2.3" Monochrome 128x32 OLED Display Module

Created by lady ada
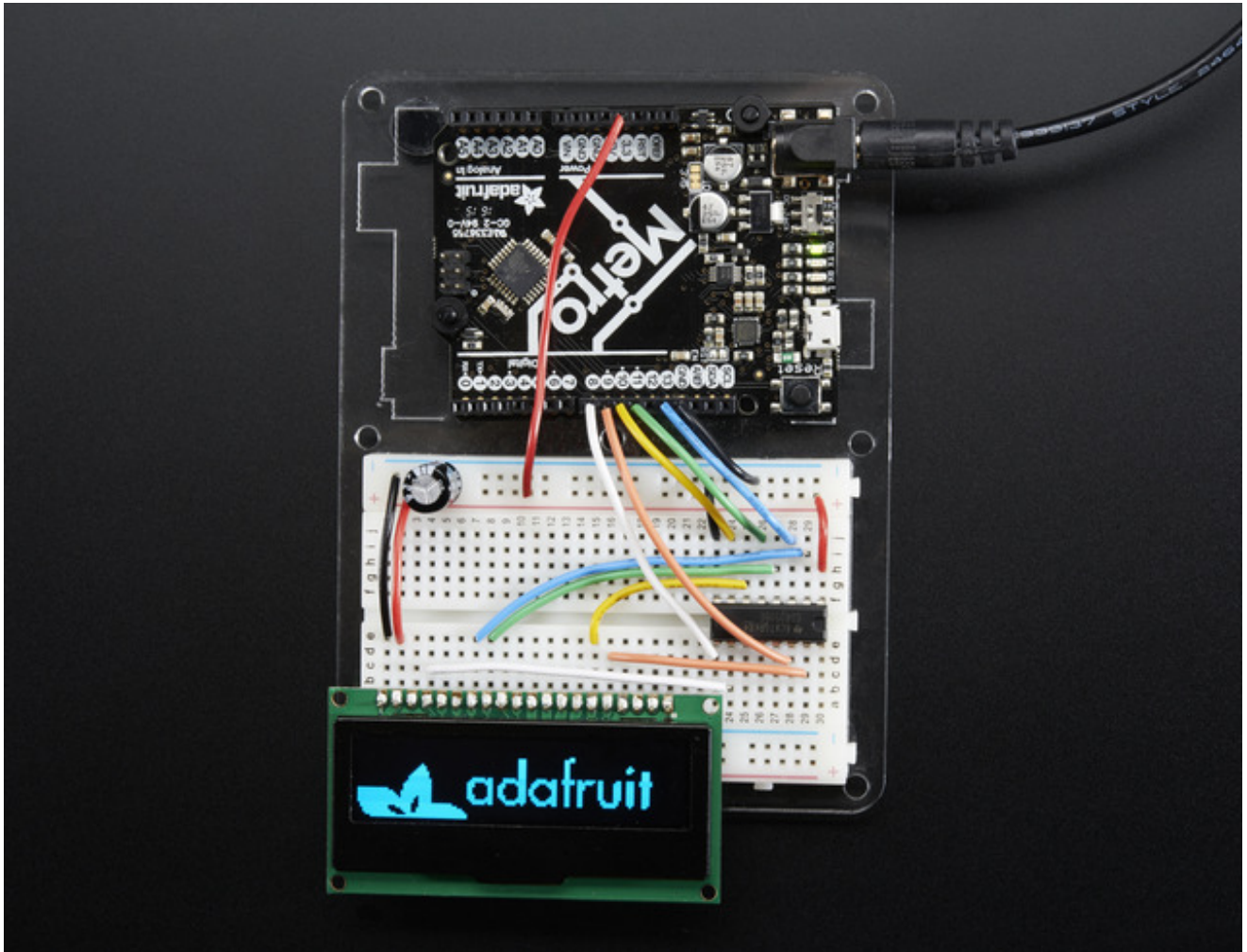
Guide Contents

Overview

If you've been diggin' our monochome OLEDs but need something bigger, this display will delight you. These displays are 2.3" diagonal, and very readable due to the high contrast of an OLED display. This display is made of 128x32 individual blue OLED pixels, 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 graphic display for its crispness!
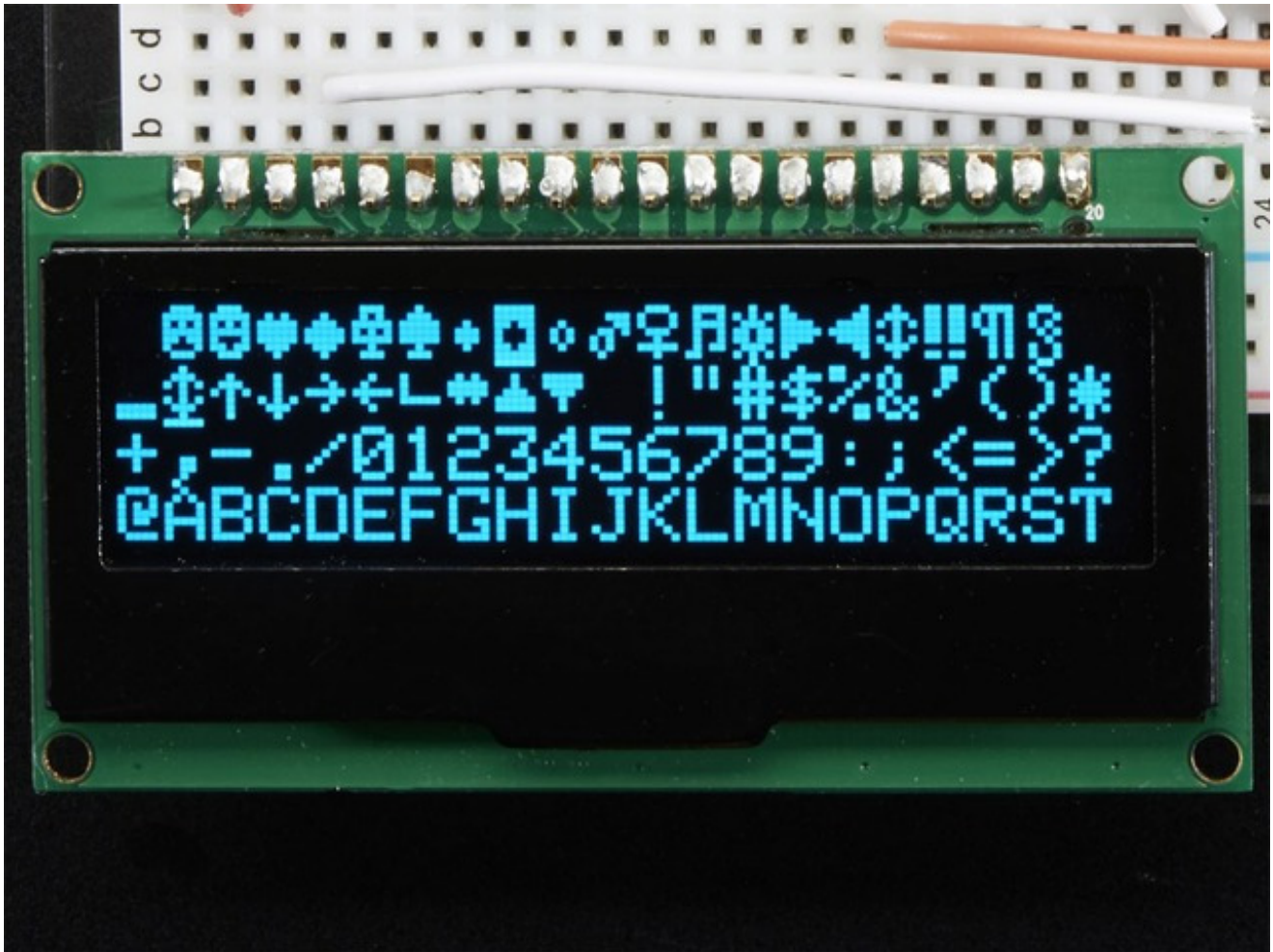
The driver chip, SSD1305 can communicate in three ways: 8-bit, I2C or SPI. Personally we think SPI is the way to go, only 4 or 5 wires are required and its very fast. The OLED itself requires a 3.3V power supply and 3.3V logic levels for communication. We include a breadboard-friendly level shifter that can convert 3V or 5V down to 3V, so it can be used with 5V-logic devices like Arduino.
