Introduction

This projects demonstrates how to use a $10 Si-PIN photodiode, a low voltage op amp and a microcontoller running CircuitPython as an alpha particle detector. This project is inspired by the BFYIII paper *Alpha Particle Spectroscopy and Energy Loss*, by Rob Knobel, Bei Cai, Sean Musclow, and Will Thompson. I have made no attempt to make a signal suitable for energy spectroscopy. There are actually many papers published on using PIN photodiodes as radiation detectors for both energy spectroscopy and counting.

Knobel et al. found that Hamamatsu S1223-01 PIN photodiodes work well to detect alpha particles. They have an active area of 10 mm^2, and an unbiased capacitance of 20 pF. Alpha particles cannot penetrate the glass window of the photodiode, so the window of the diode has to be removed. When alpha particles strike the active area, they are absorbed and almost all of their energy goes into creating electron-hole pairs.

A convenient source for alpha particles is a ^{241}Am source from a smoke detector. The ^{241}Am emits alpha particles with an energy of 5.486 MeV 85% of the time. You have to check with your institution's occupational safety department about the handling of these source even though they are technically exempt radiation sources. I bought them from eBay.

A rough estimate of the signal can be made: About 100 % of the alpha energy is deposited in the photodiode creating electron-hole pairs of energy about 3.5 eV. That is about 1.6 x 10^6 pairs or about 2.5 x 10^{-13} C. The 20 pF photodiode capacitance discharged through a 10 MΩ resistor give a time constant of 4 x 10^{-4} s and a voltage of V_α = 13 mV. This is a reasonable signal to work with.

The challenge for this project is to design a low voltage circuit powered by the 3.3 V of the microcontoller that will convert the current pulse of the alpha particles to a 3.3 V digital pulse the microcontoller can count. I have found that 60 Hz noise is a large noise source for this project.

I made a 3D printed case for the experiment that includes Lego pieces for changing the distance from the source to the detector.

The Microcontroller

I chose to use a Lolin S2 Mini microcontroller, based on the ESP32-S2 chip, for this project. Follow the steps in the *Setting Up CircuitPython* to load CircuitPython onto the Lolin S2Mini microcontroller.

One shortcoming of CircuitPython (CPy) over MicroPython is that CPy does not allow you to write interrupt driven functions. However, version 7 does include a **`countio`** library to read and count digital edge transitions on pins. This is all you need to make a radiation counter.

Once you can count the pulses you can count for 1 second intervals, for example, and get a Poisson distribution for the counts per second. Another interesting radiation measuring technique is to time the *interval* between counts which has an exponential distribution. I did program that on a microcontroller using MicroPython instead of CircuitPython.

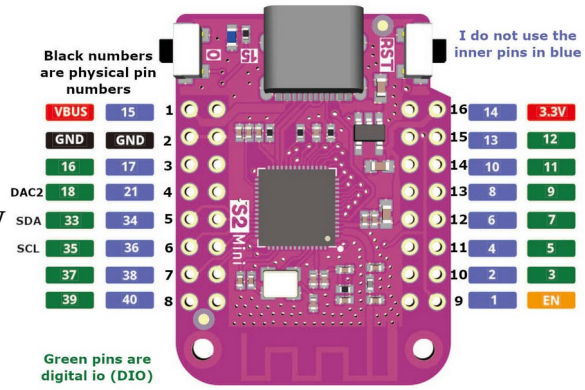


Figure 1: The pin designation for the S2Mini microcontroller.The purple and green pin numbers are accessed as ***`board.io#`*** *in CPy code.*

The Circuit

I wanted a device that was only powered over a USB cable. There is 5 V available from the microcontoller, but I wanted to work with the 3.3 V level the microcontoller runs on to avoid any possible damage to the microcontoller. I posted on the now extinct ALPhA listserv and got a recommendation of using the MCP602X family of op amps. They can run on 3.3 V and have rail-to-rail input and output.

