This is a quick tutorial for our 128x64 and 128x32 pixel monochrome OLED displays. These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. Each OLED display is made of 128x64 or 128x32 individual white OLEDs, each one is turned on or off by the controller chip. 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!
The driver chip, SSD1306 can communicate in multiple ways including I2C, SPI and 8-bit parallel. However, only the 128x64 display has all these interfaces available. For the 128x32 OLED, only SPI is available. Frankly, we prefer SPI since its the most flexible and uses a small number of I/O pins so our example code and wiring diagram will use that.
Power Requirements

OLED Power Requirements
The OLED and driver require a 3.3V power supply and 3.3V logic levels for communication. The power requirements depend a little on how much of the display is lit but on average the display uses about 20mA from the 3.3V supply. Built into the OLED driver is a simple switch-cap charge pump that turns 3.3v-5v into a high voltage drive for the OLEDs. You can run the entire display off of one 3.3V supply or use 3.3V for the chip power and up to 4.5V for the OLED charge pump or 3.3V for the chip power and a 7-9V supply directly into the OLED high voltage pin.

5V- ready 128x64 and 128x32 OLEDs
Unless you have the older v1 128x64 OLED, you can rest assured that your OLED is 5V ready. All 1.3” 128x64 and the small 128x32 SPI and I2C are 5V ready, if you have a v2 0.96” 128x64 OLED with the 5V ready mark on the front, it’s also 5V safe. If you have an older 0.96” OLED (see below) you’ll need to take extra care when wiring it to a 5V microcontroller. The OLED is designed to be 5V compatible so you can power it with 3-5V and the onboard regulator will take care of the rest.

All OLEDs are safe to use with 3.3V logic and power.

Simply connect GND to ground, and Vin to a 3 to 5V power supply. There will be a 3.3V output on the 3Vo pin in case you want a regulated 3.3V supply for something else.

0.96” 128x64 OLED
The older 0.96” 128x64 OLED is a little more complex to get running as it is not 5V compatible by default, so you have to provide it with 3.3V power.
- **VDD** is the 3.3V logic power. This must be 3 or 3.3V
- **VBAT** is the input to the charge pump. If you use the charge pump, this must be 3.3V to 4.2V
- **VCC** is the high voltage OLED pin. If you're using the internal charge pump, this must be left unconnected. If you're not using the charge pump, connect this to a 7-9V DC power supply.

For most users, we suggest connecting **VDD** and **VBAT** together to 3.3V and then leaving **VCC** unconnected.
Arduino Library & Examples

For all of the different kinds of small OLED monochrome displays, you'll need to install the Arduino libraries. The code we have is for any kind of Arduino, if you're using a different microcontroller, the code is pretty simple to adapt, the interface we use is basic bit-twiddling SPI or I2C.

Install Arduino Libraries

Using these OLEDs with Arduino sketches requires that two libraries be installed: Adafruit_SSD1306, which handles the low-level communication with the hardware, and Adafruit_GFX, which builds atop this to add graphics functions like lines, circles and text.

In recent versions of the Arduino IDE software (1.6.2 and later), this is most easily done through the Arduino Library Manager, which you'll find in the “Sketch” menu: Sketch→Include Library→Manage Libraries...

Enter “ssd1306” in the search field, locate the Adafruit SSD1306 library and select “Install” (or “Upgrade” if you have an older version). Then repeat the same for “gfx” and the Adafruit GFX library.

We also have a great tutorial on Arduino library installation here: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafruit.it/aYM)

Run Demo!

After installing the Adafruit_SSD1306 and Adafruit_GFX library, restart the Arduino IDE. You should now be able to access the sample code by navigating through menus in this order: File→Sketchbook→Libraries→Adafruit_SSD1306→SSD1306...
After you've finished wiring the display as indicated on the following pages, load the example sketch to demonstrate the capabilities of the library and display.

The OLED SSD1306 driver is based on the Adafruit GFX library which provides all the underlying graphics functions such as drawing pixels, lines, circles, etc. For more details about what you can do with the OLED check out the GFX library tutorial (https://adafruit.it/aPx)
Create Bitmaps
You can create bitmaps to display easily with the LCD assistant software (https://adafruit.it/aPs). First make your image using any kind of graphics software such as photoshop or Paint and save as a Monochrome Bitmap (bmp)

Select the following options (You might also want to try Horizontal if Vertical is not coming out right)
and import your monochrome bitmap image. Save the output to a .cpp file

You can use the output directly with our example code
Wiring 128x64 OLEDs

Solder Header

Before you start wiring, a strip of header must be soldered onto the OLED. It is not possible to "press-fit" the header, it must be attached!

