EE 1301: Introduction to Computing Systems

IoT Laboratory #2

Getting Started with Sensors and Actuators

Created by: David Orser, Kia Bazargan, and John Sartori

Many thanks to the students, teaching assistants, and faculty who work to continually improve this document. Together, we make better labs!

Please send comments and suggestions to orser@umn.edu

Background

A smart device interacts with the world using Sensors and Actuators. A **sensor** is a component that detects changes in the environment. A thermometer, light detector, and soil humidity detector are all examples of sensors. An **actuator** is a device that can manipulate the world around your smart device. A valve, light, motor, or display are all actuators.

Purpose

In this lab, you will familiarize yourself with the inputs and outputs available on your Photon. Using these inputs/outputs, we will explore how your Photon can interact with the world around it. The idea is to give you an overview of what is possible and, in turn, stimulate ideas about what your project may contain.

Supplemental Resources

Device Description - Light Sensor (NOTE: Not yet updated for new LDR.)
Device Description - Temperature Sensor (TMP36)
Device Description - Individually Addressable LEDs (8mm WS2812B RGB LED)
Device Description - Simple Speaker (Piezo speaker)
Device Description - Servo Motor

Pre-Lab Requirements

Before coming to the lab, you should review a fair amount of reading material. Reading materials are provided in a “Quick Lesson” format -- stand-alone documents that cover a single topic. Please read through all the materials on the pre-lab checklist below.

Pre-Lab Checklist

- ❏ Complete Homework problem 3B (Random Walk)
- ❏ Read the Quick Lesson - Electrical Circuits
- ❏ Read the Quick Lesson - Getting to Know Your Pins: Power Supplies, Analog, and Digital Pins
- ❏ Read the SparkFun Breadboard Tutorial - https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard
- ❏ Read the second half of the SparkFun Tutorial - “How to read a schematic” http://learn.sparkfun.com/tutorials/how-to-read-a-schematic#name-designators-and-values

Reminder: If you use your breadboard space efficiently, you should never need to take apart your breadboard. You may have a temperature sensor attached to the A0 pin and the iLEDs on D4 simultaneously; the code you flash will determine what is run. There will not be a need to unplug devices and rewire them for each new lab.

Required Components

LDR (Light Sensor) 220 Ohm Resistor 4.7k Ohm Resistor 100k Ohm Resistor	3 Individually Addressable LEDs Push button switch Potentiometer

Lab Procedure

Exercise 1: First Sensor - Light Sensor

A light sensor can be handy for projects. For example, it can tell us when someone enters a lab (turns on the light), when the sun shines into an office, or when a cupboard or locker is opened.

The light sensor utilized in this lab is called a light-dependent resistor or LDR. An LDR conducts current depending on the amount of light that hits it. Since our Particle devices only measure voltage, we use a resistor to convert the current into a voltage we can read (V=I*R). The schematic in Figure 1 shows the sensing circuit. As light hits the photo-transistor (LDR), its resistance decreases, causing more current to flow through resistor R3, dropping the voltage on the sense pin A1 (more on this in EE2015!) Therefore, more light means a **lower** measured voltage.

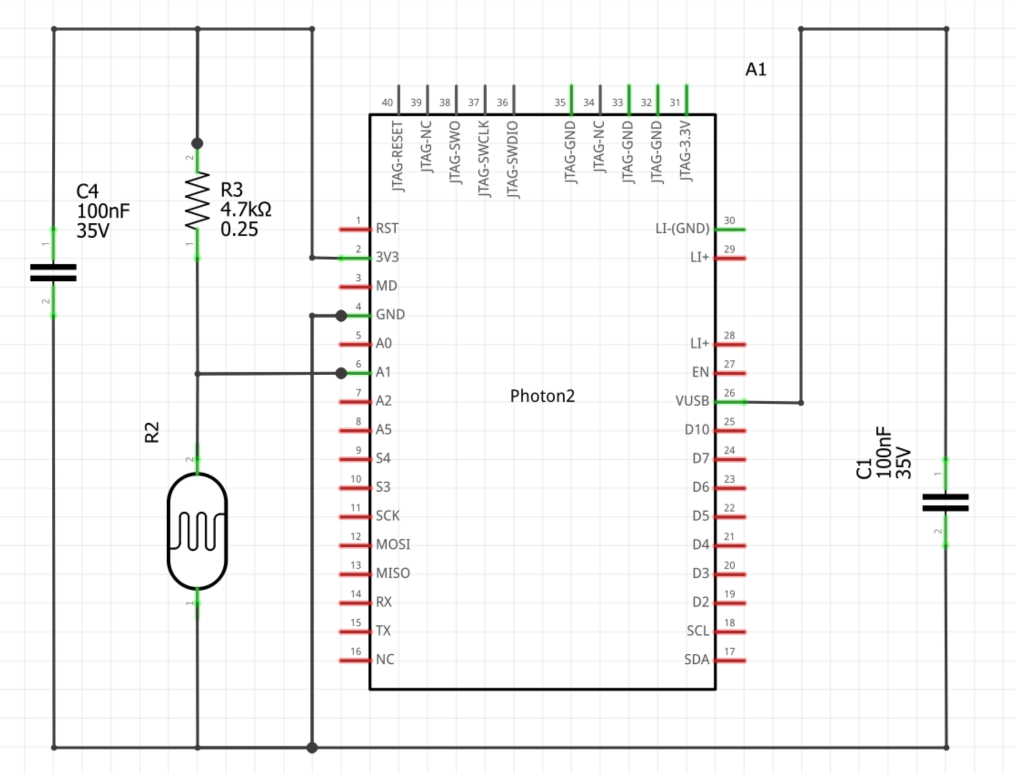


Figure 1: Schematic diagram of the test circuit for an LDR

Testing and retrieving data from a sensor is the first task in evaluating any sensor. In the example below, we will be using the serial port to evaluate our sensor's response and calibrate our final code (see " Quick Lesson - Power Supplies, Analog, and Digital Pins " if the term "serial port" seems unfamiliar).

NOTE: Many things can affect sensor readings: temperature, battery voltage, variations between manufactured parts, and noise. These can all affect the values read by your Photon. If you have issues later in the lab, it is a good idea to return to this step and verify that the sensor reports the values you expect.

1) Wire up your Particle and the LDR as shown below.