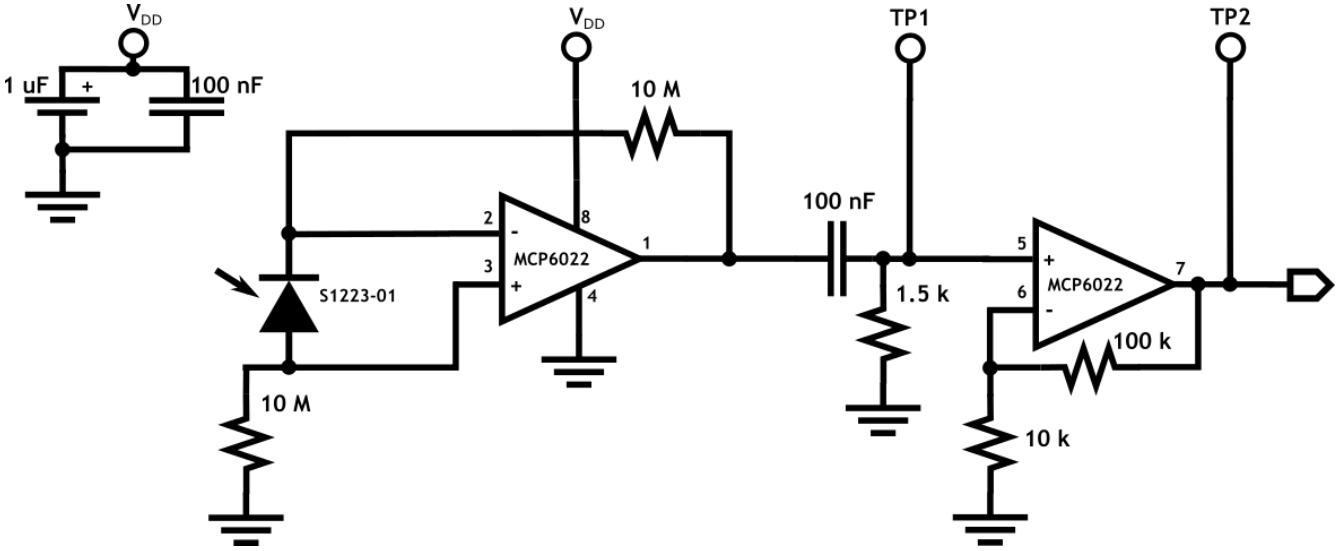
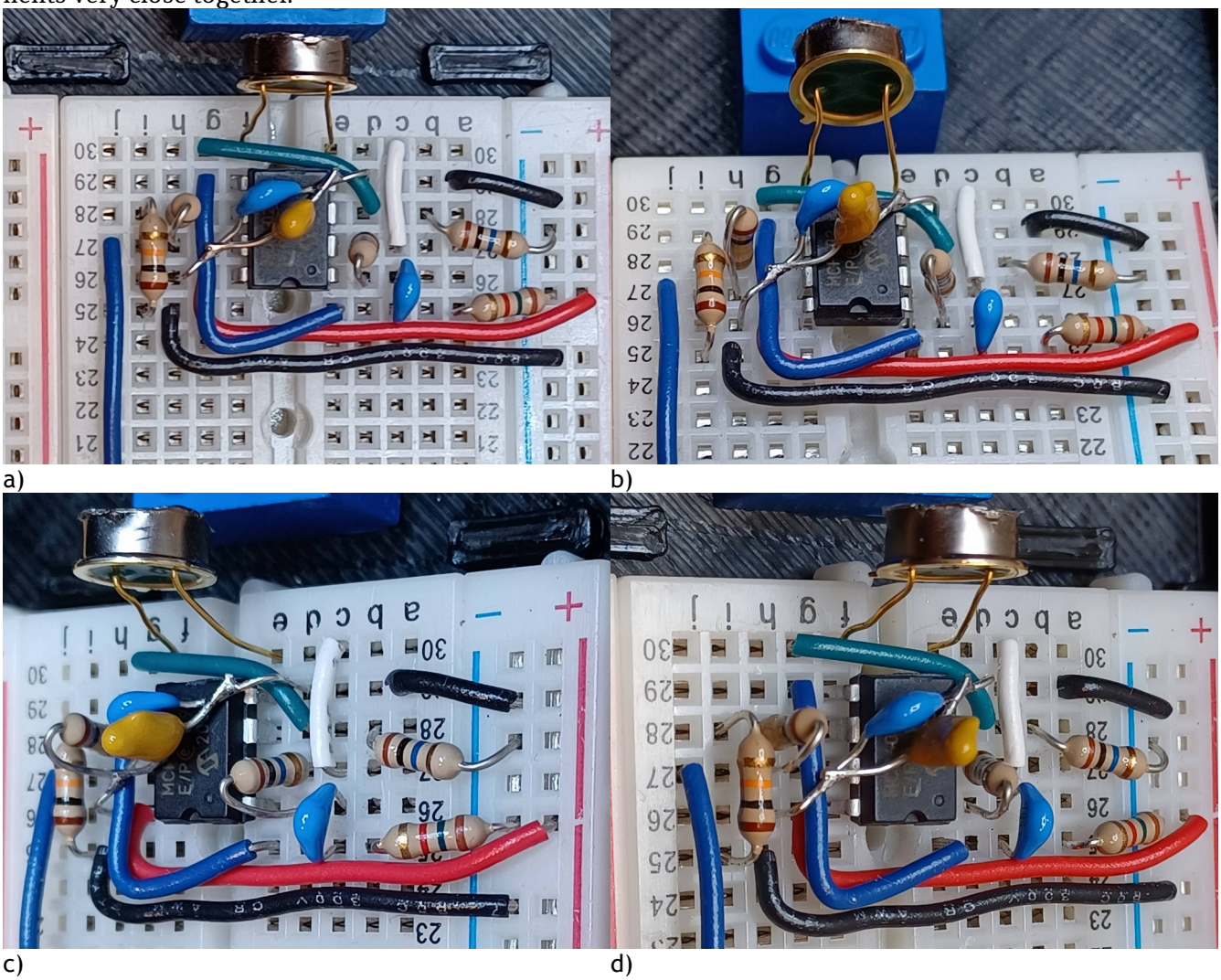


