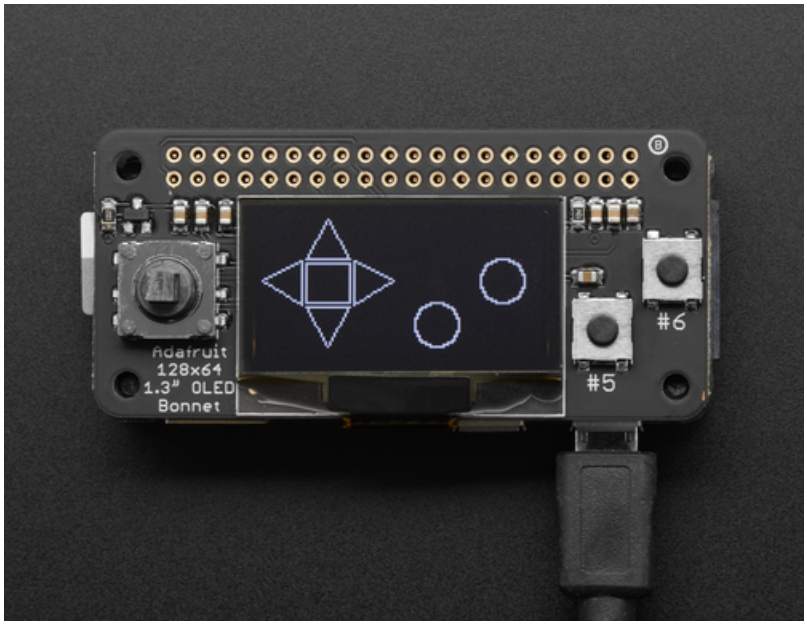
adafruit learning system

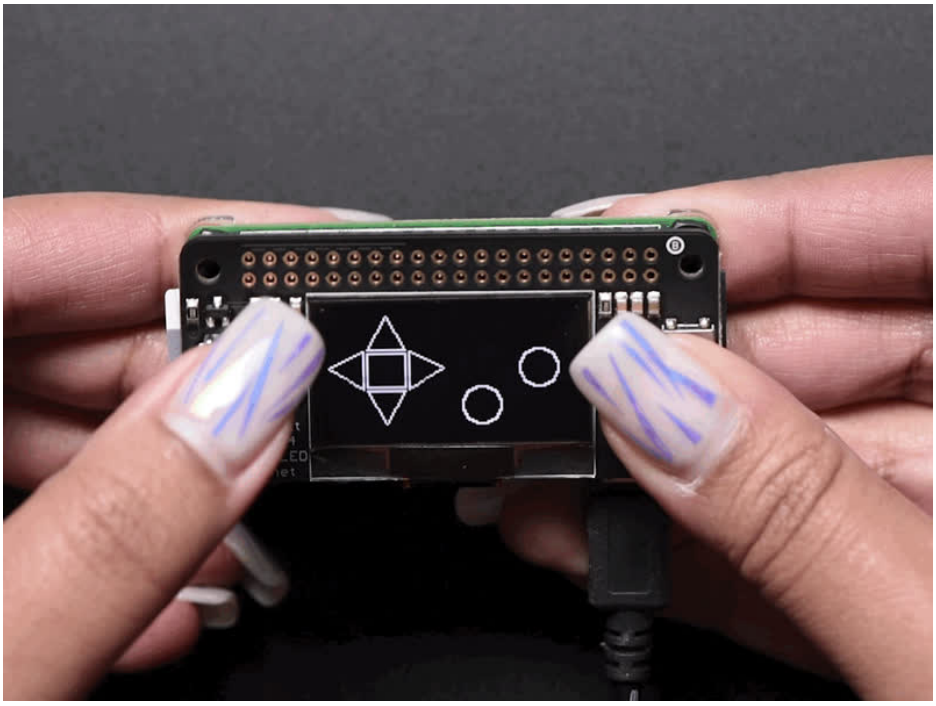
Adafruit 128x64 OLED Bonnet for Raspberry Pi

