Monochrome OLED Breakouts

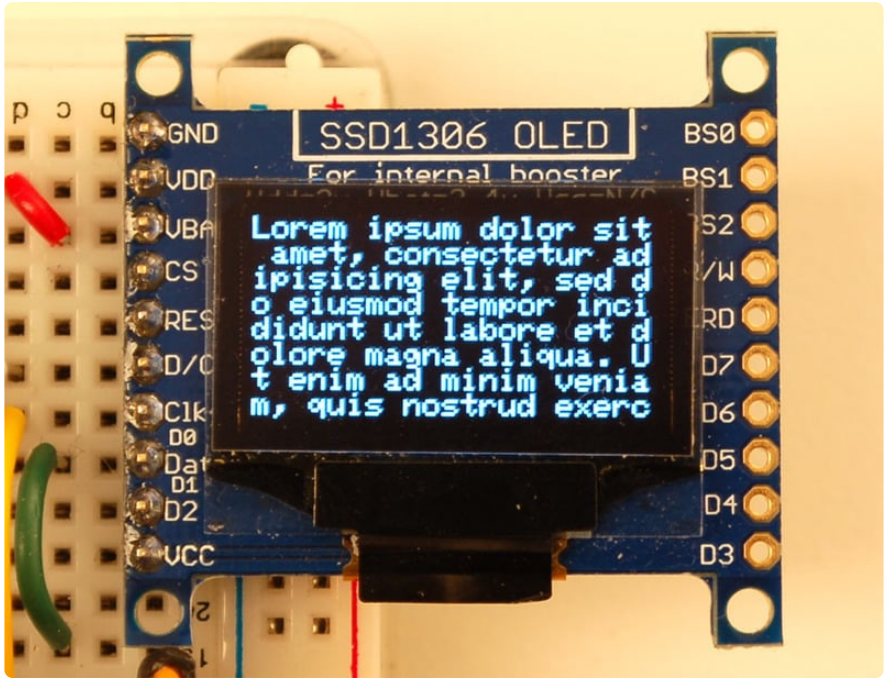
Created by lady ada

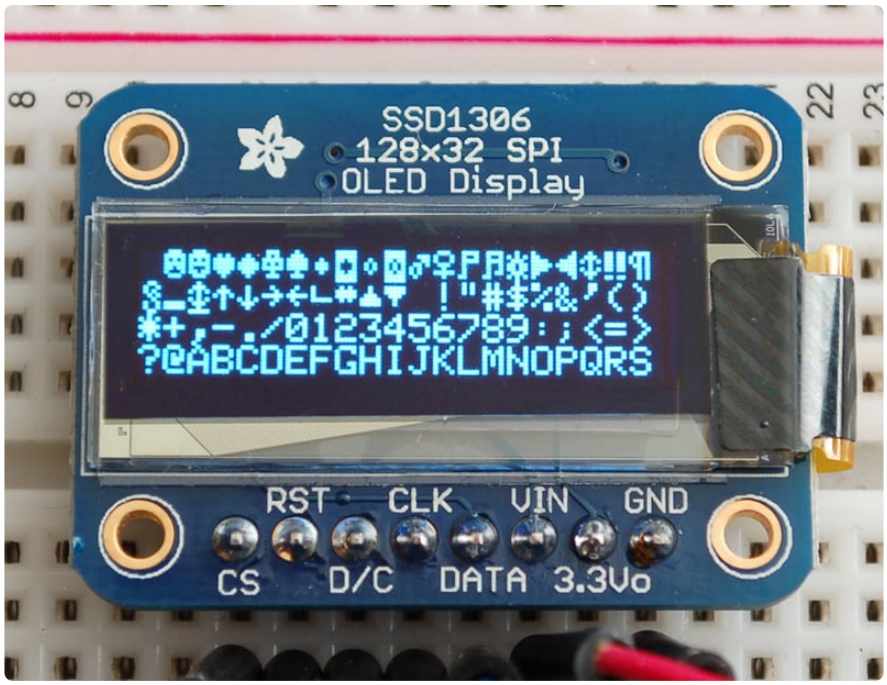
https://learn.adafruit.com/monochrome-oled-breakouts

Last updated on 2024-07-17 07:38:18 PM EDT

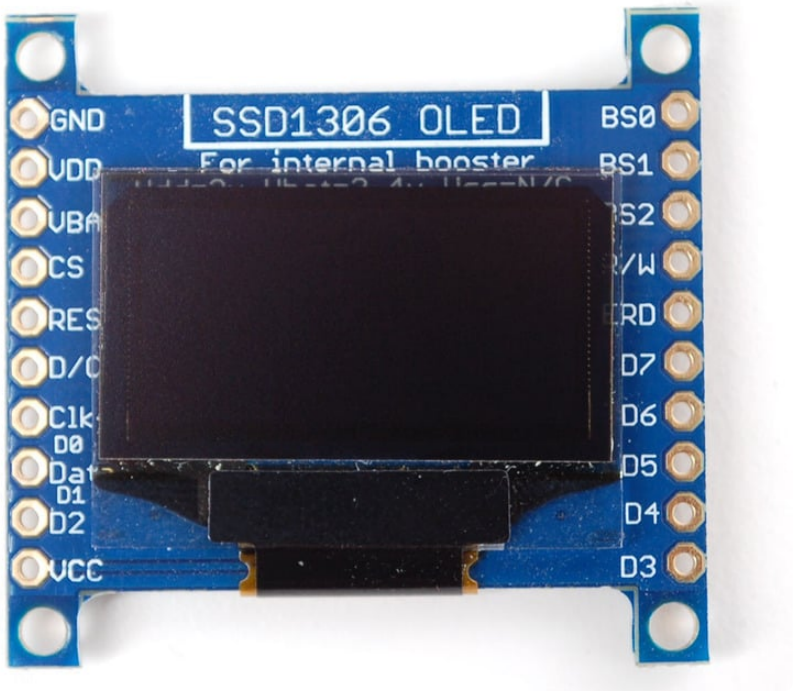
Table of Contents

Overview



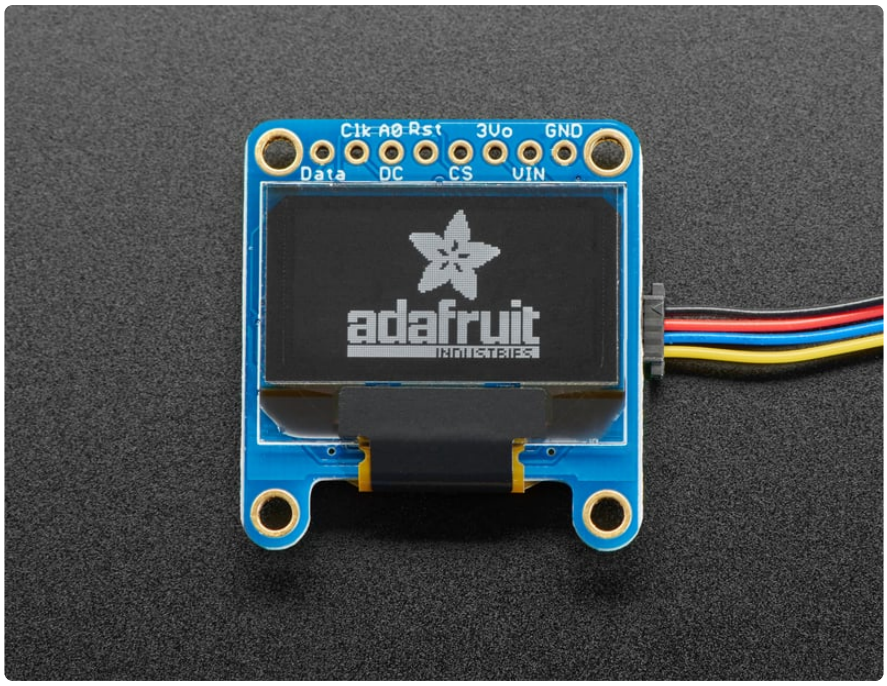


This is a quick tutorial for our 128x64 and 128x32 pixel monochrome OLED displays. These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. Each OLED display is made of 128x64 or 128x32 individual white OLEDs, each one is turned on or off by the 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.

For the 0.96" STEMMA QT version, **we've updated the design to add auto-reset circuitry** so that the reset pin is optional, since it speaks I2C you can easily connect it up with just two wires (plus power and ground!). We've even included SparkFun qwiic (https://adafru.it/Fpw) compatible STEMMA QT **(https://adafru.it/Ft4)** connectors for the I2C bus so **you don't even need to solder!**



Power Requirements

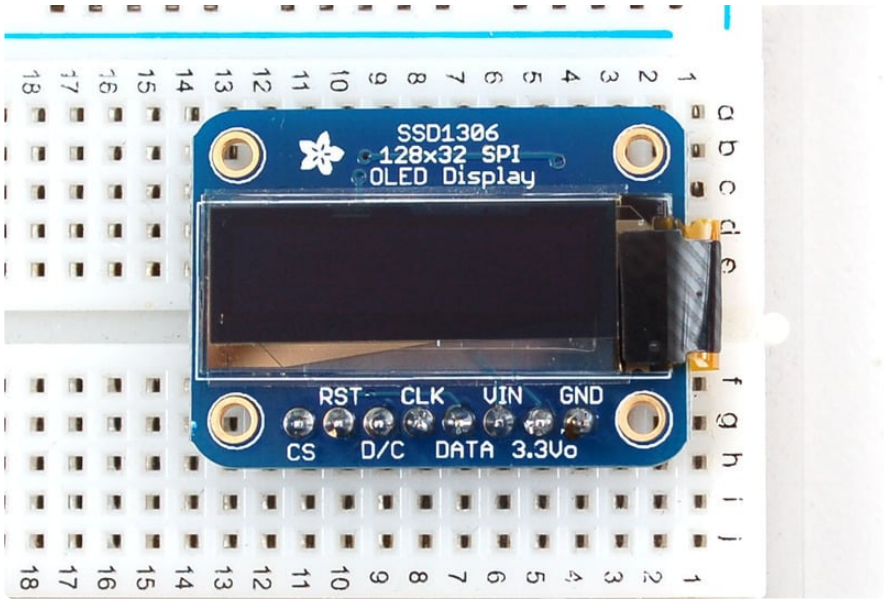
OLED Power Requirements

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 micontroller. 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.

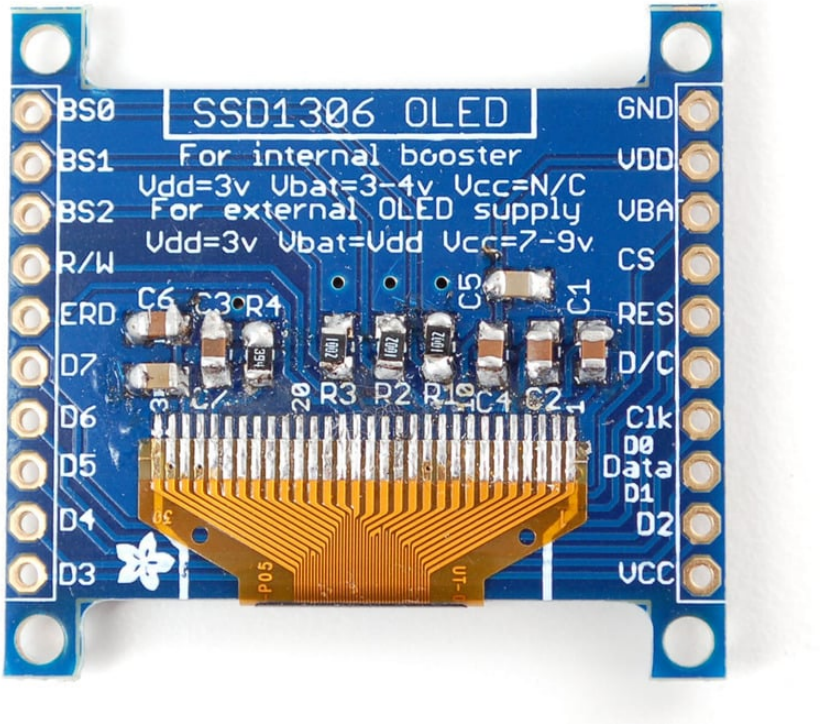
All OLEDs are safe to use with 3.3V logic and power.



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

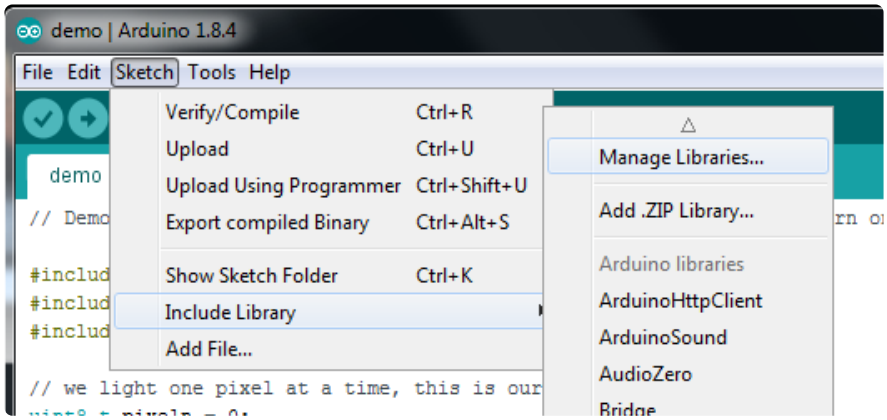
For all of the different kinds of small OLED monochrome displays, you'll need to install the Arduino libraries. The code we have is for any kind of Arduino, if you're using a different microcontroller, the code is pretty simple to adapt, the interface we use is basic bit-twiddling SPI or I2C

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.

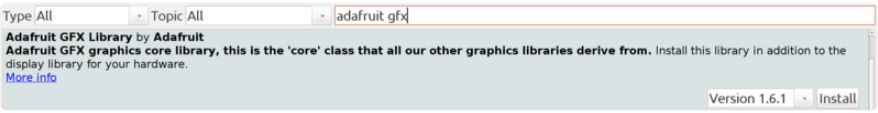
In recent versions of the Arduino IDE software (1.6.2 and later), this is most easily done through the Arduino Library Manager.

Open up the Arduino library manager:



Search for the **Adafruit SSD1306** library and install it

Search for the **Adafruit GFX** library and install it

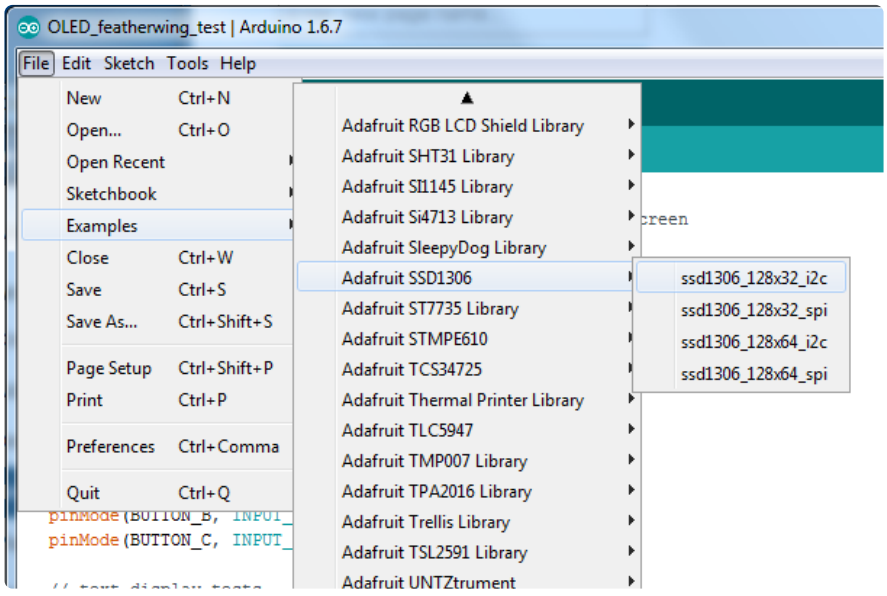


If using an earlier version of the Arduino IDE (prior to 1.8.10), also locate and install **Adafruit_BusIO** (newer versions will install this dependency automatically).

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

Run Demo!