Figure 2: Schematic for analog detection of alpha particles. The bypass capacitors and photodiode are mounted near the op amp. The photodiode has the glass window removed.

The first stage of the circuit is a transconductance amplifier as shown in *Figure 2*. The supply voltage, V_{DD}, is the 3.3V regulated output from the microcontoller. The bypass capacitors are a 1 µF tantalum and a 100 nF ceramic capacitor. I actually span these over the top of the op amp. The first op amp is a transconductance amplifier. An example of the signal at Test Point 1 (TP1) is show in *Figure 4a*. The signal at TP2 is the input to the microcontoller counter.

Wiring the Circuit

The figures in the table below are close-ups of the wiring. The photodiode is mounted on the last row of a breadboard and the op amp is mounted next to it. It is important to keep the wires short and the components very close together.



a) b)

c) d)

Table 1 – Pictures of the wiring. a) Top view. The bypass capacitors are wired together and span the op amp. b) Front view. The power and ground are provided by the rails on the right. c) Left view. These components are for op amp 1. d) Right view. Op amp 2 buffers the passive high-pass filter and provides gain so the pulses are digital for the microcontoller to count.

Circuit Signals

Figure 3 is an example of the output. The signal after high-pass filter actually goes negative. The signal into the micro-contoller is saturated at 3.3 V. The width of the pulse is about 10 µs wide. A second pulse is also captured in the figure.

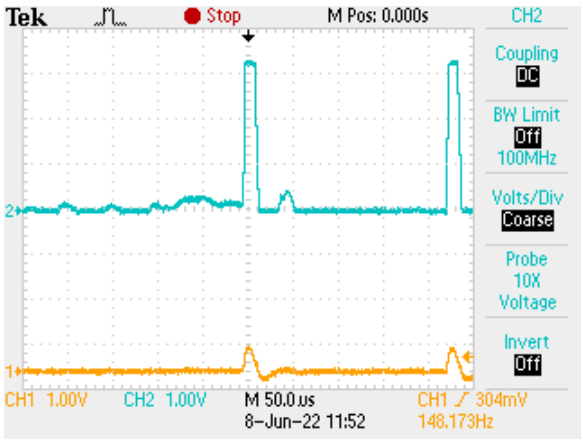


Figure 3: Output at TP1 (orange) and TP2 (cyan).

CircuitPython Code

The code is not very complicated. Here is my comments on the code **`Count_Alpha.py`**.