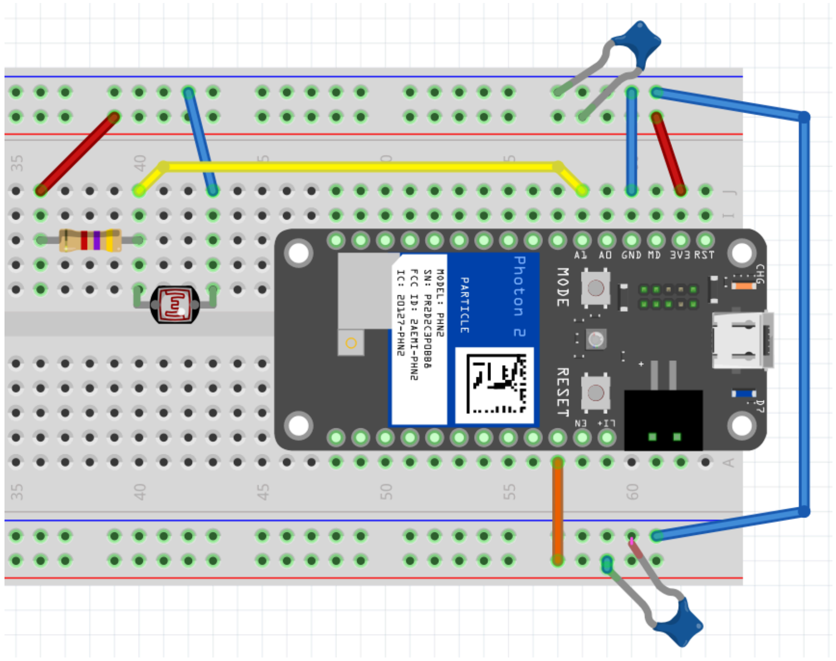


Figure 2: Breadboard Layout of the test circuit for an LDR

2) Connect a USB cable from your computer to your Particle device.
3) Open the Particle Workbench IDE (VS Code)
4) Create a new project

   REMEMBER: Always create a new project for each sub-section of a lab. (HINT: CTRL-SHIFT P, “Particle: Create New Project”

   REMEMBER: Set your DeviceOS and Select Photon2/P2 as your target. (HINT: CTRL-SHIFT-P, “Particle: Configure Project for Device”)

5) Before we declare any functions, we should declare a variable (type: int) to hold our measurement results. Do this in the global variable portion of your program.

```
int data0;
```

6) **In the setup()** function, setup your serial port by doing the following:

```
// Open the serial port for communication with the computer
Serial.begin(9600);
```

The line above may look a little cryptic. (Where did the function Serial.begin() come from?)

Two critical points:

- The Particle IDE is designed to be a *very* user-friendly programming environment and is designed to be used with only the Particle line of devices; as such, it can make many assumptions. First, the serial port is always available. Second, the library (Particle.h) is always loaded. As such, we do not need to declare “Serial” or specify its type. Just tell it when to initialize.
- If you want to know how a function or object (more on the difference later) operates, “Google it!!!” → “particle reference Serial”

7) The loop() function will contain the working payload of our program. First, we read from the Analog pin into a variable (say “data0”.)

```
// Read data from analog pins (returns a number from 0 to 4095)
data0 = analogRead(A1);
```

8) Next, we print this data in a readable format to the serial port.

```
// Print the data to the serial port
Serial.print("My Data is: ");
Serial.print(data0);
Serial.println(";");
```

Heartbeat LED

A heartbeat LED allows us to visually verify that our program successfully loaded and is running (or reloaded after a change). Adding three pieces of code to our App will allow us to implement a heartbeat LED easily.

9) Add a heartbeat LED

a.) In setup()

```
// Setup D7 pin to output a heartbeat
pinMode(D7, OUTPUT);
```

b.) At the beginning of loop()

```
// Heartbeat, show we're alive
digitalWrite(D7, HIGH);
delay(250);
```

c.) At the end of loop()

```
// Heartbeat, show we're alive
digitalWrite(D7, LOW);
delay(250);
```

NOTE: The heartbeat LED can be handy when your Particle Photon behaves oddly. Try changing the heartbeat rate of the LED significantly (i.e., change "`delay(250)`" to "`delay(1000)`") and re-flashing your Photon. You will immediately know if the code has been updated and is running.

10) Save (CTRL-s), Compile (CTRL-SHIFT-P → Particle: Compile application (local)) , and Flash (CTRL-SHIFT-P → Particle: Flash application (local)) your Code to your Particle Photon.

REMEMBER: Your Photon will do nothing until it connects to WiFi! Watch the RGB LED as it progresses through the modes. (WHITE → GREEN → Breathing CYAN). If your Photon doesn't breathe cyan, try to fix it yourself or talk to your instructor/TA for help.

11) When your Photon has reset and is "breathing cyan", check the serial monitor by doing the following:
    a.) Open a Particle CLI Terminal using CTRL + SHIFT + P "Particle: Launch CLI"
    b.) Type in the terminal:

```
particle serial monitor
```

NOTE: You should be in the Particle CLI rather than PowerShell, cmd, or bash.

NOTE: You can also use the "Launch CLI" button in the top right or via Particle Button.