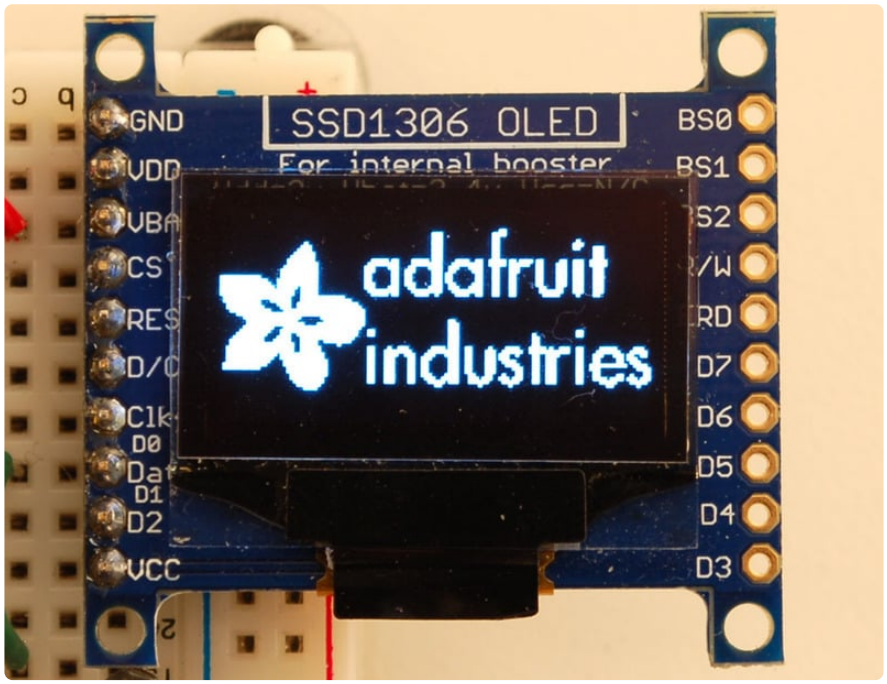
After installing the **Adafruit_SSD1306** and **Adafruit_GFX** library, restart the Arduino IDE. You should now be able to access the sample code by navigating through menus in this order: **File→Examples→Adafruit_SSD1306→SSD1306...**

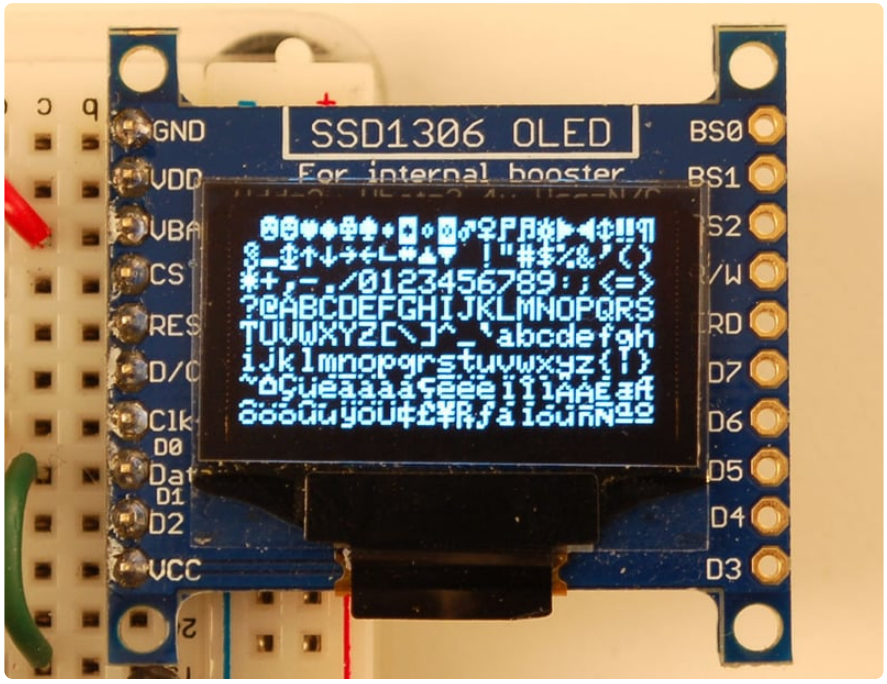


After you've finished wiring the display as indicated on the following pages, load the example sketch to demonstrate the capabilities of the library and display.

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

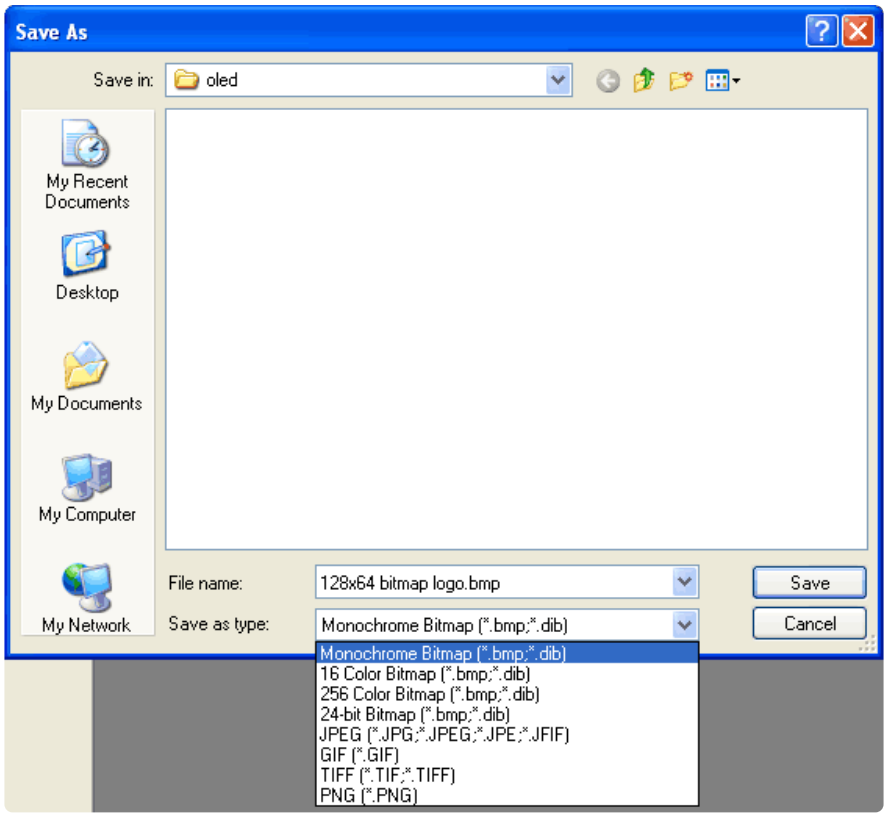
details about what you can do with the OLED check out the GFX library tutorial (https://adafru.it/aPx)



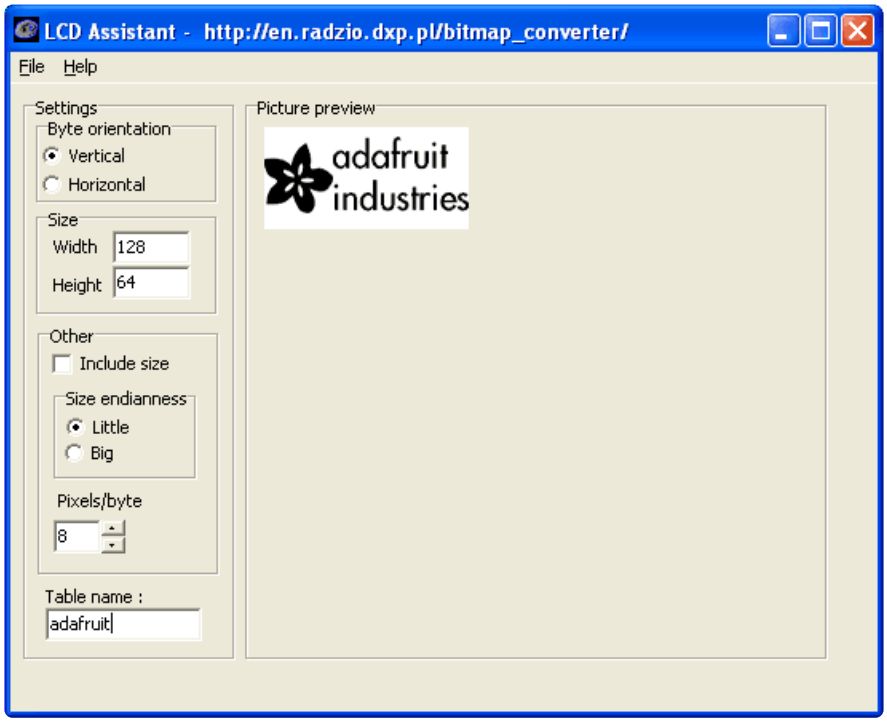


Create Bitmaps with LCD Assistant

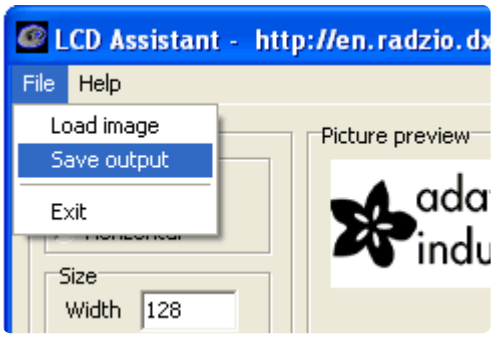
You can create bitmaps to display easily with the LCD assistant software (https://adafru.it/aPs). First make your image using any kind of graphics software such as photoshop or Paint and save as a **Monochrome Bitmap (bmp)**



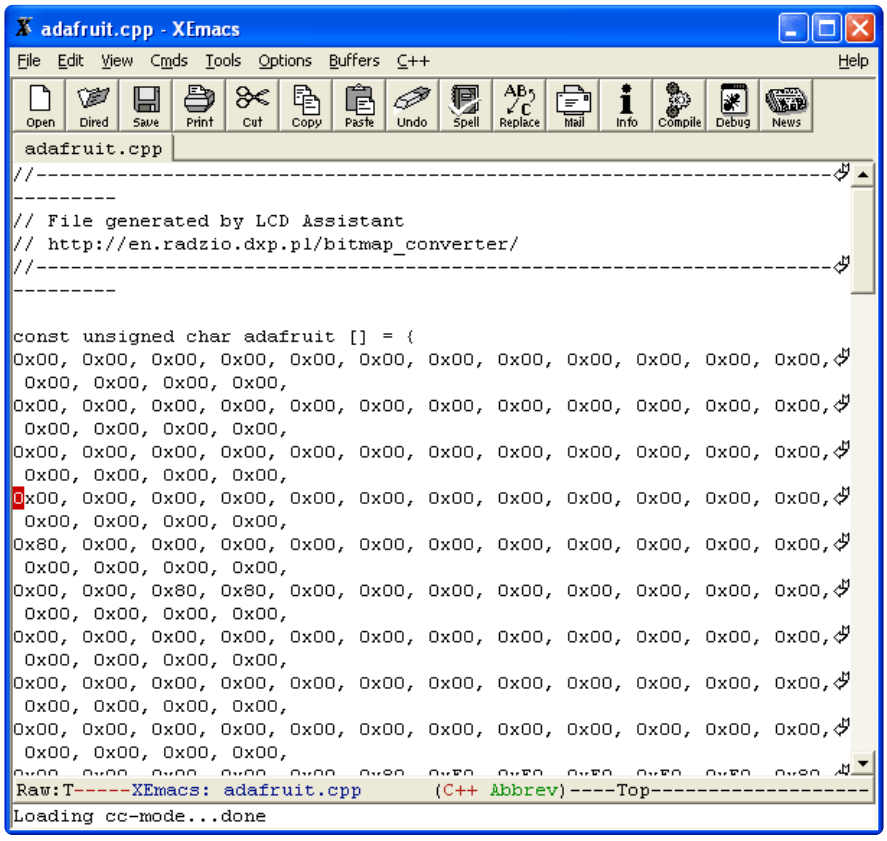
Select the following options (You might also want to try **Horizontal** if **Vertical** is not coming out right)



and import your monochrome bitmap image. Save the output to a **cpp** file



You can use the output directly with our example code



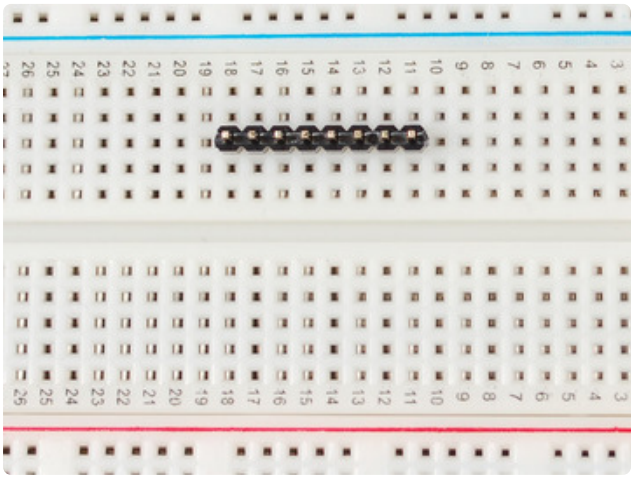
Create Bitmaps with image2cpp

image2cpp was created by GitHub user javl and provides a handy way to create bitmaps without installing any additional software. Just visit https://javl.github.io/image2cpp/, (https://adafru.it/11bl) upload an image, put in any settings that you would like to use, select a format and generate the code. You can copy the code right into your sketch. If you're interested, you can view the source in their GitHub repository (https://adafru.it/11bJ).

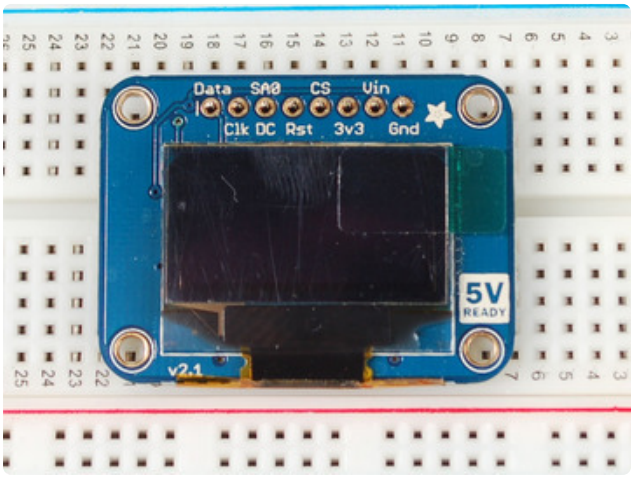
Wiring 128x64 OLEDs

Solder Header

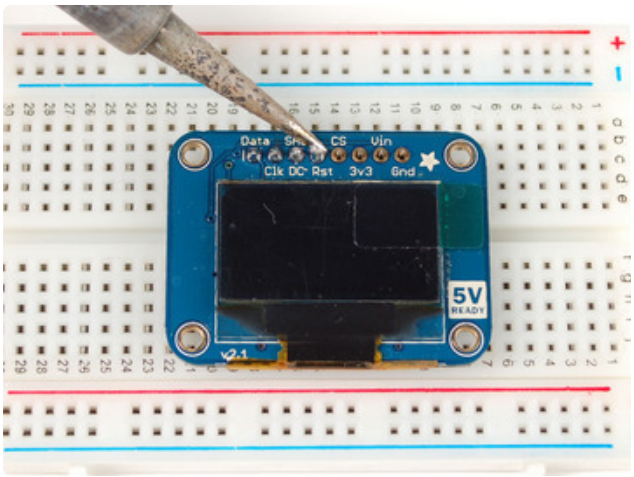
Before you start wiring, a strip of header must be soldered onto the OLED. It is not possible to "press-fit" the header, it must be attached!



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



Finish by soldering each of the 8 pins to the 8 pads!

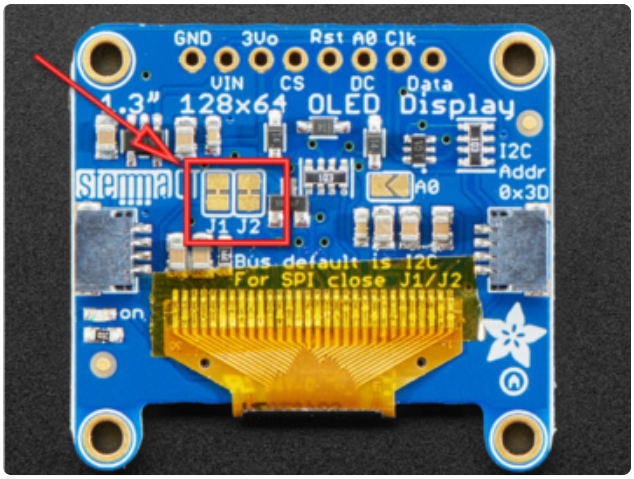
I2C or SPI

The nice thing about the 128x64 OLEDs is that they can be used with I2C (+ an optional 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).

