Ardunio RNG and interfacer : Report

Sanket Doshi: 140010001 and Rakshit Jain: 140260018

November 3, 2016

Abstract

We have created a hardware random number generator using the principle of avalanche noise in diodes. We have also added a python interface for analysis and real time plotting of the random numbers generated. Overall project's perspective is to provide an interface to use such a random numbers in a more general sense.

1 Introduction

When pn junctions are operated in a region close to the point of reverse bias avalanche breakdown, avalanche noise occurs. This phenomenon is utilized by placing two bipolar junction transistors with bases touching to generate true random numbers unlike algorithmic generated pseudo random number. By sampling required number of bits from this string of random 1s and 0s, a random number can be generated in any desired range.

2 The Circuit

A picture of the circuit used is given below: The transistor Q_1's emitter is not connected, hence

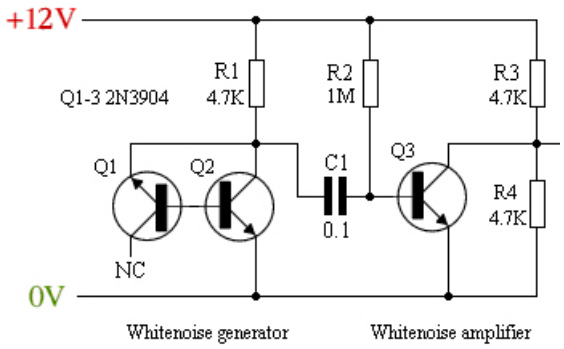

Figure 1: Circuit diagram for the Hardware Random Number Generator

by kirchoff's law $I_B + I_C = 0$ and hence Collector is in reverse biased condition on a voltage of around 12 V(it comes to be around 11.4 V) (Can be seen clearly as the I_B is very less.). According to the data sheet of 2N3904 it's breakdown voltage is 10.8 V, hence at around 12 V it will be operational in Breakdown mode.
This creates fluctuations in I_B fed to the next BJT and gets amplified in two stages The resistance values in the last Q_3 can be changed in order to a get an output between 0 - 5V. A very good description is given in reference 2 and reference 5. This type of circuit is well known to have a white noise characteristics.

3 Arduino Code and Functioning

Through this section, we will explain briefly how the sampling algorithm works and elaborate upon some important snippets from the Arduino code.

3.1 Calibration

While the calibration stage is active, Arduino reads the analog values on the output pin and maps to a corresponding integer from 0 to 255. These integers are further dumped into a bin corresponding to their values. By increasing calibration counter, we can get a better estimate of this threshold. There is, however, a time-accuracy trade off associated with this.

3.2 Setting a Threshold

The bin structure introduced in the previous subsection is used to compute the median value towards the end of calibration process, which will be used as a reference value for analog to digital conversion of all further values obtained on the output pin. We have attached the code below, for reference and understanding:

```
unsigned int findThreshold(){     // sets threshold for analog to digital conversion
  unsigned long half;
  unsigned long total = 0;
  int i;

  for(i=0; i < BINS_SIZE; i++){
    total += bins[i];
  }

  half = total >> 1;
  total = 0;
  for(i=0; i < BINS_SIZE; i++){
    total += bins[i];
    if(total > half){
      break;
    }
  }
  return i;                       // returns bin corresponding to the median value
}
```

The function that builds the string of output is implemented as follows

```
void buildByte(boolean input)
{
  if (output_format == ASCII_BOOL) Serial.print(input, DEC);
  if (output_format == BINARY) Serial.print(input, BIN);

}
```

3.3 Processing

Via the processing step we pass the digital value obtained from comparing the analog value at the output pin with the calibration set median value, to the bias removal function which further appends the appropriate binary value to the output byte.