The first part of the code imports libraries needed. Note the **`countio`** import which does the counting.

```
import time
import board
import countio
from digitalio import DigitalInOut, Direction
```

Next I print the name of the code and set important variables. The **`counterPin`** variable sets which pin the digital data is coming in on. One important note is that pin **`board.D5`** is **not** the same pin as **`board.IO5`**! I wasted a day trying to find this bug. The pin numbers in the S2 Mini figure are **`IO#`** pins in CircuitPython. It will print results once per second and every 100 seconds, print a comment line labeling the data. I use nanoseconds later, so the variable **`reportNanos`** is the time in ns.

```
print("# Count_Alpha")
reportTime = int(1) # in seconds
counterPin = board.IO5 # Any IO pin will work
nPrint = 100 # how often to print label
reportNanos = 1_000_000_000 * reportTime
```

Make sure the on board LED is off.

```
pLED = DigitalInOut(board.LED)
pLED.direction = Direction.OUTPUT
pLED.value = 0
```

Set variables up for looping.

```
# Set stop time for loop
stopNanos  = time.monotonic_ns() + reportNanos
# Set variables used in the loop
print(f"# Time (s), Counts, Avg Counts in {reportTime} seconds")
iPrint = 0
Total = 0
nLoops = 0
```

Make a **countio** instance and use **reset()** to set it to 0.

```
# Set up the counter
pin_counter = countio.Counter(counterPin)
# Reset counter and go
pin_counter.reset()
```

Finally the code that takes the data. The entire loop is in a **try** – **except** structure. When **<Ctrl>+C** is pressed the code stops and an exception is called. The exception is caught with the **except** statement and the counter is released with the **deinit()** function. This is important because if the microcontoller is restarted, the code can fail because the counting pin is already busy with the old counter.
In the **while True** loop, first the code waits for right time, then grabs the current counts in the variable **counts**. Next it resets the counter to 0. The loop updates the total counts, number of loops and the print variable. If it is time to print it does. The next time to print is set in the variable **stopNanos**. Finally, the data is printed.

```
try:
    while True:
        # Wait until time is up
        while time.monotonic_ns() < stopNanos:
            time.sleep(0.001)
        # Calculate Counts per Minute
        counts = pin_counter.count
        # Reset counter
        pin_counter.reset()

        Total += counts
        nLoops += 1
        iPrint += 1
        if iPrint >= nPrint:
            iPrint = 0
            print(f"# Time (s), Counts, Avg Counts in {reportTime} seconds")
        # Set next report time
        stopNanos += reportNanos
        print(f"{time.monotonic():10.0f}, {counts:6d}, {Total/nLoops:.3f}")
except:
    pin_counter.deinit()
```

LEGO SPACERS

Building the Blocks

In Lego terminology a *plate* is a piece 1/3rd the height of a *block*. Legos can be sanded to get a smooth surface. The dimensions are shown in *Figure 4*.

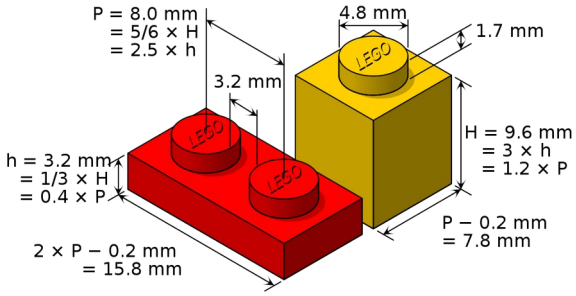


Figure 4: Official dimensions of Lego pieces.

- Use genuine Lego pieces. The dimensions are very tightly controlled.
- I recommend using a gel superglue to glue Lego to the case and to each other. The gel superglue gives you time to get the pieces in place and it stays in place when gluing pieces vertically.
- *Figure 5*. Glue a 2x8 Lego block to the case bottom. It should be centered with one end just touching the inside wall of the case.

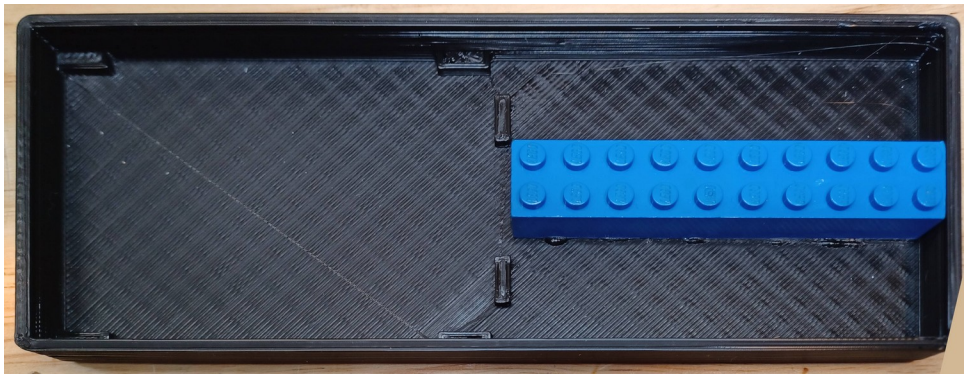


Figure 5: Glue the 2x8 black to the bottom of the case. The breadboard will be mounted on the left.