Created by lady ada

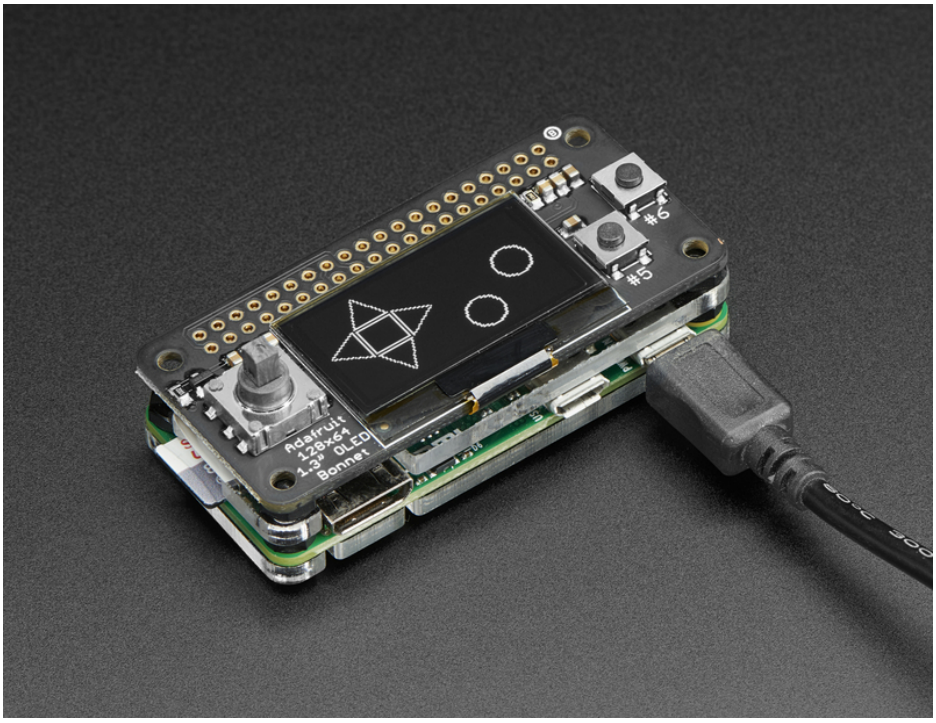


Last updated on 2020-06-24 02:59:56 PM EDT

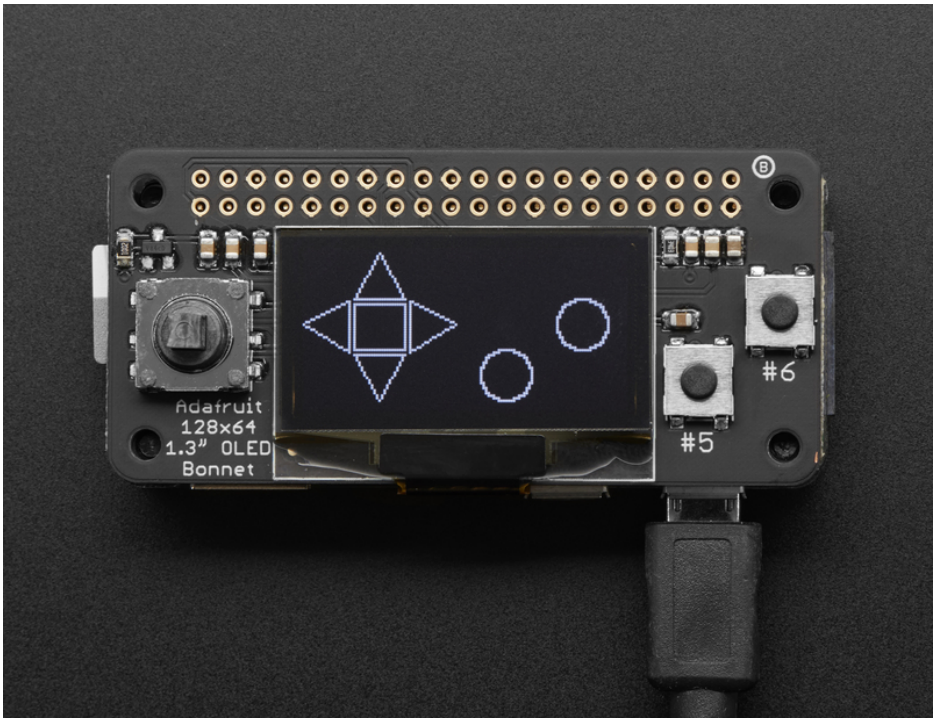
Overview



If you'd like a compact display, with buttons and a joystick - we've got what you're looking for. The Adafruit 128x64 OLED Bonnet for Raspberry Pi is the big sister to our mini PiOLED add-on (https://adafru.it/wVd). This version has 128x64 pixels (instead of 128x32) and a much larger screen besides. With the OLED display in the center, we had some space on either side so we added a 5-way joystick and two pushbuttons. Great for when you want to have a control interface for your project.

