Pulse Interval Timing

Update: December 27, 2020

1   Introduction

Electronic pulses, like that shown in Fig. 1 have always played a key role in experimental work. They can signal that some event has occurred, because they either they exist or they do not. Their existence can mean "yes, it occurred," while their absence can mean "nothing has happened." Such pulses can also be generated sequentially, at regular intervals to drive some experimental need, like the flashing of an excitation pulse (of light) for example. The cabling you see among lab equipment is typically the conduit for pulses to travel between devices, that collectively make an experiment work.

Pulses can come in a variety of shapes and sizes. As mentioned, a single "square" pulse is shown in Fig. 1.
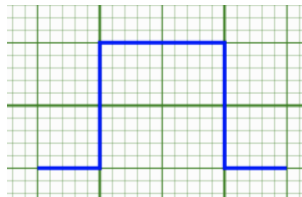

Figure 1: Example of a square pulse.

Such a pulse sits at 0V, rises quickly to +5V for some period of time, then rapidly falls to 0V again. Pulse amplitudes can vary, but formal "logic" pulses typically are either at 3.3 and 5V. Nuclear detection pulses (from a pre-amp) can even go negative in voltage.

A pulse *train* is a sequence of pulses, one arriving after another. They may be equally spaced (as from a function generator), or resemble this with the unequal spacing.
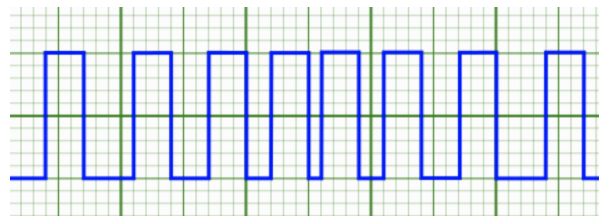
Pulse trains may also be a serial data stream, or a collection of many single pulses arriving at different times, each representing the occurrence of some event.

In this lab, we'll consider the presence of a pulse as a signal that something happened and in particular, we'll focus on examining the timing interval between such pulses for three physical processes:

1. Gamma-ray emission,

2. avalanche breakdown of a Zener diode,

3. and arrival of a Muon from outer space.

The focus here will be to statistically analyze the intervals between pulse arrivals, as generated by these processes.

1.1 Lab Motivation: Random Numbers

Believe it or not, the world is always in need of truly random numbers, or number that are unpredictable. Cryptography and Internet security for example, all critically depend on random numbers. As do gaming (i.e betting and gambling) and forecasting and simulations. Suppose an electronic slot machine in Las Vegas had a winning sequence that was somehow predictable?

As you may agree, there are processes in nature tend to be truly random and inherently unpredictable. This includes small (noise) fluctuations in a variety of scenarios, such as radioactive decay, beam splitting of photons, photoelectric emission, and thermal noise in electronic circuits. These are processes that no one can entirely predict the outcome of.

In this lab, we'll consider the randomness of the emission of a gamma ray from a nucleus, the reverse breakdown of a Zener diode, and a Muon decay. These events are truly randomness. We'll also address if they could they be used to generate random numbers (think: could you put such inside of a slot machine?).[1]

2 Random Process I: Muon Detection

Muons are created in the earth's upper atmosphere when cosmic particles (mostly protons) from outer space collide with molecules in the atmosphere. Muons then shower down to earth. We can detect them down here at the earth's surface by simply setting up a scintillation detector and leaving it on. When they strike the detector, a Muon will stop in the detector, then decay since Muons are not stable particles. The stop and decay each generate a flash of light in the scintillator. Each flash will cause–you guessed it–a square pulse into the detection electronics. The muon lifetime is shorter than the time itll take them to reach the earths surface, so we shouldnt see them down here save for time dilation in our reference frame, at the near c speeds at which they travel.

---

[1]A scientist at FermiLab has a gamma-ray detection system connected to the Internet, as seen here: `https://www.fourmilab.ch/hotbits/`. Sequences of random bits are posted.