If you have the older non-STEMMA version of the OLED, you'll need to solder the two jumpers on the back of the OLED. **Both** must be soldered 'closed' for I2C to work!



For the new STEMMA-capable version, the **J1** and **J2** jumpers are closed so that the display is **by default in I2C mode!**

There's a typo on the board, to put it into SPI, open the two jumpers (as they're closed by default)

Converting From I2C to SPI Mode

The original version of this display was SPI by default, and you could convert to I2C with some light soldering. Many folks using these displays did not know how to solder, didn't own an iron or were not comfortable with soldering, so we converted the board to STEMMA QT 'plug and play' I2C so no soldering is required to use in I2C mode.

To convert it back to SPI is very easy, and requires a thin screwdriver or other sharp-tipped item **be careful not to cut towards you as always so you do not accidentally cut yourself!**

Wiring It Up!

For the STEMMA QT version of this board, you do not need to connect RST - this revision added auto-reset circuitry so the RESET pin is not required.

For some microcontrollers with on-board STEMMA QT, such as the Adafruit ESP32-S2 feathers, you may need to enable the STEMMA QT pins first. Refer to the guide for the specific board you are using.

Finally, connect the pins to your Arduino

- **GND** goes to ground **(black wire on STEMMA QT version)**
- **Vin** goes to 5V **(red wire on STEMMA QT version)**
- **Data** to I2C SDA (on the Uno, this is **A4** on the Mega it is **20** and on the Leonardo digital **2**) **(blue wire on STEMMA QT version)**
- **Clk** to I2C SCL (on the Uno, this is **A5** on the Mega it is **21** and on the Leonardo digital **3**) **(yellow wire on STEMMA QT version)**
- **RST** to digital **4** (you can change this pin in the code, later) (Not necessary on 0.96" STEMMA QT version)

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. If you wired the RST pin to a GPIO pin, change `#define OLED_RESET -1` to the new pin number.

```
/*************************************************************************
 This is an example for our Monochrome OLEDs based on SSD1306 drivers

 Pick one up today in the adafruit shop!
 ------> http://www.adafruit.com/category/63_98

 This example is for a 128x64 pixel display using I2C to communicate
 3 pins are required to interface (two I2C and one reset).

 Adafruit invests time and resources providing this open
 source code, please support Adafruit and open-source
 hardware by purchasing products from Adafruit!

 Written by Limor Fried/Ladyada for Adafruit Industries,
 with contributions from the open source community.
 BSD license, check license.txt for more information
 All text above, and the splash screen below must be
 included in any redistribution.
 *************************************************************************/

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
// The pins for I2C are defined by the Wire-library.
// On an arduino UNO:       A4(SDA), A5(SCL)
// On an arduino MEGA 2560: 20(SDA), 21(SCL)
// On an arduino LEONARDO:   2(SDA),  3(SCL), ...
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3D ///< See datasheet for Address; 0x3D for 128x64, 0x3C
for 128x32
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
#define LOGO_WIDTH    16
static const unsigned char PROGMEM logo_bmp[] =
{ 0b00000000, 0b11000000,
  0b00000001, 0b11000000,
  0b00000001, 0b11000000,
  0b00000011, 0b11100000,
  0b11110011, 0b11100000,
  0b11111110, 0b11111000,
  0b01111110, 0b11111111,
  0b00110011, 0b10011111,
  0b00011111, 0b11111100,
  0b00001101, 0b01110000,
  0b00011011, 0b10100000,
  0b00111111, 0b11100000,
  0b00111111, 0b11110000,
  0b01111100, 0b11110000,
  0b01110000, 0b01110000,
  0b00000000, 0b00110000 };

void setup() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
```

```
  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...

  testdrawline();      // Draw many lines

  testdrawrect();      // Draw rectangles (outlines)

  testfillrect();      // Draw rectangles (filled)

  testdrawcircle();    // Draw circles (outlines)

  testfillcircle();    // Draw circles (filled)

  testdrawroundrect(); // Draw rounded rectangles (outlines)

  testfillroundrect(); // Draw rounded rectangles (filled)

  testdrawtriangle();  // Draw triangles (outlines)

  testfilltriangle();  // Draw triangles (filled)

  testdrawchar();      // Draw characters of the default font

  testdrawstyles();    // Draw 'stylized' characters

  testscrolltext();    // Draw scrolling text

  testdrawbitmap();    // Draw a small bitmap image

  // Invert and restore display, pausing in-between
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}

void loop() {
}

void testdrawline() {
  int16_t i;

  display.clearDisplay(); // Clear display buffer

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, 0, i, display.height()-1, SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn line
    delay(1);
  }
```

```
  for(i=0; i<display.height(); i+=4) {
    display.drawLine(0, 0, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(0, display.height()-1, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=display.width()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000); // Pause for 2 seconds
}

void testdrawrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i,
SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testfillrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=3) {
    // The INVERSE color is used so rectangles alternate white/black
```

```
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2,
SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testdrawcircle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillcircle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
    // The INVERSE color is used so circles alternate white/black
    display.fillCircle(display.width() / 2, display.height() / 2, i,
SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn circle
    delay(1);
  }

  delay(2000);
}

void testdrawroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    // The INVERSE color is used so round-rects alternate white/black
    display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawtriangle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
    display.drawTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
```

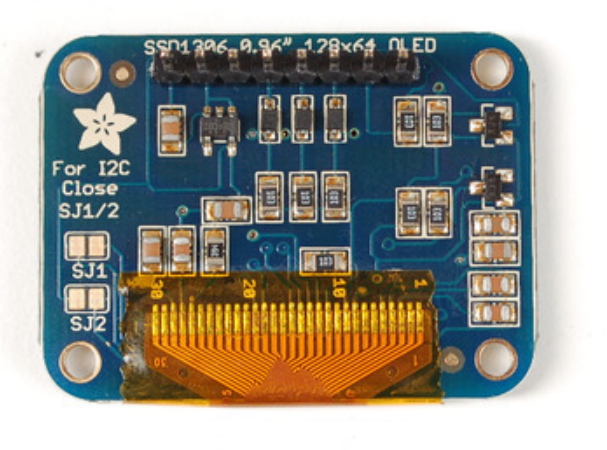
```
      display.width()/2+i, display.height()/2+i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfilltriangle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
    // The INVERSE color is used so triangles alternate white/black
    display.fillTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawchar(void) {
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.cp437(true);         // Use full 256 char 'Code Page 437' font

  // Not all the characters will fit on the display. This is normal.
  // Library will draw what it can and the rest will be clipped.
  for(int16_t i=0; i<256; i++) {
    if(i == '\n') display.write(' ');
    else          display.write(i);
  }

  display.display();
  delay(2000);
}

void testdrawstyles(void) {
  display.clearDisplay();

  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));

  display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
  display.println(3.141592);

  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);

  display.display();
  delay(2000);
}

void testscrolltext(void) {
  display.clearDisplay();

  display.setTextSize(2); // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 0);
  display.println(F("scroll"));
```

```
  display.display();      // Show initial text
  delay(100);

  // Scroll in various directions, pausing in-between:
  display.startscrollright(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
}

void testdrawbitmap(void) {
  display.clearDisplay();

  display.drawBitmap(
    (display.width()  - LOGO_WIDTH ) / 2,
    (display.height() - LOGO_HEIGHT) / 2,
    logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
  display.display();
  delay(1000);
}

#define XPOS   0 // Indexes into the 'icons' array in function below
#define YPOS   1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  int8_t f, icons[NUMFLAKES][3];

  // Initialize 'snowflake' positions
  for(f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
    icons[f][YPOS]   = -LOGO_HEIGHT;
    icons[f][DELTAY] = random(1, 6);
    Serial.print(F("x: "));
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(F(" y: "));
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(F(" dy: "));
    Serial.println(icons[f][DELTAY], DEC);
  }

  for(;;) { // Loop forever...
    display.clearDisplay(); // Clear the display buffer

    // Draw each snowflake:
    for(f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h,
SSD1306_WHITE);
    }

    display.display(); // Show the display buffer on the screen
    delay(200);        // Pause for 1/10 second

    // Then update coordinates of each flake...
    for(f=0; f< NUMFLAKES; f++) {
      icons[f][YPOS] += icons[f][DELTAY];
      // If snowflake is off the bottom of the screen...
      if (icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
```

```
        icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
```

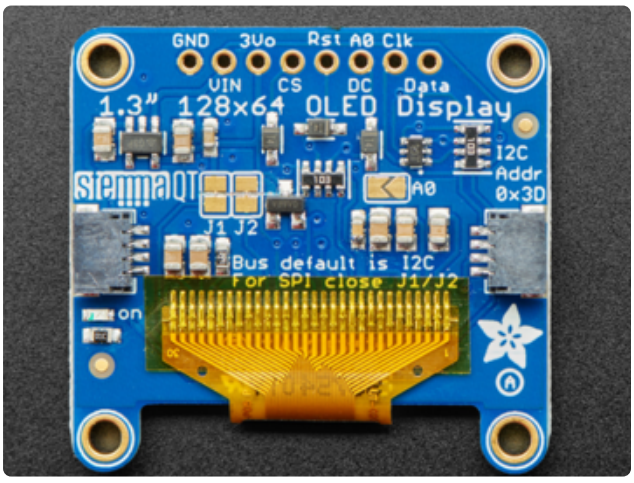
Using with SPI

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!



If you have the older non-STEMMA version of the OLED, the breakouts are ready for SPI by default.

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!



If you have the newer STEMMA QT version **cut** the two jumpers instead!

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

```
/*************************************************************************
 This is an example for our Monochrome OLEDs based on SSD1306 drivers

 Pick one up today in the adafruit shop!
 ------> http://www.adafruit.com/category/63_98

 This example is for a 128x64 pixel display using SPI to communicate
 4 or 5 pins are required to interface.

 Adafruit invests time and resources providing this open
 source code, please support Adafruit and open-source
 hardware by purchasing products from Adafruit!

 Written by Limor Fried/Ladyada for Adafruit Industries,
 with contributions from the open source community.
 BSD license, check license.txt for more information
 All text above, and the splash screen below must be
 included in any redistribution.
 *************************************************************************/

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for SSD1306 display connected using software SPI (default case):
#define OLED_MOSI   9
#define OLED_CLK   10
#define OLED_DC    11
#define OLED_CS    12
#define OLED_RESET 13
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
  OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);

/* Comment out above, uncomment this block to use hardware SPI
#define OLED_DC     6
#define OLED_CS     7
#define OLED_RESET  8
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
  &SPI, OLED_DC, OLED_RESET, OLED_CS);
*/

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
```

```
#define LOGO_WIDTH    16
static const unsigned char PROGMEM logo_bmp[] =
{ 0b00000000, 0b11000000,
  0b00000001, 0b11000000,
  0b00000001, 0b11000000,
  0b00000011, 0b11100000,
  0b11110011, 0b11100000,
  0b11111110, 0b11111000,
  0b01111110, 0b11111111,
  0b00110011, 0b10011111,
  0b00011111, 0b11111100,
  0b00001101, 0b01110000,
  0b00011011, 0b10100000,
  0b00111111, 0b11100000,
  0b00111111, 0b11110000,
  0b01111100, 0b11110000,
  0b01110000, 0b01110000,
  0b00000000, 0b00110000 };

void setup() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }


  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...

  testdrawline();      // Draw many lines

  testdrawrect();      // Draw rectangles (outlines)

  testfillrect();      // Draw rectangles (filled)

  testdrawcircle();    // Draw circles (outlines)

  testfillcircle();    // Draw circles (filled)

  testdrawroundrect(); // Draw rounded rectangles (outlines)

  testfillroundrect(); // Draw rounded rectangles (filled)

  testdrawtriangle();  // Draw triangles (outlines)

  testfilltriangle();  // Draw triangles (filled)

  testdrawchar();      // Draw characters of the default font
```

```
  testdrawstyles();    // Draw 'stylized' characters

  testscrolltext();    // Draw scrolling text

  testdrawbitmap();    // Draw a small bitmap image

  // Invert and restore display, pausing in-between
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}

void loop() {
}

void testdrawline() {
  int16_t i;

  display.clearDisplay(); // Clear display buffer

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, 0, i, display.height()-1, SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn line
    delay(1);
  }
  for(i=0; i<display.height(); i+=4) {
    display.drawLine(0, 0, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(0, display.height()-1, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=display.width()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
```

```
  }
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000); // Pause for 2 seconds
}

void testdrawrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i,
SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testfillrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=3) {
    // The INVERSE color is used so rectangles alternate white/black
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2,
SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testdrawcircle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillcircle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
    // The INVERSE color is used so circles alternate white/black
    display.fillCircle(display.width() / 2, display.height() / 2, i,
SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn circle
    delay(1);
  }

  delay(2000);
}

void testdrawroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_WHITE);
```

```
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    // The INVERSE color is used so round-rects alternate white/black
    display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawtriangle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
    display.drawTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfilltriangle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
    // The INVERSE color is used so triangles alternate white/black
    display.fillTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawchar(void) {
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.cp437(true);         // Use full 256 char 'Code Page 437' font

  // Not all the characters will fit on the display. This is normal.
  // Library will draw what it can and the rest will be clipped.
  for(int16_t i=0; i<256; i++) {
    if(i == '\n') display.write(' ');
    else          display.write(i);
  }

  display.display();
  delay(2000);
```

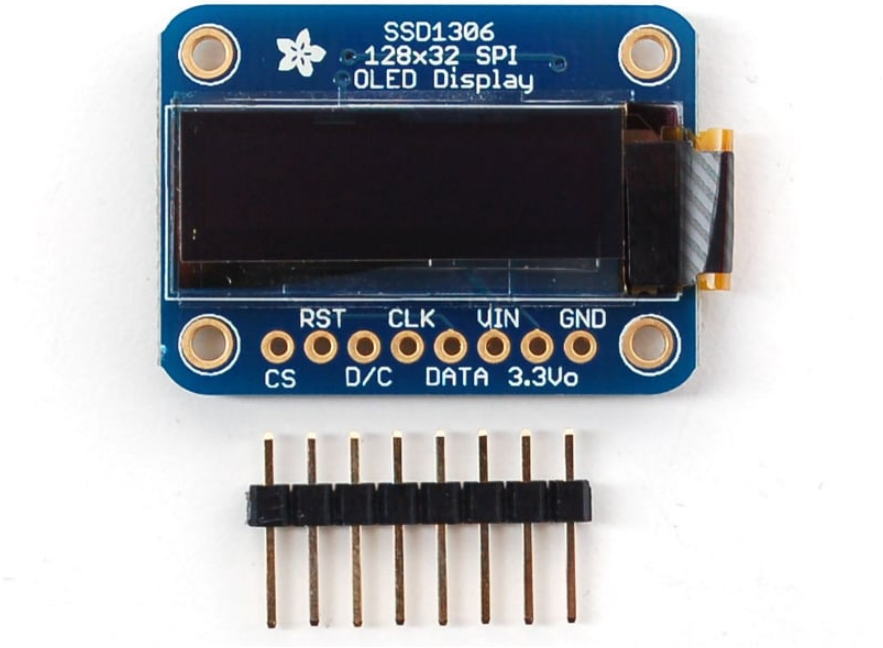
```
}

void testdrawstyles(void) {
  display.clearDisplay();

  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));

  display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
  display.println(3.141592);

  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);

  display.display();
  delay(2000);
}

void testscrolltext(void) {
  display.clearDisplay();

  display.setTextSize(2); // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 0);
  display.println(F("scroll"));
  display.display();      // Show initial text
  delay(100);

  // Scroll in various directions, pausing in-between:
  display.startscrollright(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
}

void testdrawbitmap(void) {
  display.clearDisplay();

  display.drawBitmap(
    (display.width()  - LOGO_WIDTH ) / 2,
    (display.height() - LOGO_HEIGHT) / 2,
    logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
  display.display();
  delay(1000);
}

#define XPOS   0 // Indexes into the 'icons' array in function below
#define YPOS   1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  int8_t f, icons[NUMFLAKES][3];

  // Initialize 'snowflake' positions
  for(f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
```

```
    icons[f][YPOS]   = -LOGO_HEIGHT;
    icons[f][DELTAY] = random(1, 6);
    Serial.print(F("x: "));
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(F(" y: "));
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(F(" dy: "));
    Serial.println(icons[f][DELTAY], DEC);
  }

  for(;;) { // Loop forever...
    display.clearDisplay(); // Clear the display buffer

    // Draw each snowflake:
    for(f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h,
SSD1306_WHITE);
    }

    display.display(); // Show the display buffer on the screen
    delay(200);        // Pause for 1/10 second

    // Then update coordinates of each flake...
    for(f=0; f< NUMFLAKES; f++) {
      icons[f][YPOS] += icons[f][DELTAY];
      // If snowflake is off the bottom of the screen...
      if (icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
        icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
```