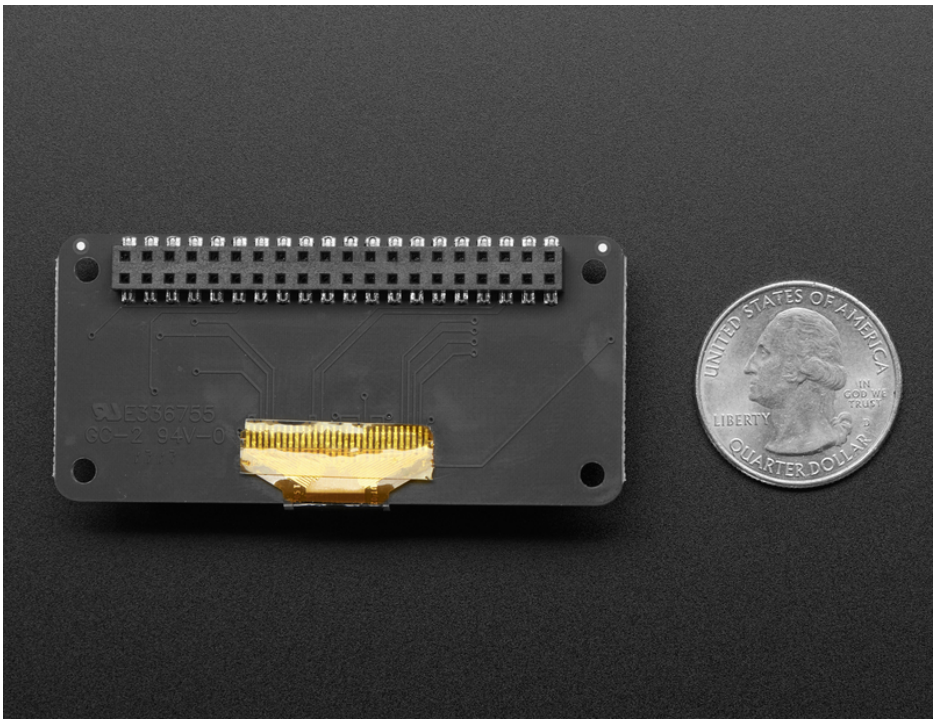


These displays are small, only about 1.3" diagonal, but very readable due to the high contrast of an OLED display. This screen is made of 128x64 individual white OLED pixels and because the display makes its own light, no backlight is required. This reduces the power required to run the OLED and is why the display has such high contrast; we really like this miniature display for its crispness!



Please note that this display is too small to act as a primary display for the Pi (e.g. it can't act like or display what would normally be on the HDMI screen). Instead, we recommend using **pygame** for drawing or writing text.

Using the display and controls in python is very easy, we have a library ready-to-go for the SSD1306 OLED chipset and the joystick/buttons are connected to GPIO pins on the Pi. Our example code allows you to draw images, text, whatever you like, using the Python imaging library. We also have example code for using the joystick/buttons/OLED together. Our tests showed 15 FPS update rates once you bump the I2C speed to 1MHz, 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

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

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:

https://adafru.it/Deo

https://adafru.it/Deo

To install the library for the Pi OLED (https://adafru.it/u1f), 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