As you might guess, Muon creation is our first natural and random process to study. No one knows when they will be created or detected. Further, once a Muon hits a detector, when it will decay is also totally random and unpredictable.

This document will not discuss Muon detection further. There is a section on the lab Github page here `https://github.com/tbensky/PhysicsLabs#lab-10-detecting-muons`, and there is also an established lab manual on this experiment you should follow.

# 3 Random Process II: Zener Diode Breakdown

In our second random natural event, we'll look at reverse avalanche breakdown of a Zener diode. Interestingly, you don't need a bunch of sophisticated electronics to study this. It can be done using an Arduino and a small circuit you can build yourself. This idea is based on a paper out of Stanford[2]. This group actually developed a circuitboard that will deliver truly random bits to some downstream computer. What would this be based on?

You may recall a Zener diode does not conduct in the reverse direction until an applied voltage across it exceeds some level. It then begins conducting. As soon as the applied voltage falls below that same level, the diode will cease conducting once again. Suppose we look at at 12.1V Zener diode. The 12.1V is the conduction voltage level. The Zener will not conduct as long as the voltage applied to it is below 12.1V, but will suddenly conduct if the applied voltage exceeds 12.1V. Zeners are often used to protect downstream electronics. Suppose one sits at the input of a sensitive device that can only sustain a 12V input signal. Should the input exceed 12V (or 12.1V), the Zener will conduct, shunting the input signal away from the sensitive input.

Here, we exploit an interesting behavior of Zener diodes. Suppose for the 12.1V Zener, we slowly start increasing the applied voltage across it. As it the voltage reaches around 10V or so, spurious thermal electrons at the PN layer may "jump across" the layer, being accelerated by the applied voltage. As one such electron accelerates, it'll collide with others, causing them to do the same, then others and others; an "avalanche" of electrons will start to jump the junction, causing a spurious and brief conduction event within the Zener. These conduction events are shown here, by the large spikes that grow downward to the CH1 ground level.

This will not be sustained conduction. It'll be brief, only lasting for a few microseconds, but it will be detectable. And note the use of "spurious" twice here, to mean *random*—truly random. No one can predict when a thermal electron will initiate the avalanche process. Thus, a Zener will exhibit truly random fits of temporary conduction. If one builds a circuit to monitor such conduction, the observation of the conduction will be witness to a truly random event in a do-it-yourself circuit.

---

[2]https://sing.stanford.edu/site/publications/rng-sensys16.pdf
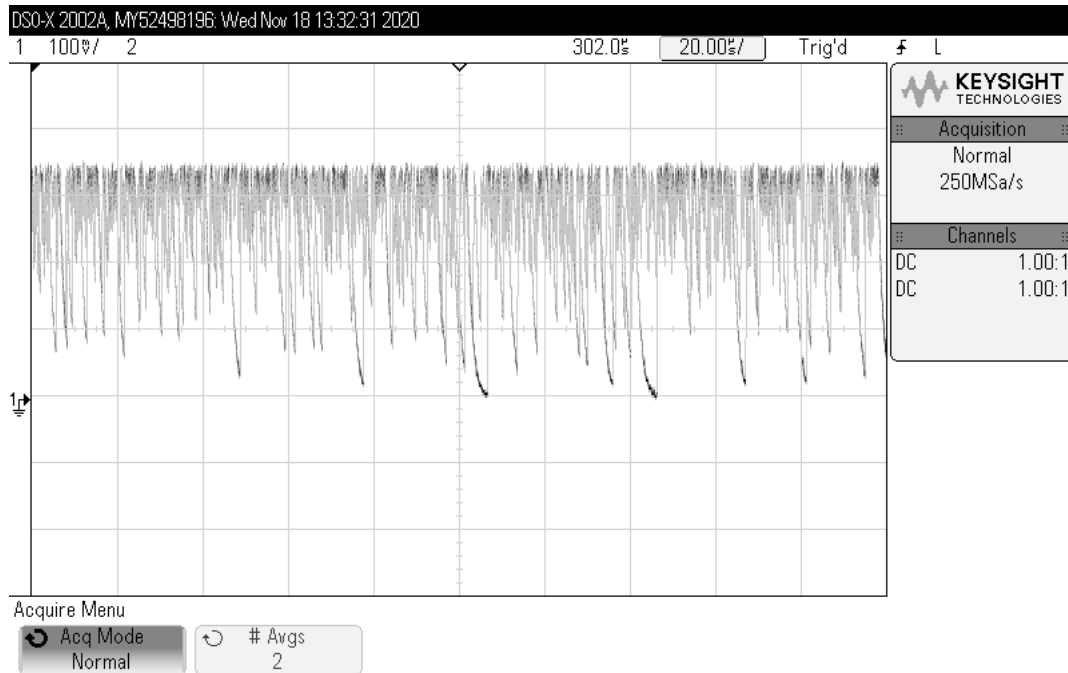
3.1 The Circuit

