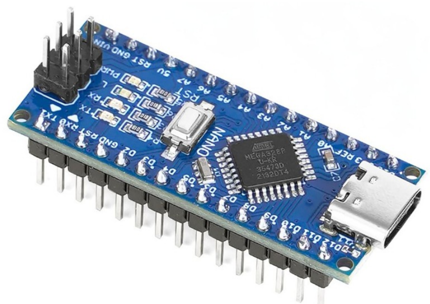
Lab 5: Serial output

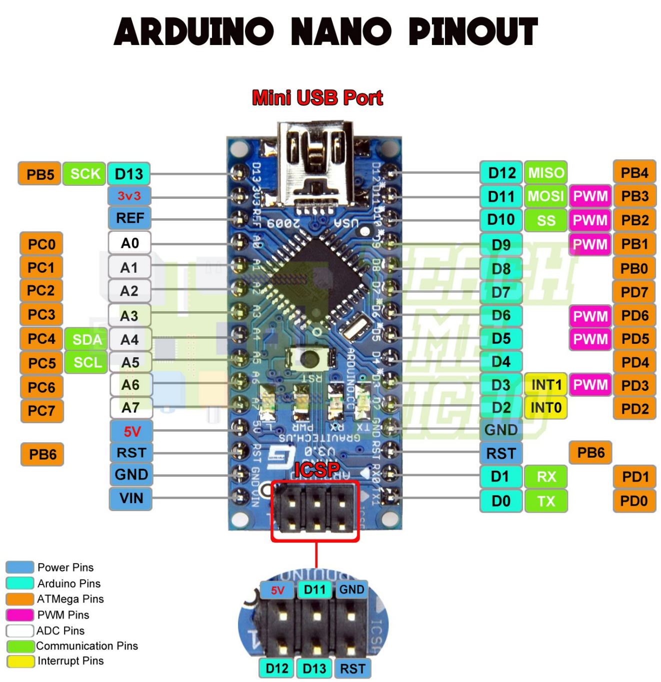


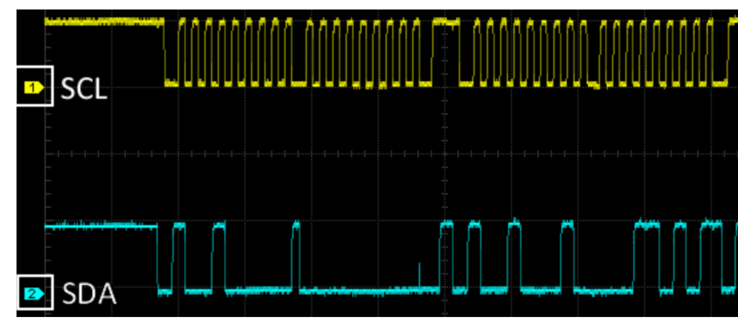
Lab goals

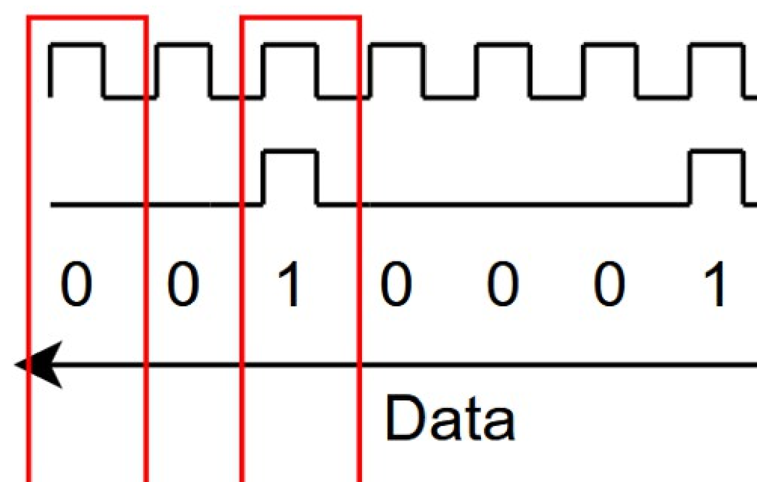
- Get acquainted with the microcontroller's capability for working with output devices via serial communications

Required hardware/software

- Everything from lab 4, plus:
- OLED display x 1







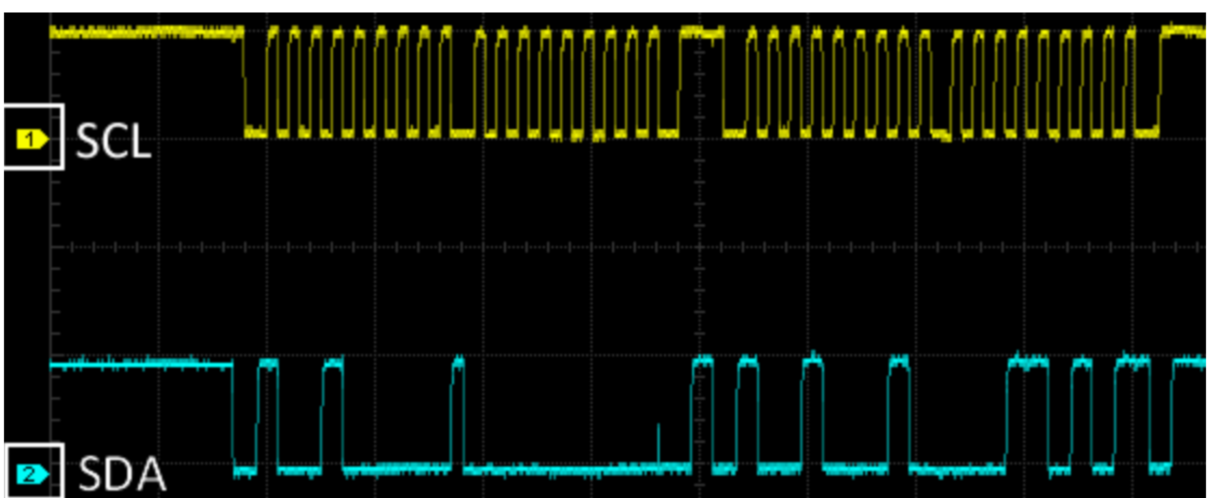
Lab 5: Serial output (background)

