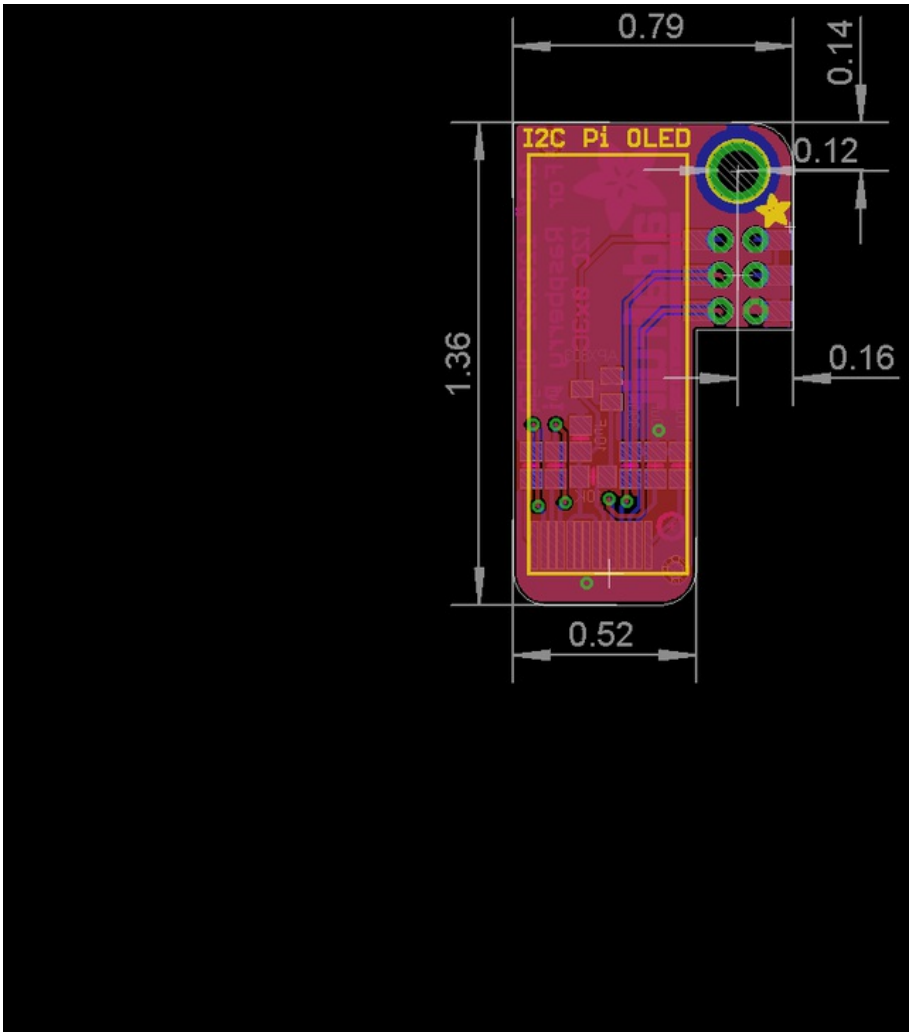
Downloads

Files

- EagleCAD PCB files on GitHub (http://adafru.it/wFa)
- UG-2832HSWEG02 (http://adafru.it/qrf) Datasheet
- SSD1306 (http://adafru.it/aJK) Datasheet
- Fritzing object in Adafruit Fritzing Library (http://adafru.it/c7M)

Schematic & Fabrication Print




0.79
0.14
I2C Pi OLED
0.12
1.36
0.16
0.52