3.1.1 Parts needed

For this, you'll need the following parts.

1. 2 9V batteries

2. 18V battery clip

3. breadboard

4. Arduino Uno

5. Jumper wires

6. LM311

7. 10K resistor

8. Zener Diode

9. (optional) small speaker element

10. 1K resistors (2)

11. 51K resistor

12. Installed Arduino IDE. Free here: `https://www.arduino.cc/en/software`

In addition to building this circuit, you'll need an Arduino Uno to perform this experiment. It'll act like lab equipment that will take your data.

3.1.2 Building the circuit

Here is the circuit you'll need to build, along with a pinout of the LM311.
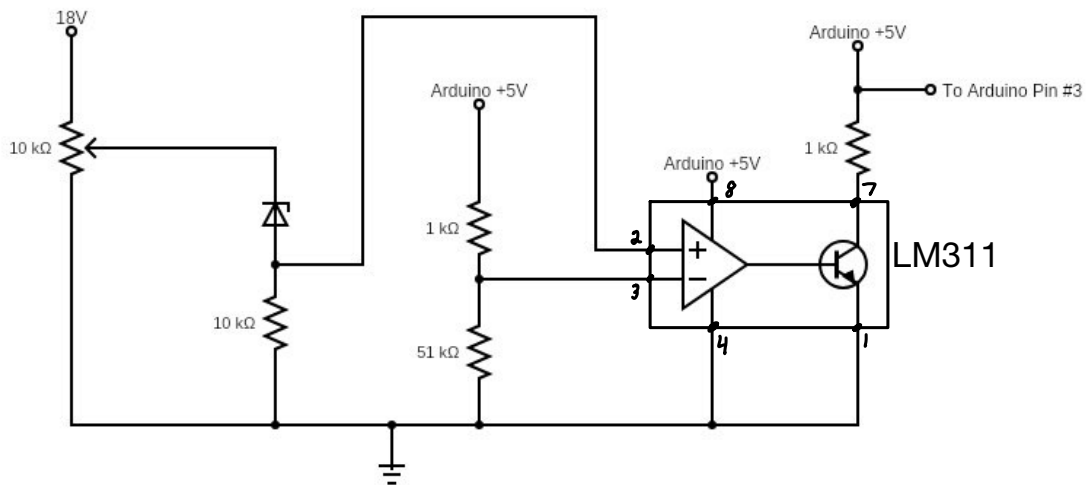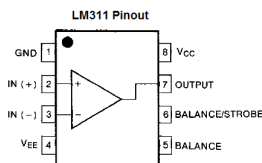

Figure 2: Zener diode avalanche to Arduino circuit.

This pin diagram of the LM311 should be useful.

