Serial Communication: Button Press Counter with OLED Display

Digital information transmission takes one of two flavors. First is parallel communication. In this method, each input or output information gets its own communication line. In our case that's a pin on the Arduino board. You are already experts at this! You've sent information to buzzers, LEDs, and other peripherals by asserting a pin on the Arduino as HIGH or LOW. You've also read data in a similar fashion. By reading one pin, you've detected a beam break or button press. Things get more complicated if you or your peripheral device needs to know more than one bit of information at a time. What could we do? "We'll just add more pins!". A wonderful idea. You've done that too! You used 3 pins on the Arduino to control a RGB LED, or 7 pins to control the seven segment display. That wasn't too bad, but what if you need to control thousands of pixels on an OLED screen. Now things are getting out of hand. We only get 14 digital input/output (I/O) pins on our Arduino Uno.

Enter the second method of digital communication. Serial communication sends many bits of data sequentially/serially using fewer pins. By sacrificing a bit of time (which is miniscule due to the speed of modern computing) we can send Gigabits of data per second using only a few pins![1]

The word 'Serial' may have rang a few bells. That's because you've been using serial communication throughout the course! You've used it to program your Arduino board and to print its output on the Serial Monitor via the Universal Serial Bus or USB. I hope you find this tidbit demystifying.

There are many methods of serial communication. Although USB is ubiquitous, we will use an older, slower, but simpler serial communication protocol called Inter-integrated Circuit or I2C[2]. I2C can't give us Gigabits of bandwidth but it will do just fine for almost everything around the lab. More importantly, there are cool peripherals available for purchase that support I2C and come with easy to use libraries[3].

In this exercise we will use I2C to control an OLED screen with over 8 thousand pixels! I hope the prospect of which has rendered your socks knocked off.

[1] Our little microcontroller can't handle such speeds but many, not as easy to program, microcontrollers can.

[2] Pronounced 'Ay-too-see' or 'Ay-squared-see' and never fully uttered in its unabbreviated form.

[3] In other words, someone has already done all the difficult programming for you as long as you buy their product. I suggest you exchange your dollars for their platform. This shall not dent your RO1s and it will save you much time.

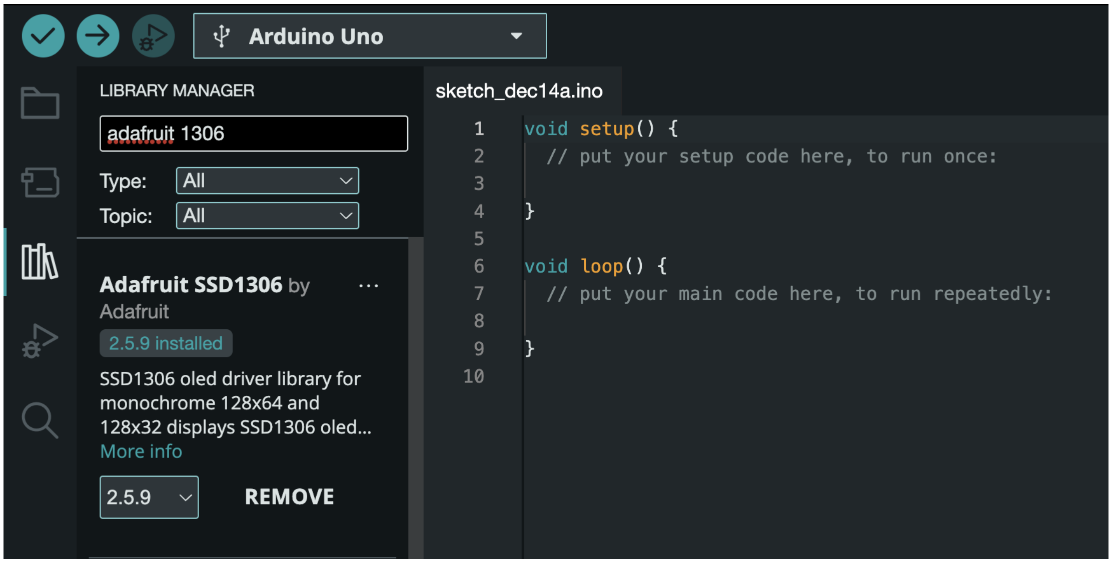
First things first, we need to connect our OLED screen to the arduino. The I2C protocol uses two pins: Data and Clock. Arduino's versions of Data and Clock are named SDA (Serial Data) and SCL (Serial Clock) respectively. Those are pins A4 and A5 on the Arduino Uno. In addition to data and clock, the screen needs power. We will use the 5V and Ground pins on the Arduino to power our screen.

Use the provided connector to access the screen's pins:

- **Red**:	Power
- **Black**:	Ground
- **Blue**:	Data / SDA
- **Yellow**:	Clock / SCL

Our hardware should be all set and if your Arduino is connected to your computer, a green LED in the back should indicate successful power delivery!

To program the OLED display, we first need to install the library that the manufacturer has provided to us: Open the Arduino IDE and navigate to the library manager. The library manager can be accessed via a button on the left control panel[4], or by going to **Sketch>Include Library>Manage Libraries**[5]. From there, we need to tell the IDE which library to download. Type 'Adafruit SSD1306' and install the library[6][7].



adafruit 1306

Type: All

Topic: All

Adafruit SSD1306 by Adafruit

2.5.9 installed

SSD1306 oled driver library for monochrome 128x64 and 128x32 displays SSD1306 oled...

More info

2.5.9

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}

```

[4] Only in Arduino IDE 2.0

[5] Adafruit is pretty good at publishing their libraries on the Arduino Library Repository. Others might not. Refer to this link if you're interested in how to install libraries that don't appear on the library manager.

[6] At this point you're asked if you'd be ok with installing the librarie's required libraries. Say yes.

[7] I've tested the hardware on version 2.5.9

Before we get to programming our own application. Let's marvel at all the cool things you could do with such a screen. Conveniently, this library comes with an example program to showcase the capability of the hardware as well as its library[8]. To access it, navigate to **File>Examples>Adafruit SSD1306>SSD1306_128x64_i2c**[9]. Compile the example code and amaze at the graphics that it generates. At this point I encourage you to skim the example code and gain some familiarity with the library commands[10].

You'll notice that some libraries are imported to make the program work.

- **SPI.h**: We actually don't need this because we are not going to use SPI
- **Wire.h**: This library allows us to instantiate I2C
- **Adafruit_GFX.h**: This is Adafruit's graphics library that is used for many of their products and allows for generation of simple graphics with minimal code.
- **Adafruit_ssd1306.h**: This is the library we'll use to communicate with the OLED

Now let's implement something. Wire up a push button based on instructions from the first class. Remind yourself that the pushbutton's normally open terminal is connected to 5V. Therefore, the pushbutton goes to 0V when it's pressed.

We're going to write code to count how many times the button is released[11] and display it on the screen. Note that you could do this with many input signals: the beam breaker, a sensor, or a square pulse generated by your lab instruments.

Copy the skeleton code from the course website and complete it to implement the button counter.

[8] Most libraries do

[9] You may have noticed that another version of the example code ends with SPI instead of I2C. Turns out our handy dandy screen supports Serial Peripheral Interface (SPI) as well.

[10] Look here for a detailed documentation of the library

[11] You want to count up when the voltage is going from 0V to 5V.