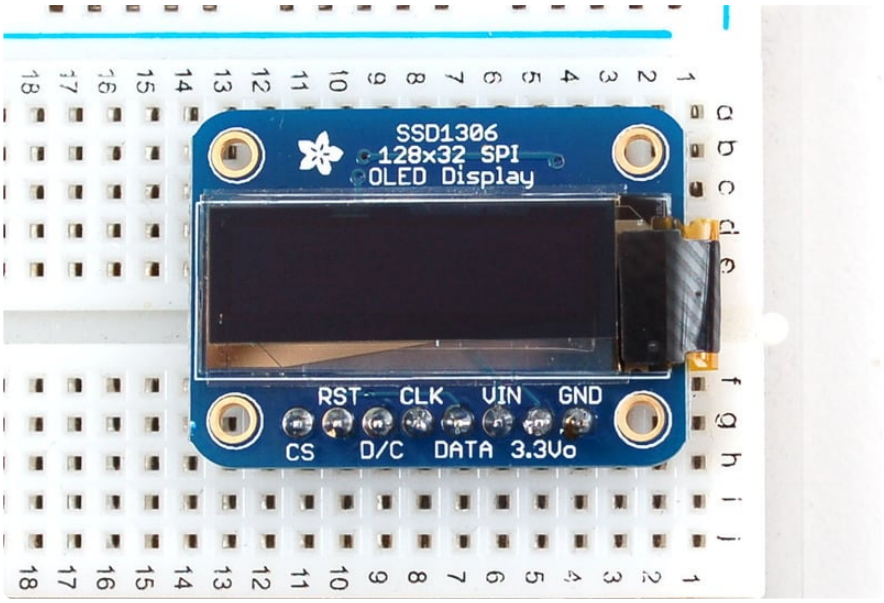
Wiring 128x32 SPI OLED display

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.

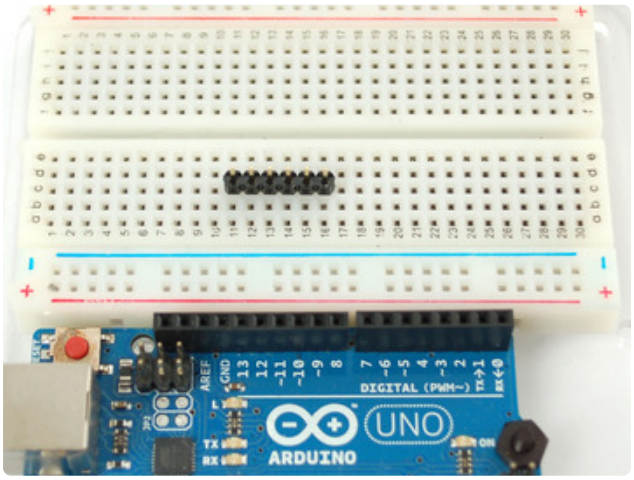
Finally you can run
the **File→Sketchbook→Libraries→Adafruit_SSD1306→SSD1306_128x32_SPI example**

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

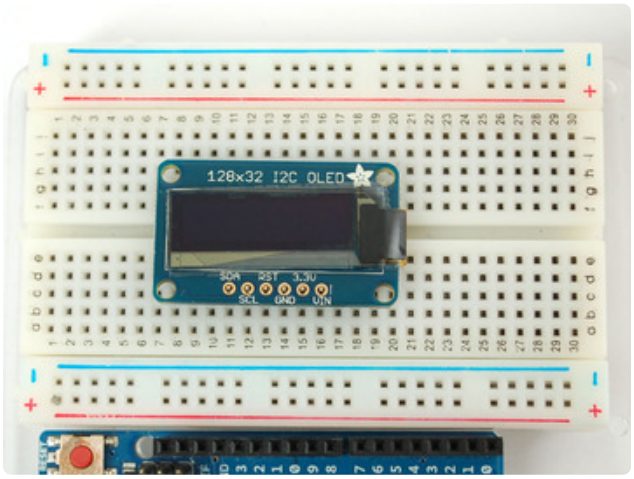
Wiring 128x32 I2C Display

128x32 I2C OLED

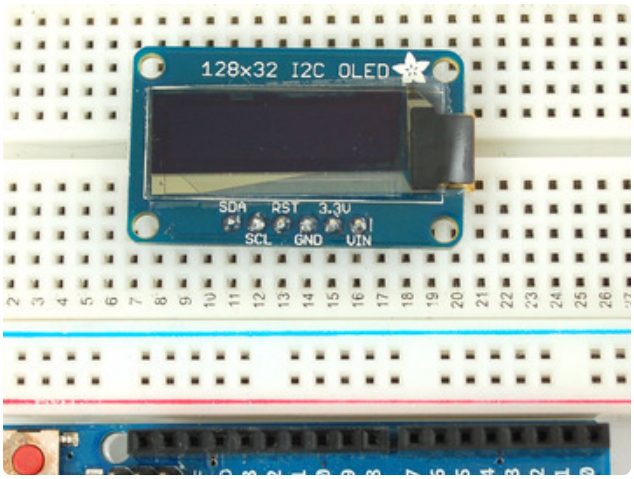
The 128x32 I2C OLED is very easy to get up and running because it has built in level shifting and regulator. First up, take a piece of 0.1" header 6 pins long.



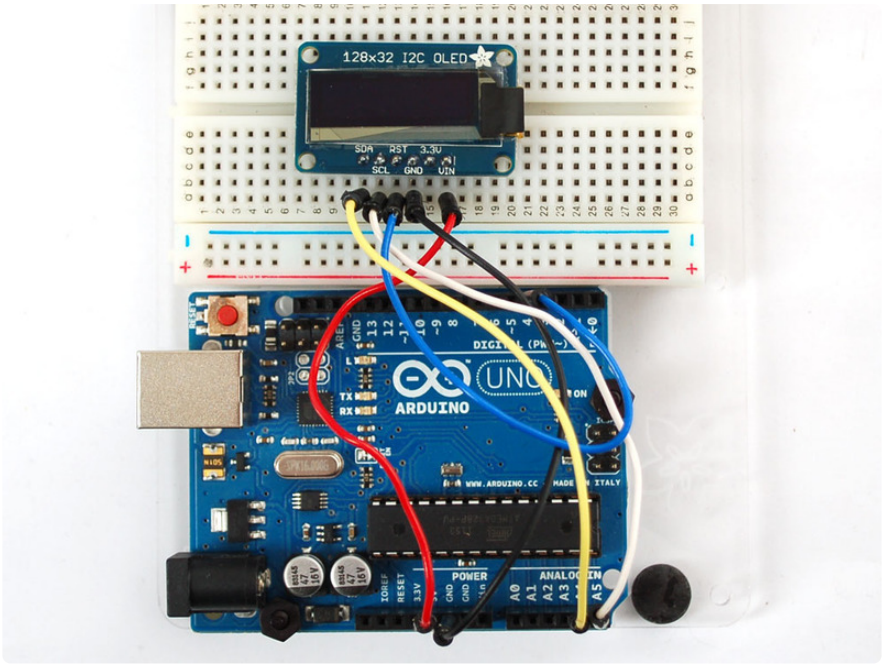
Plug the header long end down into a breadboard



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 **SDA** pin (on the Uno, this is **A4** on the Mega it is **20** and on the Leonardo digital **2**)
- **SCL** to I2C Clock **SCL** pin (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_1**

28x32_i2c example. If you wired the RST pin to a GPIO pin, change `#define OLED_RESET -1` to the new pin number.

```
/*************************************************************************
 This is an example for our Monochrome OLEDs based on SSD1306 drivers

 Pick one up today in the adafruit shop!
 ------> http://www.adafruit.com/category/63_98

 This example is for a 128x32 pixel display using I2C to communicate
 3 pins are required to interface (two I2C and one reset).

 Adafruit invests time and resources providing this open
 source code, please support Adafruit and open-source
 hardware by purchasing products from Adafruit!

 Written by Limor Fried/Ladyada for Adafruit Industries,
 with contributions from the open source community.
 BSD license, check license.txt for more information
 All text above, and the splash screen below must be
 included in any redistribution.
 *************************************************************************/

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
// The pins for I2C are defined by the Wire-library.
// On an arduino UNO:       A4(SDA), A5(SCL)
// On an arduino MEGA 2560: 20(SDA), 21(SCL)
// On an arduino LEONARDO:   2(SDA),  3(SCL), ...
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C
for 128x32
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
#define LOGO_WIDTH    16
static const unsigned char PROGMEM logo_bmp[] =
{ 0b00000000, 0b11000000,
  0b00000001, 0b11000000,
  0b00000001, 0b11000000,
  0b00000011, 0b11100000,
  0b11110011, 0b11100000,
  0b11111110, 0b11111000,
  0b01111110, 0b11111111,
  0b00110011, 0b10011111,
  0b00011111, 0b11111100,
  0b00001101, 0b01110000,
  0b00011011, 0b10100000,
  0b00111111, 0b11100000,
  0b00111111, 0b11110000,
  0b01111100, 0b11110000,
  0b01110000, 0b01110000,
  0b00000000, 0b00110000 };

void setup() {
  Serial.begin(9600);

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
```

```
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...

  testdrawline();      // Draw many lines

  testdrawrect();      // Draw rectangles (outlines)

  testfillrect();      // Draw rectangles (filled)

  testdrawcircle();    // Draw circles (outlines)

  testfillcircle();    // Draw circles (filled)

  testdrawroundrect(); // Draw rounded rectangles (outlines)

  testfillroundrect(); // Draw rounded rectangles (filled)

  testdrawtriangle();  // Draw triangles (outlines)

  testfilltriangle();  // Draw triangles (filled)

  testdrawchar();      // Draw characters of the default font

  testdrawstyles();    // Draw 'stylized' characters

  testscrolltext();    // Draw scrolling text

  testdrawbitmap();    // Draw a small bitmap image

  // Invert and restore display, pausing in-between
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}

void loop() {
}

void testdrawline() {
  int16_t i;

  display.clearDisplay(); // Clear display buffer

  for(i=0; i<display.width(); i+=4) {
```

```
    display.drawLine(0, 0, i, display.height()-1, SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn line
    delay(1);
  }
  for(i=0; i<display.height(); i+=4) {
    display.drawLine(0, 0, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.width(); i+=4) {
    display.drawLine(0, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(0, display.height()-1, display.width()-1, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=display.width()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, i, 0, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=display.height()-1; i>=0; i-=4) {
    display.drawLine(display.width()-1, display.height()-1, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000); // Pause for 2 seconds
}

void testdrawrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i,
SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testfillrect(void) {
```

```
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=3) {
    // The INVERSE color is used so rectangles alternate white/black
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2,
SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testdrawcircle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillcircle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
    // The INVERSE color is used so circles alternate white/black
    display.fillCircle(display.width() / 2, display.height() / 2, i,
SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn circle
    delay(1);
  }

  delay(2000);
}

void testdrawroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    // The INVERSE color is used so round-rects alternate white/black
    display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawtriangle(void) {
  display.clearDisplay();
```

```
  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
    display.drawTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfilltriangle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
    // The INVERSE color is used so triangles alternate white/black
    display.fillTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawchar(void) {
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.cp437(true);         // Use full 256 char 'Code Page 437' font

  // Not all the characters will fit on the display. This is normal.
  // Library will draw what it can and the rest will be clipped.
  for(int16_t i=0; i<256; i++) {
    if(i == '\n') display.write(' ');
    else          display.write(i);
  }

  display.display();
  delay(2000);
}

void testdrawstyles(void) {
  display.clearDisplay();

  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));

  display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
  display.println(3.141592);

  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);

  display.display();
  delay(2000);
}

void testscrolltext(void) {
  display.clearDisplay();
```

```
  display.setTextSize(2); // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 0);
  display.println(F("scroll"));
  display.display();      // Show initial text
  delay(100);

  // Scroll in various directions, pausing in-between:
  display.startscrollright(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
}

void testdrawbitmap(void) {
  display.clearDisplay();

  display.drawBitmap(
    (display.width()  - LOGO_WIDTH ) / 2,
    (display.height() - LOGO_HEIGHT) / 2,
    logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
  display.display();
  delay(1000);
}

#define XPOS   0 // Indexes into the 'icons' array in function below
#define YPOS   1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  int8_t f, icons[NUMFLAKES][3];

  // Initialize 'snowflake' positions
  for(f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
    icons[f][YPOS]   = -LOGO_HEIGHT;
    icons[f][DELTAY] = random(1, 6);
    Serial.print(F("x: "));
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(F(" y: "));
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(F(" dy: "));
    Serial.println(icons[f][DELTAY], DEC);
  }

  for(;;) { // Loop forever...
    display.clearDisplay(); // Clear the display buffer

    // Draw each snowflake:
    for(f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h,
SSD1306_WHITE);
    }

    display.display(); // Show the display buffer on the screen
    delay(200);        // Pause for 1/10 second

    // Then update coordinates of each flake...
    for(f=0; f< NUMFLAKES; f++) {
```

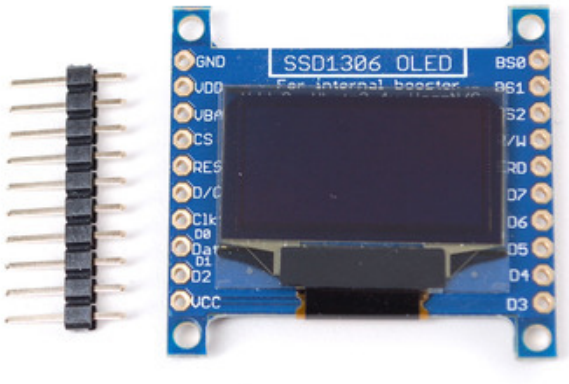
```
      icons[f][YPOS] += icons[f][DELTAY];
      // If snowflake is off the bottom of the screen...
      if (icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
        icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
```

Wiring OLD 0.96" 128x64 OLED

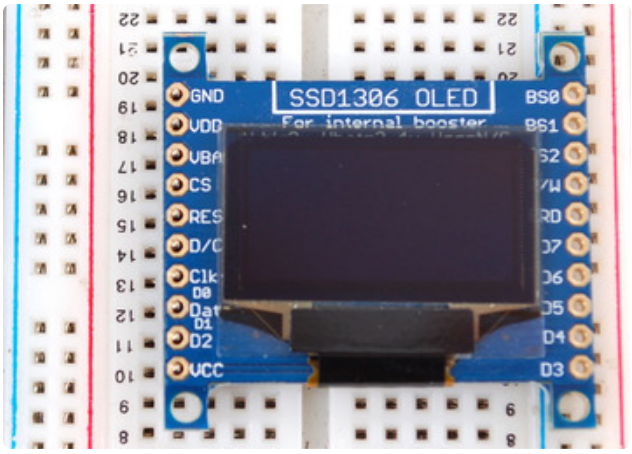
This wiring diagram is only for the older 0.96" OLED that comes with a level shifter chip. If you did not get a level shifter chip, you have a V2.0 so please check out the other wiring tutorial!