Note when wiring up your LM311:

- Pin 8 is to be at +5 (power to the LM311)
- Pin 4 is to be at ground (grounding the power to the LM311)
- Pin 1 is to be at ground too (emitter of transistor)
- Pins 5 and 6 will be left unconnected
- Pins 2 and 3 are the + and - inputs as shown in the circuit
- The comparator and NPN transistor are all part of the LM311.

5

Circuit assembly hints can be found here: `https://youtu.be/wJv6fEVaaRA`, with a neater version here `https://youtu.be/2udPSj1wQOE`. A tour of the working circuit can be found here `https://youtu.be/XYe5NGDQSmE`.

3.1.3   Running the circuit

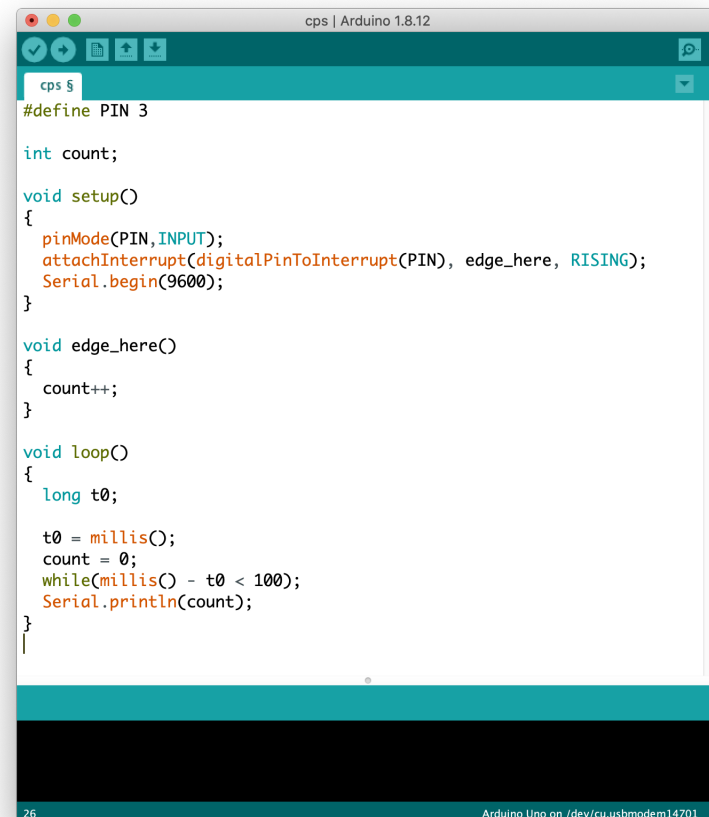It is hard to know if your circuit is working without an oscilloscope, but here what you may try.

1. Connect the 2 9V batteries to the 18V battery clip (get 2 for $1 at the dollar store).

2. Power up your Arduino by connecting its USB cable to your laptop or a USB plug-in charger.

3. Use your voltmeter to check the voltage on the wiper of the 10K potentiometer. As you turn the knob, this voltage should change. You should be able to vary it from 0 to 18V.

4. Check the voltage at the connection point between the 1K and 51K resistors. It should read about 0.1V.

5. Break the connection between the +5V and the top of the 1K resistor coming out of LM311, connect your small speaker to bridge this break in your circuit.

6. Turn the potentiometer until you hear a hissing/white-noise sound (these are the zener pulses generating audio frequencies via the speaker). The hissing should happen around 10.3V or so when you turn the potentiometer.

7. No hissing sound? Be sure the zener is inserted properly. How? Put your meter on the diode checker (look for the little diode symbol). Now, touch your leads to the zener's leads until the meter beeps. When it does, the zener's lead that the black voltmeter lead was touching should be connected to the +18V variable voltage (the middle wiper of the potentiometer) (see Fig. 2).

You can remove the speaker from the circuit, but be sure to reconnect the top of the 1K resistor coming out of the LM311 to +5V.