Start by placing an 8-pin piece of header with the long ends down into a breadboard for stability.

Place the OLED on top so all the short ends of the header stick thru the header pads.
I2C or SPI

The nice thing about the 128x64 OLEDs is that they can be used with I2C (+ a reset line) or SPI. SPI is generally faster than I2C but uses more pins. It's also easier for some microcontrollers to use SPI. Anyways, you can use either one with this display.

Using with I2C

The display can be used with any I2C microcontroller. Because the I2C interface is for ‘writing’ to the display only, you'll still have to buffer the entire 512 byte frame in the microcontroller RAM - you can't read data from the OLED (even though I2C is a bidirectional protocol).

To start, you'll need to solder the two jumpers on the back of the OLED. Both must be soldered 'closed' for I2C to work!
Finally, connect the pins to your Arduino

- GND goes to ground
- Vin goes to 5V
- Data to I2C SDA (on the Uno, this is A4 on the Mega it is 20 and on the Leonardo digital 2)
- Clk to I2C SCL (on the Uno, this is A5 on the Mega it is 21 and on the Leonardo digital 3)
- RST to digital 4 (you can change this pin in the code, later)

This matches the example code we have written. Once you get this working, you can try a different Reset pin (you can’t change the SDA and SCL pins).

Finally you can run the File→Sketchbook→Libraries→Adafruit_SSD1306→SSD1306_128x64_i2c example

Using with SPI
The breakouts are ready for SPI by default, but if you used them for I2C at some point, you’ll need to remove the solder jumpers. Use wick or a solder sucker to make sure both are clear!
Finally, connect the pins to your Arduino -

- **GND** goes to ground
- **Vin** goes to 5V
- **DATA** to digital 9
- **CLK** to digital 10
- **D/C** to digital 11
- **RST** to digital 13
- **CS** to digital 12

*(Note: If using the display with other SPI devices, D/C, CLK and DAT may be shared, but CS must be unique for each device.)*

This matches the example code we have written. Once you get this working, you can try another set of pins.

Finally you can run the File→Sketchbook→Libraries→Adafruit_SSD1306→SSD1306_128x64_spi example
Wiring 128x32 SPI OLED display

128x32 SPI OLED

The 128x32 SPI OLED is very easy to get up and running because it has built in level shifting. First up, take a piece of 0.1" header 8 pins long.

Plug the header long end down into a breadboard and place the OLED on top. Solder the short pins into the OLED PCB.
Finally, connect the pins to your Arduino - GND goes to ground, Vin goes to 5V, DATA to digital 9, CLK to
digital 10, D/C to digital 11, RST to digital 13 and finally CS to digital 12.

(Note: If using the display with other SPI devices, D/C, CLK and DAT may be shared, but CS must be unique for each
device.)

This matches the example code we have written. Once you get this working, you can try another set of pins.

Finally you can run the File→Sketchbook→Libraries→Adafruit_SSD1306→SSD1306_128x32_SPI example

If you’re using the 128x32 OLED, be sure to uncomment the "#define SSD1306_128_32" in the top of
Adafruit_SSD1306.h to change the buffer size
Wiring 128x32 I2C Display

128x32 I2C OLED
The 128x32 I2C OLED is very easy to get up and running because it has built in level shifting and regulator. First up, take a piece of 0.1" header 6 pins long.

Plug the header long end down into a breadboard.

Place the OLED on top.

Solder the short pins into the OLED PCB.
Finally, connect the pins to your Arduino

- **GND** goes to ground
- **Vin** goes to 5V
- **SDA** to I2C Data (on the Uno, this is **A4** on the Mega it is **20** and on the Leonardo digital **2**)
- **SCL** to I2C Clock (on the Uno, this is **A5** on the Mega it is **21** and on the Leonardo digital **3**)
- **RST** to digital **4** (you can change this pin in the code, later)

This matches the example code we have written. Once you get this working, you can change the RST pin. You cannot change the I2C pins, those are 'fixed' in hardware

Finally you can run the File→Sketchbook→Libraries→Adafruit_SSD1306→SSD1306_128x32_i2c example
Wiring OLD 0.96" 128x64 OLED

This wiring diagram is only for the older 0.96" OLED that comes with a level shifter chip. If you did not get a level shifter chip, you have a V2.0 so please check out the other wiring tutorial!