128x64 Version 1.0 OLED

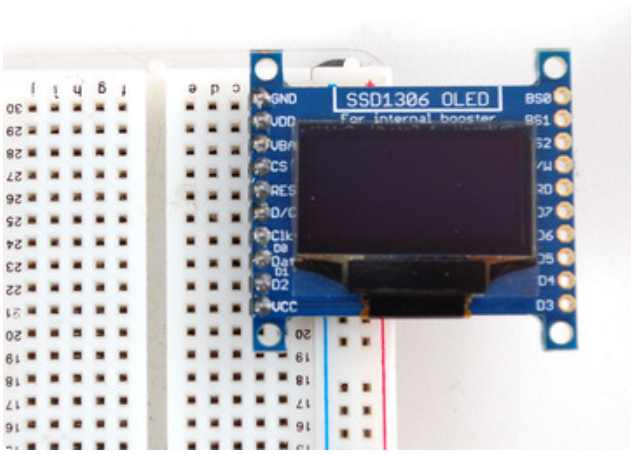
The version 1 128x64 OLED runs at 3.3V and does not have a built in level shifter so you'll need to use a level shifting chip to use with a 5V microcontroller. The following will assume that is the case. If you're running a 3.3V microcontroller system, you can skip the level shifter.



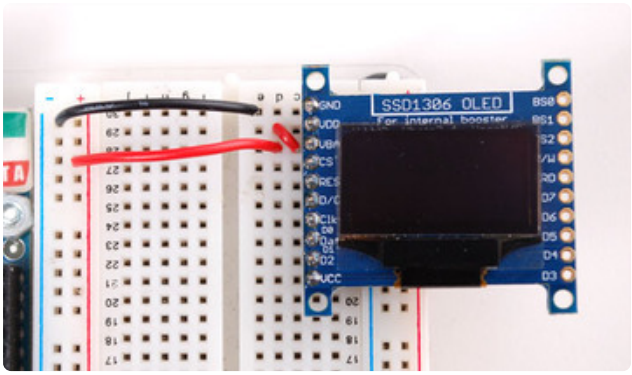
We'll assume you want to use this in a breadboard, take a piece of 0.1" header 10 pins long.



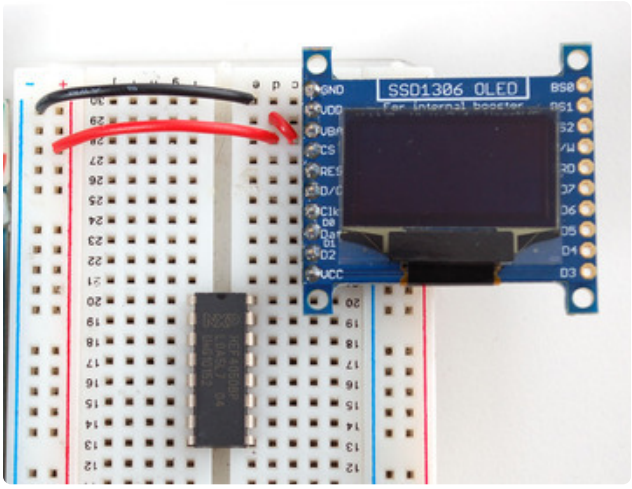
Place the header in a breadboard and then place the left hand side of the OLED on top.



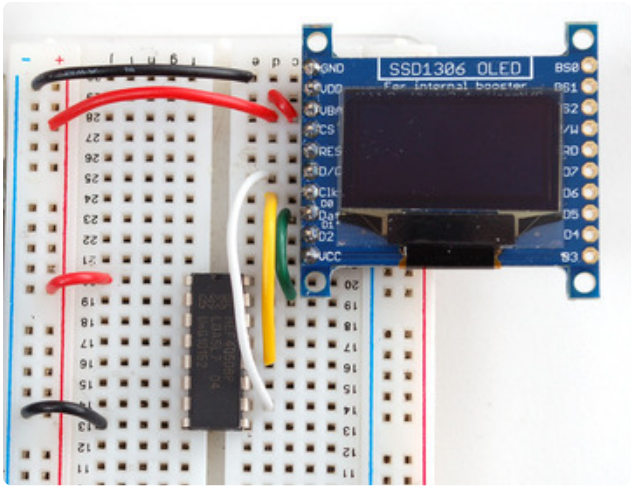
And solder the pins



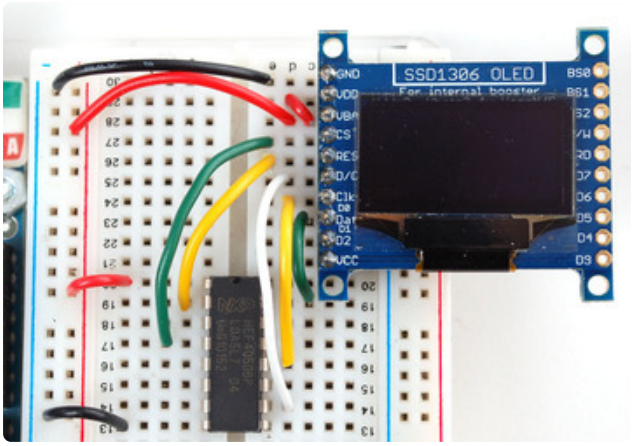
We'll be using the internal charge pump so connect **VDD** and **VBAT** together (they will connect to 3.3V). GND goes to ground.



Place a CD4050 level shifter chip so pin one is at the top.

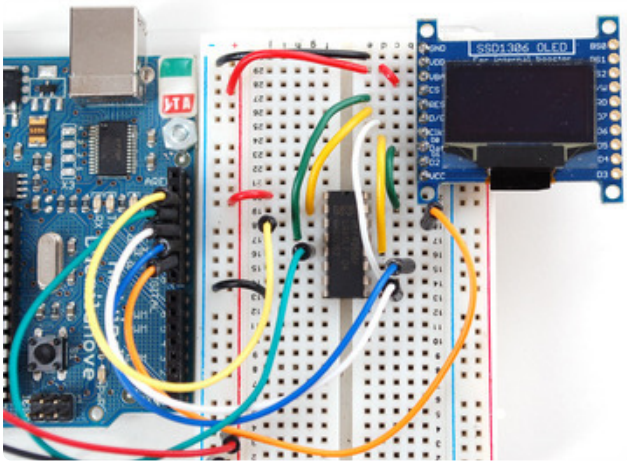


Connect pin 10 to **D/C** pin 12 to **CLK** (SPI clock) and pin 15 to **DAT** (SPI data).



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

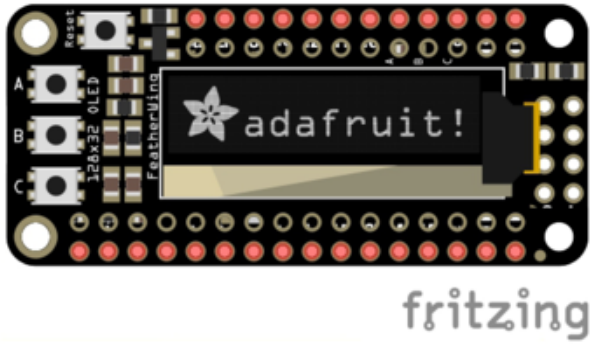
It's easy to use OLEDs with CircuitPython and the Adafruit CircuitPython DisplayIO SSD1306 (https://adafru.it/FRA) module. This module allows you to easily write CircuitPython code to control the display.

You can use this sensor with any CircuitPython microcontroller board.

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

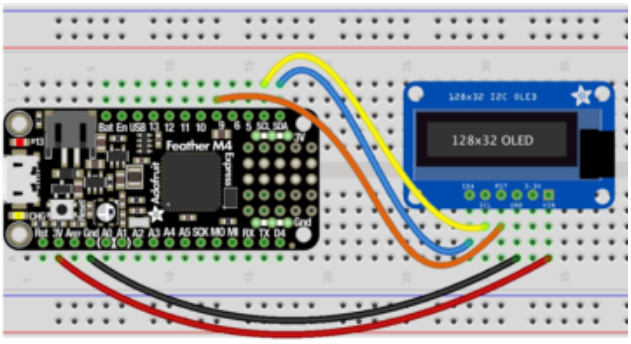
Connect the OLED to your microcontroller board as shown below.

Adafruit OLED FeatherWing



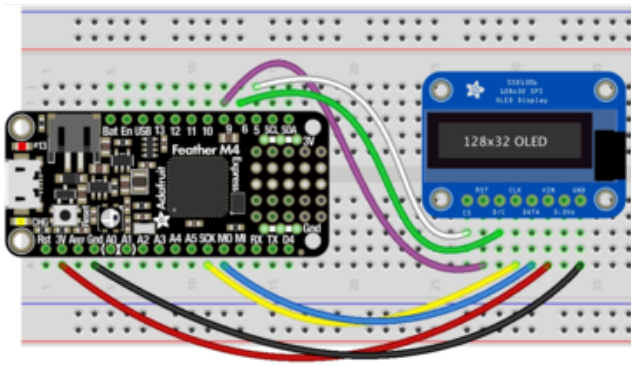
Solder the Feather with female headers on top or stacking headers.
Attach the OLED FeatherWing using the stacking method.

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**

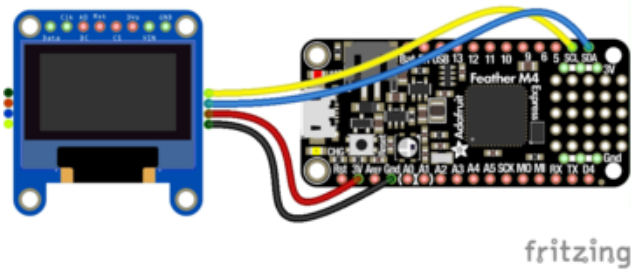
Adafruit 128x32 SPI OLED Display



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" 128x64 OLED Display STEMMA QT Version - I2C Wiring

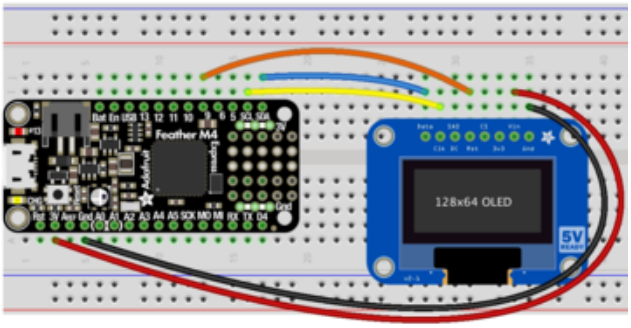
You do not need to alter the jumpers on the back - I2C is the default configuration on this display!

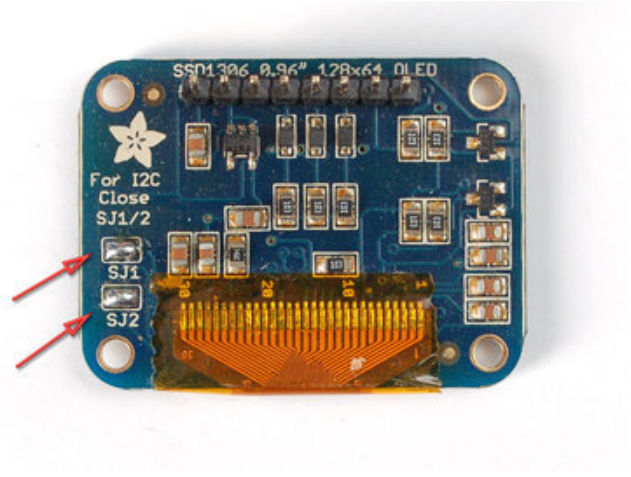


Microcontroller 3V to **OLED Vin**
Microcontroller GND to **OLED Gnd**
Microcontroller SCL to **OLED Clk**
Microcontroller SDA to **OLED Data**
Note: Connecting the OLED RST is not necessary as this revision added auto-reset circuitry so the RESET pin is not required.

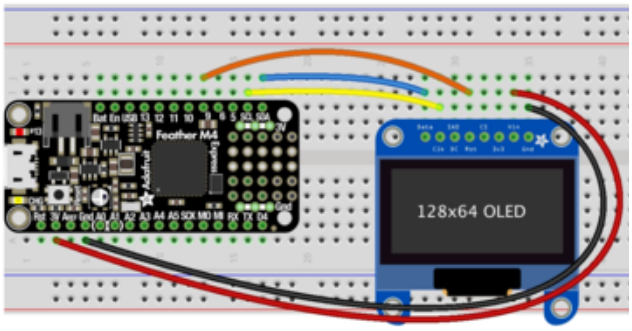
Adafruit 0.96" or 1.3" 128x64 OLED Display Original Version - I2C Wiring

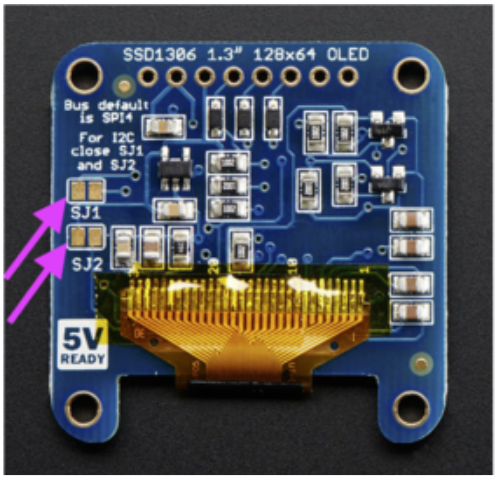
Check that the two jumpers are CLOSED on the back of the display to use with I2C





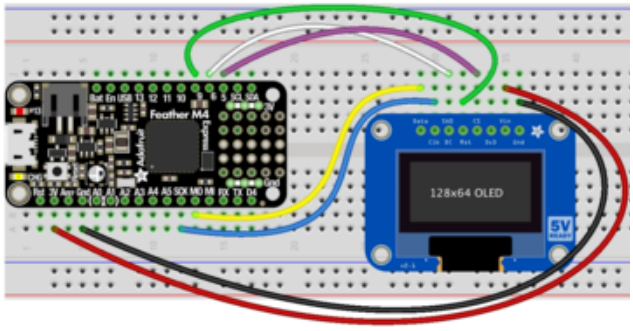
Microcontroller **3V** to **OLED Vin**
Microcontroller **GND** to **OLED Gnd**
Microcontroller **SCL** to **OLED Clk**
Microcontroller **SDA** to **OLED Data**
Microcontroller **D9** to **OLED Rst**

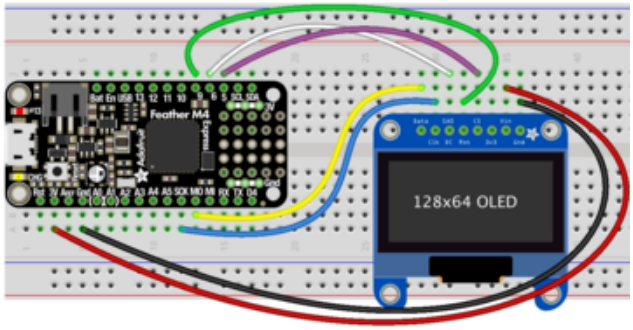




Adafruit 0.96" or 1.3" 128x64 OLED Display - SPI Wiring

Check that the two jumpers are OPEN on the back of the display to use with SPI





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

Note that there is a non-displayio driver available as well and you want the displayio version.

To use the SSD1306 OLED with your Adafruit CircuitPython board you'll need to install the Adafruit CircuitPython DisplayIO SSD1306 (https://adafru.it/FRA) module on your board.

First make sure you are running the latest version 5.0 or later of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

You must be using CircuitPython 5 or later for this to work!

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 (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

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 (https://adafru.it/Awz) 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.

Adafruit_CircuitPython_Display_Text

https://adafru.it/FRB

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

Displayio is only available on express board due to the smaller memory size on non-express boards.

It's easy to use OLEDs with Python and the Adafruit CircuitPython DisplayIO SSD1306 (https://adafru.it/FRA) 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:

```
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):

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

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

```
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:

```
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!

Initializing with CircuitPython 9

Because displayio has grown so much since it was first written, it has been split up into separate modules to make better use of space on boards with limited storage. Due to splitting things up, the initialization has changed slightly in CircuitPython 9.

```
import displayio
import i2cdisplaybus
import adafruit_displayio_ssd1306
display_bus = i2cdisplaybus.I2CDisplayBus(i2c, device_address=0x3c)

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

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:

```
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:

```
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:

```
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

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
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

# Compatibility with both CircuitPython 8.x.x and 9.x.x.
# Remove after 8.x.x is no longer a supported release.
try:
    from i2cdisplaybus import I2CDisplayBus

    # from fourwire import FourWire
except ImportError:
    from displayio import I2CDisplay as I2CDisplayBus

    # from displayio import FourWire

import terminalio
from adafruit_display_text import label
import adafruit_displayio_ssd1306

displayio.release_displays()

oled_reset = board.D9

# Use for I2C
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
display_bus = I2CDisplayBus(i2c, device_address=0x3C, reset=oled_reset)

# Use for SPI
# spi = board.SPI()
# oled_cs = board.D5
# oled_dc = board.D6
# display_bus = 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()
display.root_group = 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.