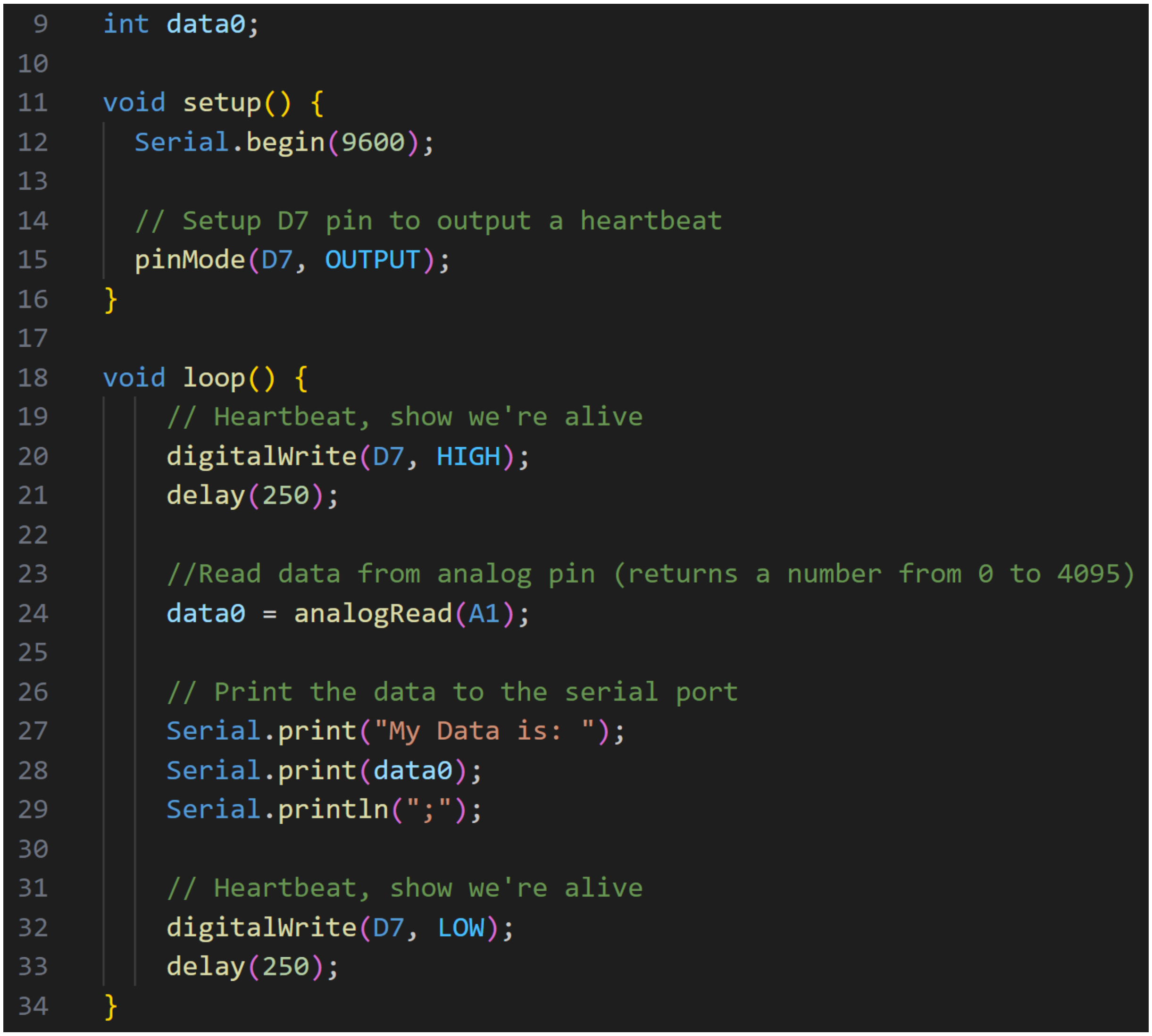
12) Take data with the sensor to prove that it is working.

For example, position the sensor so it can see the room lights then cover and uncover the sensor with your hand. Try using your phone flashlight to see even higher light levels.

Condition	ADC reading
Covered by your hand	
Room with bright fluorescent lights	
…	
Strong Phone Light	

NOTE: "ADC reading" stands for Analog to Digital Converter reading. It is a 12-bit digital representation of the voltage on the pin A1 described the the equation:

Voltage = ADC_reading / 4095.0 * 3.3



```
int data0;

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

  // Setup D7 pin to output a heartbeat
  pinMode(D7, OUTPUT);
}

void loop() {
    // Heartbeat, show we're alive
    digitalWrite(D7, HIGH);
    delay(250);

    //Read data from analog pin (returns a number from 0 to 4095)
    data0 = analogRead(A1);

    // Print the data to the serial port
    Serial.print("My Data is: ");
    Serial.print(data0);
    Serial.println(";");

    // Heartbeat, show we're alive
    digitalWrite(D7, LOW);
    delay(250);
}
```

Figure: One possible version of AnalogRead2Serial.ino

Exercise 2: Temperature Sensor

This time, on your own, build a serial evaluation setup for your TMP36 temperature sensor using the schematic and example code offered in Device Description - Temperature Sensor

WARNING: Unlike CLI Labs you can't have multiple *.cpp files with multiple loop() and setup() functions in the same directory. You must create a new project for each IoT exercise! Feel free to cut-n-paste the file content from your previous exercise into the new exercise.

Take some data from the sensor to prove it is working (for example, air temp and temp after being pinched for 20 seconds).

Condition	ADC reading	Temp(C)	Temp(F)
Room Temperature			
Finger Temp			
…<other>...			

WARNING: Be careful not to touch the leads on the bottom of the package, as your body resistance might alter your measurements.

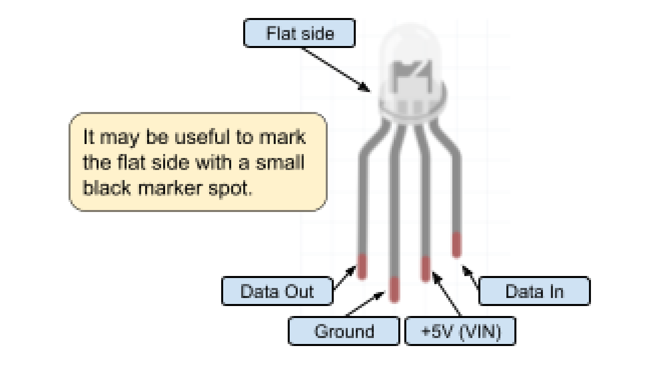
Actuators - Display Elements

So far you have worked with LEDs to display a single color. If you want to see what old-school LEDs are capable of, check out the YouTube of CSE's Winter Light Show. Imagine what *you* could do with a lot of these!

This section will display information on external display elements connected to the Photon. The advent of very cheap integrated circuits and new packaging technologies has allowed the embedding of encoders and decoders directly into display elements. No longer do you need 8+ wires to communicate with a display element. It is common today to use a variation of the serial port (2, 3, or 4 wires) utilized above to transmit configuration information (often dozens/hundreds of variables) to external devices.

Individually Addressable LEDs

We will now look at our first complex output device (sometimes called an actuator). iLEDs stand for individually addressable LEDs. (Adafruit's "NeoPixel" is the brand name for some of the earliest popular iLEDs.) Every iLED can be individually set to a different color when wired up in a chain. Each LED requires a shared power supply and ground pin. Additionally, each LED has a Data_In and Data_Out pin. Notice the "Flat side" marked on the diagram below. You can find this by examining the head of the bulb (or feeling with your fingers.)