- *Figure 6*. Snap the 2x4 top block (red in the figure above) to two 2x4 blocks as a jig and stand it on end. Glue a 1x2 plate to the side of the block with a gap of about 1 mm.
- *Figure 7*. Glue an alpha source to a 1x2 *cap*. It is a 1x2 plate without pegs on the top. Caps are hard to find; you can cut the pegs off of a 1x2 plate instead and sand it to make it flat.
- In addition to these pieces, you need two 1x2 plates and one 1x2 block.

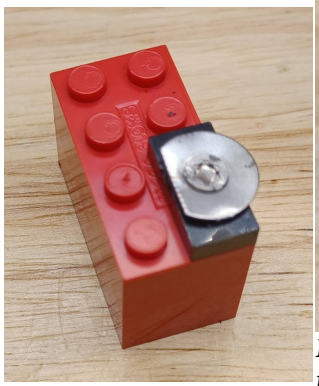


Figure 7: Glue the alpha source on a 1x2 cap. Here the cap is snapped to a block to hold it while gluing.

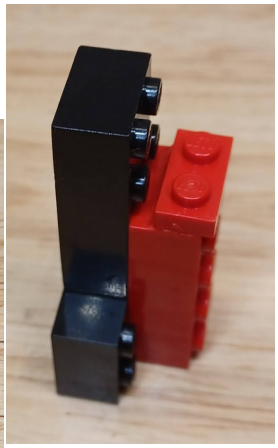
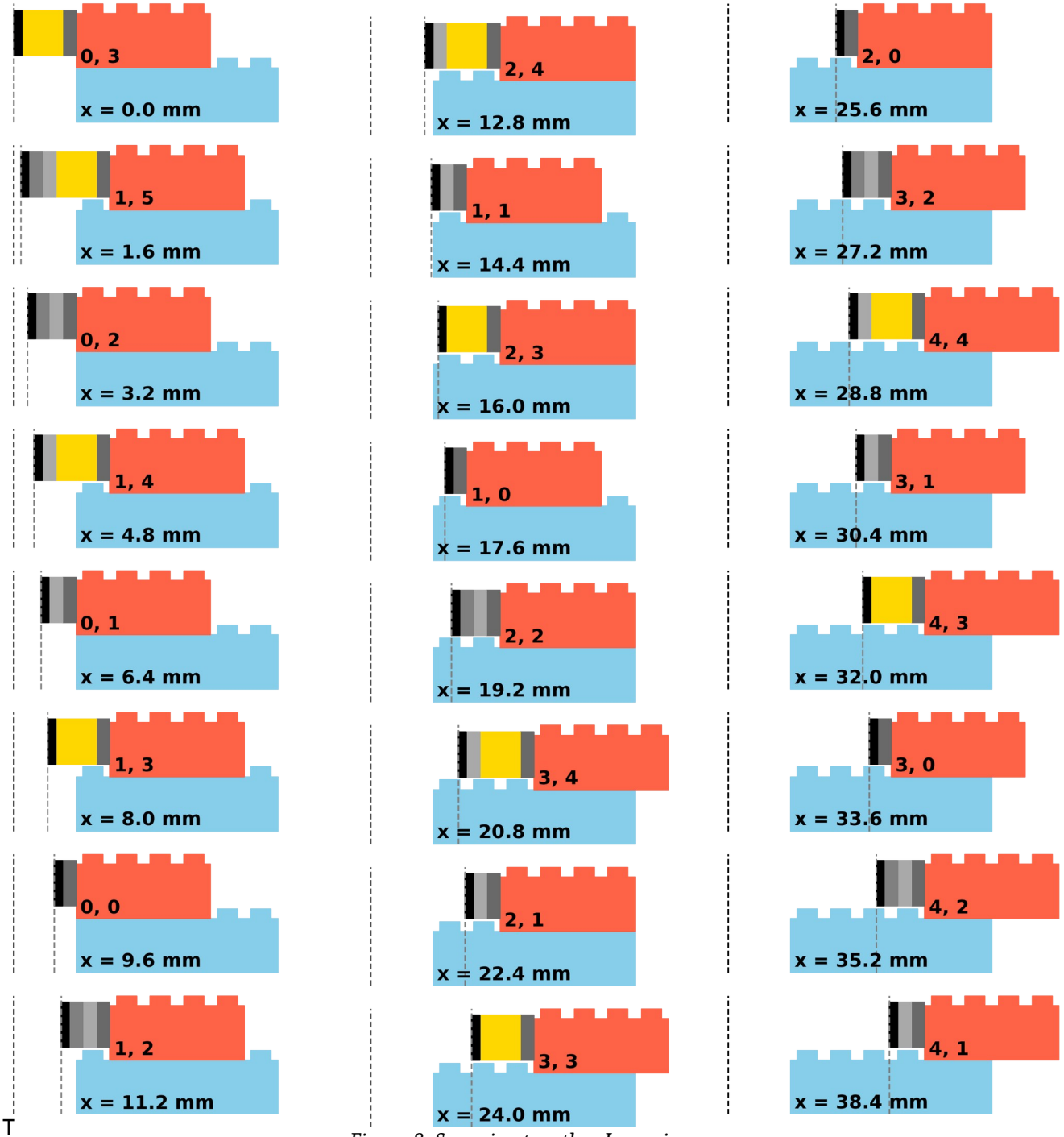