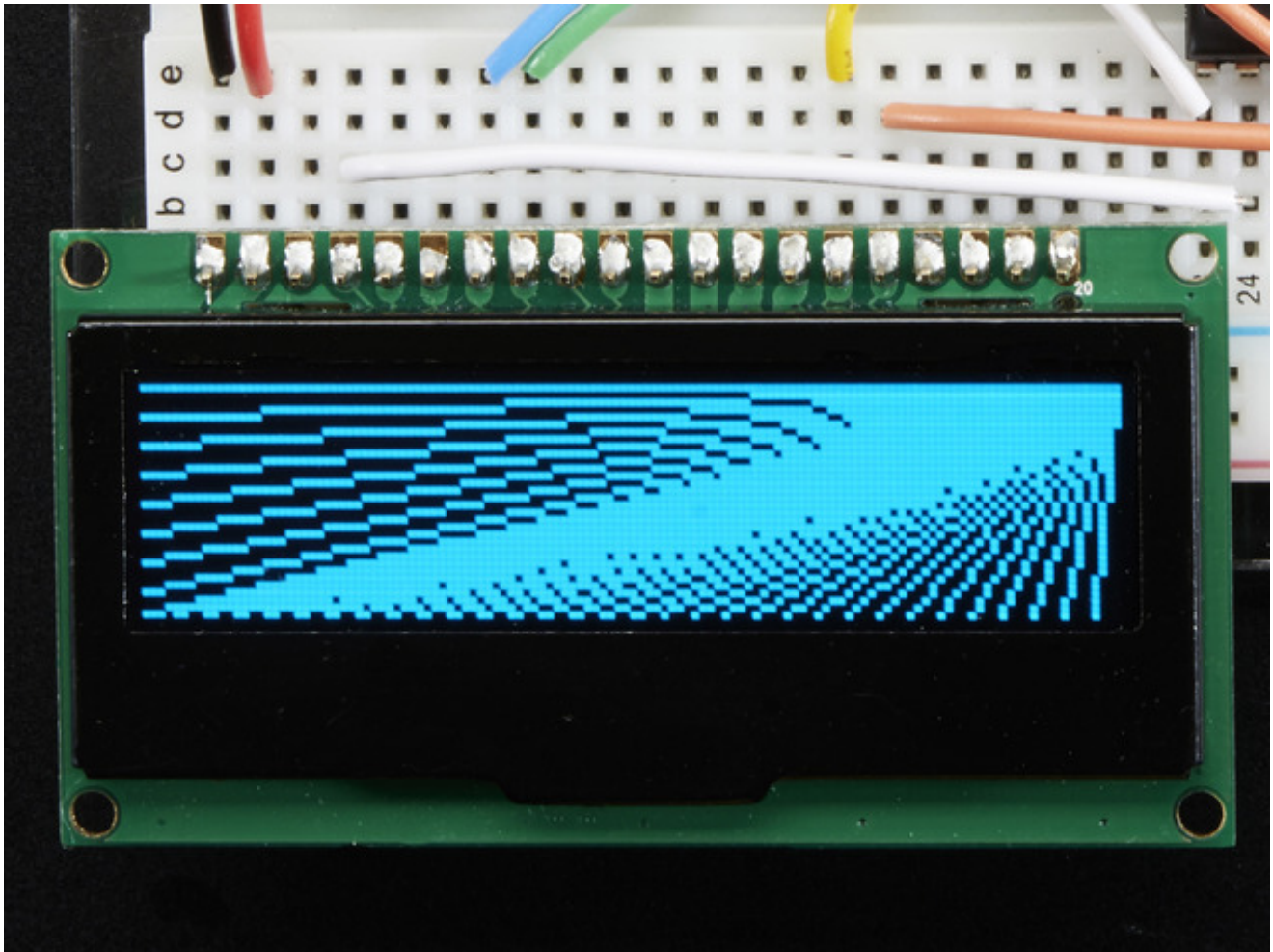
The power requirements depend a little on how much of the display is lit but on average the display uses about 50mA from the 3.3V supply. Built into the OLED driver is a simple switch-cap charge pump that turns 3.3V into a high voltage drive for the OLEDs.
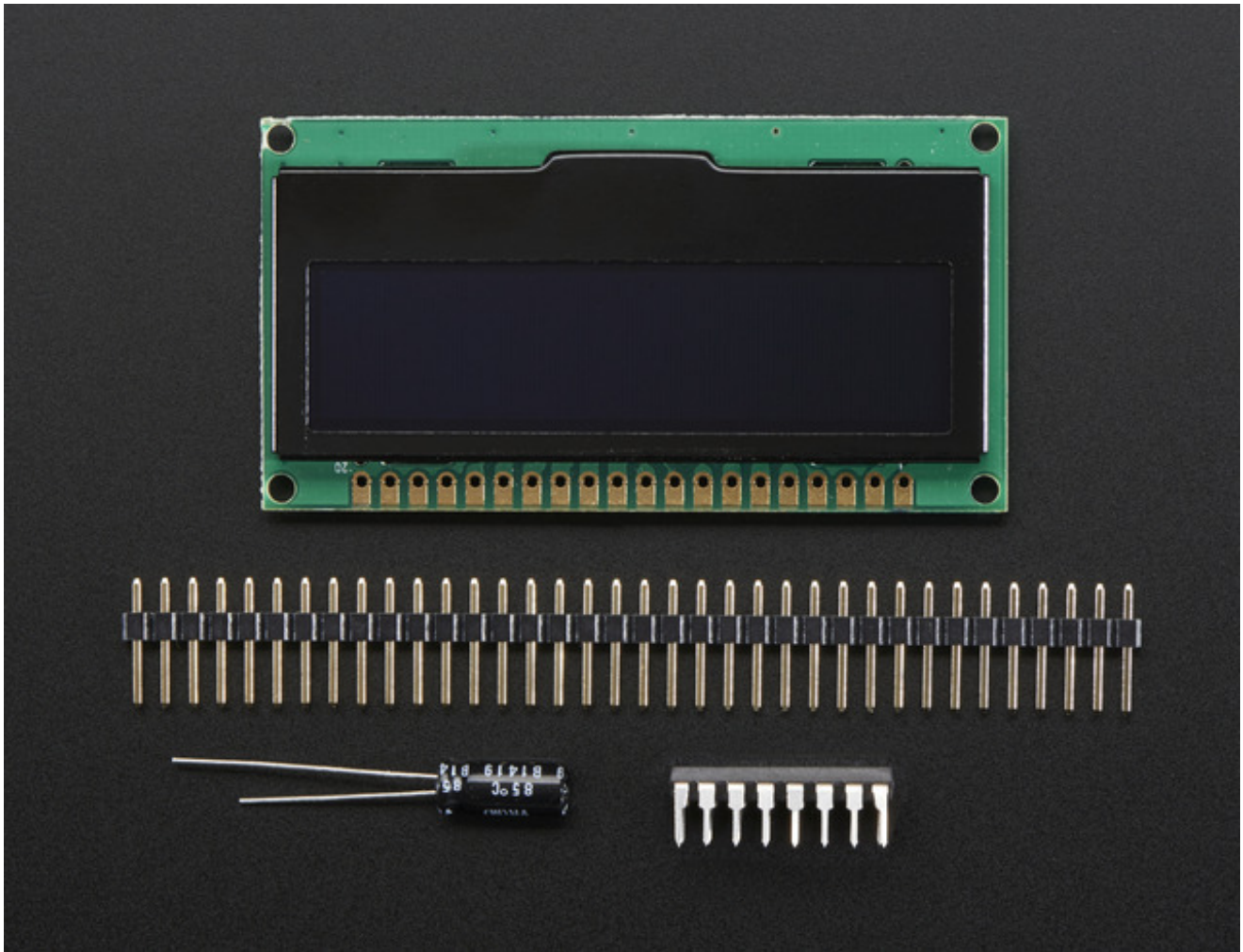
Each order comes with one assembled OLED module with a nice bezel and 4 mounting holes. The display is 3V logic & power so we include a HC4050 level shifter. We also toss in a 220uF capacitor, as we noticed an Arduino may need a little more capacitance on the 3.3V power supply for this big display! This display does not come with header attached but we do toss in a stick of header you can solder on. Also, the display may come in 8-bit mode. You can change modes from 8-bit to SPI with a little soldering, check out the tutorial for how to do so. (http://adafru.it/iTa)