Flat side
It may be useful to mark the flat side with a small black marker spot.
Data Out
Data In
Ground
+5V (VIN)

iLED Pins

When iLEDs are connected in series, the first iLED in the string grabs the first 24-bits (first color setting) and then passes the rest of the data to the next iLED down the chain, which grabs the next 24-bits (second color setting), passing the remaining data, etc.

In this context, 24-bits means 24 ones and/or zeros. You do not need to worry about the details of how 24 bits are represented, as there are predefined library functions that handle those details. However, you may ask, why do we have 24 bits to represent the color? Well, each LED internally has three small LEDs: Red, Green, and Blue. Each of these mini-LEDs can show its color with intensity from 0 (off) to 255 (the brightest)[1]. So, if we set the Red value to 128 and the

[1] Remember that 8 binary digits (also called bits) are needed to represent numbers between 0 and 255.

Green and Blue values to zero, we get a half-brightness red color. We get the brightest purple if we set Red=255, Green=0, and Blue=255.

NOTE: These devices need a +5V power supply (on the Photon, this pin is labeled "VUSB"). For more information, please read Quick Lesson - Power Supplies, Analog, and Digital Pins

WARNING: Plugging these LEDs in backward (even for a second) will destroy them!!! Find the flat side before beginning wiring!

iLEDs NOTE: Manufacturers sometimes ship slightly different types of iLEDs with slightly different signaling standards."The lab document was written assuming WS2812B iLEDs. There is currently no way to determine if your iLED is a WS2811, WS2812, or WS2812B simply by looking at it. If you see strange behavior, check your wiring first. Then try to adjust the order in which color arguments are entered. If those don't work, consult your TA.

Exercise 3A: Build the iLED Circuit

It's now time to start wiring up your iLED.

13) Connect three iLEDs to your Photon as shown below.

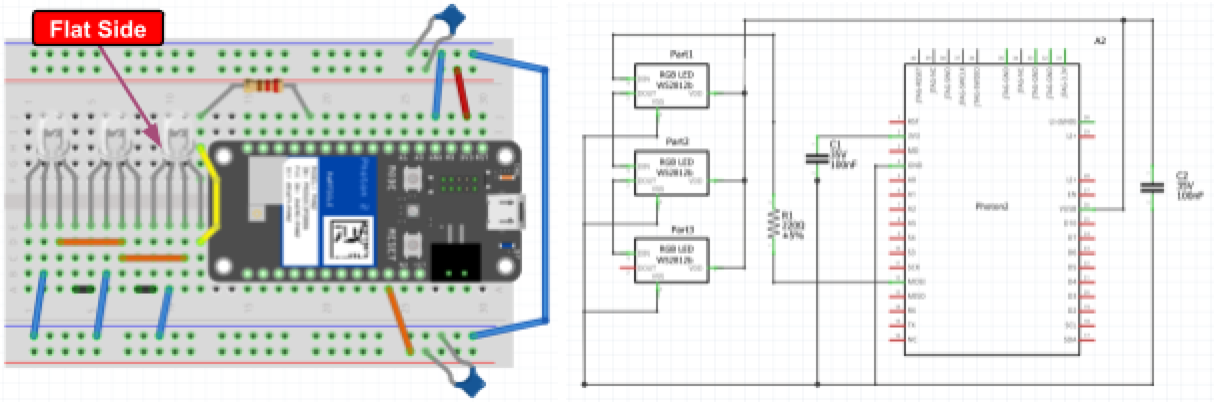


Figure: iLED Wiring Diagram

NOTE: We must use either the SPI or SPI1 peripheral to control the NeoPixels. The SPI "MOSI" pin has many names in the documentation including → `S0`, `MO`, and `MOSI`. See library update..

NOTE: Those looking carefully might notice that the power has not been connected to the iLEDs. This is intentionally done to protect the LEDs while we write the code. The power connection will be carefully added in the final steps.

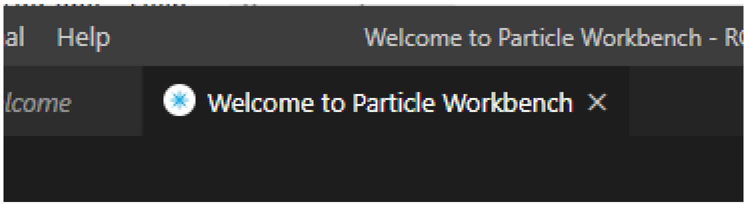
NOTE: The blue ceramic capacitors on the top and bottom power rails keep the voltage supplies stable under varying current demands, similar to how water towers keep the water pressure constant under varying water demands. They are "good practice" but not required.

Libraries

Libraries are collections of predefined variables and functions that are commonly utilized. They are usually maintained by companies or interested individuals. Libraries make complicated tasks easier. In the Particle development environment, they are handled in a special way.

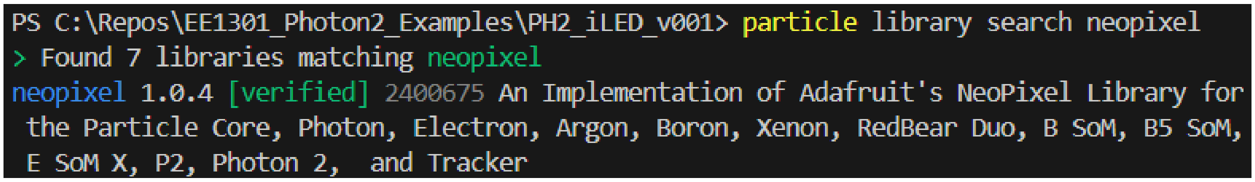
Exercise 3B: Writing Code for the iLED

14) Go to VS Code and create a new project
    a.) Click on the “Welcome to the Particle Workbench” tab


al Help
Welcome to Particle Workbench - R(
lcome
Welcome to Particle Workbench ×

    b.) Click on “Create a Project” under the “Getting Started” section.
    c.) Choose the parent folder and the name of the project.
15) Bring up the command pallet (⇧⌃P (Windows, Linux) | ⇧⌘P (macOS))
16) Type “libraries” and choose “Particle: Find Libraries”
17) When asked what the library name is, type “neopixel”.