Figure 6: Glue a 1x2 plate to the side of a 2x4 block. Here it is standing on end to help gluing.

Using the Lego

The horizontal Lego base has a pitch of 8 mm between pegs. By gluing a Lego plate at right angles to the base, there is an additional pitch of 3.2 mm. By combining these two you can move the alpha source in increments of 1.6 mm. *Figure 8* below shows one way of arranging the Lego to get increments of 1.6 mm out to 35.2 mm plus one additional increment of 3.2 mm. The red, yellow and blue pieces are blocks; the rest are plates.

The terminology **`1,2`**, for example, means the 2x4 block is 1 peg to the right, and 2 plates are snapped on horizontally. 3 plates equal 1 block, so to make a 4 use one plate and one block.



T

Figure 8: Snapping together Lego pieces to take alpha counts versus distance.

Data Capture

I use *putty* on Windows and the terminal app *tio* on a Mac or Linux system to capture the data. With *putty* you can set up logging to a file with the name of the date and time. I start a separate *putty* session for each data set.

Here is a typical session with a Mac:

- Start the *Terminal* app.
- Change directories to the data folder.
  - **`cd ~/Dropbox/Projects/2022-mBFY-Alpha\ Detector/CircuitPython/Data/`**
- Start *tio* with logging to a data file of the format **`YYYY-mm-dd-HHMM.csv`**.
  - **`tio -l`date '+%Y-%m-%d-%H%M.csv'` /dev/tty.usbmodem487F304FDE251`**
- Output sample

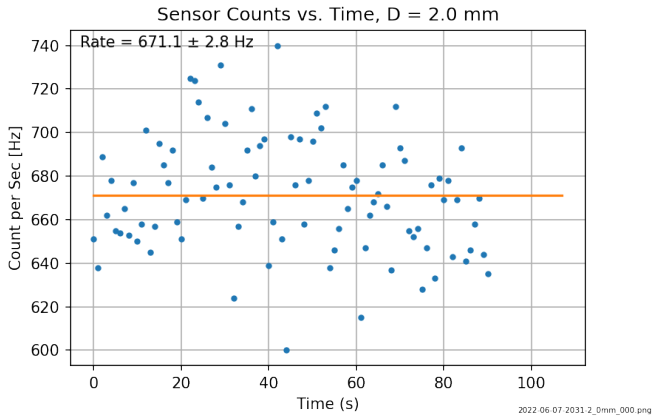
```
      199,    494, 491.990
      200,    486, 491.959
      201,    504, 492.020
      202,    487, 491.995
# Time (s), Counts, Avg Counts in 1 seconds
      203,    522, 492.145
      204,    467, 492.020
```

- The output is in CSV format. The fields are the time after booting the microcontoller, counts in the last second, and the running average counts per second. Comment lines start with a **`#`**.
- To stop the session and the data set, type **`<Ctrl>T`** then **`q`**.
- Rename the file **`2022-06-08-1212-5_2mm.csv`** because the source is about 5.2 mm from the detector.
- Edit the file to remove all lines not either data or comment.