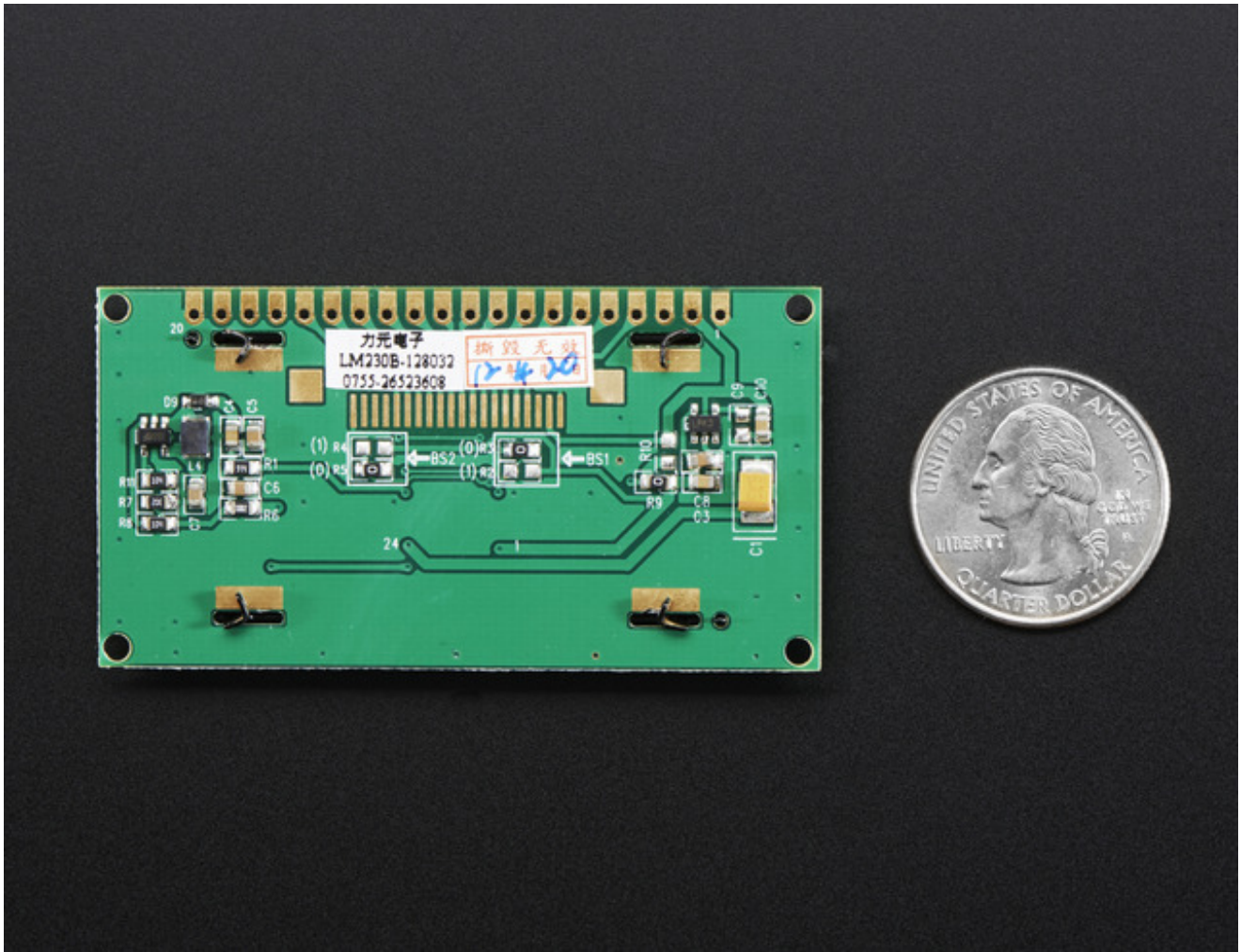
Getting started is easy! We have a detailed tutorial and example code in the form of an Arduino library for text and graphics. (http://adafru.it/j6a) You'll need a microcontroller with more than 512 bytes of RAM since the display must be buffered. The library can print text, bitmaps, pixels, rectangles, circles and lines. It uses 512 bytes of RAM since it needs to buffer the entire display but its very fast! The code is simple to adapt to any other microcontroller.
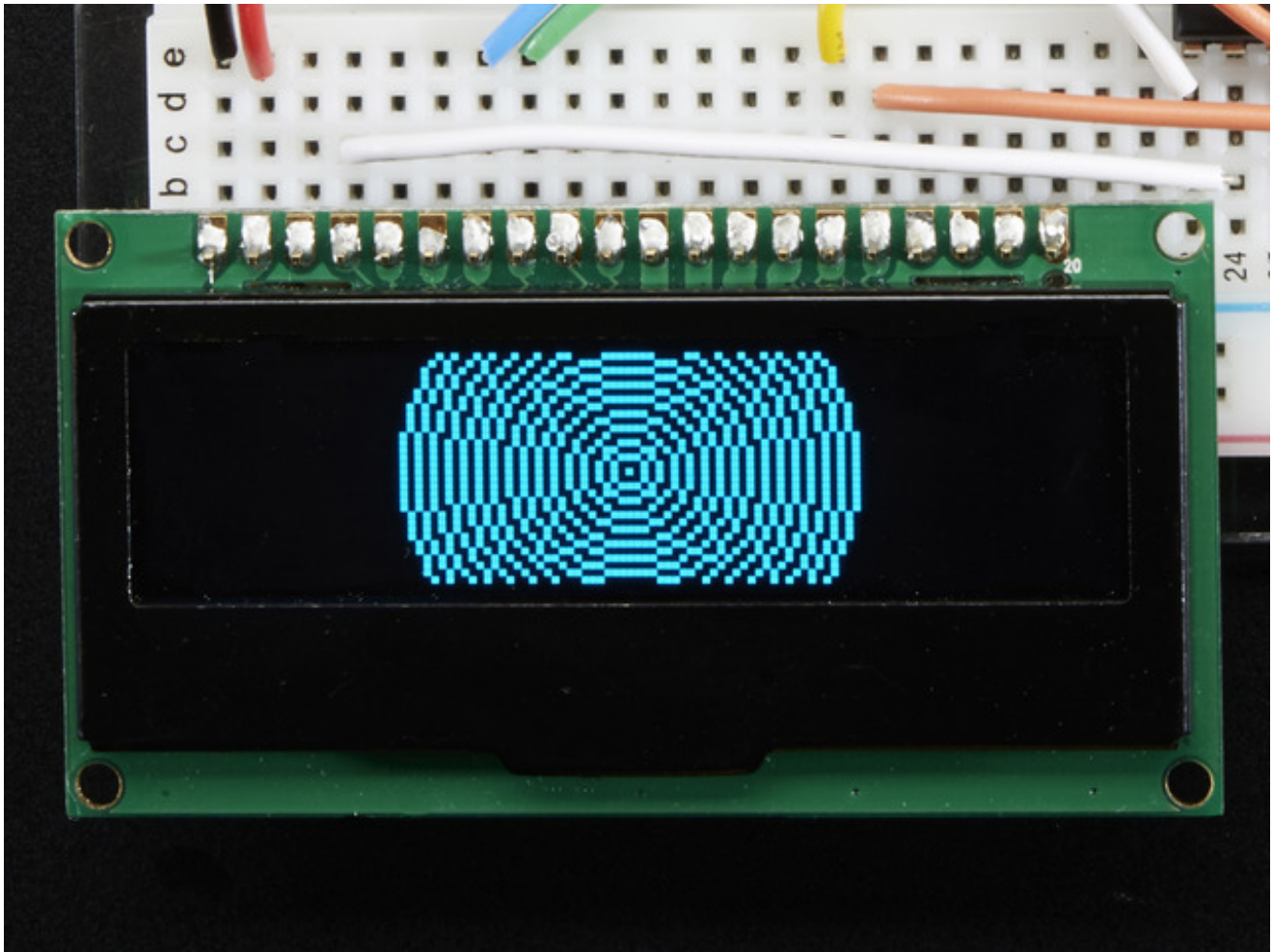
Pinouts
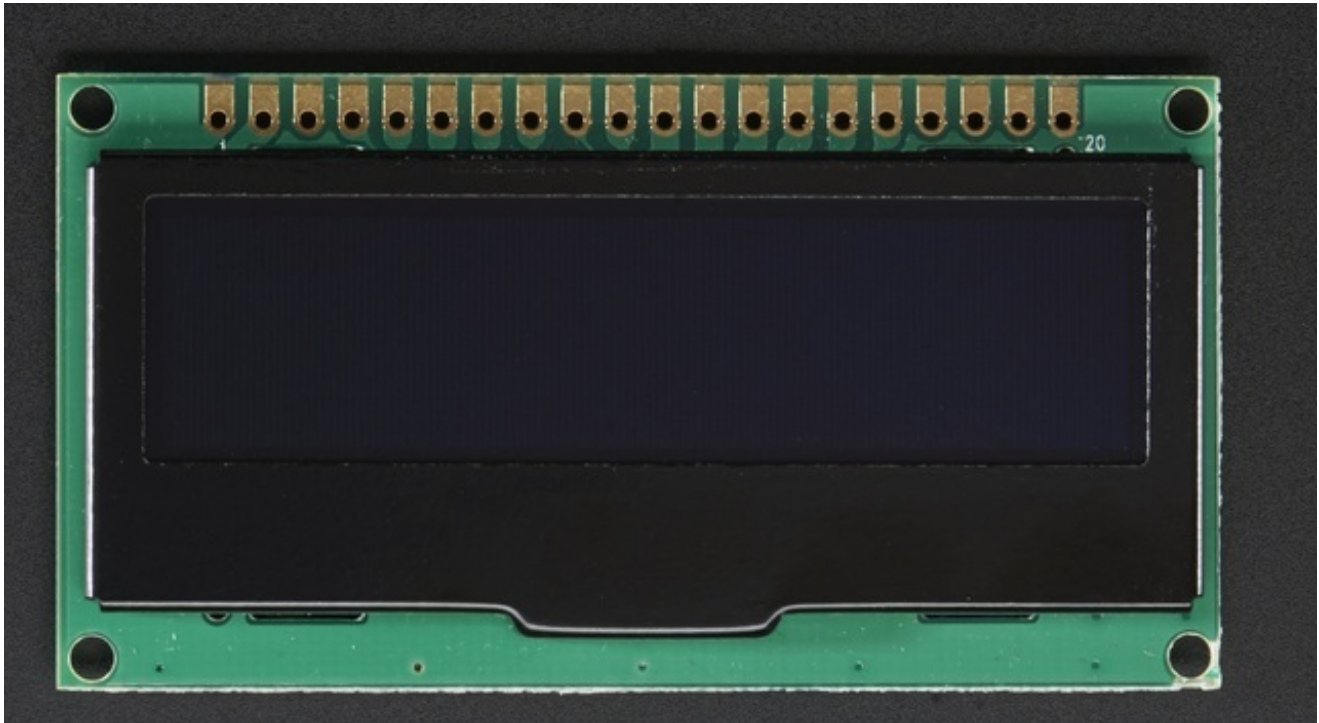
The pins on these modules are not well marked, but the one on left is #1 and the pins increment in order until the one on the very right, #20