```
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.

```
# 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.

```
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.

```
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.

```
splash = displayio.Group()
display.root_group = 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.

```
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.

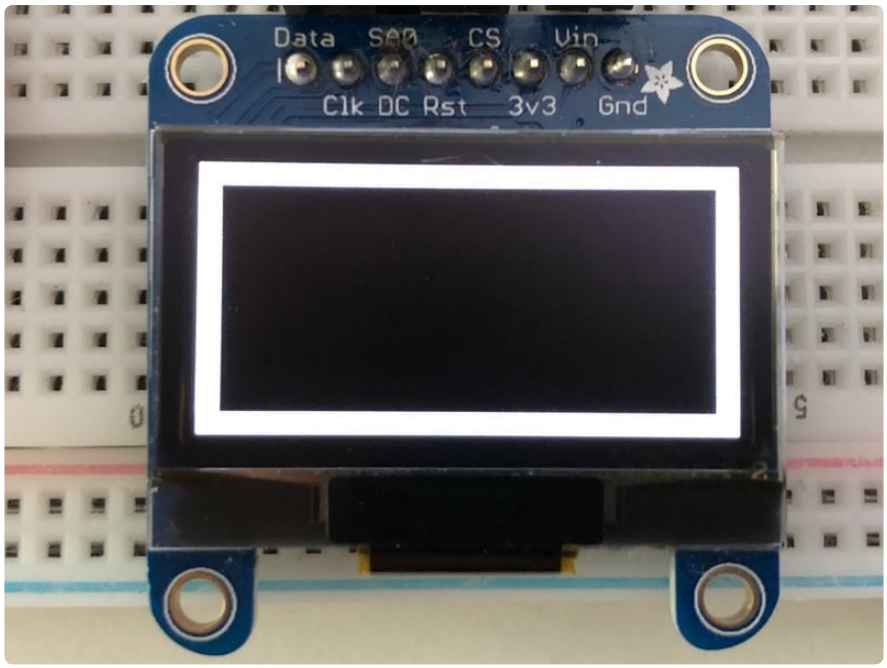
```
bg_sprite = displayio.TileGrid(color_bitmap,
                               pixel_shader=color_palette,
                               x=0, y=0)
splash.append(bg_sprite)
```

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 `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 **BORDER** variable comes into use. It makes calculating the size of the second rectangle much easier. The screen we're using here is **128x64** and we have the BORDER set to **5** , so we'll want to subtract 10 from each of those numbers.

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

```
# 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.

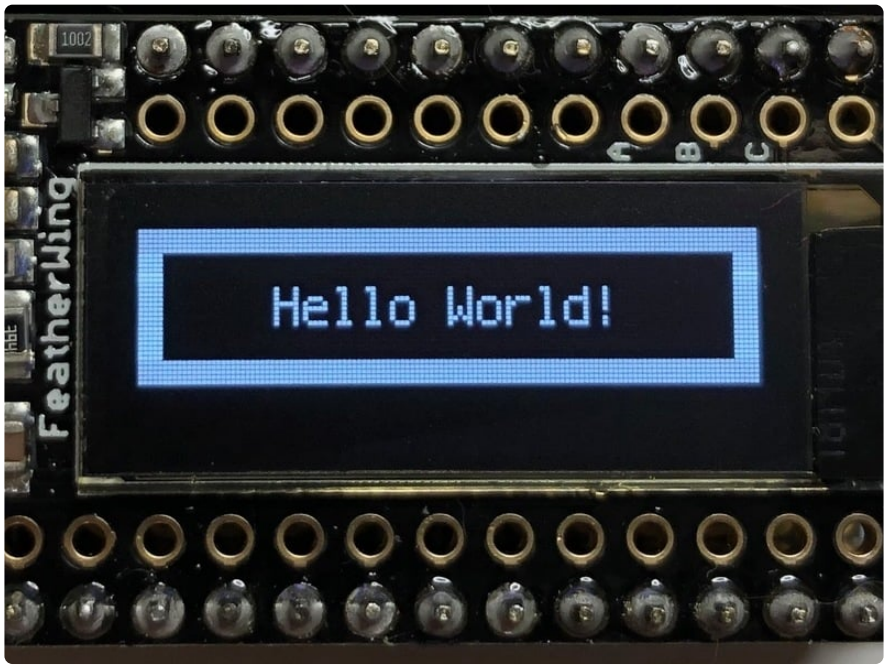
Labels are centered vertically, so we'll place it at half the HEIGHT for the Y coordinate and subtract one so it looks good. We use the `//` operator to divide because we want a whole number returned and it's an easy way to round it. We'll set the width to around 28 pixels make it appear to be centered 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`.

```
# 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.

```
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://adafru.it/EGh)

Python Wiring

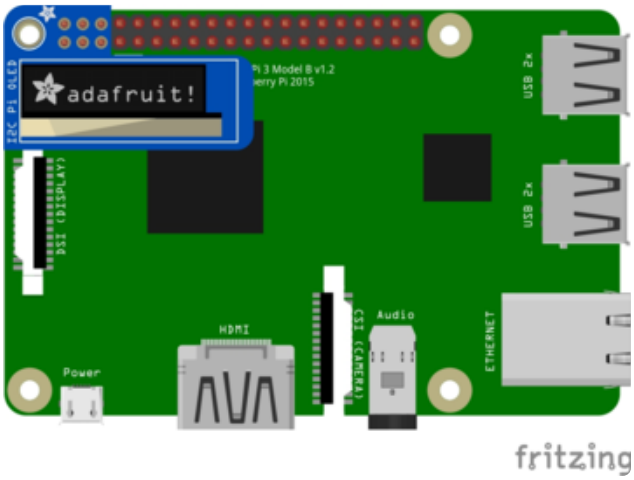
It's easy to use OLEDs with Python and the Adafruit CircuitPython SSD1306 (https://adafru.it/u1f) module. This module allows you to easily write Python code to control the display.

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 (https://adafru.it/BSN).

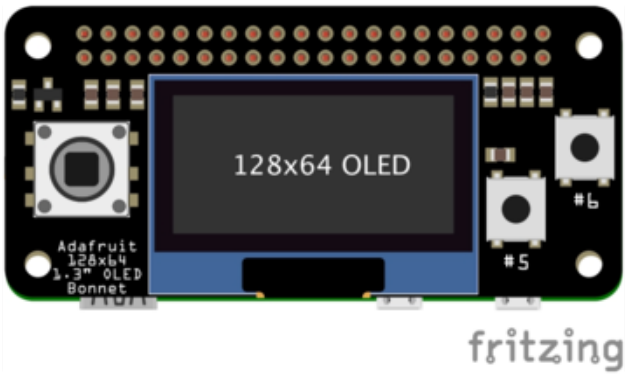
Connect the OLED as shown below to your Raspberry Pi.

Adafruit PIOLED



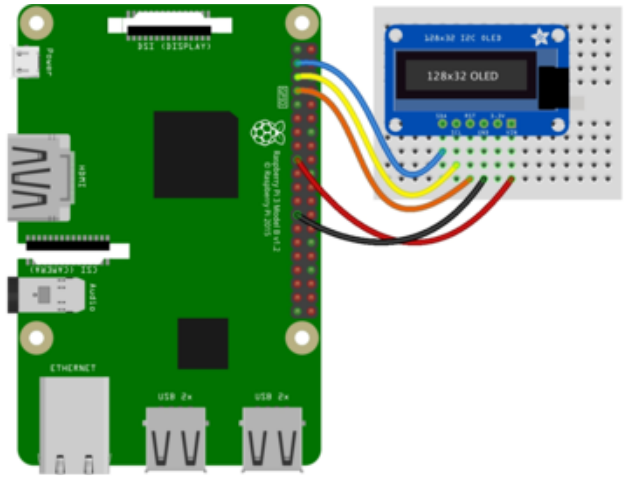
The PiOLED comes fully assembled. Simply plug into any Raspberry Pi as shown.

Adafruit 128x64 OLED Bonnet for Raspberry Pi



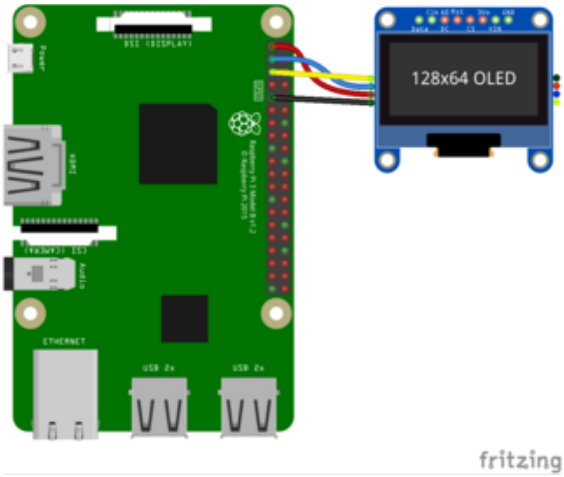
The OLED Bonnet comes fully assembled. Simply plug into the Raspberry Pi as shown.

Adafruit 128x32 I2C OLED Display



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)

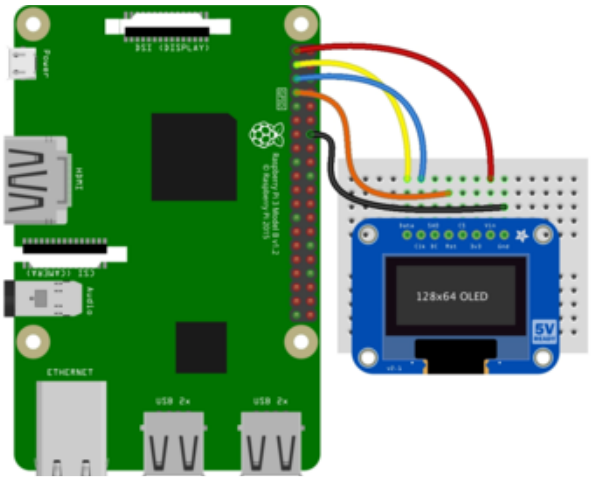
Adafruit 0.96" 128x64 OLED Display STEMMA QT Version - I2C Wiring

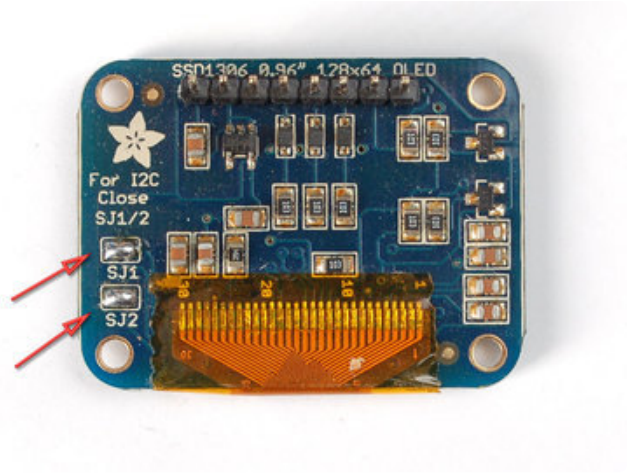


You do not need to alter the jumpers on the back - I2C is the default configuration on this display!

Pi 3.3V to **OLED Vin (red wire)**
Pi GND to **OLED Gnd (black wire)**
Pi SCL to **OLED Clk (yellow wire)**
Pi SDA to **OLED Data (blue wire)**
Note: Connecting the OLED RST is not necessary as this revision added auto-reset circuitry so the RESET pin is not required.

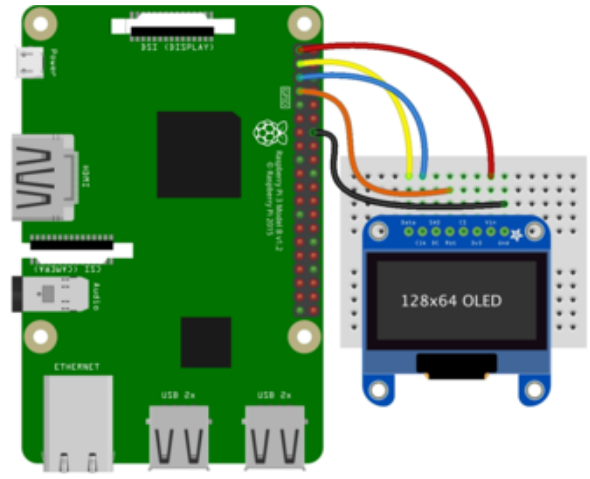
Adafruit 0.96" or 1.3" 128x64 OLED Display Original Version - I2C Wiring

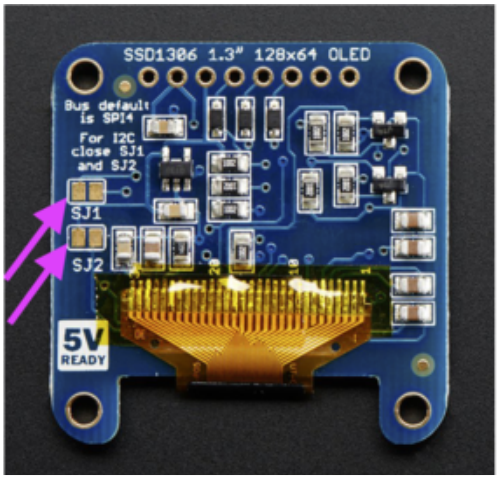




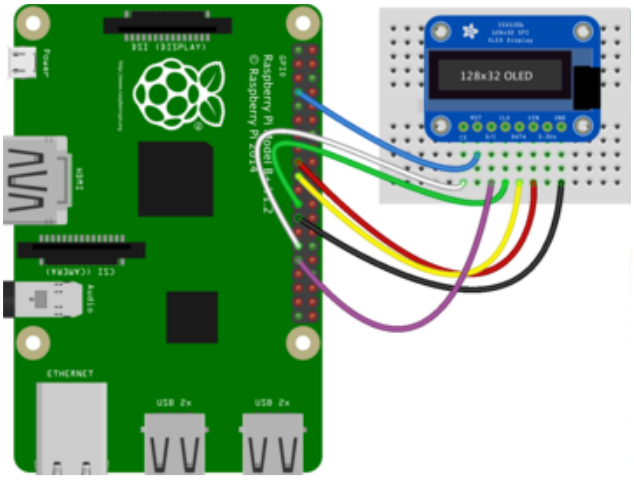
You must solder two jumpers closed on the back of the display to use with I2C!

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)



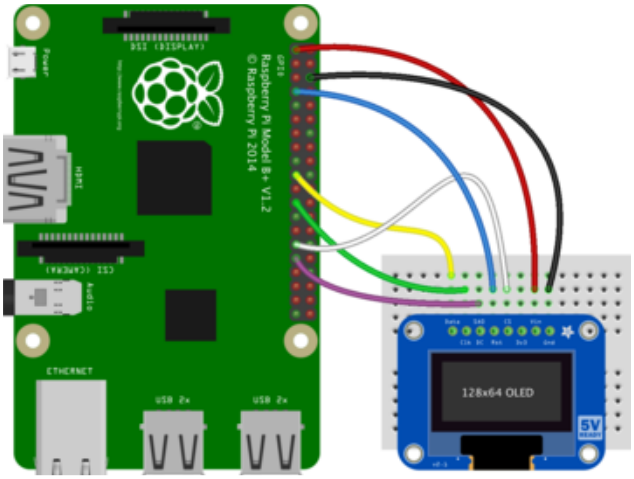


Adafruit 128x32 SPI OLED Display



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:

```
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
```

NumPy Library

NumPy is needed for the **circuitpython_typing** library. This can be installed with the following command:

```
sudo apt-get install python3-numpy
```

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

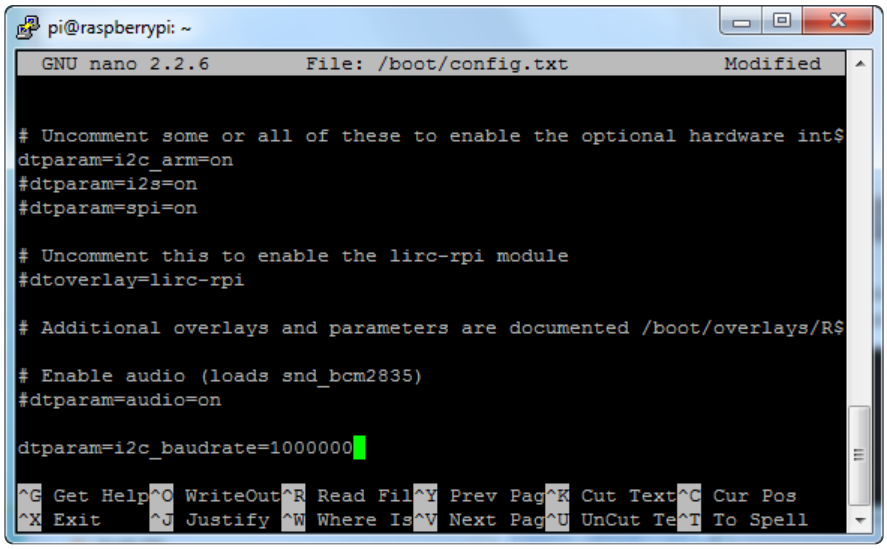
Speeding up the Display on Raspberry Pi

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/firmware/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 (https://adafru.it/u1f) 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 (https://adafru.it/BSN).