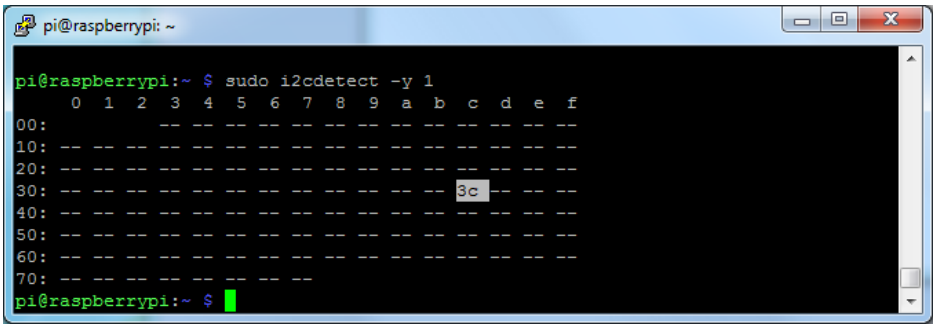
To enable i2c, you can follow our detailed guide on configuring the Pi with I2C support here. (https://adafru.it/Deo)

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



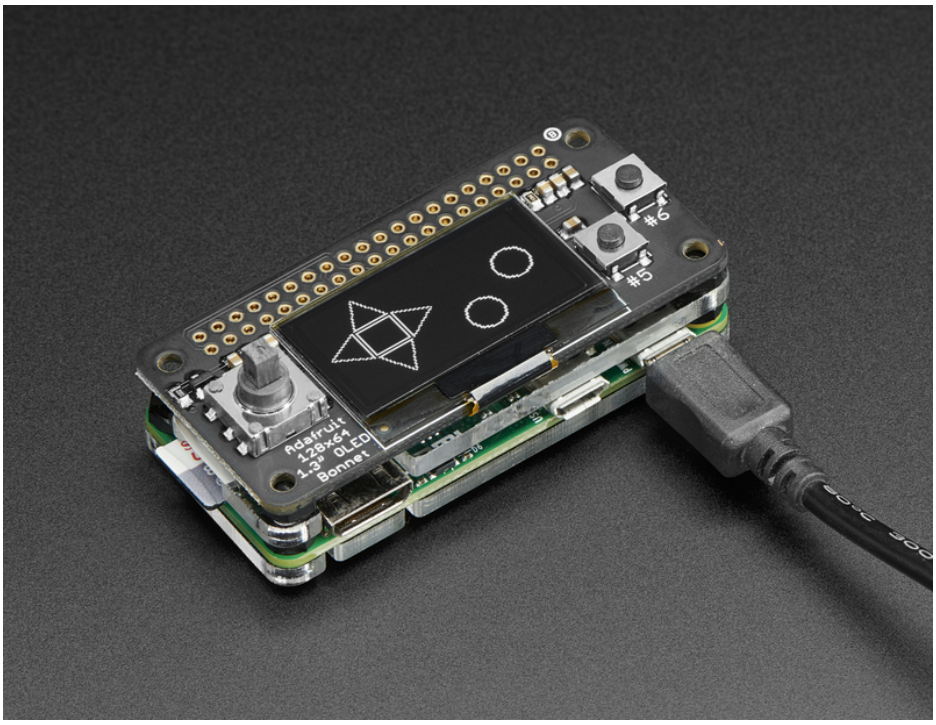
Verify I2C Device

You can run our buttons example, which will let you press various buttons and see them mimicked on the OLED.

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

```
Temporarily unable to load content:
```

Run `sudo python3 bonnet_buttons.py` to run the demo, you should see something like the below:



Press buttons to interact with the demo. Press the joystick + buttons at once for an Easter egg!

Running Scripts on Boot

You can pretty easily make it so this program (or whatever program you end up writing) run every time you boot your Pi.

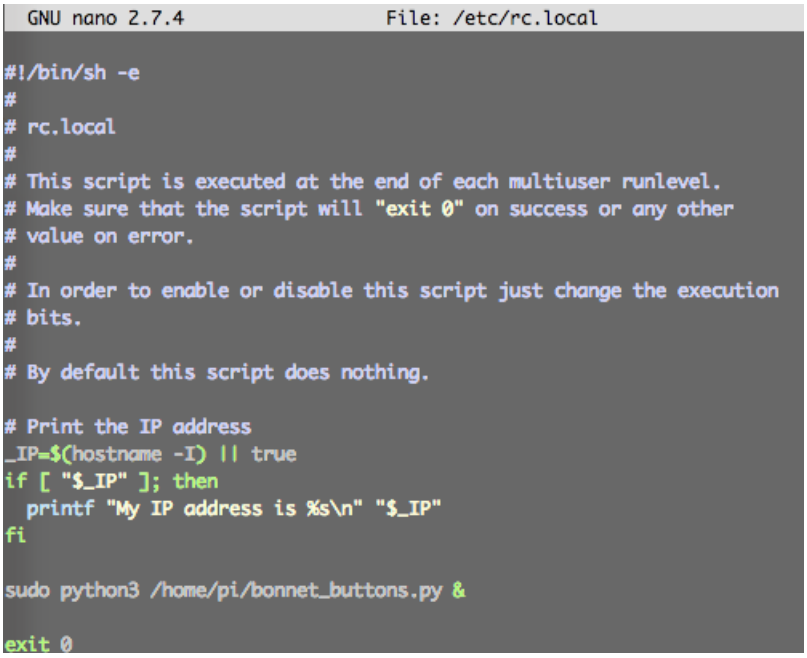
The fastest/easiest way is to put it in **/etc/rc.local**