Power Pins

- Pin #1 is power and signal **Ground**
- Pin #2 is **3V Power In** - provide 3V with 50-75mA current capability
- Pin #3 is not used, do not connect to anything

Signal Pins

- Pin #4 is **DC** - the **data/command pin**. This is a 3V logic level input pin and is used for both SPI and 8-bit connections
- Pin #5 is **WR** - the 8-bit write pin. This is a 3V logic level input pin and is used for 8-bit connections. Do not connect if using SPI/I2C
- Pin #6 is **RD** - the 8-bit read pin. This is a 3V logic level input pin and is used for 8-bit connections. Do not connect if using SPI/I2C
- Pin #7 is **Data0** - this pin is the **SPI Clock pin, I2C Clock pin** and the 8-bit data bit 0 pin. This is a 3V logic level input pin when used with I2C/SPI, and an input/output when used in 8-bit.
- Pin #8 is **Data1** - this pin is the **SPI Data In pin, I2C Data pin** and the 8-bit data bit 1 pin. This is a 3V logic level input pin when used with I2C/SPI, and an input/output when used in 8-bit.

- Pins #9-14 are **Data2-7** - Used for 8-bit mode. These is a 3V input/output when used in 8-bit. Do not connect if using SPI or I2C
- Pin #15 is **CS** - the **chip select** pin. This is a 3V logic level input pin and is used for both SPI and 8-bit connections
- Pin #16 is **RESET** - the **reset** pin. This is a 3V logic level input pin and is used for I2C, SPI and 8-bit connections

Remaining Pins

- Pins #17-19 are not connected, do not use
- Pin #20 is the 'frame ground' pin and is connected to the metal case around the OLED, you can connect to ground or leave floating.

Assembly

Changing "modes"

These modules can be used in SPI or 8-Bit mode. Somewhat annoyingly, the only way to switch modes is to desolder/solder jumpers on the back of the modules.

SPI Mode

This is the mode you likely want to be in. Your module probably came with this setting by default.

Make sure the **R5** and **R3** resistors are soldered in and the **R2** and **R4** spots are blank
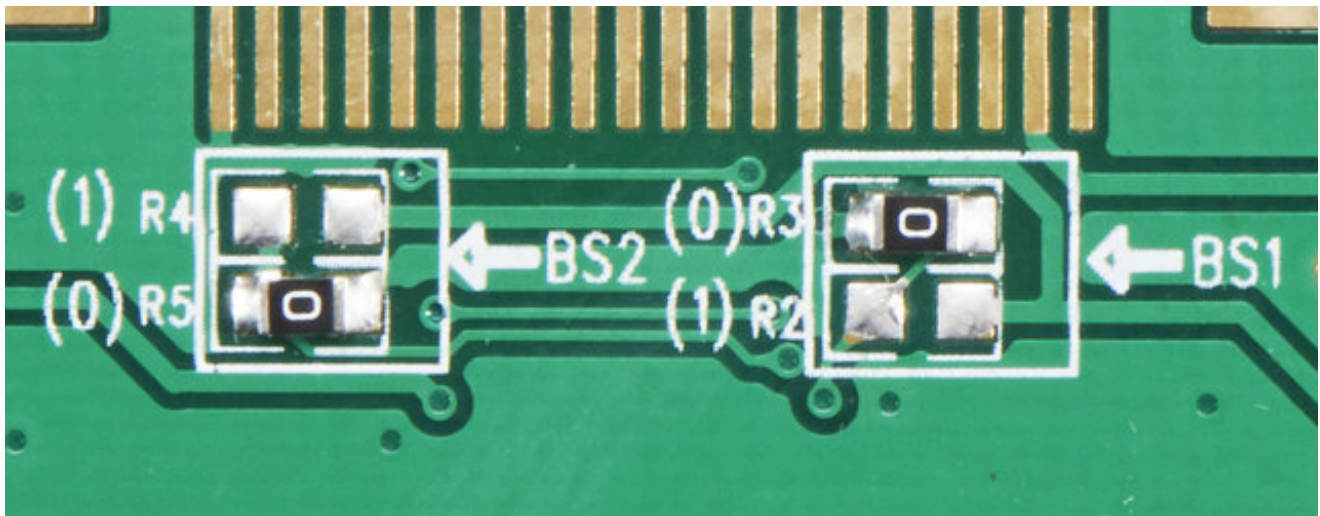
8-Bit "6800" mode