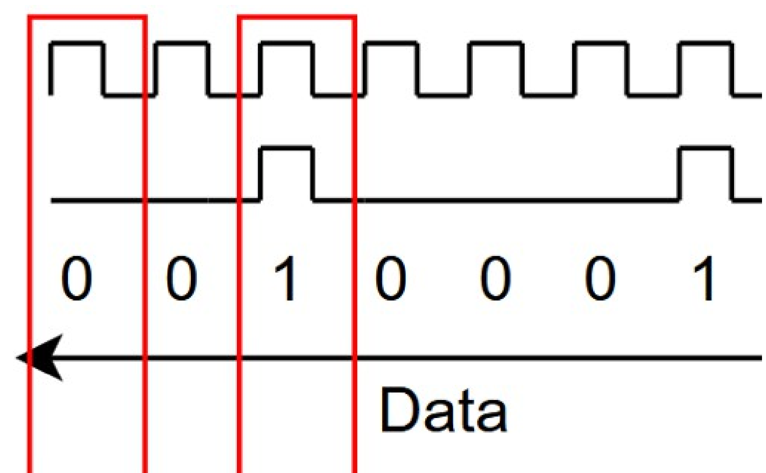
Recall that microcontrollers often use “serial communication” to interact with outside peripherals (e.g. displays, sensors, etc) – to “talk to one another.”

Serial communication allows a microcontroller to use a minimum of wires to exchange data with outside peripherals.

This communication is digital / binary, meaning that the data exchanged is represented in terms of 0's and 1's / LOW and HIGH voltage levels / ON and OFF states.

We can represent digital data over an analog medium (voltage on a wire) using square waves, meaning that voltage is effectively either completely present (1 / HIGH / ON) or completely absent (0 / LOW / OFF).

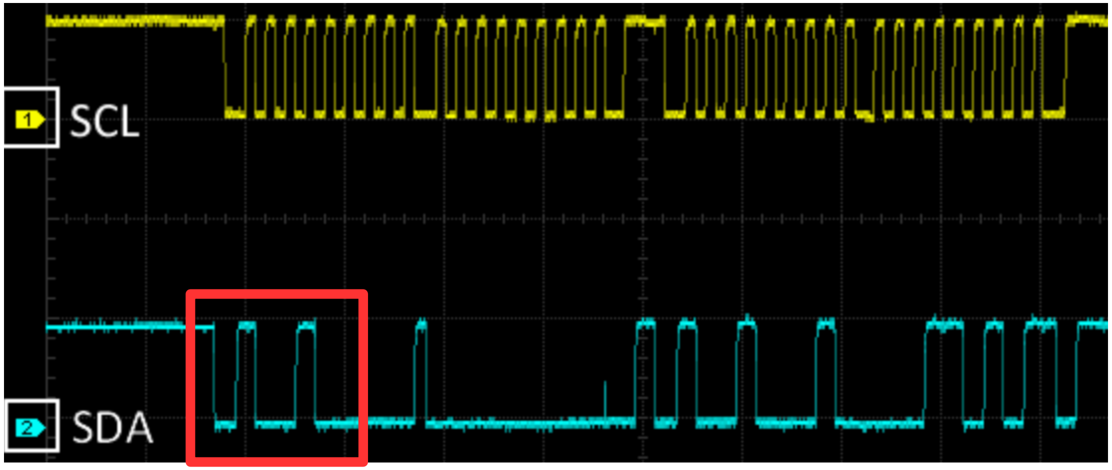


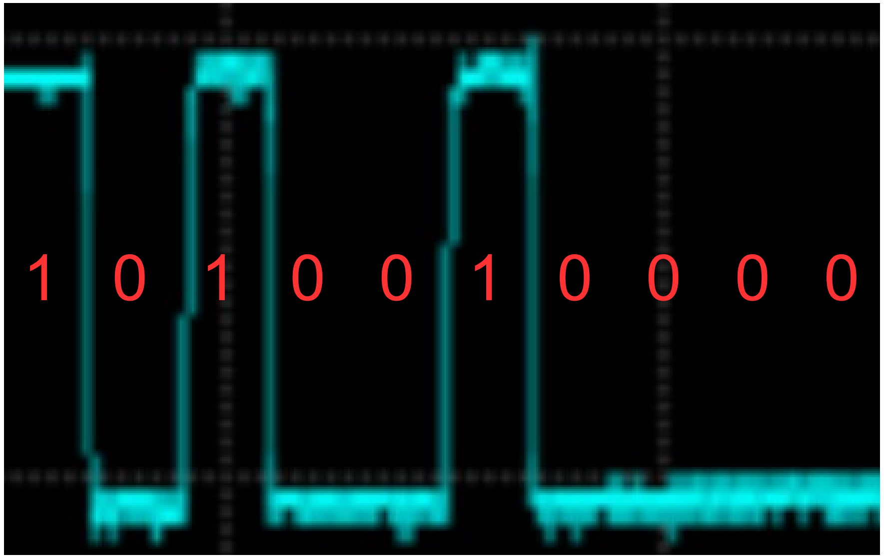


Lab 5: Serial output (background)

Serial communications work by sending 0's and 1's in the form of two different voltage levels, commonly 0 VDC (LOW) and 3.3 VDC (HIGH) or 0 VDC (LOW) and 5 VDC (HIGH).

Because there are only two voltage levels used in this form of communication, we refer to these as “square waves” and they tend to look something like this when observed with an oscilloscope:



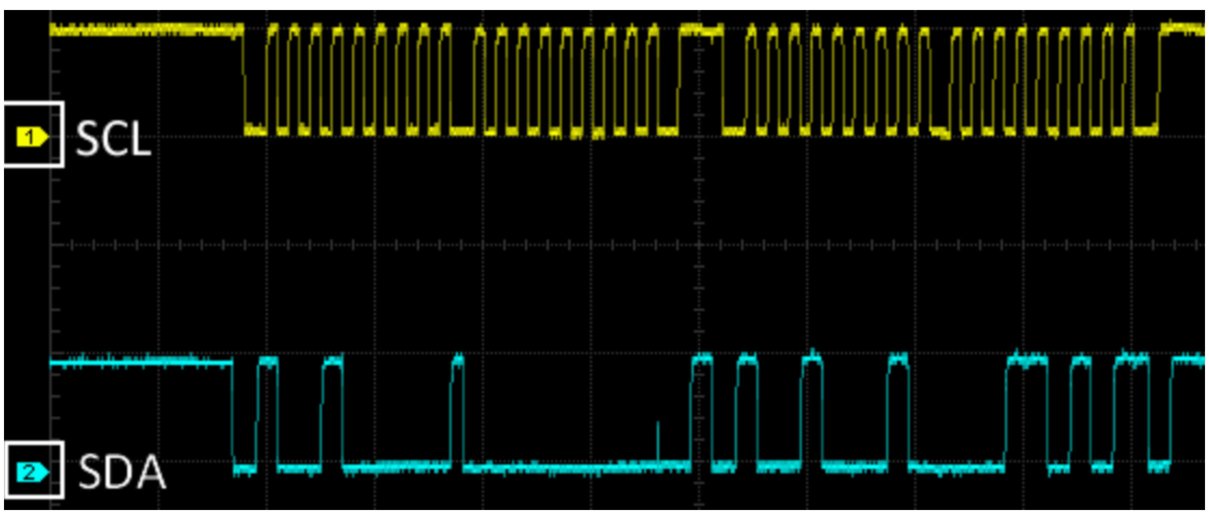


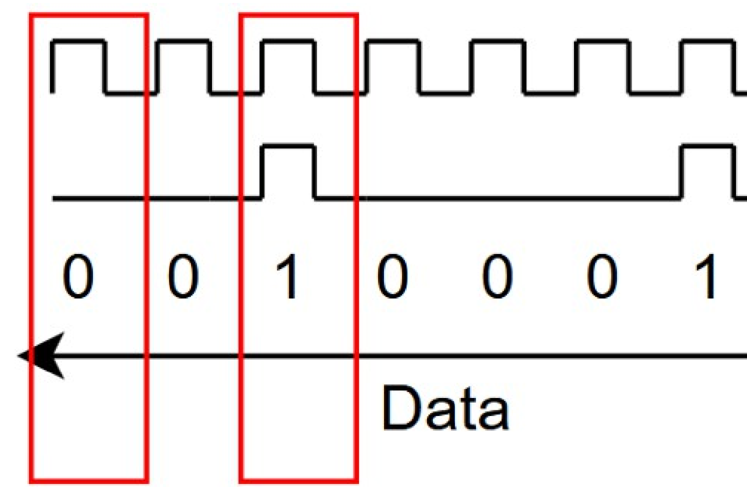
In actuality, the more you “zoom in”, the less square things actually look. There will always be a certain amount of noise present and there will always be a measurable rise/fall time when changing logic levels. ***This is the underlying analog reality of digital communications.***

Lab 5: Serial output (background)

Also recall that it is common for a “clock” signal to be present alongside the “data” signal, although this is not always the case. The clock signal helps differentiate bits and helps keep the sender and receiver synchronized.

Further recall that the time between bits is the “bit rate” or “data rate”, and can generally be referred to as the “baud rate” (there are very real exceptions to this but they are out of the scope of this introductory course – everything we cover today will have “bit rate” = “baud rate”).





Lab 5: Serial output (background)

The most common serial communications “protocols” used by microcontrollers fall into two categories: synchronous (separate clock signal) and asynchronous (no separate clock signal):

Synchronous	Asynchronous
I^2C	UART
SPI	
USART	

We will only directly interact with I^2C-based peripherals today but will also briefly touch on SPI due to both its popularity and its similarity to I^2C.

Lab 5: Serial output (background)

<u>I2C</u>
Inter-Integrated Circuit (aka IIC aka I^2C)

Pronounced "eye-squared-see"

I^2C essentially equivalent to Two Wire Interface (TWI) although they are not technically completely equivalent

Uses two wires:
- SCL: serial clock
- SDA: serial data

Master-slave architecture

Must include "address" of intended slave in header data because no *slave select* line

Typical maximum bit rate: 400,000 bits/second

<u>SPI</u>
Serial Peripheral Interface

Pronounced "spy"

Uses four wires:
- SCLK: serial clock
- SS (CS): slave select (chip select)
- MOSI: serial data from master to slave
- MISO: serial data from slave to master

Master-slave architecture

Need not include address of intended slave in header data

Bit rates exceeding 10,000,000 bits/second not uncommon (significantly faster than I^2C)

Lab 5: Serial output (background)

Recall that binary numbers can be converted to hexadecimal and decimal numbers and vice versa.

For instance, we can refer to the binary number “1001011” as:

- 4B in hexadecimal
- 75 in decimal

By the way, “1001011” would be represented “on the wire” (by voltage changes over a specified period of time) as HIGH, LOW, LOW, HIGH, LOW, HIGH, HIGH.

Each peripheral will have a “datasheet” and part of that datasheet will define what data can be exchanged with it.

(Next slide)

Lab 5: Serial output (background)

SOLOMON SYSTECH
SEMICONDUCTOR TECHNICAL DATA

SSD1306

Advance Information

128 x 64 Dot Matrix
OLED/PLED Segment/Common Driver with Controller

Lab 5: Serial output (background)

Lab 5: Serial output (background)

10.1.12 Set Display ON/OFF (AEh/AFh)

These single byte commands are used to turn the OLED panel display ON or OFF.
When the display is ON, the selected circuits by Set Master Configuration command will be turned ON.
When the display is OFF, those circuits will be turned OFF and the segment and common output are in V_{SS} state and high impedance state, respectively. These commands set the display to one of the two states:

- AEh : Display OFF
- AFh : Display ON

We are skipping a couple of additional details here, but essentially, sending “AE” (hexadecimal) turns the OLED display off and sending “AF” (hexadecimal) turns the display on.

For example:

Display on:
AF (hexadecimal), written as 0xAF or Afh

0xAF = 1010 1111

We would send 1010 1111 (HIGH LOW HIGH LOW HIGH HIGH HIGH HIGH) on the I^2C bus, addressed to the OLED display controller, to turn the OLED display on.

Lab 5: Serial output (background)

Datasheets tell us exactly how to communicate with devices via serial communications.

But don't worry about all of these “scary” details!

The “drivers” or “libraries” handle all of this for you (hide the details from you).