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

`sudo python /home/pi/bonnet_buttons.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 (https://adafru.it/wFR)

Library Usage

In the examples subdirectory of the Adafruit_CircuitPython_SSD1306 repository (https://adafru.it/EsZ), you'll find more examples which demonstrate the usage of the library.

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

Python Library Setup

```
import board
import busio
from digitalio import DigitalInOut, Direction, Pull
from PIL import Image, ImageDraw
import adafruit_ssd1306
```

First, a few modules are imported, including the `adafruit_ssd1306` module which contains the OLED driver classes. The code also imports board (containing the Raspbery Pi pin definitions), busio (communication with the i2c and spi buses), and digitalio (to control the Raspberry Pi's pins).

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!

Display Setup

```
# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)
# Create the SSD1306 OLED class.
disp = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c)
```

The next bit of code creates the I2C interface (which the display on the bonnet communicates over) and creates a SSD1306 OLED class. Note that we are passing `SSD1306_I2C` 128 and 64, those values correspond to the bonnet's OLED display.

Pin Setup

```
# Input pins:
button_A = DigitalInOut(board.D5)
button_A.direction = Direction.INPUT
button_A.pull = Pull.UP

button_B = DigitalInOut(board.D6)
button_B.direction = Direction.INPUT
button_B.pull = Pull.UP

button_L = DigitalInOut(board.D27)
button_L.direction = Direction.INPUT
button_L.pull = Pull.UP

button_R = DigitalInOut(board.D23)
button_R.direction = Direction.INPUT
button_R.pull = Pull.UP

button_U = DigitalInOut(board.D17)
button_U.direction = Direction.INPUT
button_U.pull = Pull.UP

button_D = DigitalInOut(board.D22)
button_D.direction = Direction.INPUT
button_D.pull = Pull.UP

button_C = DigitalInOut(board.D4)
button_C.direction = Direction.INPUT
button_C.pull = Pull.UP
```

Next up we define the pins that are used for the joystick and buttons. The Joystick has Left, Right, Center (press in), Up and Down. There's also the A and B buttons on the right. Each one should be set as an input with pull-up resistor (`Pull.UP` in the code)

Display Initialization

```
# 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!

Button Input and Drawing

```
while True:
    if button_U.value: # button is released
        draw.polygon([(20, 20), (30, 2), (40, 20)], outline=255, fill=0)  #Up
    else: # button is pressed:
        draw.polygon([(20, 20), (30, 2), (40, 20)], outline=255, fill=1)  #Up filled

    if button_L.value: # button is released
        draw.polygon([(0, 30), (18, 21), (18, 41)], outline=255, fill=0)  #left
    else: # button is pressed:
        draw.polygon([(0, 30), (18, 21), (18, 41)], outline=255, fill=1)  #left filled

    if button_R.value: # button is released
        draw.polygon([(60, 30), (42, 21), (42, 41)], outline=255, fill=0) #right
    else: # button is pressed:
        draw.polygon([(60, 30), (42, 21), (42, 41)], outline=255, fill=1) #right filled

    if button_D.value: # button is released
        draw.polygon([(30, 60), (40, 42), (20, 42)], outline=255, fill=0) #down
    else: # button is pressed:
        draw.polygon([(30, 60), (40, 42), (20, 42)], outline=255, fill=1) #down filled

    if button_C.value: # button is released
        draw.rectangle((20, 22, 40, 40), outline=255, fill=0) #center
    else: # button is pressed:
        draw.rectangle((20, 22, 40, 40), outline=255, fill=1) #center filled