128x64 Version 1.0 OLED

The version 1 128x64 OLED runs at 3.3V and does not have a built in level shifter so you'll need to use a level shifting chip to use with a 5V microcontroller. The following will assume that is the case. If you're running a 3.3V microcontroller system, you can skip the level shifter.

We'll assume you want to use this in a breadboard, take a piece of 0.1" header 10 pins long.

Place the header in a breadboard and then place the left hand side of the OLED on top.
And solder the pins.

We'll be using the internal charge pump so connect **VDD** and **VBAT** together (they will connect to 3.3V). GND goes to ground.

Place a CD4050 level shifter chip so pin one is at the top.
Connect pin 10 to D/C pin 12 to CLK (SPI clock) and pin 15 to DAT (SPI data).

Connect pin 2 to RES (reset) and pin 4 to CS (chip select). Pin 1 goes to 3.3V and pin 8 to ground.

(Note: If using the display with other SPI devices, D/C, CLK and DAT may be shared, but CS must be unique for each device.)
You can connect the inputs of the level shifter to any pins you want but in this case we connected digital I/O 13 to pin 3 of the level shifter, 12 to pin 5, 11 to pin 9, 10 to pin 11 and 9 to pin 14. This matches the example code we have written. Once you get this working, you can try another set of pins.
CircuitPython Wiring

It's easy to use OLEDs with CircuitPython and the Adafruit CircuitPython SSD1306 module. This module allows you to easily write Python code to control the display.

You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library.

We'll cover how to wire the OLED to your CircuitPython microcontroller board. First assemble your OLED.

Connect the OLED to your microcontroller board as shown below.

Adafruit OLED FeatherWing

- Solder the Feather with female headers on top or stacking headers.
- Attach the OLED FeatherWing using the stacking method.

Adafruit 128x32 I2C OLED Display

- Microcontroller 3V to OLED VIN
- Microcontroller GND to OLED GND
- Microcontroller SCL to OLED SCL
- Microcontroller SDA to OLED SDA
- Microcontroller D9 to OLED RST

Adafruit 128x32 SPI OLED Display
Adafruit 0.96" or 1.3" 128x64 OLED Display - I2C Wiring

You must solder two jumpers closed on the back of the display to use with I2C!

- Microcontroller 3V to OLED VIN
- Microcontroller GND to OLED GND
- Microcontroller SCK to OLED CLK
- Microcontroller MOSI to OLED Data
- Microcontroller D5 to OLED CS
- Microcontroller D6 to OLED D/C
- Microcontroller D9 to OLED RST
Adafruit 0.96" or 1.3" 128x64 OLED Display - SPI Wiring
- Microcontroller 3V to OLED Vin
- Microcontroller GND to OLED Gnd
- Microcontroller SCK to OLED Clk
- Microcontroller MOSI to OLED Data
- Microcontroller D5 to OLED CS
- Microcontroller D6 to OLED DC
- Microcontroller D9 to OLED Rst
CircuitPython Setup

CircuitPython Installation of DisplayIO SSD1306 Library

To use the SSD1306 OLED with your Adafruit CircuitPython board you’ll need to install the Adafruit CircuitPython DisplayIO SSD1306 module on your board.

First make sure you are running the latest version 5.0 or later of Adafruit CircuitPython for your board.

Next you’ll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle. Our CircuitPython starter guide has a great page on how to install the library bundle.

If you choose, you can manually install the libraries individually on your board:

- `adafruit_displayio_ssd1306`
- `adafruit_bus_device`

Before continuing make sure your board’s lib folder or root filesystem has the `adafruit_displayio_ssd1306.mpy` and `adafruit_bus_device` files and folders copied over.

Next connect to the board’s serial REPL so you are at the CircuitPython >>> prompt.

Code Example Additional Libraries

For the Code Example, you will need an additional library. We decided to make use of a library so the code didn’t get overly complicated.