NOTE: We knew the library's search term because we looked at the manufacturer's website and did a search for “adafruit neopixel photon” when we bought our iLEDs!

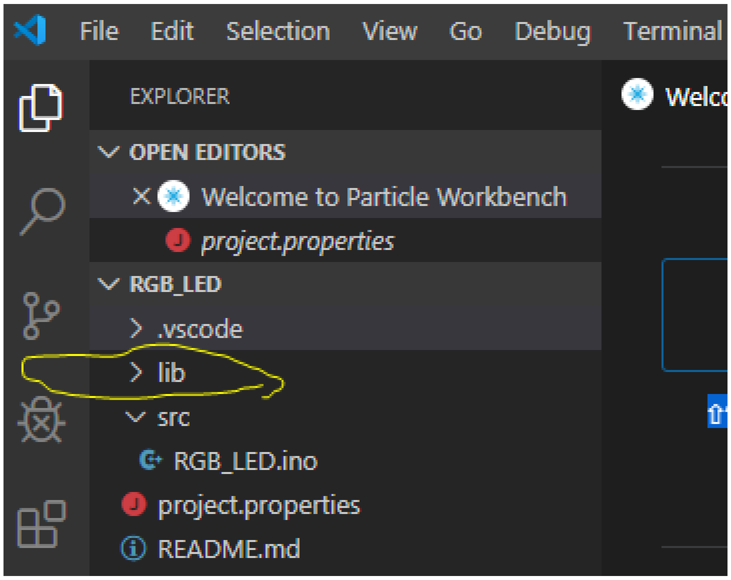


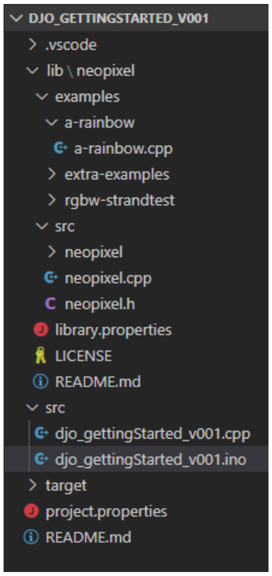
```
PS C:\Repos\EE1301_Photon2_Examples\PH2_iLED_v001> particle library search neopixel
> Found 7 libraries matching neopixel
neopixel 1.0.4 [verified] 2400675 An Implementation of Adafruit's NeoPixel Library for
 the Particle Core, Photon, Electron, Argon, Boron, Xenon, RedBear Duo, B SoM, B5 SoM,
 E SoM X, P2, Photon 2,  and Tracker
```

In the list of found libraries, the first item is called “neopixel,” and the description shows that it is **[verified]** and compatible with Photon.

18) Bring up the command pallet again, and this time use “Particle: Install Library”.
19) Enter “neopixel” without the quotation marks as the library name.

20) Notice that a new item called “lib” was added to your project:





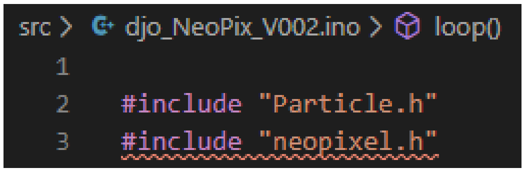
21) Click the arrow > next to lib, you will notice subdirectories for “examples” and “src”. The src directory contains the *source code* for controlling the “neopixel” color LED.

22) Look through the *examples* directories and look at a couple of the *.cpp files. You may see the following lines. These lines are necessary to integrate the neopixel library with your code.

```
#include "Particle.h"
#include "neopixel.h"
```

23) Add these lines to the top of **your** source file (the one created when you created your new project, something like: “djo_TestNeoPixel_v001.ino”).

24) If you see a red underline beneath “neopixel.h” close and re-open VS Code.



```

#include "Particle.h"
#include "neopixel.h"
```

Object Models

Going into detail on object-oriented programming is beyond the scope of this document (but will be covered in detail later in EE1301.) It is sufficient to understand that a class is another data type in C++ programming (just like an int, a bool, etc., but a bit more complex because it holds both values AND actions/functions to be taken on those values). The internal actions/functions of an object are sometimes called methods. An object is a single *instance* of a class.[2]

We will need some basic information to create and set up our NeoPixel object:

- The number of pixels (3) in the chain
- The pin number (SPI) to which the string of pixels is attached

[2] Generally speaking you can have many objects of one class in a program. For example, if you had multiple iLED strips you needed to control, each with a different pin, number of LEDs, color pattern, etc.

- The type of controller chip (WS2812) (see https://www.adafruit.com/products/1734)

Exercise 3C: Creating a NeoPixel Object and Using it

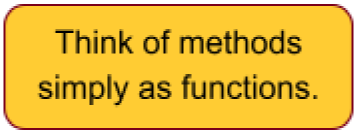
You can then create a new NeoPixel object similar to how we declare a new integer, except we use a function to fill the object with its initialization data. The ints/defines below are used for better readability when we declare the new NeoPixel object below. Note that we will give you code pieces in the next few sections. You have to put these together for the code to work.

25) Insert the following code above setup(), in the Global declarations section.

```
// These lines of code should appear AFTER the #include statements, and before
// the setup() function.
// IMPORTANT: Set pixel COUNT, PIN and TYPE
int PIXEL_COUNT = 3;
#define PIXEL_PIN SPI // Only use SPI or SPI1 on Photon 2 (SPI is MO or S0 pin; SPI1 is D2)
                      // NOTE: On the Photon 2, this must be a compiler constant!
int PIXEL_TYPE = WS2812;

Adafruit_NeoPixel strip = Adafruit_NeoPixel(PIXEL_COUNT, PIXEL_PIN, PIXEL_TYPE);
```

This creates “strip”, an object based on the “Adafruit_NeoPixel” class, which is defined in the “NeoPixel” library (Confused yet? Hold onto your shorts...)


Think of methods simply as functions.

As you saw with Serial, in C++ (and many other programming languages), we access the methods of an object with the syntax “object.method()”. To initialize the strip, we call the method “**strip.begin()**”. We only need to do this once, so the best place is in our Photon code's setup() function.

26) Insert the following code in setup()

```
void setup() {
    ...
    strip.begin();
    ...
}
```

Finally, we have declared and initialized our object; we’re ready to use it to do something useful! We will use three methods from the Adafruit_NeoPixel class of objects. The table below defines these methods.