3.1.4   Arduino as a scalar

At this point, your circuit is producing random pulses originating from the zener diode. In the lab, counting how many pulses per time interval are produced is a key measurement. This is what the "scalar" did in the gamma-ray labs (big red digits). Here, you'll program your Arduino to work as a scalar.

The following code will count how many random pulses the Arduino sees coming from your circuit in 100ms (0.1 s). Compile and upload this code into your Arduino. Open the serial monitor. A list of numbers should start scrolling down the screen. Turning the potentiometer should change the overall size of each number. This is the number of random pulses/100ms your circuit is generating.


```
#define PIN 3

int count;

void setup()
{
  pinMode(PIN,INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN), edge_here, RISING);
  Serial.begin(9600);
}

void edge_here()
{
  count++;
}

void loop()
{
  long t0;

  t0 = millis();
  count = 0;
  while(millis() - t0 < 100);
  Serial.println(count);
}
```

Figure 3: Code to have the Arduino count how many pulses are seen coming in on digital input pin 3 in 100ms. The code is posted here https://github.com/tbensky/PhysicsLabs/tree/master/Labs/PulseIntervalTiming/ArduinoCode as well.

3.1.5 Arduino as a pulse interval timer

The following code will log the times *between* pulses coming from your circuit. This is the "pulse interval" timing. Compile and upload this code into your Arduino. Open the serial monitor. A list of numbers should start scrolling down the screen. which are the times (in microseconds) the Arduino is measuring between incoming pulses.

4 Random Process III: Gamma-ray arrivals

When dealing with gamma-rays, we're most used to discussing how many our detection system detects in some time interval, like 845 counts/second, etc. Suppose we instead wondered "once we detect a gamma ray, how long to we have to wait for the next gamma-ray?" Would it be 1/845 seconds? If so, would this be a constant for a given source? Measuring the "waiting time" or "interval" between gamma-ray emissions is the goal of this experiment. An Arduino Uno will be used to do the timing.

See `https://github.com/tbensky/PhysicsLabs#gamma-rays` for more details and data.

5 Where does the data come from?

All three processes generate the same kind of data. You'll have to think some about it all to tie this lab together. Here are some hints.

Muon. A pulse comes out of the pre-amp with every Muon stop or decay. These are "the pulses" generated by the system of study. The TAC does the interval timing for you. Think carefully about what the MCA software does in response to the TAC pulses as input. Also, think about how TAC and software limits the available data to you vs the Arduino use with the zener and gamma-ray. The muon anaylsis should have one graph and one curve fit.

Zener diode. The Arduino gives you two data sets. One is the counts per 100ms coming from the zener circuit, the other is the pulse intervals times. When you run your circuit, copy and numbers out of the serial monitor (turn autoscroll off) and paste them into your analysis software. How will you process the long lists of numbers? (You are looking to make two different plots and two different curve fits.) The fit parameters from both should be comparable. Find which ones, show how and why they should compare.

Gamma-ray. Like the zener data, once again the Arduino gives you two data sets. One is the gamma-rays detected per second. The other is the interval times between each gamma-ray arrival. How will you process the data sets? Once again, you are looking to make two different plots and two different curve fits. Like the zener data, each should have a fit done to them. The fit parameters from both should be comparable. Find which ones, show how they compare, any why they should compare.

The Muon, zener, and gamma-ray data all show a similar time-interval distribution. This is because these processes each show events that have something in common: each event is independent of any other event, the events continuously occur, and occur at at a constant average rate. These are called "Poisson Point Processes."

6 Analysis for randomness

Let's return now to randomness, discussed in Sect. 1.1. The Fermilab experiment with Hotbits does the following to generate random bits.[3]

Hotbits' premise is that gamma-ray arrivals are random, so the time intervals between their arrivals are also random. So a random series of bits can be generate as follows. Assume the time interval data numbers are labeled $t_0, t_1, t_2...t_N$. If one goes through the times one by one, bits can be generated like this:

$$bit(n) = \begin{cases} 1, & \text{if } t_n < t_{n+1} \\ 0, & \text{if } t_n > t_{n+1} \\ \text{skip} & \text{otherwise,} \end{cases}$$

for $n < N$ (note this is a forward looking algorithm).

