Microprocessor Lab Project Report

Examining Random numbers generated using Johnson noise and chaotic mapping

Anshul Avasthi

Anushrut Sharma

Rishabh Khandelwal

Guide : Prof. Pradeep Sarin

Department of Physics, IIT Bombay [4cm]

Acknowledgments

We would like to thank Saurabh Mogre, Niladri Chatterji and Kartik Kothari for their excellently documented project, using which as a foundtion, we have built ours. We thank Prof. Pradeep Sarin for his help throughout the course of this project and Mr. Nitin Pawar for his valuable insights in moments of dire need.
We are, of course, grateful to Texas Instruments Inc. for providing us with free samples of INA 217 and REF 02 without which this project would not have been possible.

Contents

1 Introduction

In this report we summarize the work we did over the past eight weeks for the microprocessor lab project. The aim of our experiment is to generate truly random numbers, since there exist numerous applications that would benefit greatly from the existence of easily accessible truly random numbers. Computational fields like cryptography are among the first that come to mind.

1.1 Stern-Gerlach Experiment

We started the project with a very ambitious plan of generating random numbers using quantum mechanical phenomenon. We initially planned to use the probabilistic nature of the collapse of wave function to generate a random number. For this we decided to make use of the well established Stern-Gerlach experiment. This experiment was to be conducted with an external magnetic field applied along, say, the z axis. In that case, the spin of the electron would collapse randomly along either +z(bit 1) or -z(bit 0) axis with each of them having an equal probability. This way, we would have succeeded in making a random sequence of binary numbers.

However, Prof. K. G. Suresh pointed out that it would be very difficult for us to get the appropriate magnetic field required for the Stern-Gerlach experiment(a magnetic field with a linear gradient) and Prof. D. K. Ghosh told us that a much simpler way would be to use optical phenomenon instead. For instance, the polarization of light instead of the spin of electrons could be the quanta used to generate random numbers. We then decided to conduct this experiment using polarization of light.

1.2 Random Number generation using polarization of light

Truly random numbers can be generated using polarization of light by using a polarized source of light and passing it through a polarizer whose axis is inclined at 45° with respect to the polarization of the incident light. In this case, the probability for a photon to pass through the polarizer would be 0.5. Thus, we can make random sequences of binary numbers by assigning a bit value of 1 to a photon which passes through the polarizer and a bit value 0 to one which fails to pass through the polarizer. Again, we found out that this experiment wasn't feasible as Prof. Parinda Vasa informed us that we could only get a laser of wavelength 450nm and above, whereas the PMT available from Prof. Pragya Das's lab was only capable of detecting wavelengths greater than or equal to 350nm. Given these constraints, we had to again rethink on our experiment. Moreover the available intensity of the laser was way too high to be able

to detect a single photon at a time.

1.3 Johnson Noise

It is at this point that we decided to use the past as a reference to decide how to move forward. Saurabh, Niladri and Karthik had used Johnson noise to obtain white noise as a signal and then pass it via a ADC to get an array of truly random numbers instead. Since we had already spent too much time deliberating upon processes irrelevant to electronic systems, we decided to use the same approach as well. We are currently generating random numbers using Johnson noise (thermal noise in a resistor), as was used last year.

Johnson-Nyquist noise is the electronic noise generated by the thermal agitation of the charge carriers inside an electrical conductor at equilibrium, which happens regardless of any applied voltage. The noise is then amplified and processed on a micro controller to obtain strings of random numbers.

2 Circuit

2.1 Noise Generation

Thermal noise has a magnitude of approximately 20μVolt (calculated via back-calculation). We used a balanced wheatstone bridge, made up of four 930Ω resistors using resistor gate arrays, for precision. The voltage across the balanced arms is the Johnson thermal noise, superimposed upon a small DC signal caused due to the mismatch in the resistance ratios. We amplify this noise in three steps, and attenuate the DC signal using an RC filter. After this, we examine the randomness of the final signal.

2.2 Voltage Supply

The voltage source for the sytem was regulated to 5V using the Texas instruments IC REF02. The voltage regulator provides a steady 5V supply within 0.3% and has a low noise rating of 15μV.

2.3 Wheatstone Bridge

The wheatstone bridge was designed using precision resistors connected via resistor gate arrays of 930Ω. The values of the resistors were chosen such that they were within the load driving capacity of the Voltage regulator (REF02).

2.4 Two stage amplication

The signal from the arms of the wheatstone bridge was fed to instrumentation amplier INA214 (manufactured by Texas Instruments). Two gain stages were used to reduce bandwidth limiting and to provide enough amplication. The gains in the two stages were 500 and 5 respectively. These values of the gain give a signal with a peak to peak value of 140mV, and a DC offset of 10V.

2.5 Using OpAmp LM741 as a voltage buffer

The INA217 is an IC incapable of providing the sheer amount of current needed at the next step - and would start heating up whenever we tried to do so. After trying to make several tweaks in the load, we decided to introduce an OpAmp as a Unity gain amplifier instead. The OpAmp provides upto 2mA of current.