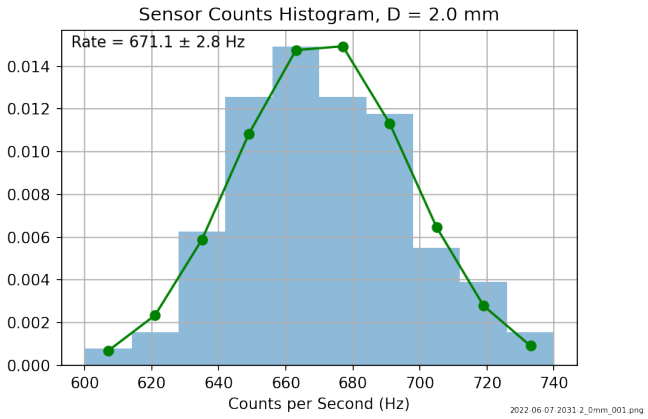
The data file can be analyzed with your choice of program. I use jupyter notebooks running **`python`**, **`numpy`**, and **`matplotlib`**.

Data Analysis

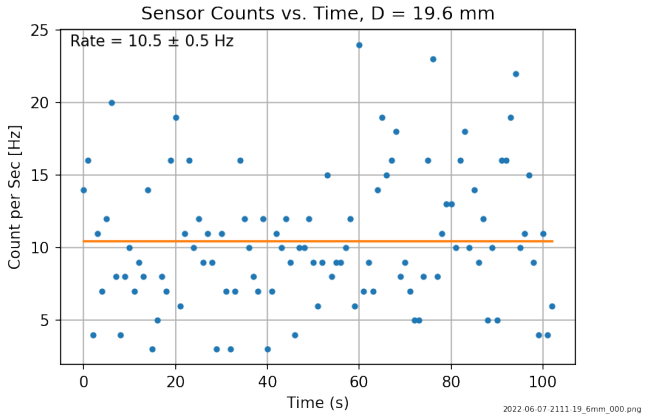
I use the **numpy** function **genfromtxt** to read CSV files and matplotlib to make the plots. Here are plots from a run I did before the workshop.



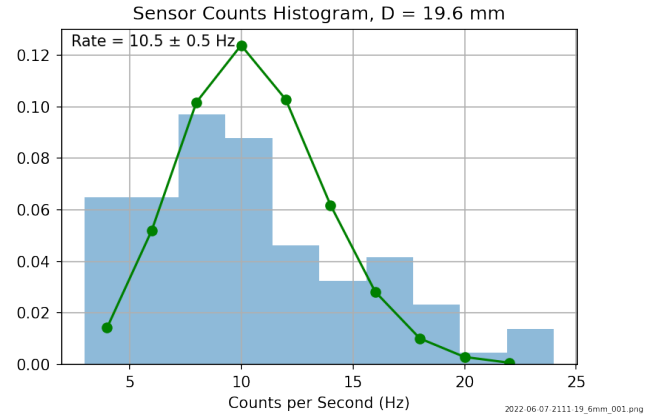
a)



b)



c)



d)

Figures a) and b) are from the shortest distance. The green line in b) is a Poisson distribution with a matching mean value. You can see the distribution is symmetric, as it should be for a large mean. In contrast, the data used in figures c) and d) are 19.6 mm distance and the mean is 10.5 Hz. The data are significantly skewed.

Extensions to the Project

I always start my advanced lab course with a counting experiment to give the students experience in building an experiment from scratch, programming a microcontoller, analyzing data that is novel to them, and writing a lab report. In the past I have used Geiger counters and Teesny microcontollers which are programmed in C++. In the last two years the students have showed more enthusiasm and programming competence using CircuitPython. The SiPIN detector is a new development I will use next spring.

Here are some extensions to the project.

- Modeling the counts versus distance. This can be complicated because is can include:
  - The geometrical effect of smaller detector spherical area versus distance,
  - Exponential absorption of the alpha particles in air, and
  - Energy loss per alpha particle in air. At distances of 18 mm I could see on the scope that many pulses were smaller in amplitude due to the energy loss.
- Alpha spectroscopy. I intentionally saturate the alpha pulses for counting when actually there is energy information in the pulse amplitudes. You could use a MCP6024 quad op amp to do further linear pulse shaping, then instead of an S2Mini board running CircuitPython, you could try a Teensy 3.2 microcontoller running MicroPython which has been tested to have about 60,000 samples/s, 17 µs/sample.

  The maximum value of a shaped pulse can be used to log the energy of the alpha particle. If you go to C++, the Teensy 4.0 microcontoller can record up to 1 million samples/s.