4 Bias Removal

It is important to try and remove any long strings of consecutive 0s or 1s, since these add to the bias and thus skew the output depending upon where the sampling happens to end. We implement the removal of such bias through the Von Neumann filtering algorithm.
The next part of Filtering has been done by Sanket, hence you can find it in his report

5 Interfacing with Python

We have used a pyserial interface to interface the output of the arduino through the serial output dumped by the arduino to be used in python, main code which does the work is as follows -

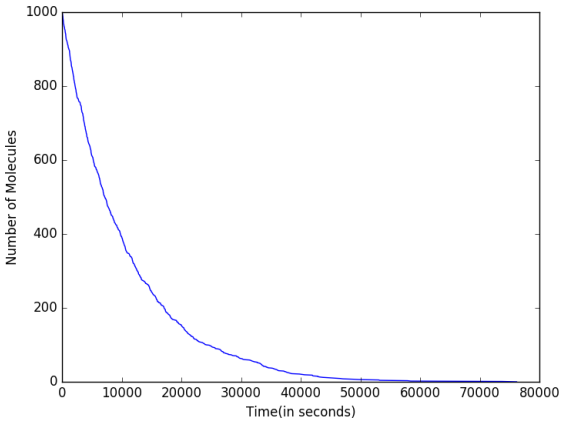

Figure 2: Resultant Plot for Gillespie Simulation

```
def random1(p):

        while 1 :
                # Checking the input bits available
                bytesavailable = arduino.inWaiting()
                arduino.write(str(chr(4)))


                if (bytesavailable) >= p :
                        # Conversion into a random number between zero and one.
                        q =  int(str(arduino.read(p)), 2)/(2.0**p - 1.0)
                        if q < 0 :
                                continue
                        break
        return q
```

You can see that this function can provide random number of any precision by sampling equivalent number of bits.
This part of testing has been done by Sanket and you can find it in his report

6 Chemical Reaction Simulations

We have written the following code in order to simulate a very simple decay chemical reaction -

$$A \rightarrow \Phi$$

It is well known that the time after the chemical reaction occurs is a log random number (Reference 6) and by using the pyserial interface as made before we have simulated this reaction to give the results as shown in figure 3.
Code is as follows -

```
timer = np.array([0])
p = np.array([])
n = 0
A = np.array([10])
t1 = time.time()
plt.ion()
while n < 1000 :
        # Defining Propensities for Log random number.
        propensity = .0001 * A[n]
        p = np.append(p, [random1(16)])
        q = (-1/propensity)*math.log(p[n])
```

```
            timer  = np.append(timer, timer[n] + [q])
            A = np.append(A, [A[n] - 1])
            print timer[n]
            print n

            n += 1
t2 = time.time()
print t2 - t1
plt.plot(timer, A)
plt.show()
```

7 Real Time data Plotting

We can plot the random number coming from the arduino using the interface we built. The code is as follows -

```
p = np.array([])
n = 0
t1 = time.time()
plt.ion()
while n < 1000 :
            p = np.append(p, [random1(16)])
            plt.hist(p, bins=20, color = 'c')
            plt.show()
            plt.pause(0.0001)
            plt.close()

            n += 1
t2 = time.time()
print t2 - t1
```

8 Contributions

The circuit is a well known circuit for generating white noise random number and has been studied extensively in references 2, 3, 4.
Rakshit did the interfacing with the python and performed chemical stochastic simulations and real time plotting of the random numbers.
Sanket performed the filtering, studied measures of the randomness and implemented the Chi-Squared test and the experimented with an implementation with Java using RXTX.

9 Summary

We have been able to achieve the following -

- Made a true random number generator and sampled it using Arduino
- Calibrating and Sampling it using Arduino.
- Applying von neumann filtering on the output generated by it.
- Tested the filtering by Chi - Squared tests.
- Interfacing it to a computer to use these random number to a desired level of accuracy
- Finally used the above structure to implement Gillespie Stochastic algorithm and Real time plotting of Random Numbers.