Go ahead and install this in the same manner as the driver library by copying the `adafruit_display_text` folder over to the lib folder on your CircuitPython device.
CircuitPython Usage

It's easy to use OLEDs with Python and the Adafruit CircuitPython DisplayIO SSD1306 module. This module allows you to easily write Python code to control the display.

To demonstrate the usage, we'll initialize the library and use Python code to control the OLED from the board's Python REPL.

I2C Initialization

If your display is connected to the board using I2C (like if using a Feather and the FeatherWing OLED) you'll first need to initialize the I2C bus. First import the necessary modules:

```python
import board
```

Now for either board run this command to create the I2C instance using the default SCL and SDA pins (which will be marked on the boards pins if using a Feather or similar Adafruit board):

```python
i2c = board.I2C()
```

After initializing the I2C interface for your firmware as described above, you can create an instance of the I2CDisplay bus:

```python
import displayio
import adafruit_displayio_ssd1306
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c)
```

Finally, you can pass the display_bus in and create an instance of the SSD1306 I2C driver by running:

```python
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=32)
```

Now you should be seeing an image of the REPL. Note that the last two parameters to the SSD1306 class initializer are the width and height of the display in pixels. Be sure to use the right values for the display you're using!

128 x 64 size OLEDs (or changing the I2C address)

If you are using a 128x64 display, the I2C address is probably different (0x3d), unless you've changed it by soldering some jumpers:

```python
display_bus = displayio.I2CDisplay(i2c, device_address=0x3d)
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=64)
```

Adding hardware reset pin
If you have a reset pin (which may be required if your OLED does not have an auto-reset chip like the FeatherWing) also pass in a reset pin like so:

```python
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c, reset=board.D9)
```

At this point the I2C bus and display are initialized. **Skip down to the example code section.**

### SPI Initialization

If your display is connected to the board using SPI you'll first need to initialize the SPI bus.

If you're using a microcontroller board, run the following commands:

```python
import board
import displayio
import adafruit_displayio_ssd1306

displayio.release_displays()

spi = board.SPI()
tft_cs = board.D5
tft_dc = board.D6
tft_reset = board.D9

display_bus = displayio.FourWire(spi, command=tft_dc, chip_select=tft_cs,
                                  reset=tft_reset, baudrate=1000000)
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=64)
```

The parameters to the FourWire initializer are the pins connected to the display's **DC**, **CS**, and **reset**. Because we are using keyword arguments, they can be in any position. Again make sure to use the right pin names as you have wired up to your board!

Note that the last two parameters to the **SSD1306** class initializer are the **width** and **height** of the display in pixels. Be sure to use the right values for the display you're using!

### Example Code

```python
***
This test will initialize the display using displayio and draw a solid white background, a smaller black rectangle, and some white text.
***

import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_displayio_ssd1306

displayio.release_displays()

oled_reset = board.D9

# Use for I2C
```
```python
i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c, reset=oled_reset)

# Use for SPI
#spi = board.SPI()
#oled_cs = board.D5
#oled_dc = board.D6
display_bus = displayio.FourWire(spi, command=oled_dc, chip_select=oled_cs,
#                                 reset=oled_reset, baudrate=1000000)

WIDTH = 128
HEIGHT = 32     # Change to 64 if needed
BORDER = 5
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=WIDTH, height=HEIGHT)

# Make the display context
splash = displayio.Group(max_size=10)
display.show(splash)

color_bitmap = displayio.Bitmap(WIDTH, HEIGHT, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF # White

bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)

splash.append(bg_sprite)

# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(WIDTH-BORDER*2, HEIGHT-BORDER*2, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000 # Black
inner_sprite = displayio.TileGrid(inner_bitmap,
                                   pixel_shader=inner_palette,
                                   x=BORDER, y=BORDER)

splash.append(inner_sprite)

# Draw a label
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFFFF, x=28, y=HEIGHT//2-1)
splash.append(text_area)

while True:
    pass
```

Let's take a look at the sections of code one by one. We start by importing the `board` so that we can initialize SPI, `displayio`, `terminalio` for the font, a `label`, and the `adafruit_displayio_ssd1306` driver.

```python
import board
import displayio
import terminalio
from adafruit_display_text import label
import adafruit_displayio_ssd1306
```

Next we release any previously used displays. This is important because if the microprocessor is reset, the display pins
are not automatically released and this makes them available for use again.