Method (Function) Definition	Description

void = strip.setPixelColor(<uint16>,<uint32>)[3]
Example:
strip.setPixelColor(PixelID, myColor)
	Store the Color for the n-th PixelID (zero-indexed) in memory. PixelID refers to the n-th physical LED in the chain of serially connected LEDs. The LEDs do not light up yet (see the last row).

<uint32_t> = strip.Color(<uint8>,<uint8>,<uint8>)
Examples:
myColor2 = strip.Color(255,0,0)
myColor = strip.Color(Red,Green,Blue) // RGB Order
// ** Or ***
myColor = strip.Color(Green,Red,Blue) // GRB Order
	Returns an encoded color representing the RGB values that neoPixel can use. In the second example, Red, Green, and Blue are variables specifying an intensity between 0 and 255.
	NOTE: Some neopixels specify the color values in the G,R,B order instead of R,G,B order

void = strip.show()
Example:
strip.show()
	Sends the colors stored in memory to the physical LED strip in single burst transmission. This statement causes the lights to change color!

We will define some colors and store the desired data in the strip object. Once everything is set, we'll call the show() method to dump the data over the digital data link to the string of pixels. If you forget to use the show() method, the iLEDs will NOT change colors.

27) Insert the following code in loop()

Loop() Example Code

```
void loop() {
    /* NOTE: Two versions of the neopixel exist.  RGB versions and GRB versions.
       The only difference is the order the colors are specified.  Can you figure
       out which type you have?  */
    //Setup some colors, RGB version
    int PixelColorCyan = strip.Color(   0, 100, 100);
    int PixelColorRed  = strip.Color(  80,   0,   0);
    int PixelColorGold = strip.Color(  60,  50,   5);
    //Setup some colors, GRB version
    /*
    int PixelColorCyan = strip.Color( 100,   0, 100);
    int PixelColorRed  = strip.Color(   0,  80,   0);
    int PixelColorGold = strip.Color(  50,  60,   5);
    */

    //Set first pixel to cyan
    strip.setPixelColor(0, PixelColorCyan);
    //set second pixel to red
    strip.setPixelColor(1, PixelColorRed);
```

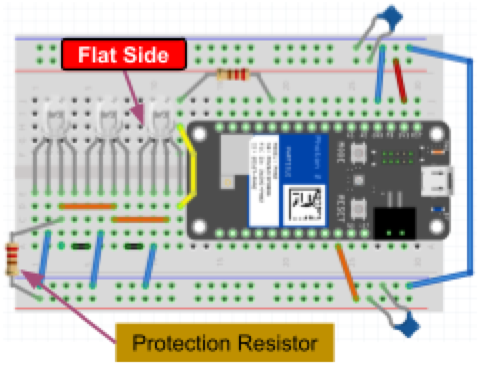
[3] You can actually see the definition of this method in the file "neopixel.c" on line 699. Opening the library definitions is a useful place to look if you need to figure out how an undocumented library function works.

```
    //set third pixel to Gopher Gold!
    strip.setPixelColor(2, PixelColorGold);
    strip.show();
    delay(1000);  //wait 1sec

    //flip the red and gold
    strip.setPixelColor(0, PixelColorCyan);
    strip.setPixelColor(1, PixelColorGold);
    strip.setPixelColor(2, PixelColorRed);
    strip.show();
    delay(1000);  //wait 1sec
}
```

28) Verify and flash your code to your Photon.
29) Add a 220 Ohm “Protection Resistor” to your iLEDs, as shown below:

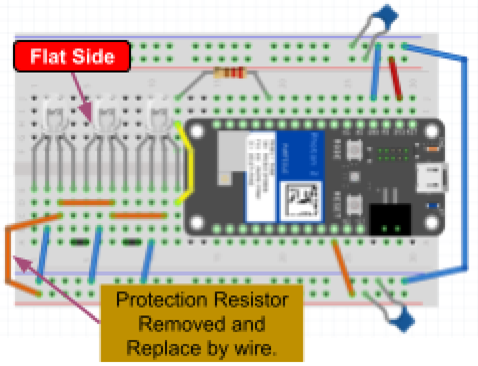
Figure: iLED Wiring Diagram (with Protection Resistor)



30) Check that the LEDs function as intended.

31) Finally, when your circuit generates colors, replace the “protection resistor” with a wire to fix any flickering issues, as shown below.

Figure: iLED Wiring Diagram (no Protection Resistor)



32) Show your iLEDs to your TA and make any modifications to the colors they request.

HINT: You should be able to change one of the LEDs to green. This will require understanding *color order* (hints in blue above.)

IMPORTANT: NeoPixels hold their color settings until the next call to the show() method. There is no need to continuously update them.

Exercise 4

Based on the circuit you built in Exercise 3, finish either Ex4A or Ex4B below.

Exercise 4A: Changing Brightness in Steps of 50

Now, change the program so that the first LED shows red, the second green, and the third blue. We want all three to start with a light intensity of 0. After a second, all three change intensity to 50, then 100, 150, 200, 250, and back to 0. Repeating forever.

(VERY STRONG HINT: Do not use the pixel.setBrightness () function. It is not intended for this purpose and will likely break and/or confuse you later.)

REMEMBER: Create a new project for each IoT exercise! When you do so, reselect “DeviceOS@5.8.0” and “Photon 2 / P2”, plus you’ll need to reinstall the “neopixel” library (and potentially re-add the line *#include “neopixel.h”*)

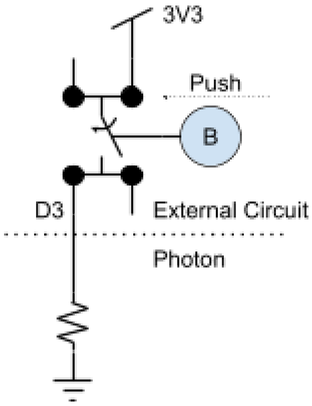
Exercise 4B: Flickering Candle Exercise

Copy your HW3 function “`int randWalk(int oldValue, int updateSize);`“ to VS Code. Use it to set the brightness of your LED(s). Try making all three flicker with the same brightness. Play with the updateSize to create different effects. Try making all three flicker independently (they will each have a separate “oldValue” and call randWalk() separately.)