For instance, calling a piece of code such as:

turnDisplayOn()

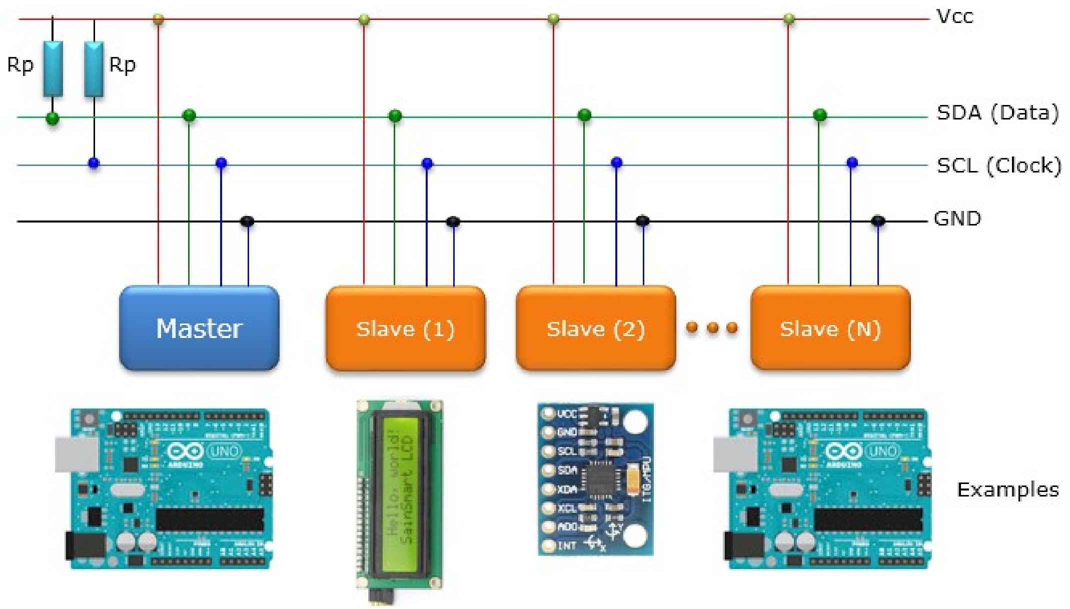
would handle all of the details of performing an I^2C transmission with the appropriate data values, etc.

Lab 5: Serial output (background)

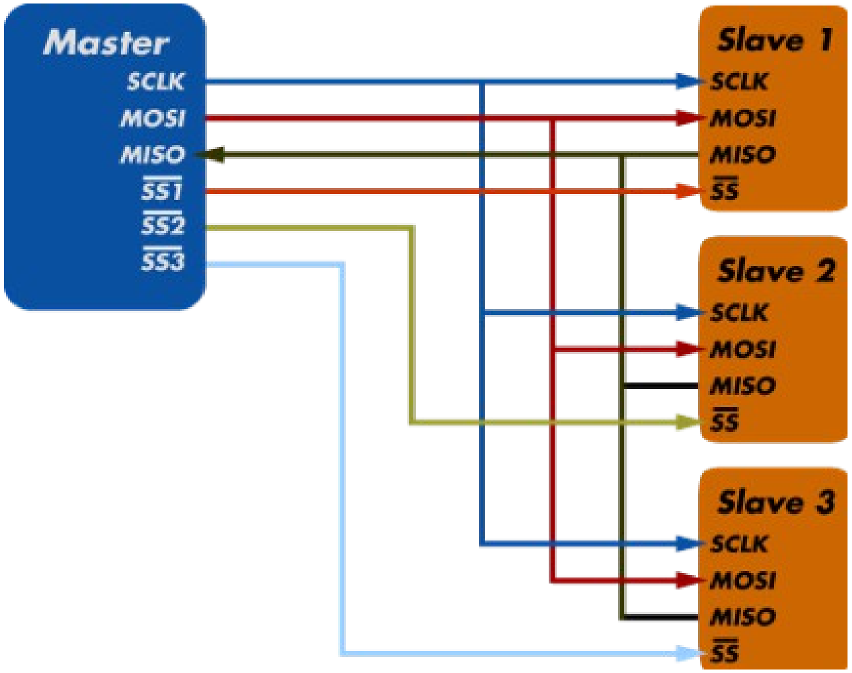
I^{2}C and SPI are both “bus” protocols. In computing, a bus is a shared physical pathway that allows multiple devices to communicate.

It is not uncommon to have multiple slave devices (peripherals) on a single I^{2}C or SPI bus.

I^{2}C bus



SPI bus



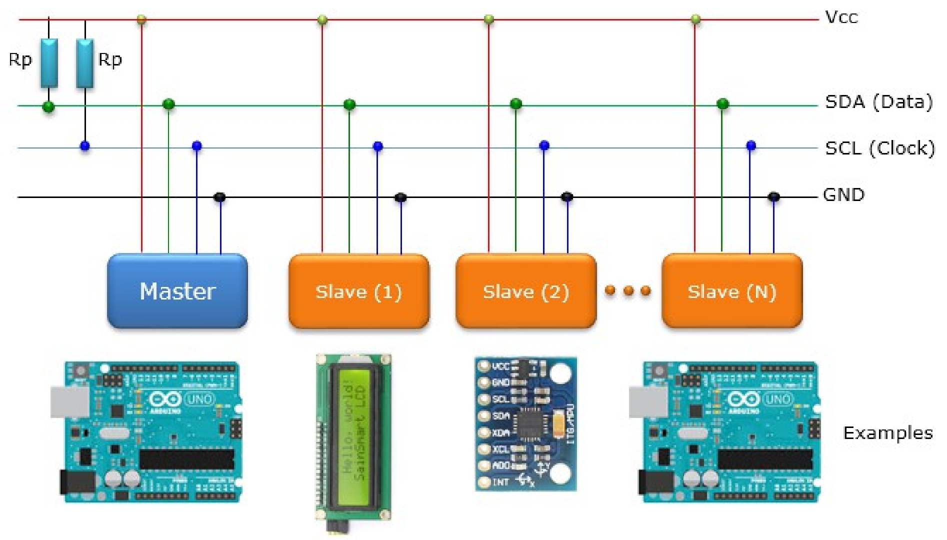
Lab 5: Serial output (background)

I^2C and SPI are both based on a “master-slave” architecture, meaning that there is one device in control of the others. The microcontroller typically acts as the master, but this is not always the case. Only the master can initiate communication on the bus, thus avoiding conflicts where more than one device tries to communicate on the bus at the same time.