displayio.release_displays()

Next we define the reset line, which will be used for either SPI or I2C.

oled_reset = board.D9

If you're using I2C, you would use this section of code. We set the I2C object to the board's I2C with the easy shortcut function `board.I2C()`. By using this function, it finds the SPI module and initializes using the default SPI parameters. We also set the display bus to I2CDisplay which makes use of the I2C bus.

```python
# Use for I2C
i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3c, reset=oled_reset)
```

If you're using SPI, you would use this section of code. We set the SPI object to the board's SPI with the easy shortcut function `board.SPI()`. By using this function, it finds the SPI module and initializes using the default SPI parameters. We set the OLED's CS (Chip Select), and DC (Data/Command) pins. We also set the display bus to FourWire which makes use of the SPI bus. The SSD1306 needs to be slowed down to 1MHz, so we pass in the additional baudrate parameter.

```python
spi = board.SPI()
oled_cs = board.D5
oled_dc = board.D6
display_bus = displayio.FourWire(spi, command=oled_dc, chip_select=oled_cs,
                                 reset=oled_reset, baudrate=1000000)
```

In order to make it easy to change display sizes, we'll define a few variables in one spot here. We have the display width, the display height and the border size, which we will explain a little further below. If your display is something different than these numbers, change them to the correct setting.

```
WIDTH = 128  
HEIGHT = 32   # Change to 64 if needed
BORDER = 5
```

Finally, we initialize the driver with a width of the `WIDTH` variable and a height of the `HEIGHT` variable. If we stopped at this point and ran the code, we would have a terminal that we could type at and have the screen update.

```python
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=WIDTH, height=HEIGHT)
```
Next we create a background splash image. We do this by creating a group that we can add elements to and adding that group to the display. In this example, we are limiting the maximum number of elements to 10, but this can be increased if you would like. The display will automatically handle updating the group.

```python
splash = displayio.Group(max_size=10)
display.show(splash)
```

Next we create a Bitmap that is the full width and height of the display. The Bitmap is like a canvas that we can draw on. In this case we are creating the Bitmap to be the same size as the screen, but only have one color. Although the Bitmaps can handle up to 256 different colors, the display is monochrome so we only need one. We create a Palette with one color and set that color to `0xFFFFFF` which happens to be white. If were to place a different color here, `displayio` handles color conversion automatically, so it may end up black or white depending on the calculation.

```python
color_bitmap = displayio.Bitmap(WIDTH, HEIGHT, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF # White
```

With all those pieces in place, we create a TileGrid by passing the bitmap and palette and draw it at `(0, 0)` which represents the display’s upper left.

```python
bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)
splash.append(bg_sprite)
```
Next we will create a smaller black rectangle. The easiest way to do this is to create a new bitmap that is a little smaller than the full screen with a single color of \texttt{0x000000} , which is black, and place it in a specific location. In this case, we will create a bitmap that is 5 pixels smaller on each side. This is where the \texttt{BORDER} variable comes into use. It makes calculating the size of the second rectangle much easier. The screen we're using here is \texttt{128x64} and we have the \texttt{BORDER} set to \texttt{5} , so we'll want to subtract 10 from each of those numbers.

We'll also want to place it at the position \texttt{(5, 5)} so that it ends up centered.

```python
# Draw a smaller inner rectangle
inner_bitmap = displayio.Bitmap(WIDTH-BORDER*2, HEIGHT-BORDER*2, 1)
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000 # Black
inner_sprite = displayio.TileGrid(inner_bitmap,
    pixel_shader=inner_palette,
    x=BORDER, y=BORDER)
splash.append(inner_sprite)
```

Since we are adding this after the first square, it's automatically drawn on top. Here's what it looks like now.
Next add a label that says "Hello World!" on top of that. We're going to use the built-in Terminal Font. In this example, we won't be doing any scaling because of the small resolution, so we'll add the label directly the main group. If we were scaling, we would have used a subgroup.