Make sure the **R3** and **R4** resistors are in place and the R2 and R5 are missing. You'll need to remove the R5 resistor to move it, by heating up the resistor with a soldering iron and maybe even melting a little solder on.
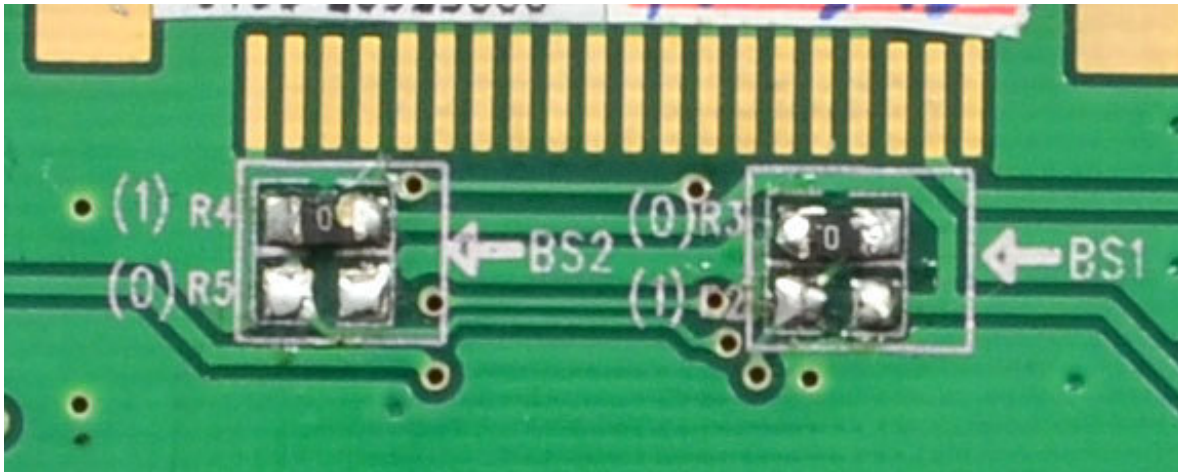
I2C mode

Make sure the **R2** and **R5** resistors are in place and the R3 and R4 are missing. You'll need to remove the R3 resistor to move it, by heating up the resistor with a soldering iron and maybe even melting a little solder on.
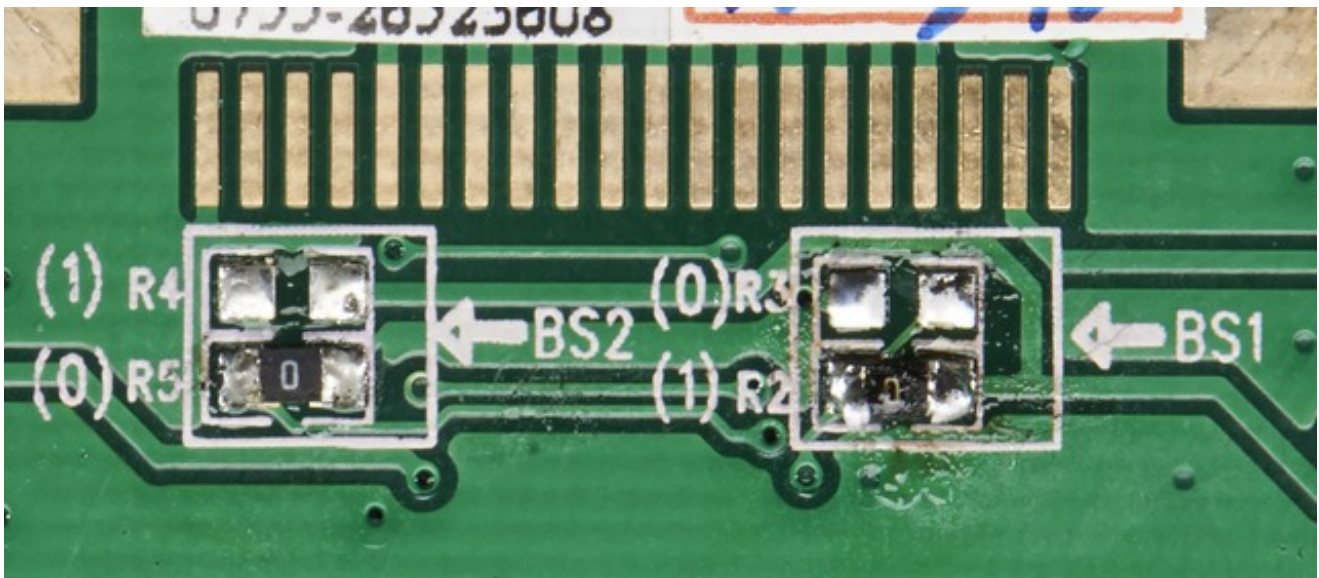
(If you solder and re-solder the resistors 10 times like I did while figuring out all the setting and wiring, you'll get a kinda messy/fluxy look like the above! In that case you can clean up with some IPA wipes. Or just revel in your punk soldering job)

Wiring & Test

We will demonstrate using this display with an Arduino UNO compatible. If you are using a 3V logic device you can skip the level shifter and connect direct from the microcontroller to display. You can also use another kind of level shifter if you like.

Any microcontroller with I2C + 1 pin or 4 or 5 pins can be used, but we recommend testing it out with an UNO before you try a different processor.

SPI Wiring
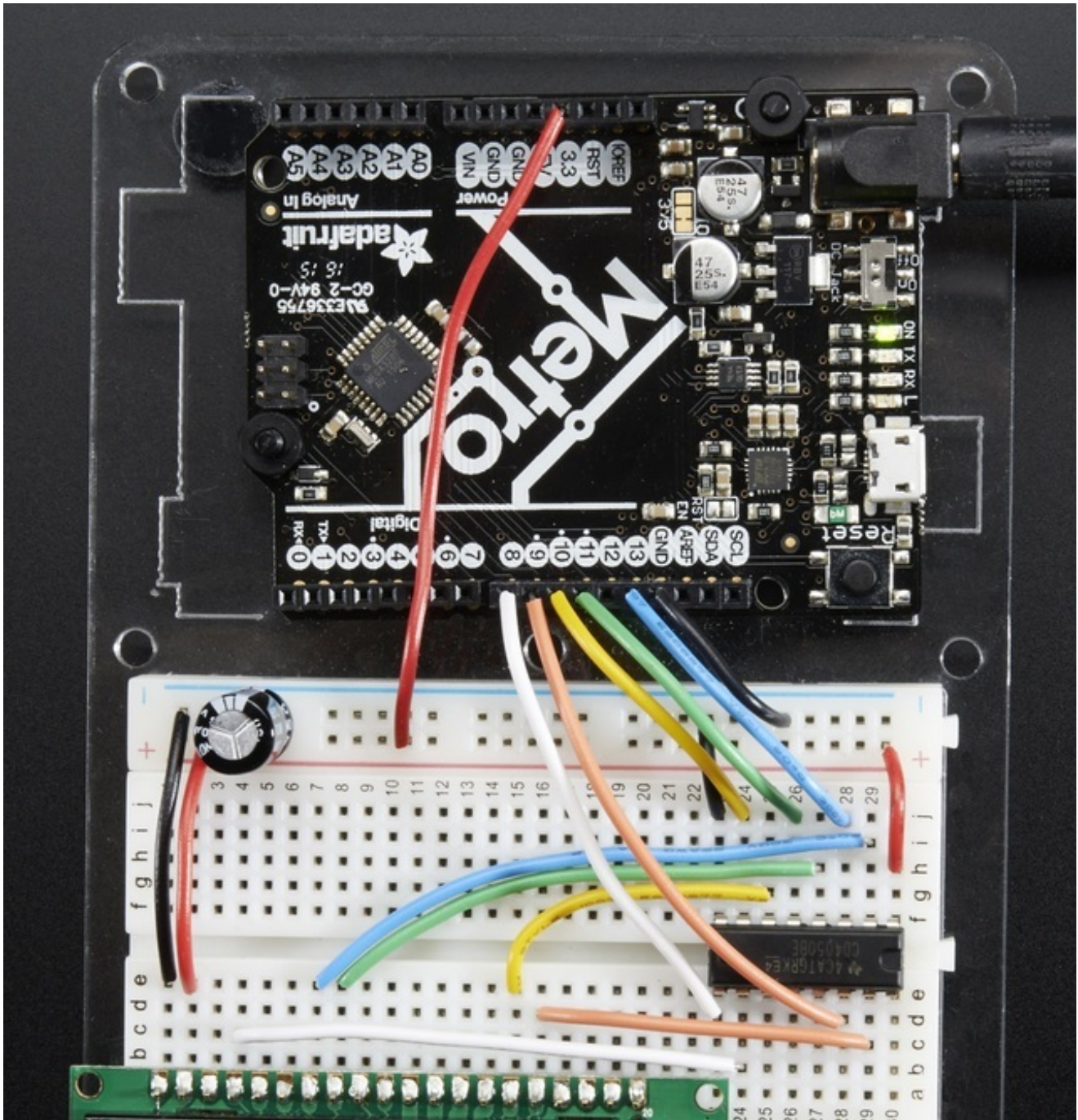
Don't forget you have to set the display to SPI mode, see the Assembly step on how to do that!