I^2C transmissions from the master to a slave device contain the “address” of the slave so that the other devices on the I^2C bus do not participate when they are not being “addressed.”

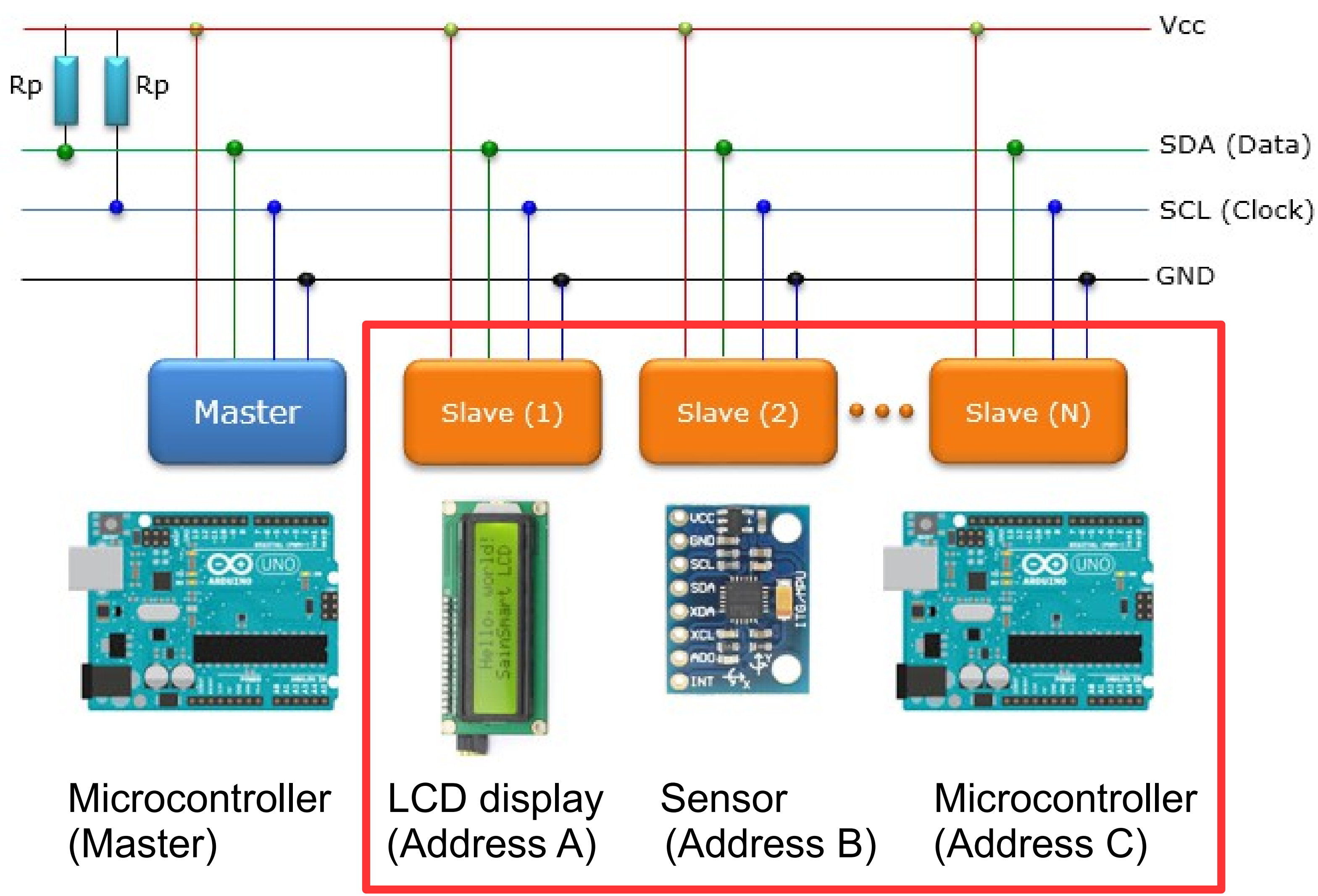
On the other hand, SPI uses a separate “slave select” or “chip select” wire to specify which slave is being addressed.

We will work with multiple slave devices on a single I^2C bus in the next lab.



Lab 5: Serial output (background)

Here is an example of a single I^2C bus (single set of wires) with one master (a microcontroller) and three slaves (an LCD display, some sort of sensor, and another microcontroller).