Labels are centered vertically, so we'll place it at half the HEIGHT for the Y coordinate and subtract one so it looks good. We use the `//` operator to divide because we want a whole number returned and it's an easy way to round it. We'll set the width to around 28 pixels make it appear to be centered horizontally, but if you want to change the text, change this to whatever looks good to you. Let's go with some white text, so we'll pass it a value of `0xFFFFFF`.

```python
# Draw a label
text = "Hello World!"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFFFF, x=28, y=HEIGHT//2-1)
splash.append(text_area)
```

Finally, we place an infinite loop at the end so that the graphics screen remains in place and isn't replaced by a terminal.

```python
while True:
    pass
```
If you've been following along with a FeatherWing or 128x32 OLED, this is what it should look like:

Where to go from here

Be sure to check out this excellent guide to CircuitPython Display Support Using displayio (https://adafruit.it/EGh)
Python Wiring

It's easy to use OLEDs with Python and the Adafruit CircuitPython SSD1306 module. This module allows you to easily write Python code to control the display.

We'll cover how to wire the OLED to your Raspberry Pi. First assemble your OLED.

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported.

Connect the OLED as shown below to your Raspberry Pi.

Adafruit PIOLED

- The PiOLED comes fully assembled. Simply plug into any Raspberry Pi as shown.

Adafruit 128x64 OLED Bonnet for Raspberry Pi

- The OLED Bonnet comes fully assembled. Simply plug into the Raspberry Pi as shown.

Adafruit 128x32 I2C OLED Display
Adafruit 0.96" or 1.3" 128x64 OLED Display - I2C Wiring

- Pi 3.3V to OLED VIN
- Pi GND to OLED GND
- Pi SCL to OLED SCL
- Pi SDA to OLED SDA
- Pi GPIO4 to OLED RST (or any available GPIO pin)

You must solder two jumpers closed on the back of the display to use with I2C!

- Pi 3.3V to OLED Vin
- Pi GND to OLED Gnd
- Pi SCL to OLED Clk
- Pi SDA to OLED Data
- Pi GPIO4 to OLED Rst (or any available GPIO pin)
Adafruit 128x32 SPI OLED Display

- Pi 3.3V to OLED VIN
- Pi GND to OLED GND
- Pi MOSI to OLED DATA
- Pi SCLK to OLED CLK
- Pi GPIO4 to OLED RST (or any available GPIO pin)
- Pi GPIO5 to OLED CS (or any available GPIO pin)
- Pi GPIO6 to OLED DC (or any available GPIO pin)

Adafruit 0.96" or 1.3" 128x64 OLED Display - SPI Wiring
- Pi 3.3V to OLED VIN
- Pi GND to OLED GND
- Pi MOSI to OLED DATA
- Pi SCLK to OLED CLK
- Pi GPIO4 to OLED RST (or any available GPIO pin)
- Pi GPIO5 to OLED CS (or any available GPIO pin)
- Pi GPIO6 to OLED DC (or any available GPIO pin)
Python Setup

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN).

Python Installation of SSD1306 Library

Once that's done, from your command line run the following command:

• sudo pip3 install adafruit-circuitpython-ssd1306

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

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

• sudo apt-get install python3-pip

Pillow Library

We also need PIL, the Python Imaging Library, 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

That's it. You should be ready to go.

Speeding up the Display on Raspberry Pi

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.
Python Usage

It's easy to use OLEDs with Python and the Adafruit CircuitPython SSD1306 module. This module allows you to easily write Python code to control the display.

You can use this sensor with any computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library.

To demonstrate the usage, we'll initialize the library and use Python code to control the OLED from the board's Python REPL.

Since we are running full CPython on our Linux/computer, we can take advantage of the powerful Pillow image drawing library to handle text, shapes, graphics, etc. Pillow is a gold standard in image and graphics handling, you can read about all it can do here.

I2C Initialization

If your display is connected to the board using I2C (like if using a PiOLED or Bonnet) you'll first need to initialize the I2C bus. First import the necessary modules:

```python
import board
import busio
```

Now for either board run this command to create the I2C instance using the default SCL and SDA pins of your I2C host:

```python
i2c = busio.I2C(board.SCL, board.SDA)
```

After initializing the I2C interface for your firmware as described above you can create an instance of the SSD1306 I2C driver by running:

```python
import adafruit_ssd1306
oled = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
```

Note that the first two parameters to the `SSD1306_I2C` class initializer are the width and height of the display in pixels. Be sure to use the right values for the display you're using!