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 (https://adafru.it/FU7).

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:

```
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:

```
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:

```
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:

```
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:

```
import digitalio

reset_pin = digitalio.DigitalInOut(board.D4) # any pin!
oled = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
```

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:

```
import adafruit_ssd1306
import board
import busio
import digitalio

spi = busio.SPI(board.SCK, MOSI=board.MOSI)
reset_pin = digitalio.DigitalInOut(board.D4) # any pin!
cs_pin = digitalio.DigitalInOut(board.D5)    # any pin!
dc_pin = digitalio.DigitalInOut(board.D6)    # any pin!

oled = adafruit_ssd1306.SSD1306_SPI(128, 32, spi, dc_pin, reset_pin, cs_pin)
```

Note the first two parameters to the `SSD1306_SPI` 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!

The next parameters to the initializer are the pins connected to the display's **DC**, **reset**, and **CS** lines in that order. Again make sure to use the right pin names as you have wired up to your board!

Example Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""
This demo will fill the screen with white, draw a black box on top
and then print Hello World! in the center of the display

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 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()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
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)
# oled = 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))

# Get drawing object to draw on image.
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!"
bbox = font.getbbox(text)
(font_width, font_height) = bbox[2] - bbox[0], bbox[3] - bbox[1]
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 `board.I2C()`. By using this function, it finds the I2C module and initializes using the default I2C parameters. We also set up the oled with SSD1306_I2C which makes use of the I2C bus.

```
# Use for I2C.
i2c = board.I2C()
oled = adafruit_ssd1306.SSD1306_I2C(WIDTH, HEIGHT, i2c, addr=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 up the OLED with SSD1306_SPI which makes use of the SPI bus.

```
# 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)
```

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

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

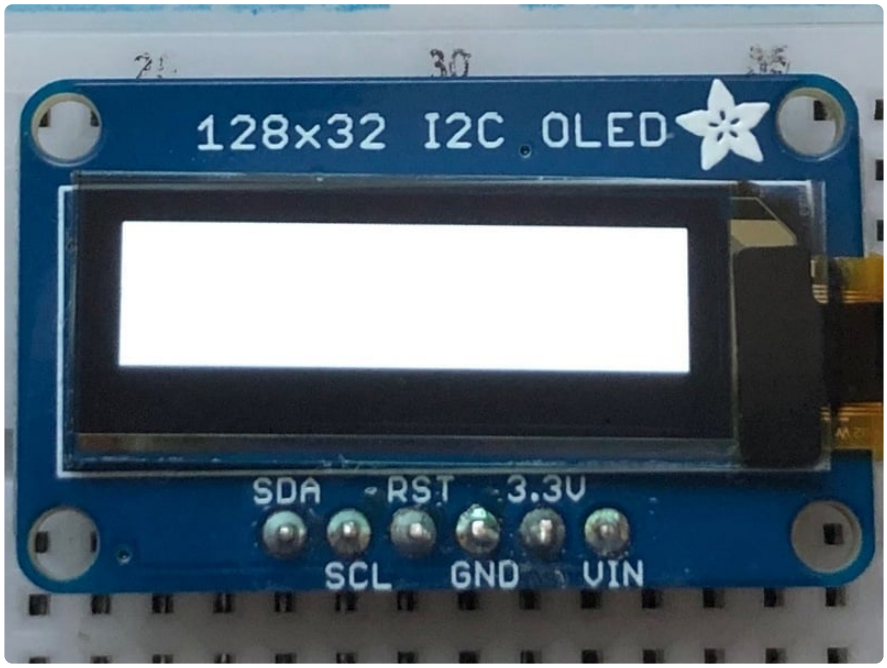
Next, we need to initialize PIL to create a blank image to draw on. Think of it as a virtual canvas. Since this is a monochrome display, we set it up for 1-bit color, meaning a pixel is either white or black. We can make use of the OLED's width and height properties as well. Optionally, we could have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

```
# 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)
```

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.

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

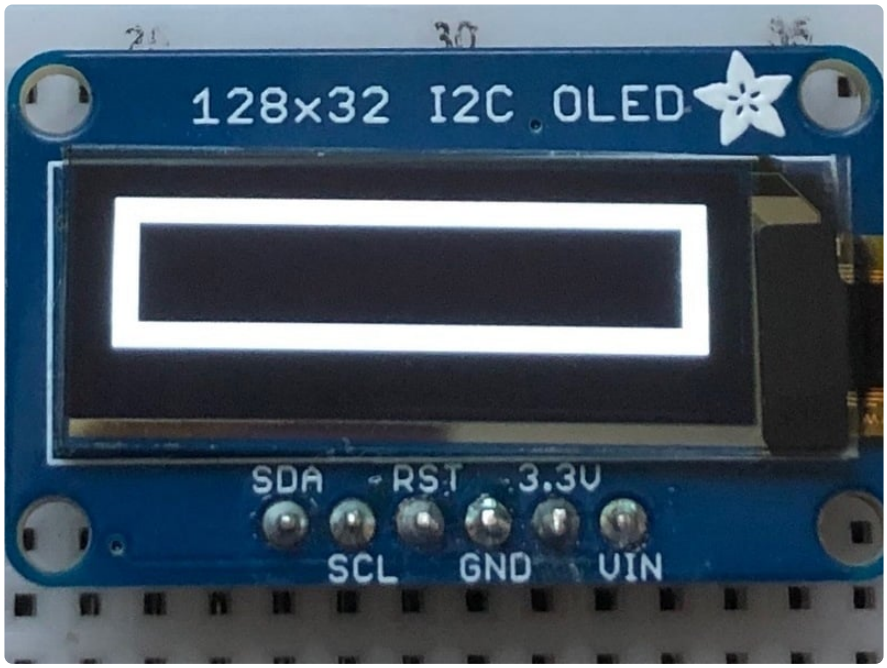
If we ran the code now, it would still show a blank display because we haven't told python to use our virtual canvas yet. You can skip to the end if you would like to see how to do that. This is what our canvas currently looks like in memory.



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.

```
# 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.

```
# 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.

```
# 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://adafru.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://adafru.it/aJa) and install it after you've gotten the OLED driver library.

You can check out a simulator for these OLEDs at https://wokwi.com/arduino/libraries/Adafruit_SSD1306 (https://adafru.it/Ncf)

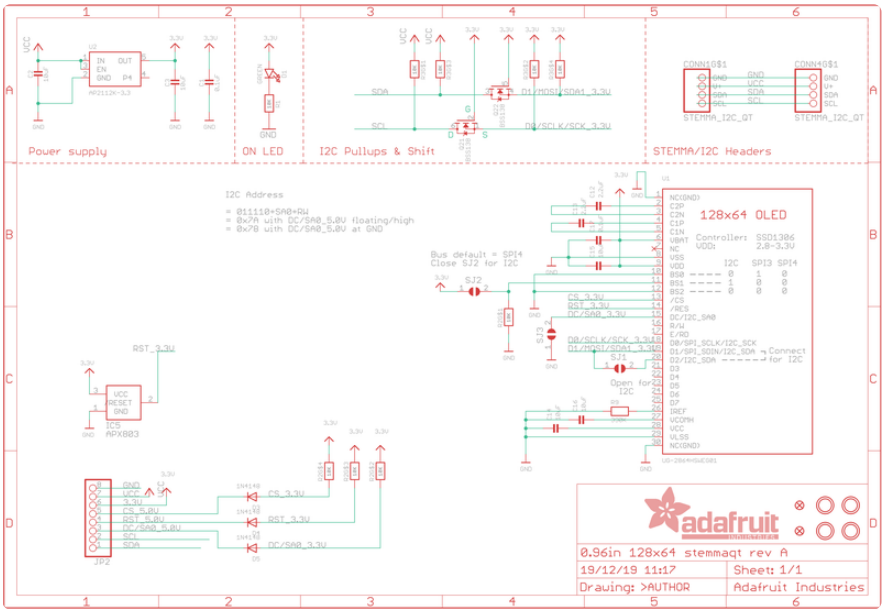
Datasheets

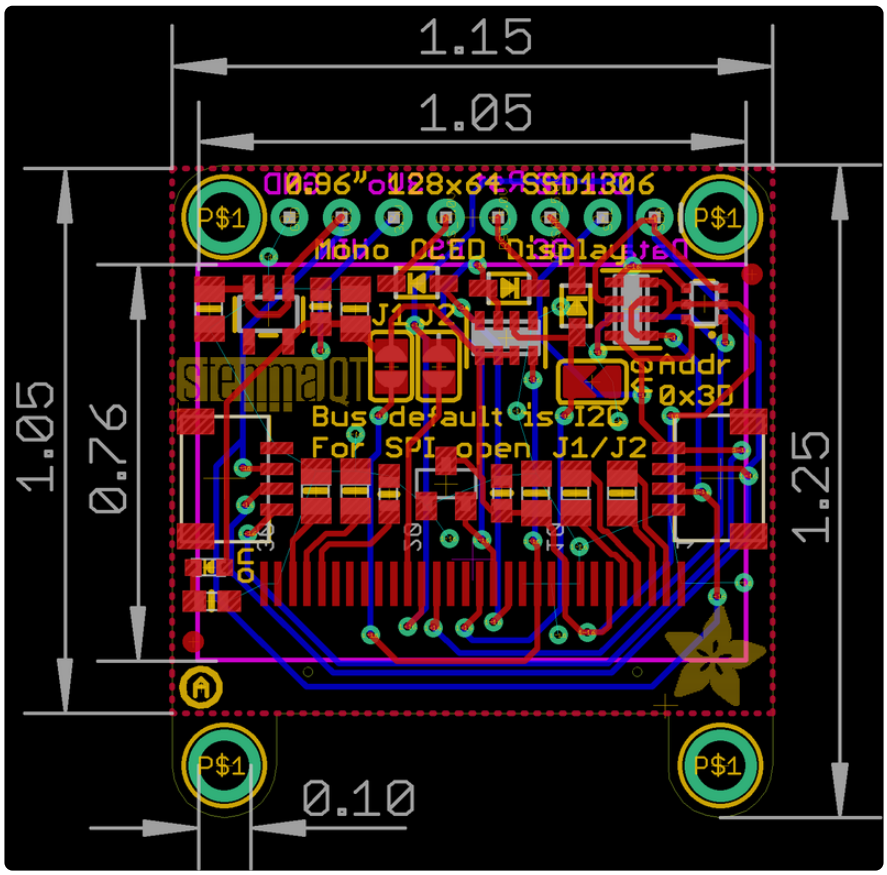
- UG-2864HSWEG01 (https://adafru.it/aJI) Datasheet
- UG-2832HSWEG02 (https://adafru.it/qrf) Datasheet
- UG-2864HSWEG01 (https://adafru.it/wWD) User Guide
- UG-2832HSWEG04 (https://adafru.it/qVA) Datasheet
- UG-2864KSWLG01 (https://adafru.it/Re8) Datasheet
- SSD1306 (https://adafru.it/aJK) Datasheet

Files

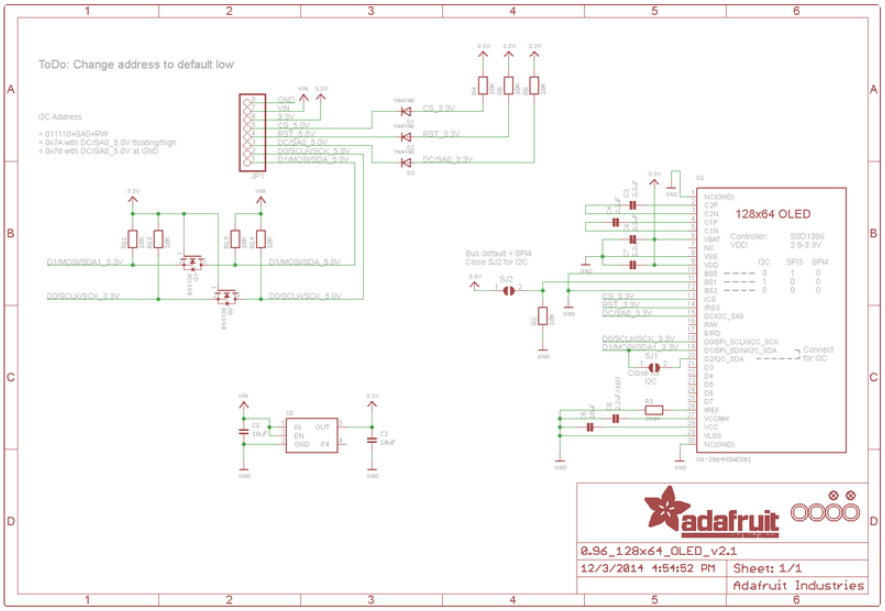
- EagleCAD PCB files for 128x32 0.91" SPI display PCB (https://adafru.it/aJL)
- EagleCAD PCB files for 128x32 0.91" I2C display on GitHub (https://adafru.it/rPF)
- EagleCAD PCB files for 128x64 0.96" display on GitHub (https://adafru.it/aJM)
- EagleCAD PCB files for 128x64 1.3" display on GitHub (https://adafru.it/rJe)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

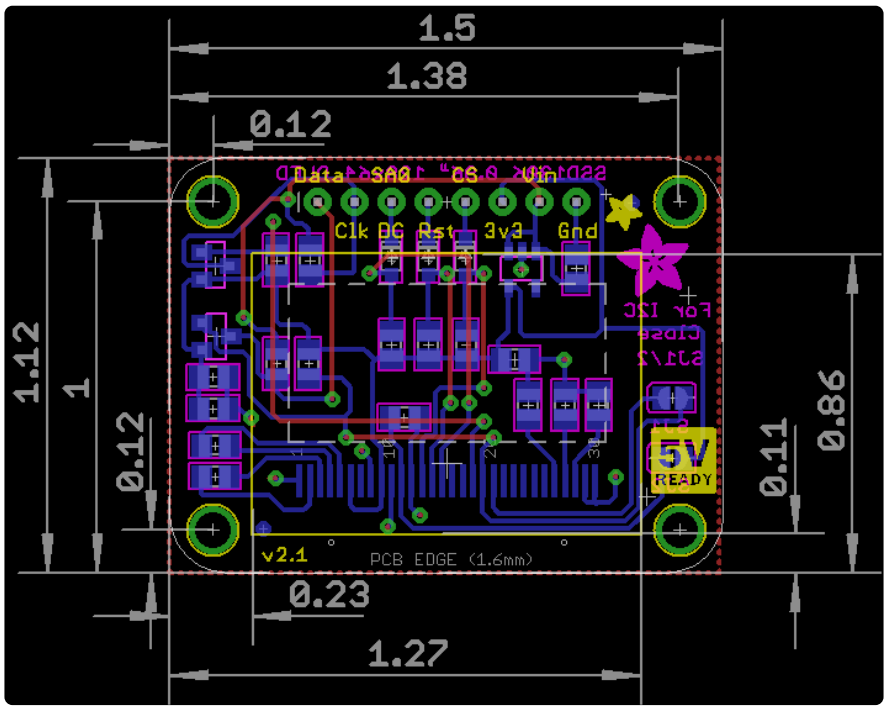
Schematic & Fabrication Print for 0.96" OLED - STEMMA QT version