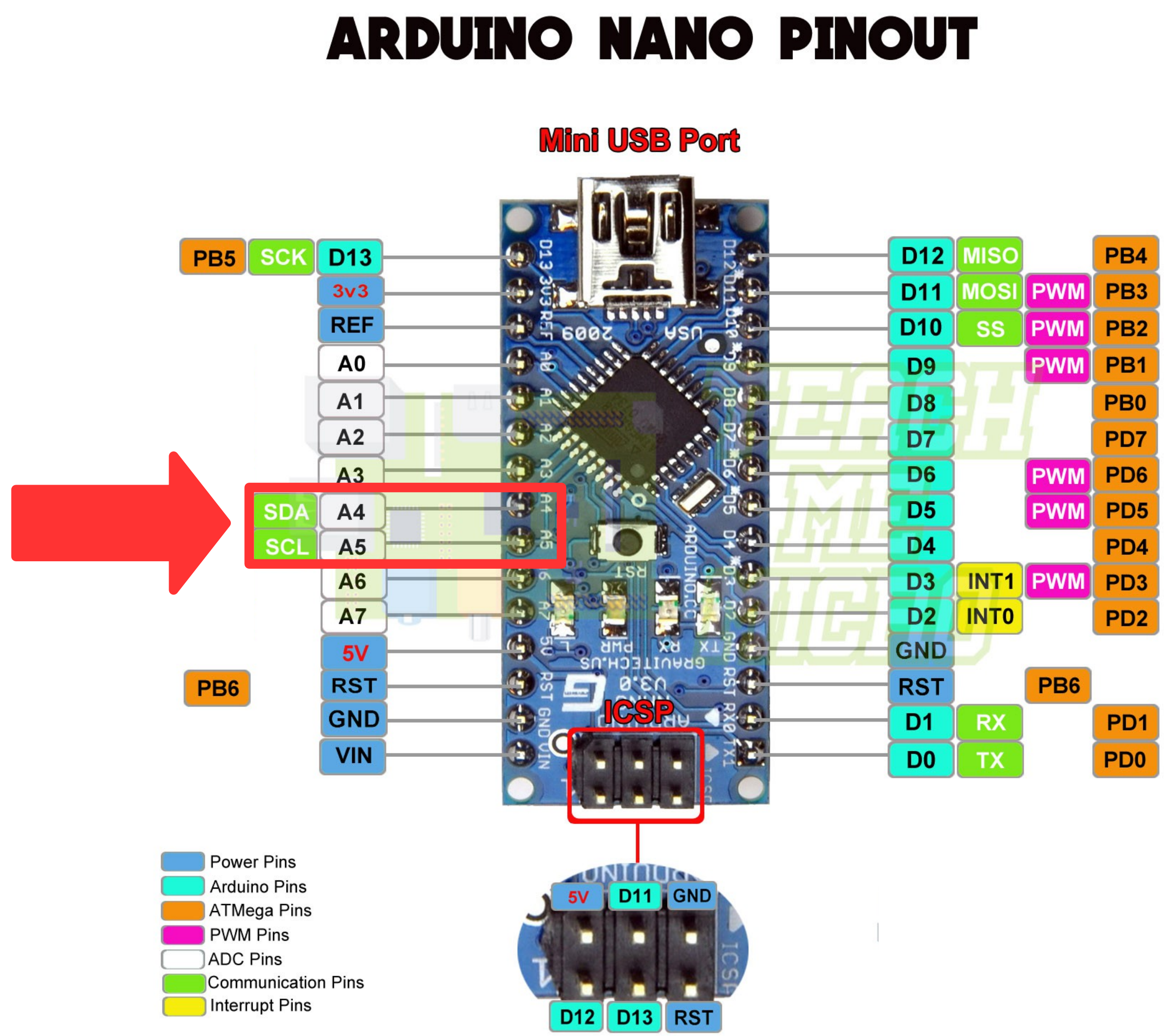


Lab 5: Serial output (background)

Recall that the Arduino Nano development board is based on the Microchip (Atmel) Atmega328P microcontroller. The Atmega328P has a built-in Two Wire Interface (TWI) peripheral for handling serial communications such as I^{2}C.

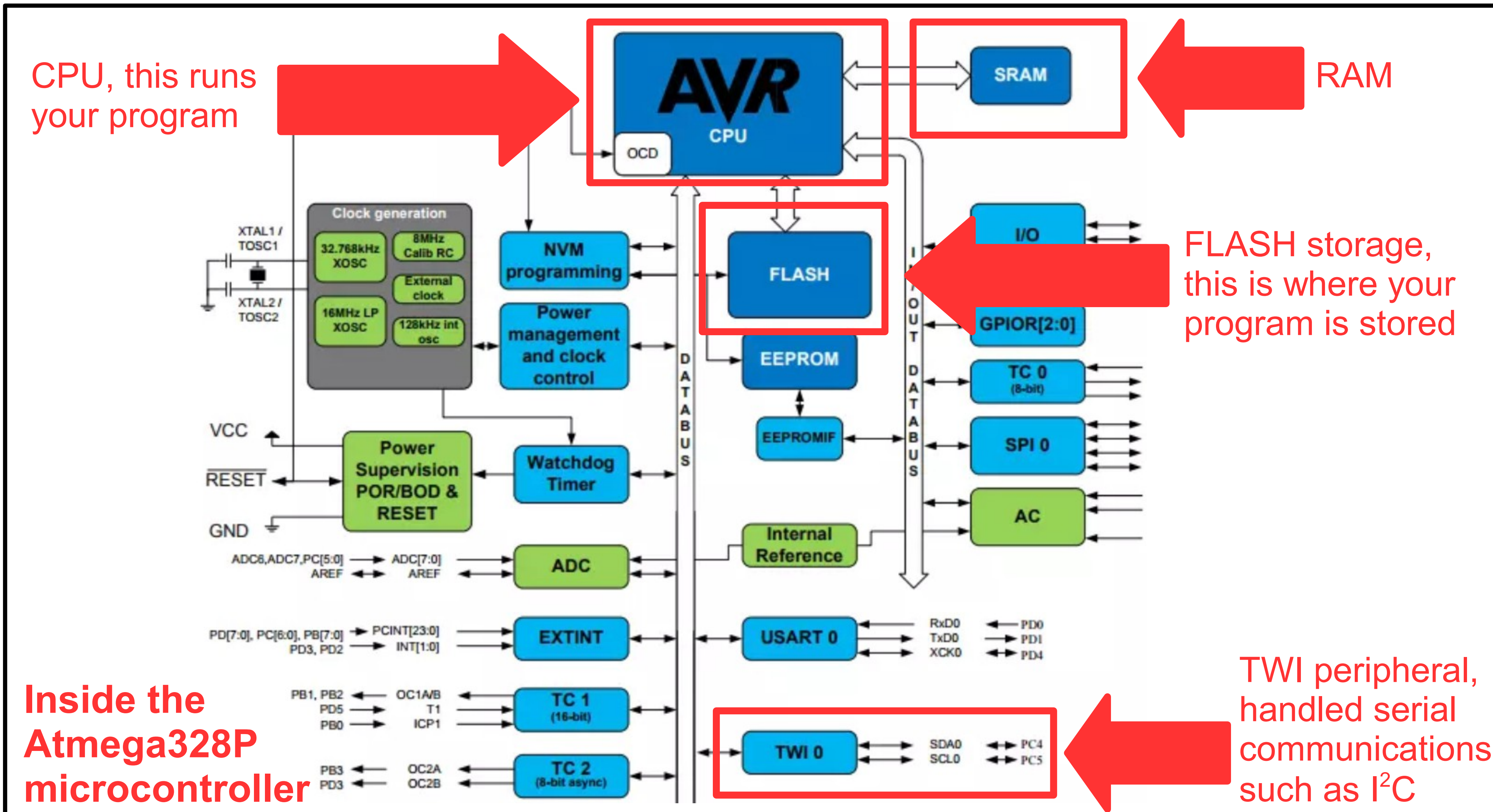
The pins marked "A4" and "A5" are directly connected to the SDA and SCL pins (respectively) of the Atmega328P's built-in TWI (I^{2}C) peripheral.