Since this is a SPI-capable display, we can use hardware or 'software' SPI. To make wiring identical on all Arduinos, we'll begin with 'software' SPI. The following pins should be used:

- Connect **Pin #1** to common power/data ground line (black wires)
- Connect **Pin #2** to the 3V power supply on your Arduino. (red wires)
- Skip pin #3
- Connect **Pin #4 (DC)** to **digital #8 via the level shifter** (white wires) any pin can be used later

- Connect **Pin #7 (SCLK)** to **digital #13 via the level shifter** (blue wires) any pin can be used later
- Connect **Pin #8 (DIN)** to **digital #11 via the level shifter** (green wires) any pin can be used later
- Skip pins #9-14
- Connect **Pin #15 (CS)** to **digital #10 via the level shifter** (yellow wires) any pin can be used later
- Connect **Pin #16 (RST)** to **digital #9 via the level shifter** (orange wires) any pin can be used later

Later on, once we get it working, we can adjust the library to use hardware SPI if you desire, or change the pins to any others.

Level Shifter Wiring

You will also want to power the HC4050 level shifter by connecting pin #1 to 3V (the red wire) and pin #8 to ground (the black wire)

3.3V Capacitor

We also include a 220uF capacitor with your order because we noticed that the 3V line can fluctuate a lot when powered via an Arduino's 3.3V regulator. We *really* recommend installing it. Clip the leads on this capacitor and connect the negatve pin to **GND** and the positive pin to **3V**

Download Adafruit_SSD1305 library

To begin reading sensor data, you will need to download Adafruit_SSD1305 (http://adafru.it/fLd) from our github repository. You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download Adafruit_SSD1305 Library from Github

http://adafru.it/fLe

Rename the uncompressed folder **Adafruit_SSD1305** and check that the **Adafruit_SSD1305** folder contains **Adafruit_SSD1305.cpp** and **Adafruit_SSD1305.h**

Place the **Adafruit_SSD1305** library folder your *arduinosketchfolder/libraries/* folder. You may need to create the libraries subfolder if its your first library. Restart the IDE.

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

Installing Adafruit_GFX

You'll also need to download the 'underlying' graphics support library that does all the lines, text,

rectangle drawing for you. You can grab it from https://github.com/adafruit/Adafruit-GFX-Library (http://adafru.it/fKs)

Or click on the button below and follow the same instructions you did for the **Adafruit_SSD1305** library. Uncompress and rename the folder to **Adafruit_GFX** (contains **Adafruit_GFX.cpp** and **.h**). Then place them inside your Arduino **libraries** folder and restart the Arduino IDE. If this is all unfamiliar, we have a tutorial introducing Arduino library concepts and installation (http://adafru.it/aYM).

<div style="background:#29abe2; color:white; text-align:center; padding:1em;">Download Adafruit GFX library</div>

http://adafru.it/cBB
Restart the IDE!

Adjust display size

Before you compile and run the demo, you'll need to make sure you have the library configured to the right size. Open up the **Adafruit_SSD1305.h** file in your ArduinoSketchFolder/**libraries**/**Adafruit_SSD1305** folder and look for the lines that have:

```
#define SSD1305_128_32
//#define SSD1305_128_64
```
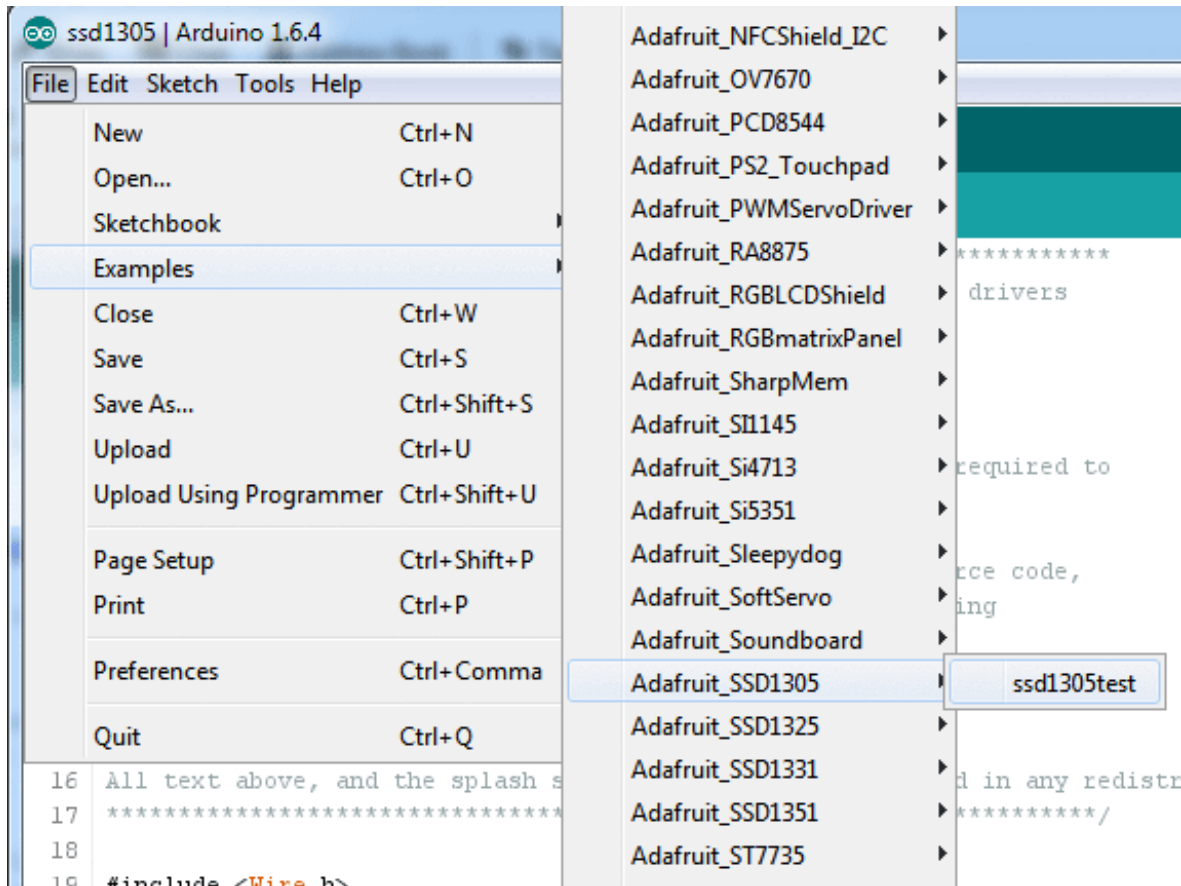
If you are running a 128x32 display, leave the second line commented

If you are running a 128x64 size display, comment the first line and uncomment the second one. Then save the file

Running the Demo

After restarting the Arduino software, you should see a new **example** folder called **Adafruit_SSD1305** and inside, an example called **ssd1305test**

Now upload the sketch to your Arduino. That's pretty much it! You should see immediate update of the display.

If nothing shows up at all, make sure you have your wiring correct, we have a diagram above you can use. Also, check that you converted the module to "SPI" mode (see the Assembly) step on how to do that

Changing Pins

Now that you have it working, there's a few things you can do to change around the pins.

If you're using Hardware SPI, the CLOCK and MOSI pins are 'fixed' and cant be changed. But you can change to software SPI, which is a bit slower, and that lets you pick any pins you like. Find these lines:

```
    // If using software SPI, define CLK and MOSI
#define OLED_CLK 13
#define OLED_MOSI 11

// These are neede for both hardware & softare SPI
#define OLED_CS 10
#define OLED_RESET 9
#define OLED_DC 8
```

Change those to whatever you like!

Using Hardware SPI

If you want a little more speed, you can 'upgrade' to Hardware SPI. Its a bit faster, maybe 2x faster to draw but requires you to use the hardware SPI pins.