For both your gamma-ray and zener interval times, implement some code that will generate the sequence of bits from each. Save them into two files named `gammabits.txt` and `zenerbits.txt`. Next, generate a third file called `softwarebits.txt` of random bits using the random number generator in your data analysis software. Lastly, generate about 500 random bits yourself, by typing them in on the keyboard. (That's right–bang away on the `1` and `0` key on your keyboard until you have at least 500 random bits. Do not use copy and paste at all.) Call this file `me.txt`.

6.1 Tests for randomness

So we have some data files containing a bunch of random 1s and 0s. But just how random are they? Is there a test for randomness? Yes, there are plenty. In fact there may be more tests for randomness that *uses* for randomness. BUT, computers are very fast and powerful these days. They consume many random numbers in their algorithms, with uses in physics, combinatorics, stochastic geometry, etc. (See the "Monte Carlo" lab in this class.)

Here are three common and largely common and empirical tests for randomness. Code these up and run them on all of your data sets. Your code should be robust enough to just need to read in one of your finals, and display the results.

6.1.1 Test your code first

Test your code against the 14,350 "random" bits in the data file here `https://github.com/tbensky/PhysicsLabs/tree/master/Labs/PulseIntervalTiming/TestBits` before running

---

[3]See `https://www.fourmilab.ch/hotbits/how3.html`.

your actual data files through it. (The results should be obvious and easy to check against your code, to check for programming errors.)

6.1.2 The tests

Frequency test. For a given data set, count the number of times a 1-bit appears. Now, count the number of time a 0-bit appears. If there are N bits in a given data set, how many times would you expect a 1 to appear if the data set was truly random? A 0? Can you think of a pattern of bits that would pass this test but be very obviously not random?

Serial test. For each data set, scan from beginning to end is steps of two-bits. For each two-bit sequence count how many times the two-bits are 00, 01, 10, or 11. (Note: this is a *non-overlapping* test, hence grouping your bit sequence into pairs of adjacent bits.) If there are N bits in a given data set, how many times would you expect each to occur?

Run test. A "run" is defined as a continuous run of bits in a sequence that are the same. Testing for randomness relies on looking for how many of each type of run exists. A run of 1s is sometimes called a "block" and a run of 0s a "gap." In this test, one looks for how many runs of each bit appear. For example, suppose your bits are 11100111110000101111. It has 4 runs of 1-bits, and 3 runs of 0-bits.

It turns out that runs of a random source are normally distributed and have an expected average number and a standard deviation. The Wald-Wolfowitz runs test (see `https://en.wikipedia.org/wiki/WaldWolfowitz_runs_test`) will be our guide. Supposing you have N_1 runs of 1-bits and N_0 runs of 0-bits, the expected number of total runs in your N total bits $(= N_1 + N_0)$ is

$$\mu = \frac{2N_1 N_0}{N} + 1, \tag{1}$$

with a standard deviation of

$$\sigma = \sqrt{\frac{(\mu - 1)(\mu - 2)}{N - 1}}. \tag{2}$$

6.1.3 Draw some conclusions on "randomness"

Using results of these tests, assess all four of your random bit data sets. Can you rank the "randomness" of the data sets? Also comment on the "randomness" of the sources that generated the bits.