Exercise 5: Sensors - Human Input Devices

Accepting input from a human being is a valuable feature for microcontrollers. While this can appear to be a simple task, several pitfalls exist. We will start by setting up a single push button and potentiometer (knob) as input devices.

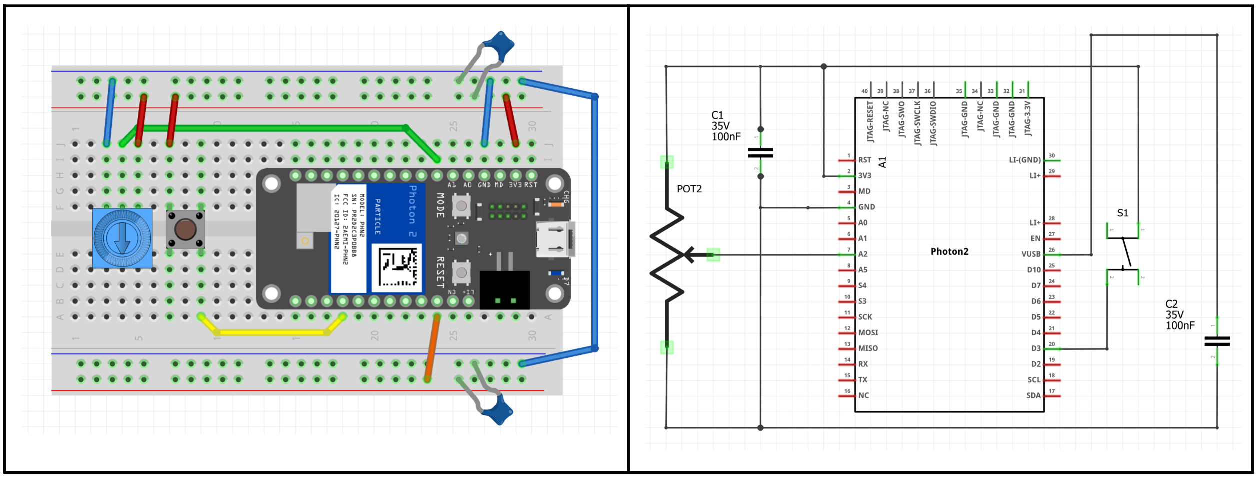
The **push button** is a normally open momentary switch; this means that if we tie one terminal of the push button to 3V3 and the other terminal to an input of the Photon set to “INPUT_PULLDOWN” it results in the circuit on the right. When the button is “not pushed”, it results in the Photon’s internal “pulldown” resistor pulling the input to ground, 0.0V, or a “low” state. Think of this as a default state. The input gets connected to 3V3 with a low resistance when the button is



pushed. The resistors have a tug of war, and the lower resistance path wins, pulling the input to 3.3V or a “high” state.

The **potentiometer** (“pot” for short) is a three-terminal device. It is implemented physically as a long resistor with a contact on each end (terminals 1 and 3) and a “swiper” that can contact anywhere in between (terminal 2). When the two fixed terminals are connected to the power rails (GND and 3V3), the “swiper” terminal will output a voltage between 0.0V and 3.3V, depending on the position of the swiper. This can easily be wired into an analog input terminal that samples the voltage value.

33) Wire up a push button and a potentiometer as described above and shown in the following circuit:



We now have two inputs. Note that the pushbutton provides a digital input (D3) and the potentiometer provides an analog input (A2). We want to sample the states of those inputs and then output them to the serial port for debugging purposes.

The following code shows one way to do this:

```
int ButtonPIN = D3;
int PotPIN = A2;

int PotOut = 0;
bool ButtonOut = FALSE;
int ButtonCount = 0;

void setup() {
    pinMode(ButtonPIN, INPUT_PULLDOWN);
    pinMode(PotPIN, INPUT);
    Serial.begin(9600);
}
```

```
void loop() {
    ButtonOut = digitalRead(ButtonPIN);
    PotOut = analogRead(PotPIN);

    if(ButtonOut == HIGH) {
        ButtonCount = ButtonCount + 1;

        Serial.print("Button Count = ");
        Serial.print(ButtonCount);
        Serial.print(" , Level = ");
        Serial.println(PotOut);
    }
}
```

You have now created a bunch of Photon programs (create new project, add lines of code to the globals, setup(), and loop().) Please continue to take these steps even if you are not prompted.

34) Write a program using the button/potentiometer code example above. Compile, Flash it.
35) Connect to the Photon with your terminal program (HINT: “particle serial monitor”).
36) Press the button a couple of times, turn the potentiometer, and press the button again. You should notice a couple of things immediately. Continue reading below.

Events vs. State

You may have noticed that pressing the button once may result in reporting more than one button count. In fact, the number of button counts depends on the speed of your microcontroller, how quick your fingers are, and the complexity of the code being run (not good things!).

In our case, we are interested in the event “Button is pushed,” not the current state of the button “Down.” Inherently the event “Button is pushed” requires knowledge of two pieces of information -- the previous state of the button “Up” and the current state of the button “Down.” We can build code to capture and build on these pieces of information. For example:

```
    ButtonNow = digitalRead(ButtonPIN);

    if(ButtonNow == HIGH && ButtonLast == LOW) {

        //Do our work here;

        ButtonLast = HIGH;
    } else if (ButtonNow == LOW) {
        ButtonLast = LOW;
    }
```

37) Finally, to complete Exercise 5, modify your code to only send information to the serial port once per button press. Show your working project to your TA.

Debouncing

Often, the human interface is made more complex because the physical buttons actually “bounce” several times before settling on a steady output value (see oscilloscope output on the right, which shows the voltage sampled from a button when pressed.) This can result in detecting multiple button presses for a single press or an artificial button press when the button is released. Usually, this bouncing lasts less than 2ms (but can sometimes last 10s of milliseconds).