- **SPI CLK** connects to SPI clock. On Arduino Uno/Duemilanove/328-based, thats **Digital 13**. On Mega's, its **Digital 52** and on Leonardo/Due its **ICSP-3** (See SPI Connections for more details (http://adafru.it/d5h))
- **SPI DATA IN** connects to SPI MOSI. On Arduino Uno/Duemilanove/328-based, thats **Digital 11**. On Mega's, its **Digital 51** and on Leonardo/Due its **ICSP-4** (See SPI Connections for more details (http://adafru.it/d5h))

To enable hardware SPI, look for these lines:

```
// this is software SPI, slower but any pins
Adafruit_SSD1305 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);

// this is for hardware SPI, fast! but fixed oubs
//Adafruit_SSD1305 display(OLED_DC, OLED_RESET, OLED_CS);
```
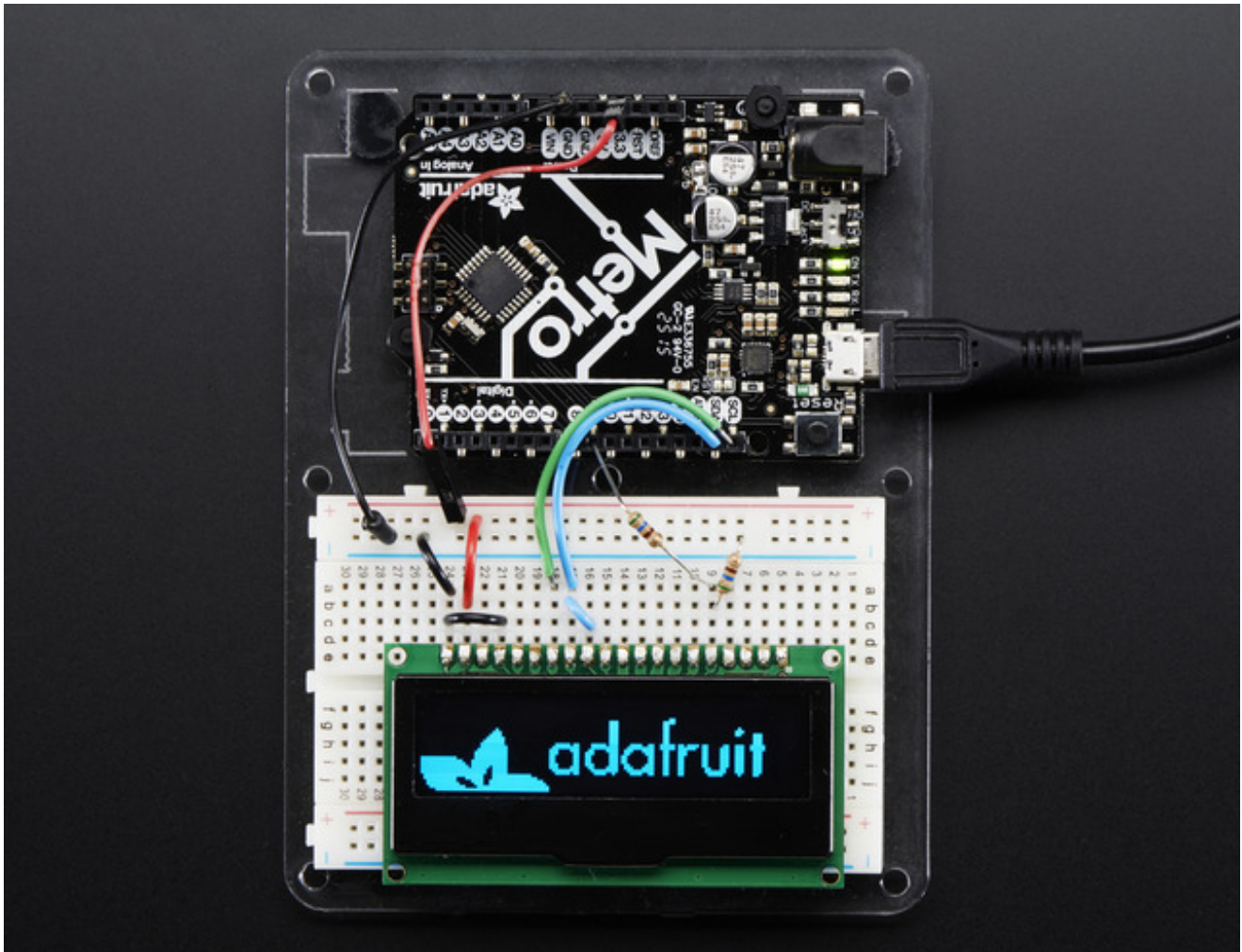
Comment out the top line and uncomment the bottom line

I2C Wiring

It is also possible to use the display in I2C mode. Its a little slower but uses way fewer pins.

Don't forget you have to set the display to I2C mode, see the Assembly step on how to do that!

For I2C you will need to use the hardware I2C pins on your Arduino or microcontroller. The following pins should be used:

- Connect **Pin #1** to common power/data ground line (black wires)
- Connect **Pin #2** to the 3V power supply on your Arduino. (red wires)
- Skip pin #3
- Connect **Pin #4 (DC & I2C Addr0)** to **ground** (black wire) to set the I2C address to 0x3C. If this is tied to 3.3V, it will set the I2C address to 0x3D
- Connect **Pin #7 (SCL)** to **Arduino SCL** (green wire)
- Connect **Pin #8 (SDA)** to **Arduino SDA** (blue wire)
- Connect **Pin #9 (SDA2)** to **Pin #8** (small blue wire)
- Skip pins #9-15
- Connect **Pin #16 (RST)** to **digital #9** by using a resistive divider as shown, two resistors from 1K to 10K both the same value can be used. Any pin can be used later

While its ideal to use level shifters on the I2C pins, you can sorta get away with this on an arduino,

because the I2C pins are open collector and there are very weak pullups on those two lines. If using with other I2C devices, we suggest using a 3V-logic arduino or an I2C-safe shifter. (http://adafru.it/757)

Later on, once we get it working, we can adjust the library to use hardware SPI if you desire, or change the pins to any others.

3.3V Capacitor

We also include a 220uF capacitor with your order because we noticed that the 3V line can fluctuate a lot when powered via an Arduino's 3.3V regulator. We *really* recommend installing it. Clip the leads on this capacitor and connect the negatve pin to **GND** and the positive pin to **3V**

I2C code changes

In the test code, change the top area where you define the protocol used by commenting out the software and hardware SPI and uncommenting the I2C version

```
// software SPI
//Adafruit_SSD1305 display(OLED_MOSI, OLED_CLK, OLED_DC, OLED_RESET, OLED_CS);
// hardware SPI
//Adafruit_SSD1305 display(OLED_DC, OLED_RESET, OLED_CS);
// I2C
Adafruit_SSD1305 display(OLED_RESET);
```

Everything else about the display is identical to SPI mode.

By default we use I2C address **0x3C** which is what we get by connecting **DC**/**A0** to ground. If you tie that pin to 3.3V instead, the address will be **0x3D** and all you have to do is call **display.begin(0x3D)** to initialize with that address.