128 x 64 size OLEDs (or changing the I2C address)

If you are using a 128x64 display, the I2C address is probably different (0x3d), unless you've changed it by soldering some jumpers:

```python
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3d)
```

Adding hardware reset pin

If you have a reset pin (which may be required if your OLED does not have an auto-reset chip like the FeatherWing) also pass in a reset pin like so:

```python
oled = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3d, reset=reset)
```
import digitalio
define the Reset Pin
reset_pin = digitalio.DigitalInOut(board.D4) # any pin!

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)!

Example Code

```
import board
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306

# Define the Reset Pin
oled_reset = digitalio.DigitalInOut(board.D4)

# Change these
# to the right size for your display!
WIDTH = 128
HEIGHT = 32  # Change to 64 if needed
BORDER = 5
```
# Use for I2C.
i2c = board.I2C()
oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=0x3c, reset=oled_reset)

# Use for SPI
#spi = board.SPI()
#oled_cs = digitalio.DigitalInOut(board.D5)
#oled_dc = digitalio.DigitalInOut(board.D6)
#echoled = adafruit_ssd1306.SSD1306_SPI(WIDTH, HEIGHT, spi, oled_dc, oled_reset, oled_cs)

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

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

draw = ImageDraw.Draw(image)

# Draw a white background
draw.rectangle((0, 0, oled.width, oled.height), outline=255, fill=255)

# Draw a smaller inner rectangle
draw.rectangle((BORDER, BORDER, oled.width - BORDER - 1, oled.height - BORDER - 1),
               outline=0, fill=0)

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

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text((oled.width//2 - font_width//2, oled.height//2 - font_height//2),
          text, font=font, fill=255)

# Display image
oled.image(image)
oled.show()

Let's take a look at the sections of code one by one. We start by importing the board so that we can initialize SPI, digitalio, several PIL modules for Image Drawing, and the adafruit_ssd1306 driver.

import board
import digitalio
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306

Next we define the reset line, which will be used for either SPI or I2C. If your OLED has auto-reset circuitry, you can set the oled_reset line to None

oled_reset = digitalio.DigitalInOut(board.D4)

In order to make it easy to change display sizes, we'll define a few variables in one spot here. We have the display
width, the display height and the border size, which we will explain a little further below. If your display is something
different than these numbers, change them to the correct setting.

| WIDTH = 128 |
| HEIGHT = 32  | # Change to 64 if needed |
| BORDER = 5   |

If you're using I2C, you would use this section of code. We set the I2C object to the board's I2C with the easy shortcut
function  \texttt{board.I2C()}. By using this function, it finds the SPI module and initializes using the default SPI
parameters. We also set up the oled with SSD1306\_I2C which makes use of the I2C bus.

\begin{verbatim}
# Use for I2C.
i2c = board.I2C()
oled = adafruit_ssd1306.SSD1306\_I2C(WIDTH, HEIGHT, i2c, addr=0x3c, reset=oled\_reset)
\end{verbatim}

If you're using SPI, you would use this section of code. We set the SPI object to the board's SPI with the easy shortcut
function  \texttt{board.SPI()}. By using this function, it finds the SPI module and initializes using the default SPI parameters. We
set the OLED's \texttt{CS} (Chip Select), and \texttt{DC} (Data/Command) pins. We also set up the OLED with SSD1306\_SPI which
makes use of the SPI bus.

\begin{verbatim}
# Use for SPI
spi = board.SPI()
oled\_cs = digitalio.DigitalInOut(board.D5)
oled\_dc = digitalio.DigitalInOut(board.D6)
oled = adafruit_ssd1306.SSD1306\_SPI(WIDTH, HEIGHT, spi, oled\_dc, oled\_reset, oled\_cs)
\end{verbatim}

Next we clear the display in case it was initialized with any random artifact data.

\begin{verbatim}
# Clear display.
oled.fill(0)
oled.show()
\end{verbatim}

Next, we need to initialize PIL to create a blank image to draw on. Think of it as a virtual canvas. Since this is a
monochrome display, we set it up for 1-bit color, meaning a pixel is either white or black. We can make use of the
OLED's width and height properties as well. Optionally, we could have used our \texttt{WIDTH} and \texttt{HEIGHT} variables.

\begin{verbatim}
# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
image = Image.new('1', (oled.width, oled.height))
# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)
\end{verbatim}

Now we start the actual drawing. Here we are telling it we want to draw a rectangle from \((0,0)\), which is the upper left,
to the full width and height of the display. We want it both filled in and having an outline of white, so we pass 255 for
both numbers.

\begin{verbatim}
# Draw a white background
draw.rectangle((0, 0, oled.width, oled.height), outline=255, fill=255)
\end{verbatim}
If we ran the code now, it would still show a blank display because we haven't told python to use our virtual canvas yet. You can skip to the end if you would like to see how to do that. This is what our canvas currently looks like in memory.