6.2   ENT Software

Fermilab has some software called "ENT" (see https://www.fourmilab.ch/random/) that will assess the randomness of your bits. Download the Windows and macOS version from here https://github.com/tbensky/PhysicsLabs/tree/master/Labs/PulseIntervalTiming/ENT. It is not point and click software. You have to run it from your computers command line. Send your files through them 1 by 1 like this:
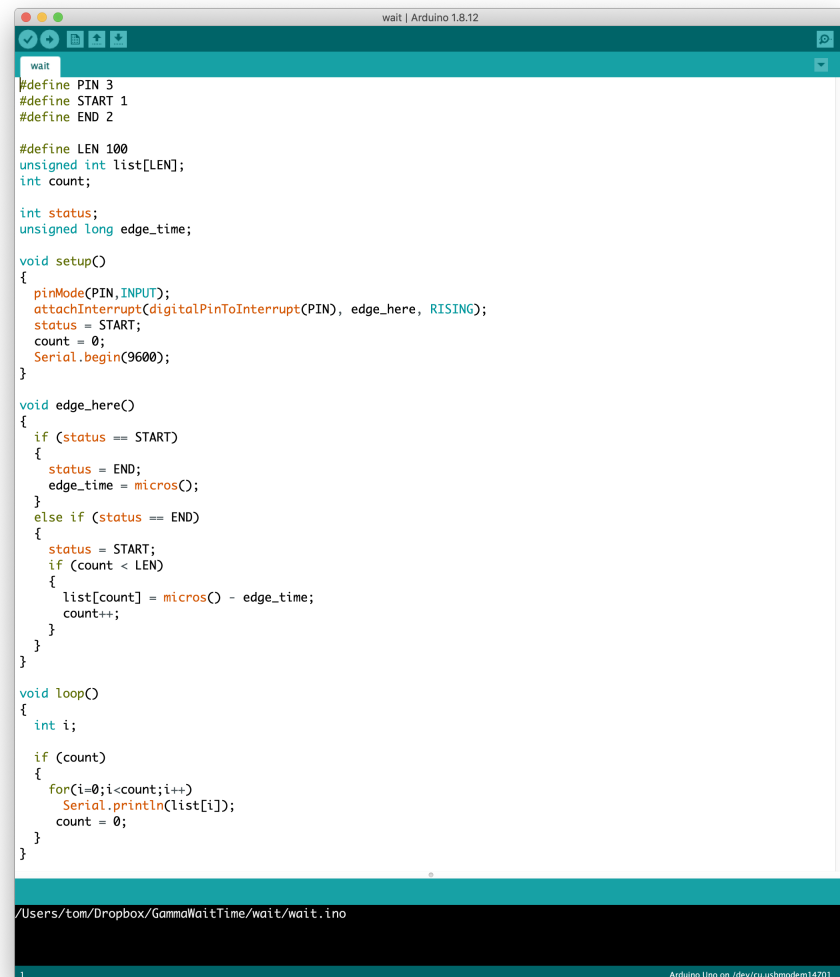
```
ent -b gammabits.txt
```

then

```
ent -b zenerbits.txt
```

and finally

```
ent -b softwarebits.txt
```

Interpret the output of each, and assess your bits' randomness. Compare your results with those from the Hotbits site. What can you say about the "entropy per byte" parameter?

```
#define PIN 3
#define START 1
#define END 2

#define LEN 100
unsigned int list[LEN];
int count;

int status;
unsigned long edge_time;

void setup()
{
  pinMode(PIN,INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN), edge_here, RISING);
  status = START;
  count = 0;
  Serial.begin(9600);
}

void edge_here()
{
  if (status == START)
  {
    status = END;
    edge_time = micros();
  }
  else if (status == END)
  {
    status = START;
    if (count < LEN)
    {
      list[count] = micros() - edge_time;
      count++;
    }
  }
}

void loop()
{
  int i;

  if (count)
  {
    for(i=0;i<count;i++)
      Serial.println(list[i]);
    count = 0;
  }
}
```

/Users/tom/Dropbox/GammaWaitTime/wait/wait.ino

Figure 4: Code to have the Arduino behave as a pulse interval timer. The code is posted here https://github.com/tbensky/PhysicsLabs/tree/master/Labs/PulseIntervalTiming/ArduinoCode as well.