Using Adafruit GFX
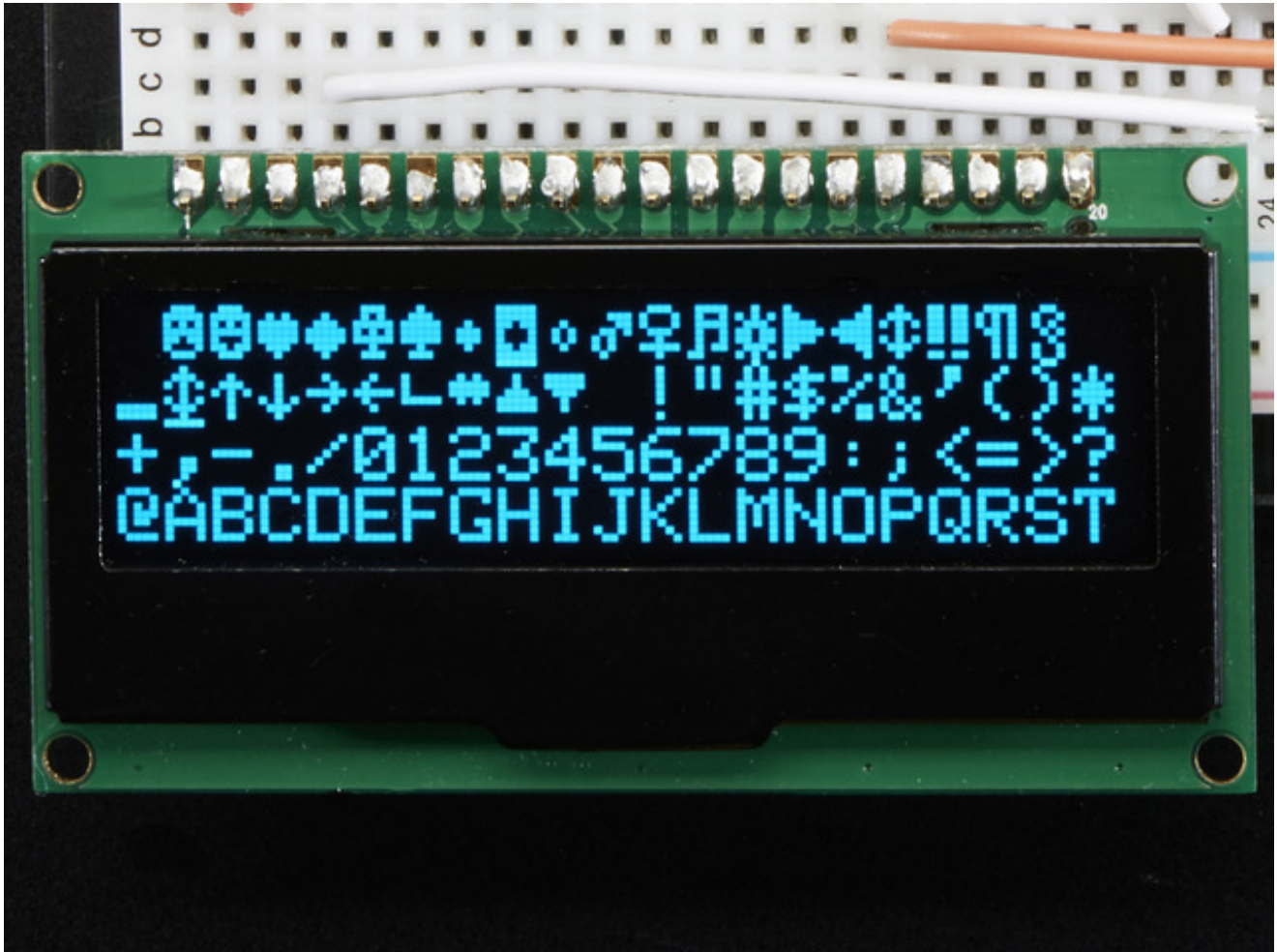
The Adafruit_GFX library for Arduino provides a common syntax and set of graphics functions for all of our TFT, LCD and OLED displays. This allows Arduino sketches to easily be adapted between display types with minimal fuss…and any new features, performance improvements and bug fixes will immediately apply across our complete offering of displays.

The GFX library is what lets you draw points, lines, rectangles, round-rects, triangles, text, etc.

Check out our detailed tutorial here http://learn.adafruit.com/adafruit-gfx-graphics-library (http://adafru.it/aPx) It covers the latest and greatest of the GFX library!

Since this is a 'buffered' display, dont forget to call the "display()" object function whenever you want to update the OLED. The entire display is drawn in one data burst, so this way you can put down a bunch of graphics and display it all at once.
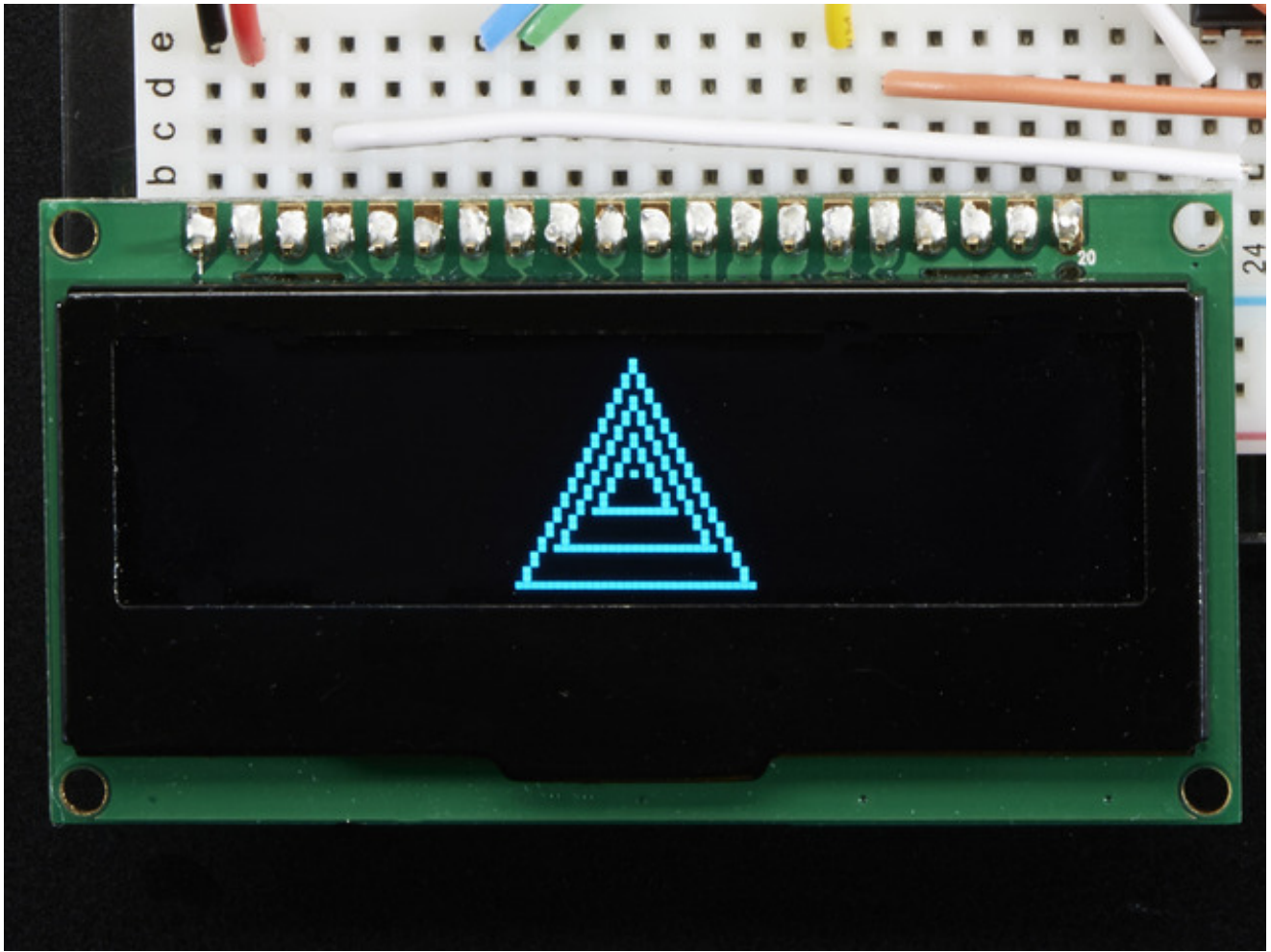
F.A.Q.

How come sometimes I see banding or dim areas on the OLED?

These OLEDs are passively drawn, which means that each line is lit at once. These displays are fairly inexpensive and simple, but as a tradeoff the built in boost converter has to drive all the OLED pixels at once. If you have a line with almost all the pixels lit it wont be as bright as a line with only 50% or less lit up.

The display works, because I can see the splash screen, but when I draw to the display nothing appears!

Don't forget you **must** call **.display()** to actually write the display data to the display. Unlike many of our TFTs, the entire display must be written at once so you should print all your text and draw all your squares, then call display()

How do I get rid of the splash screen?

Open up **Adafruit_SSD1305.cpp** in the libraries/Adafruit_SSD1305 folder and find these lines

```
static uint8_t buffer[SSD1305_LCDHEIGHT * SSD1305_LCDWIDTH / 8] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
....
0x00, 0x01, 0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x01
};
```

and delete everything after `static uint8_t buffer[SSD1305_LCDHEIGHT * SSD1305_LCDWIDTH / 8] = {` and before `};`

Downloads

Datasheets

- Datasheet for the SSD1305, the passive OLED driver chip in the module (http://adafru.it/fLg) this is the chip in the module that converts SPI/8-bit commands to OLED control signals
- Mechanical specifications for the OLED display itself (http://adafru.it/fLh)