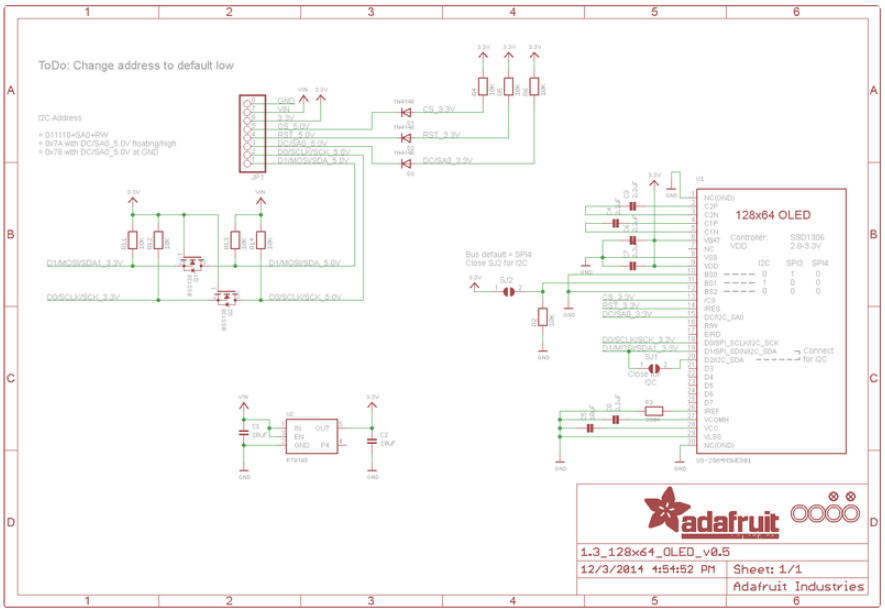


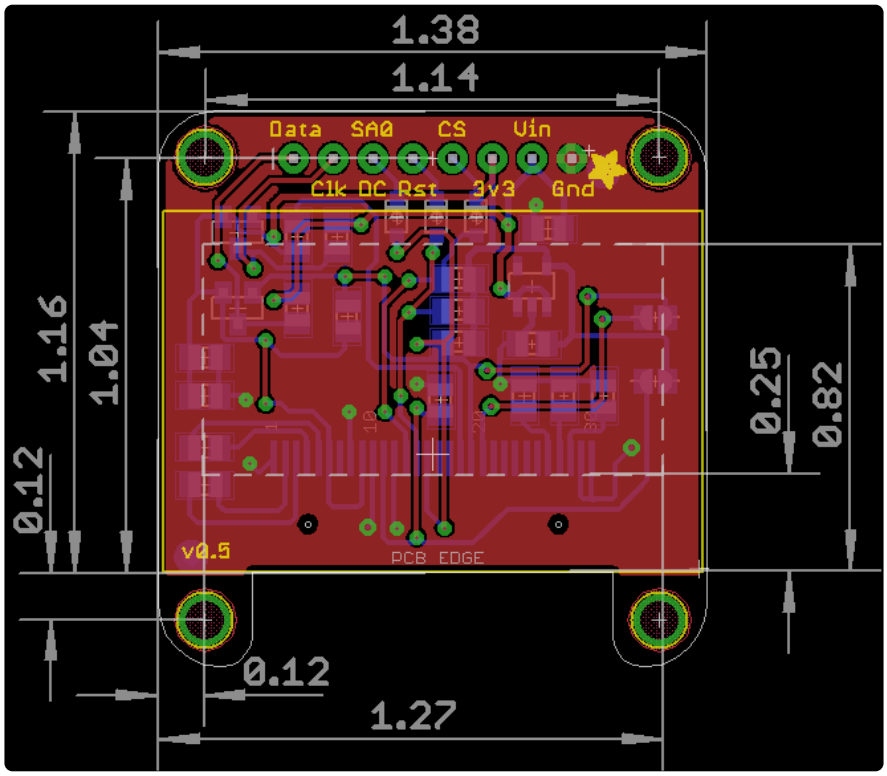
Schematic & Fabrication Print for 0.96" OLED - Original version





Schematic & Fabrication Print for 1.3" OLED

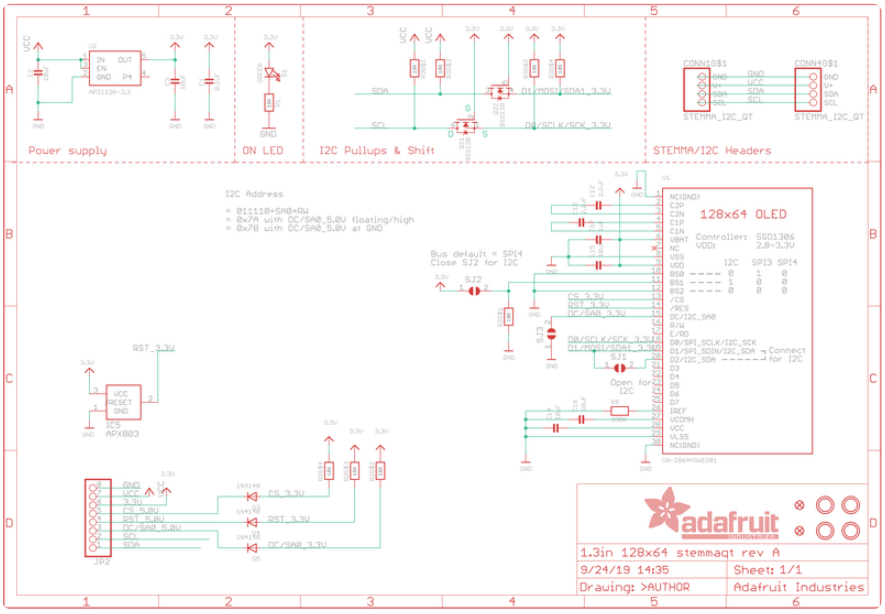


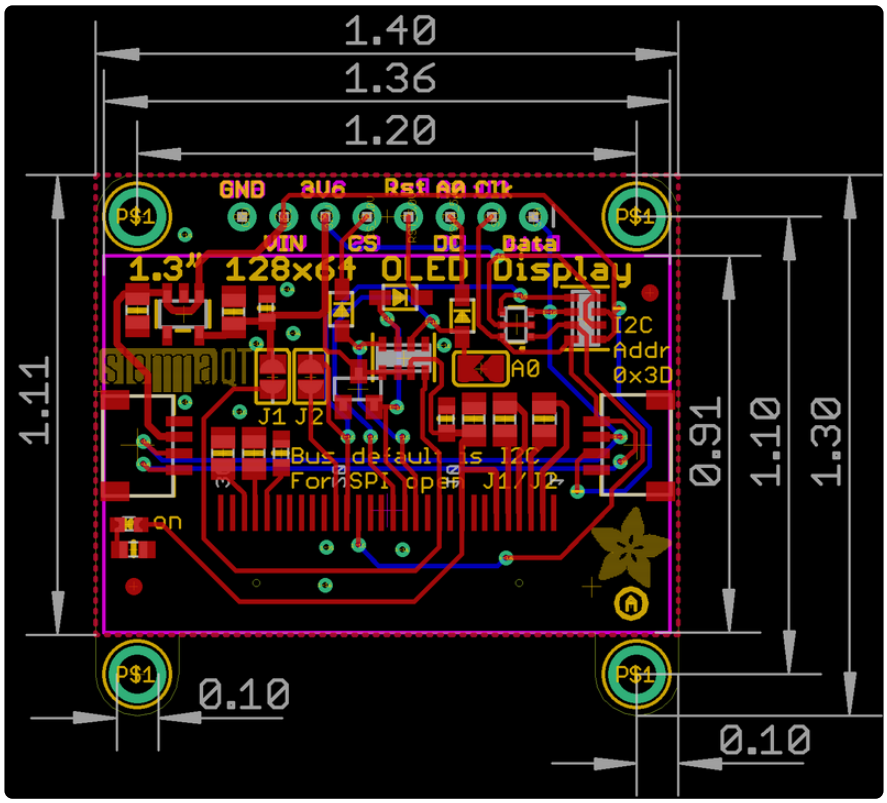


Schematic & Fabrication Print for 1.3" OLED STEMMA QT

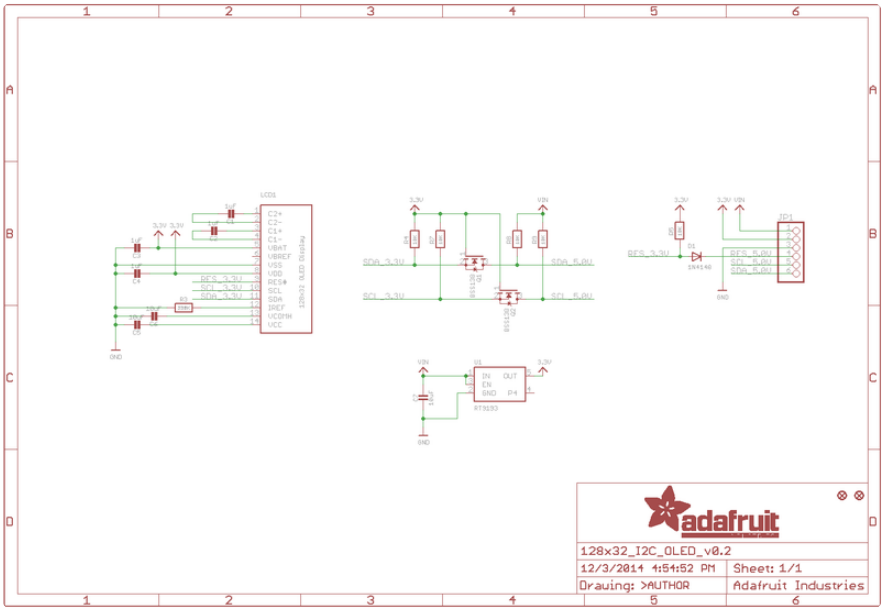
As of Nov 20, 2019 we've done a re-design to make the display more plug and play. There is now an auto-reset circuit so that it will reset the display on power up. We've also changed the default protocol to be I2C instead of SPI. **To convert to SPI mode**

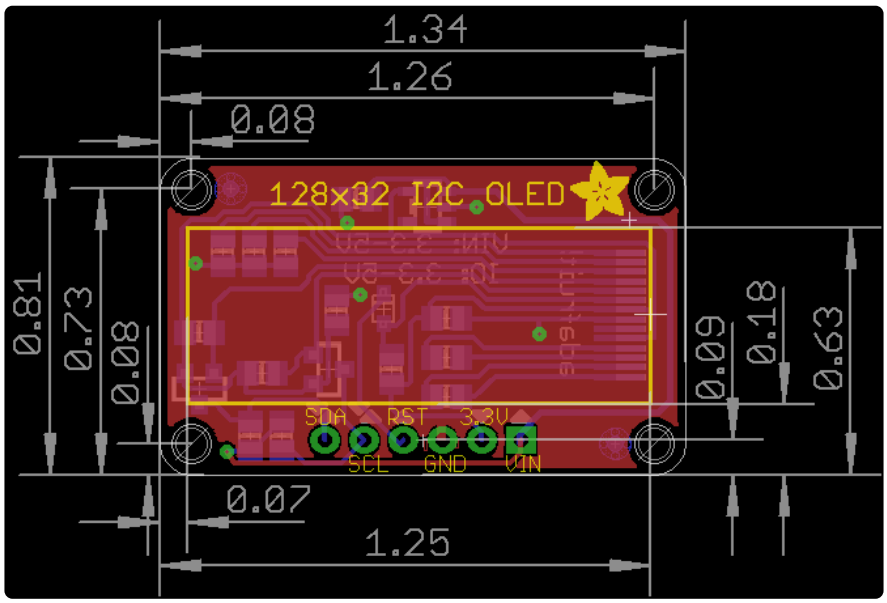
you will need to cut two jumpers (there's a typo on the PCB). We have also added two STEMMA QT / Qwiic connectors for plug and play usage! The board size, mounting holes and layout has changed slightly to accommodate these changes.





Schematic & Fabrication Print for 0.91" 128x32 I2C

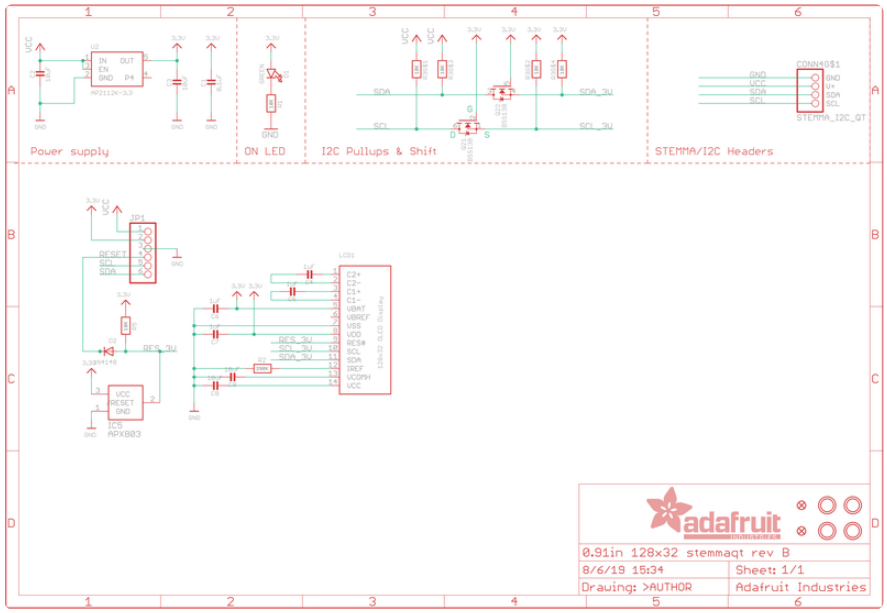


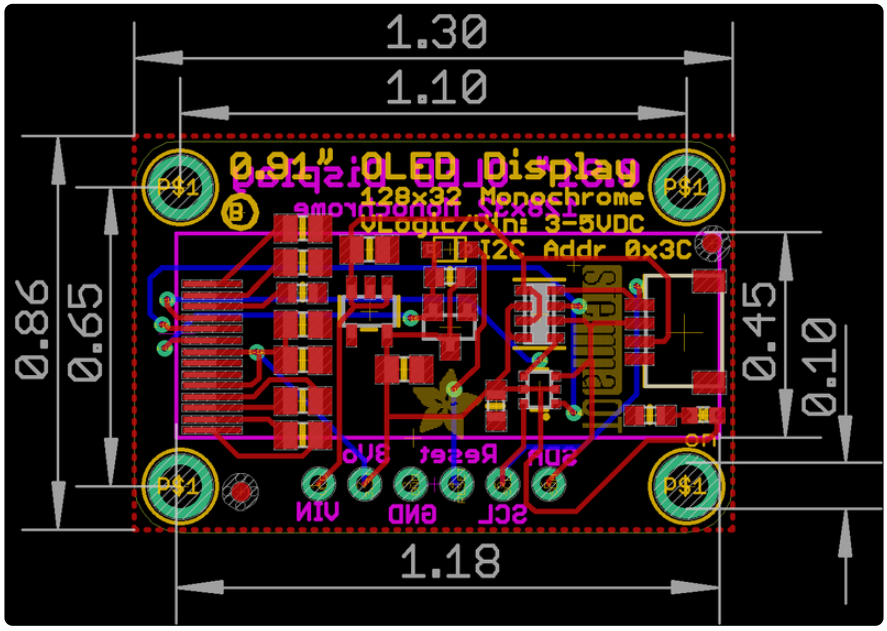


Schematic & Fabrication Print for 0.91" 128x32 I2C STEMMA QT

As of Nov 20, 2019 we've done a re-design to make the display more plug and play. There is now an auto-reset circuit so that it will reset the display on power up. We've also changed the default protocol to be I2C instead of SPI. **To convert to SPI mode you will need to cut two jumpers (there's a typo on the PCB).** We have also added

two STEMMA QT / Qwiic connectors for plug and play usage! The board size, mounting holes and layout has changed slightly to accommodate these changes.





Schematic & Fabrication Print for 0.91" 128x32 SPI