Next we will create a smaller black rectangle. The easiest way to do this is to draw another rectangle a little smaller than the full screen with no fill or outline and place it in a specific location. In this case, we will create a rectangle that is 5 pixels smaller on each side. This is where the BORDER variable comes into use. It makes calculating the size of the second rectangle much easier. We want the starting coordinate, which consists of the first two parameters, to be our BORDER value. Then for the next two parameters, which are our ending coordinates, we want to subtract our border value from the width and height. Also, because this is a zero-based coordinate system, we also need to subtract 1 from each number.

```python
# Draw a smaller inner rectangle
draw.rectangle((BORDER, BORDER, oled.width - BORDER - 1, oled.height - BORDER - 1),
                outline=0, fill=0)
```

Here's what our virtual canvas looks like in memory.
Now drawing text with PIL is pretty straightforward. First we start by setting the font to the default system text. After that we define our text and get the size of the text. We’re grabbing the size that it would render at so that we can calculate the center position. Finally, we take the font size and screen size to calculate the position we want to draw the text at and it appears in the center of the screen.

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

# Draw Some Text
text = "Hello World!"
(font_width, font_height) = font.getsize(text)
draw.text((oled.width//2 - font_width//2, oled.height//2 - font_height//2),
          text, font=font, fill=255)
```

Finally, we need to display our virtual canvas to the OLED and we do that with 2 commands. First we set the image to the screen, then we tell it to show the image.

```python
# Display image
oled.image(image)
oled.show()
```

Don't forget you MUST call oled.image(image) and oled.show() to actually display the graphics. The OLED takes a while to draw so cluster all your drawing functions into the buffer (fast) and then display them once to the oled (slow)

Here's what the final output should look like.
Troubleshooting

Display does not work on initial power but does work after a reset.

The OLED driver circuit needs a small amount of time to be ready after initial power. If your code tries to write to the display too soon, it may not be ready. It will work on reset since that typically does not cycle power. If you are having this issue, try adding a small amount of delay before trying to write to the OLED.

In Arduino, use `delay()` to add a few milliseconds before calling `oled.begin()`. Adjust the amount of delay as needed to see how little you can get away with for your specific setup.
Display is showing burn in on some pixels.

The display can have image burn in for any pixels left on over a long period of time - many days. Try to avoid having the display on constantly for that length of time.
Downloads

Software

You can download our SSD1306 OLED display Arduino library from github (https://adafruit.it/aHq) which comes with example code. The library can print text, bitmaps, pixels, rectangles, circles and lines. It uses 1K of RAM since it needs to buffer the entire display but its very fast! The code is simple to adapt to any other microcontroller. You’ll also have to install the Adafruit GFX graphics core library at this github repo (https://adafruit.it/aJa) and install it after you’ve gotten the OLED driver library.

Datasheets

- UG-2864HSWEG01 (https://adafruit.it/aJI) Datasheet
- UG-2832HSWEG02 (https://adafruit.it/qrf) Datasheet
- UG-2864HSWEG01 (https://adafruit.it/wWD) User Guide
- UG-2832HSWEG04 (https://adafruit.it/qVA) Datasheet
- SSD1306 (https://adafruit.it/aJK) Datasheet

Files

- EagleCAD PCB files for 128x32 0.91" SPI display PCB (https://adafruit.it/aJL)
- EagleCAD PCB files for 128x32 0.91" I2C display on GitHub (https://adafruit.it/rPF)
- EagleCAD PCB files for 128x64 0.96" display on GitHub (https://adafruit.it/eJM)
- EagleCAD PCB files for 128x64 1.3" display on GitHub (https://adafruit.it/rJe)
- Fritzing objects available in the Adafruit Fritzing Library (https://adafruit.it/aP3)

Schematic & Fabrication Print for 0.96" OLED
Schematic & Fabrication Print for 1.3" OLED
Schematic & Fabrication Print for 0.91" 128x32 SPI