10 References

1. Von - Neumann corrector - http://everything2.com/title/von+Neumann+corrector
2. Basic Circuit diagram for Hardware Random Generator - http://www.cryogenius.com/hardware/rng/
3. Motivation for Hardware Random Number generator - https://web.jfet.org/hw-rng.html
4. Functioning of the Random Number generator - http://holdenc.altervista.org/avalanche/
5. Pyserial Documentation - http://pyserial.readthedocs.io/en/latest/
6. Gillespie Algorithm - *https : //en.wikipedia.org/wiki/Gillespie_algorithm*

11 Appendix

11.1 Code for the Arduino

```
#define BINS_SIZE 256
#define CALIBRATION_SIZE 50000

#define NO_BIAS_REMOVAL 0                                  // bias removal scheme
#define VON_NEUMANN 1

#define ASCII_BYTE 0                                       // formats for output
#define BINARY 1
#define ASCII_BOOL 2

/***  Configure the RNG **************/
int bias_removal = NO_BIAS_REMOVAL;
int output_format = BINARY;
float baud_rate = 115200;
/************************************/


unsigned int bins[BINS_SIZE];
int adc_pin = 0;
int led_pin = 13;
boolean initializing = true;
unsigned int calibration_counter = 0;


void setup(){
  pinMode(led_pin, OUTPUT);
  Serial.begin(baud_rate);
  for (int i=0; i < BINS_SIZE; i++){
    bins[i] = 0;
  }
}

void loop(){
  byte threshold;                                          // byte stores an integer  //
from 0 to 255
  int adc_value = analogRead(adc_pin);
  byte adc_byte = adc_value >> 2;                          // mapping the value to  //0-255 range
  if(calibration_counter >= CALIBRATION_SIZE){
    threshold = findThreshold();                           // set the threshold  //once calibration is
    initializing = false;
  }
  if(initializing){
    calibrate(adc_byte);
    calibration_counter++;
  }
  else{


      processInput(adc_byte, threshold);                   // process any input that is fed after the c

  }
}

void processInput(byte adc_byte, byte threshold){
```

```
  boolean input_bool;
  input_bool = (adc_byte < threshold) ? 1 : 0;        // compare the input to //obtained threshold
  switch(bias_removal){
    case VON_NEUMANN:                                 // feeds the output of the previous step to
      vonNeumann(input_bool);
      break;
    case NO_BIAS_REMOVAL:
      buildByte(input_bool);
      break;
  }
}


void buildByte(boolean input){                        // responds to the calls of debiasing functi
  static int byte_counter = 0;
  static unsigned int out = 0;

    if (output_format == ASCII_BYTE) Serial.println(input, DEC); //Serial.print(", ");
    if (output_format == BINARY) Serial.print(input, BIN);
  if (output_format == ASCII_BOOL) Serial.print(input, DEC);


}


void calibrate(byte adc_byte){                        // while calibration is on, it updated the fr
  bins[adc_byte]++;
}

unsigned int findThreshold(){                         // screens the received data while calibratio
  unsigned long half;
  unsigned long total = 0;
  int i;

  for(i=0; i < BINS_SIZE; i++){
    total += bins[i];
  }

  half = total >> 1;
  total = 0;
  for(i=0; i < BINS_SIZE; i++){
    total += bins[i];
    if(total > half){
      break;
    }
  }
  return i;
}
```

11.1.1 Code for python

```
import serial, time, sys
import matplotlib.pyplot as plt
import math
class mySerial(serial.Serial):
    def __del__(self):
        pass
arduino = mySerial('/dev/cu.usbmodem1421', 115200)
```

```
import numpy as np

time.sleep(1) #give the connection a second to settle

def random1(p):

        while 1 :
                bytesavailable = arduino.inWaiting()
                arduino.write(str(chr(4)))


                if (bytesavailable) >= p :
                        q =  int(str(arduino.read(p)), 2)/(2.0**p - 1.0)
                        time.sleep(.01)
                        if q < 0 :
                                continue
                        break
        return q
```