The maximum execution rate of the loop function for a Photon device is 1 ms. This can mask bouncing effects very nicely. Unfortunately, this can sometimes result in a false sense of security. You can ignore debouncing as long as your App doesn’t:

- Use a switch other than the PCB mount momentary switch supplied by the ECE Depot
- Operate on the “release” of the PCB mount switch rather than the “press”
- Sample the same input multiple times in a single loop() function
- Operate your Photon in SYSTEM_MODE(MANUAL)

Exercise 6: Micro Project

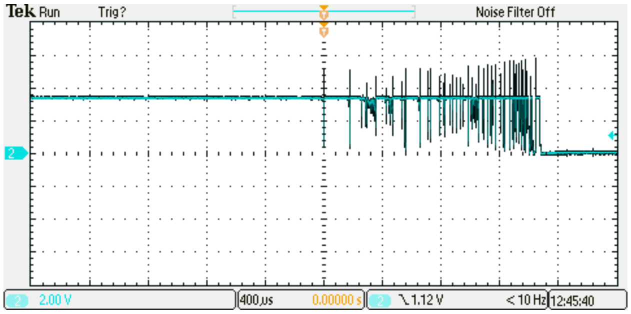
Read through the device descriptions for the Speaker and the Servo Motor. Using what you’ve learned in this lab, write an app that senses something (a button, pot, temp, light, etc.) and somehow responds (speaker, servo, led, etc.). Use any of the actuators in this lab or devices you have figured out independently. A couple of examples of potential micro-projects are:

- Automated Light: Light level rises → Turn off an LED lamp
- Automated Fan: Temperature rises → Turn on a fan
- Music Box: Press a button → play a song or multi-tone siren

While the choice of a specific micro project you do for this section of the lab is up to you, a couple of concepts are useful for embedded systems design. Now is potentially a good time to read Quick Lesson - Programming Constructs. Take a look and see if it is relevant to you.

Lab Report

You are required to submit a written report on your Micro Project. This is only Exercise 6 and is the open-ended part of this lab. Your report should contain your well-commented code and a brief (no more than half a page) description of your project.



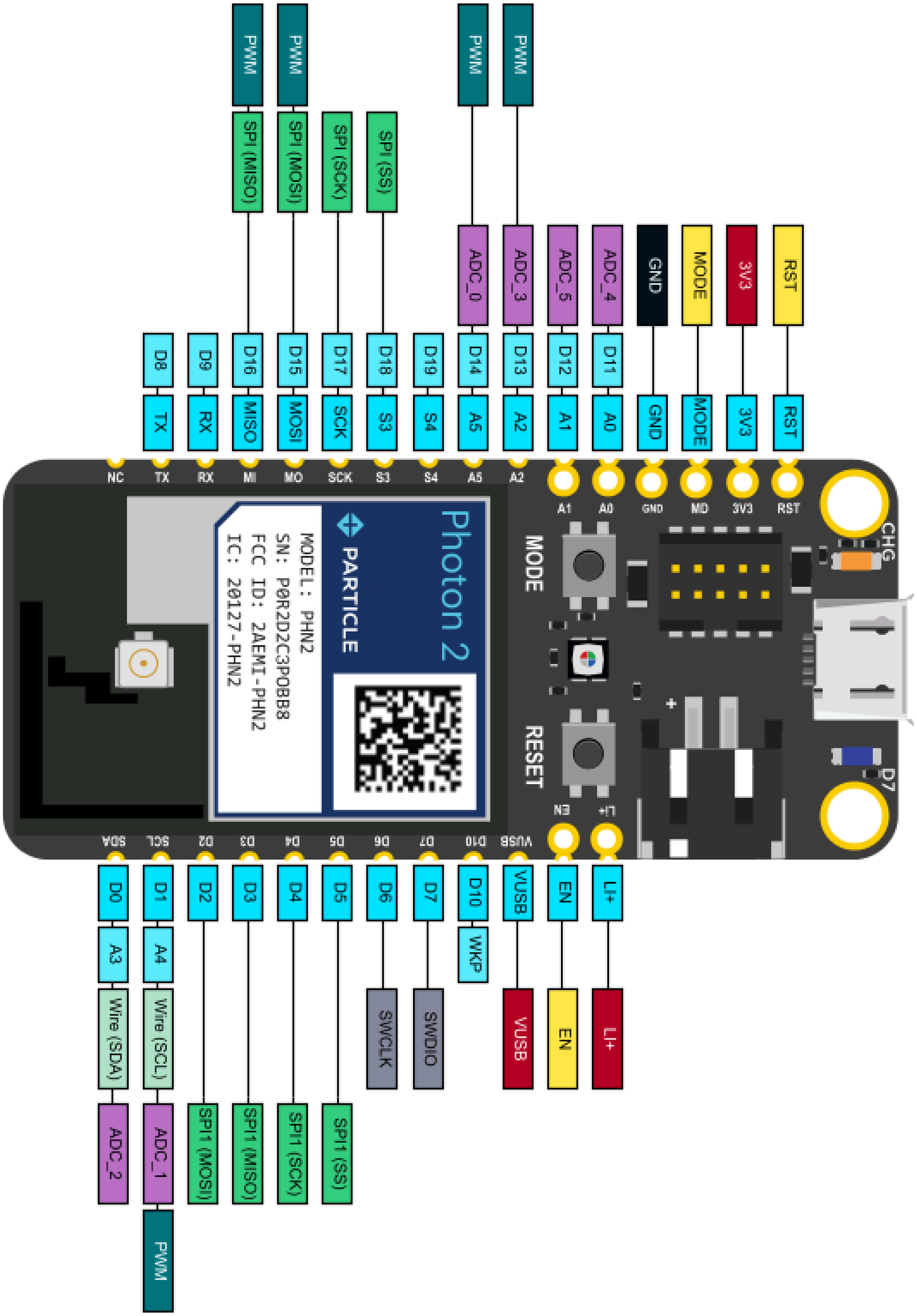
Be prepared to discuss the following:

- Your experience with each device (sensors, actuators) that you connected.
    - Did it work the first time you connected and programmed it? What mistakes did you make? How did you resolve any issues? If you didn't have any issues, just say so.
    - Include the relevant code you used to get it to work. When describing code, it is essential to break it into sections and explain how and why you chose the particular implementation you used.
    - For the sensors, include the table that shows the ADC (analog read) values under different conditions.

No submission is necessary for the other exercises, but the Lab TA must see a demo of your circuits. Make sure s/he checks off your work before you leave the lab (or set up a time outside the lab to demo if you don't have time to finish).

Appendix - Photon 2 Pinout Diagram

Many pins in the Photon can be utilized in various ways. The following pin diagram may be useful when deciding which outputs to use on a project.



Derivative work, based on: https://docs.particle.io/reference/datasheets/wi-fi/photon-2-datasheet/