They are marked "A4" and "A5" ("A" as in "Analog" or "ADC") because these pins double as part of the Atmega328P's ADC, depending on what pin configuration the firmware sets.



Lab 5: Serial output (background)

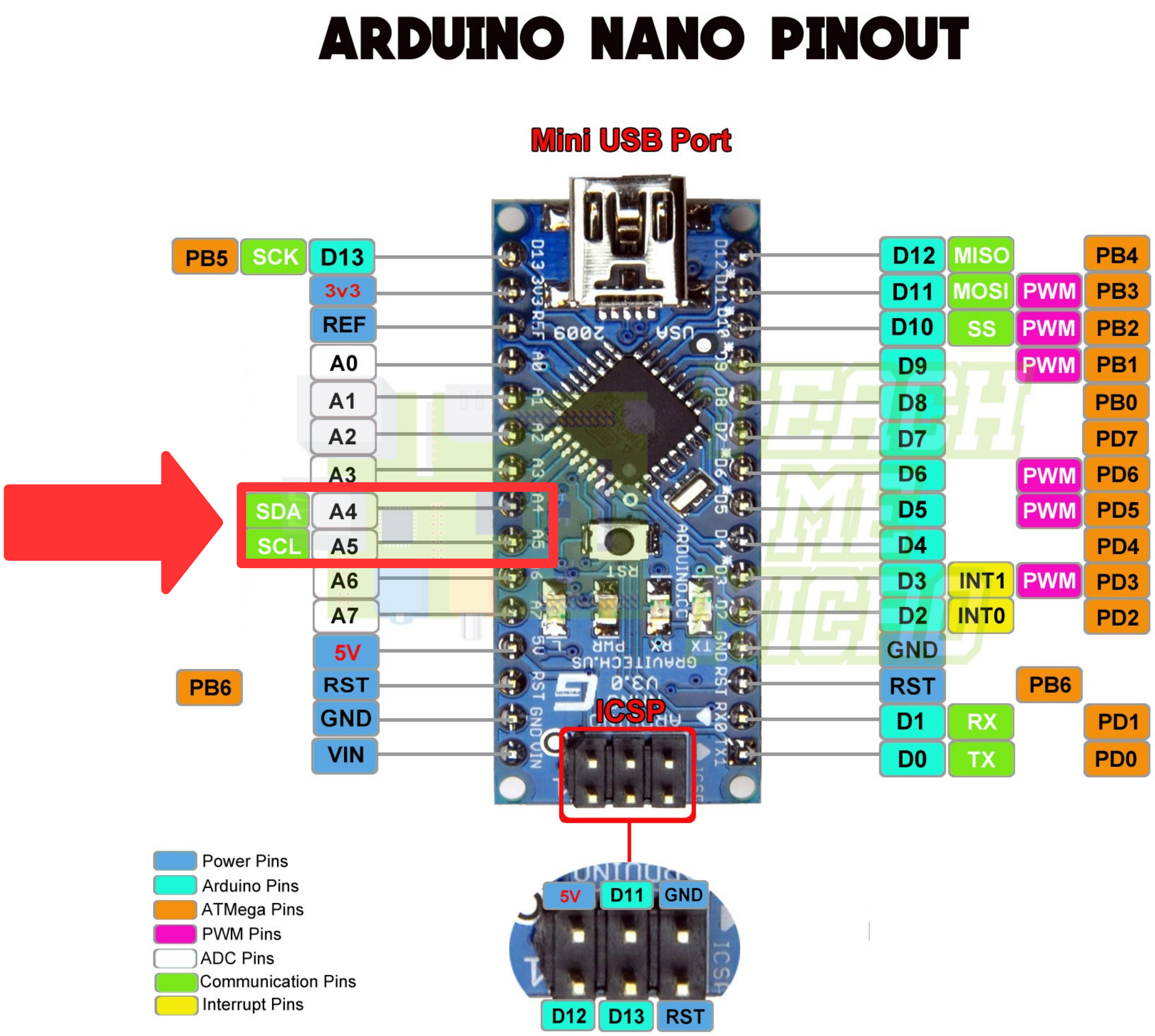
Recall that a microcontroller is more than just a microprocessor (CPU). It is a microprocessor and many other peripherals, RAM, FLASH storage, etc all on a single Integrated Circuit (IC) that together make up a fully-functional tiny computer.



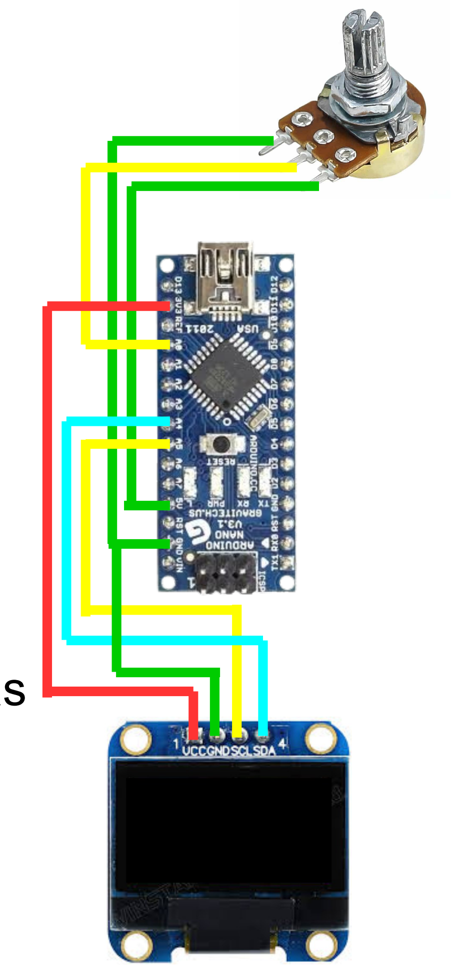
Lab 5: Serial output (background)

If the microcontroller did not have a built-in peripheral for handling serial communication then someone would have to write software that runs on the microcontroller that performs the precise timing of controlling various voltage levels in order to put the desired data on the serial bus or for making sense of the data coming in on the serial bus.