    if button_A.value: # button is released
        draw.ellipse((70, 40, 90, 60), outline=255, fill=0) #A button
    else: # button is pressed:
        draw.ellipse((70, 40, 90, 60), outline=255, fill=1) #A button filled

    if button_B.value: # button is released
        draw.ellipse((100, 20, 120, 40), outline=255, fill=0) #B button
    else: # button is pressed:
        draw.ellipse((100, 20, 120, 40), outline=255, fill=1) #B button filled

    if not button_A.value and not button_B.value and not button_C.value:
        catImage = Image.open('happycat_oled_64.ppm').convert('1')
        disp.image(catImage)
    else:
        # Display image.
        disp.image(image)

    disp.show()
```

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

This is a basic polling example - we'll check each `button.value` in order, and draw a different shape - a directional arrow or a round circle) depending on whether the button is pressed. If the button is pressed we have the shape filled in. If the button is not pressed, we draw an outline only

Then we run **`disp.image(image)`** and **`disp.show()`** to actually push the updated image to the OLED. **This is required to actually make the changes appear!**

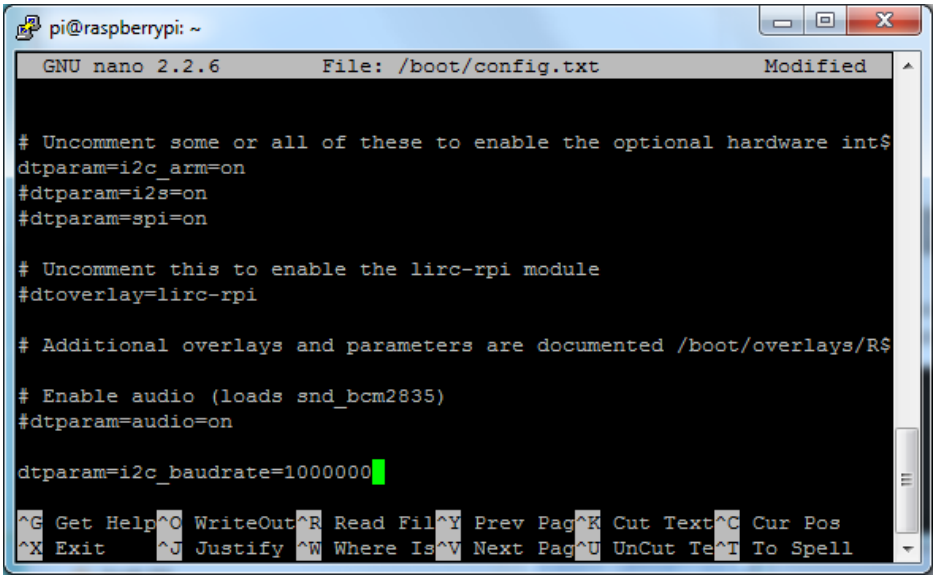
Speeding up the Display

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.

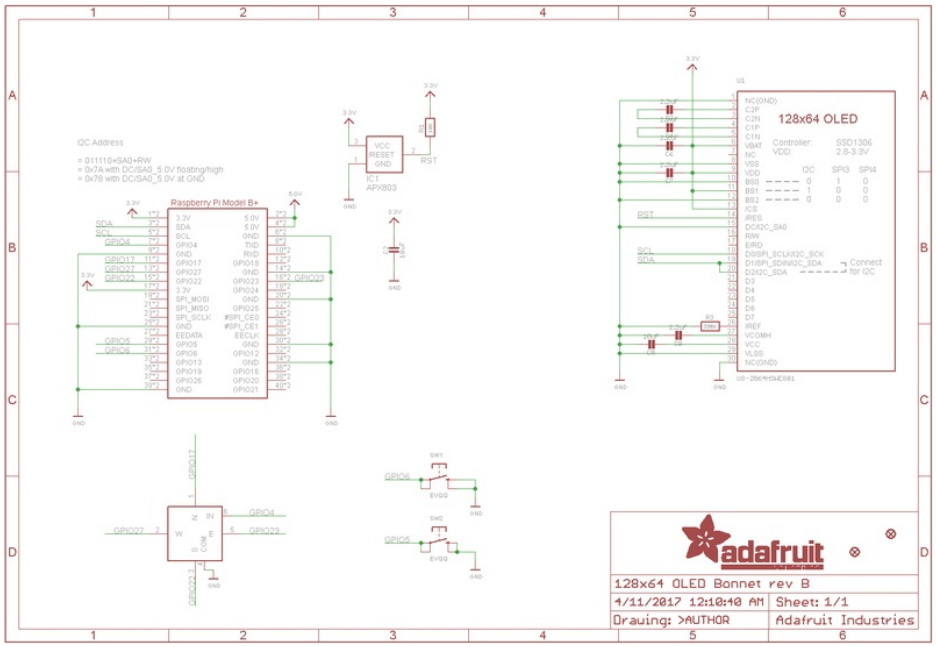
Downloads

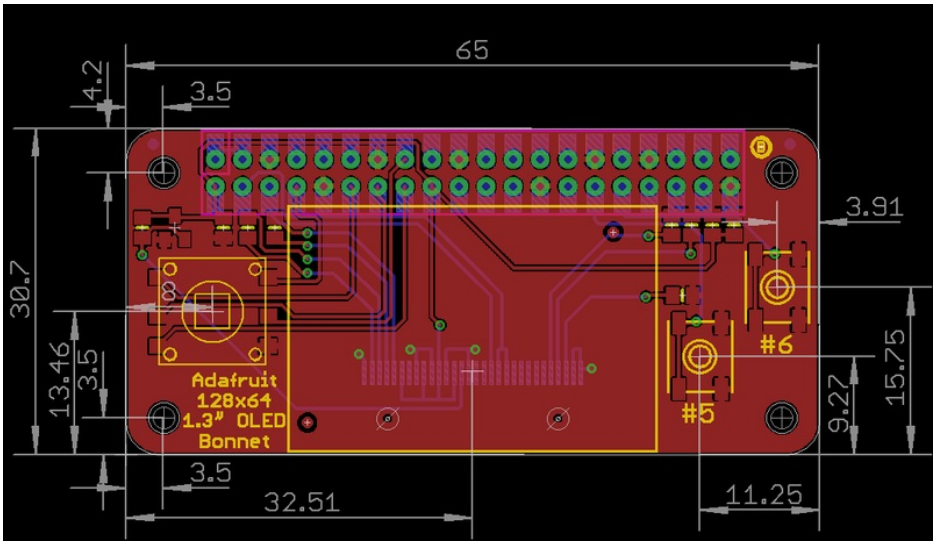
Files

- EagleCAD PCB files on GitHub (https://adafru.it/wWC)
- UG-2864HSWEG01 (https://adafru.it/aJI) Datasheet