This practice is referred to as “bit banging” and can be used in place of using a dedicated hardware peripheral provided by the microcontroller.



Lab 5: Serial output



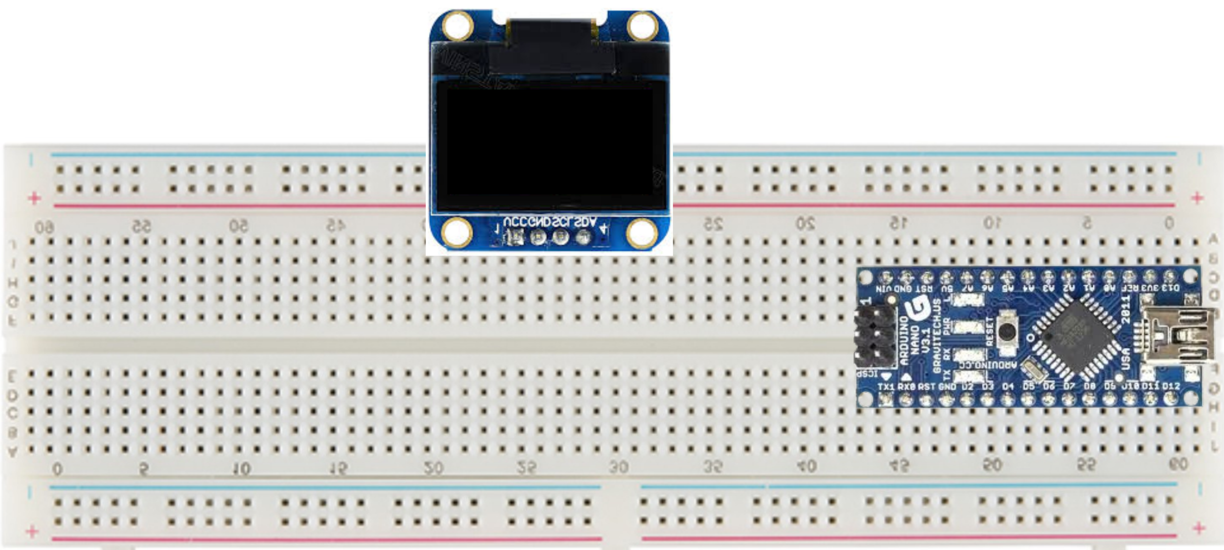
<u>Steps</u>

1. As a starting point, disconnect the USB cable from your development board and leave the components and connections from lab 4 intact.

2. Remove the LED, resistor, and associated wires from your breadboard.

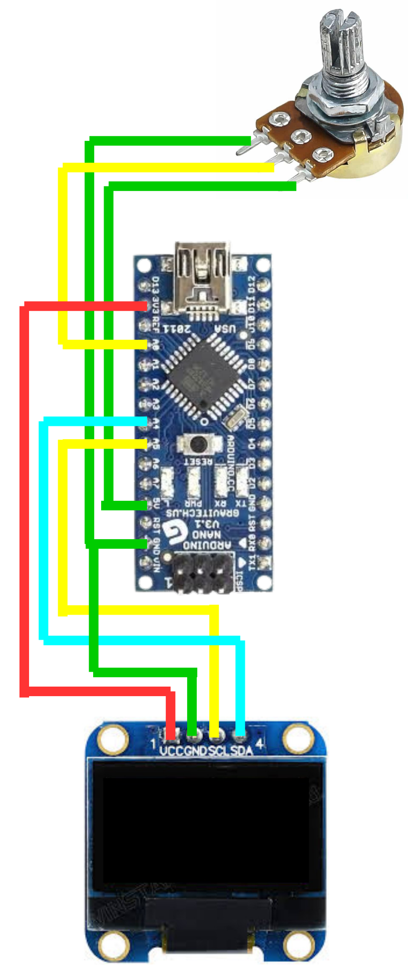
<u>NOTE: Leave the potentiometer and associated wires intact.</u>

3. Plug your OLED display into an outermost column of your breadboard as shown below:



(Next slide)

Lab 5: Serial output



Steps

4. Make the following connections between the OLED display and your development board:
- VCC to 3V3
- GND to GND
- SDA to A4
- SCL to A5

5. Copy and paste the code found at the following URL into the Arduino IDE's text editor window:
https://raw.githubusercontent.com/dandandrea/intro-microcontrollers-lab-005/refs/heads/main/intro-microcontrollers-lab-005.ino

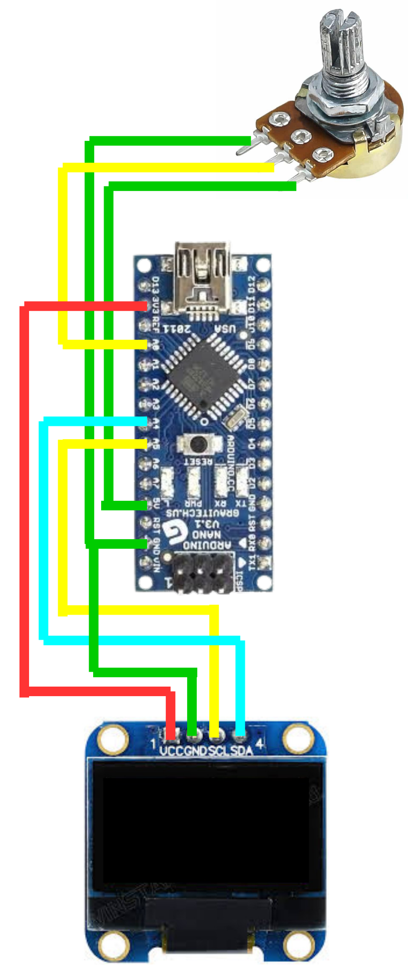
6. Save the "Arduino Sketch" (program) and name it intro-microcontrollers-lab-005
(File > Save)

8. Install Adafruit SSD1306 library
(Sketch > Include Library > Manage Libraries)

9. Connect your development board to your laptop via the included USB cable.

(Next slide)

Lab 5: Serial output



Steps

10. Program your development board with the program displayed in Arduino IDE.
(Sketch > Upload)

11. Observe that as you turn the knob to one direction, the number shown on the OLED display increases, and that as you turn the knob in the other direction, the number shown on the OLED display decreases.