- UG-2864HSWEG01 (https://adafru.it/wWD) User Guide
- SSD1306 (https://adafru.it/aJK) Datasheet
- Fritzing objects available in the Adafruit Fritzing Library (https://adafru.it/aP3)

Schematic & Fabrication Print

Dimensions in mm





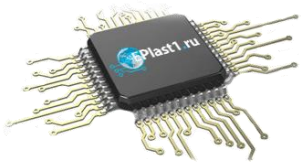
Компания «ЭлектроПласт» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Оперативные поставки широкого спектра электронных компонентов отечественного и импортного производства напрямую от производителей и с крупнейших мировых складов;
- Поставка более 17-ти миллионов наименований электронных компонентов;
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- Лицензия ФСБ на осуществление работ с использованием сведений, составляющих государственную тайну;
- Поставка специализированных компонентов (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Aeroflex, Peregrine, Syfer, Eurofarad, Texas Instrument, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits,General Dynamics и др.);

Помимо этого, одним из направлений компании «ЭлектроПласт» является направление «Источники питания». Мы предлагаем Вам помощь Конструкторского отдела:

- Подбор оптимального решения, техническое обоснование при выборе компонента;
- Подбор аналогов;
- Консультации по применению компонента;
- Поставка образцов и прототипов;
- Техническая поддержка проекта;
- Защита от снятия компонента с производства.



Как с нами связаться

Телефон: 8 (812) 309 58 32 (многоканальный)
Факс: 8 (812) 320-02-42
Электронная почта: org@eplast1.ru
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, дом 2, корпус 4, литера А.