2.6 Filtering of DC offset

There is a small DC component arising from mismatch in resistor values which gets amplied along with the noise signal. It was filtred out using a high pass filter with a very high capacitor value (8000μV). The filter has a cutoff value of 50Hz. It is this extremely high value of the capacitor which requires a large amount of current to be able to get charged in a reasonable amount of time. The final signal now has a DC offset of about 10mV.

2.7 Third stage amplication

The filtered output was then amplied with a gain of 20 to give a signal of 1.5V peak to peak value.

2.8 Noise input to Arduino

The resulting signal from the capacitor could not be fed directly to the microcontroller forprocessing, as the input range for the Arduino is 0 to 5V. To get the signal in this range, a voltage adder circuit was designed to add a stable 2.5V DC voltage from a REF02 using an OpAmp LM741.

Another LM 741 was introduced as a Unity gain amplifier to prevent the 2.5V signal (obtained via a voltage divider) from being attenuated when entering the OpAmp.

3 Processing the noise on Arduino

We have done away with the chaotic mapping used by Saurabh Mogre, Niladri Chaterjee and Karthik Kothari - since doing so would entirely supercede the noise we have obtained from the system, and give us final data points that lie on the strange attractor of the logistic map. The final data points obtained this way have only a very weak correlation with the initial conditions given to the system.
Hence, we converted the input values (which range from 0 to 1023) to 9-bit binary numbers and posited them into an array in C++ for further analysis.

4 Results

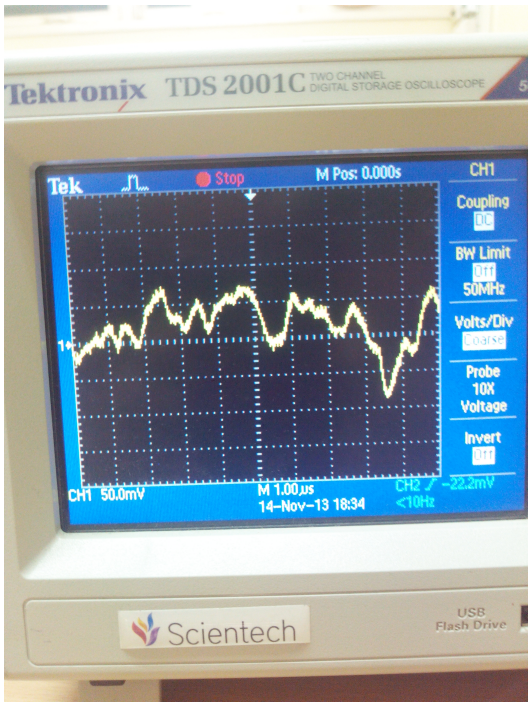

Figure :

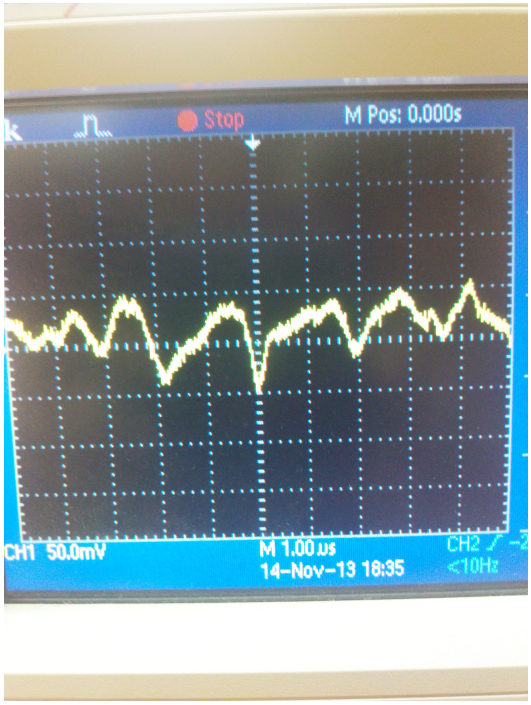

Figure :

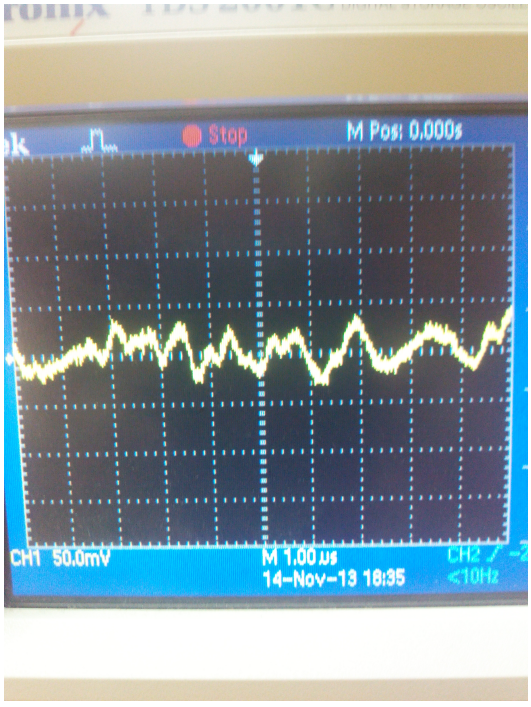

Figure :

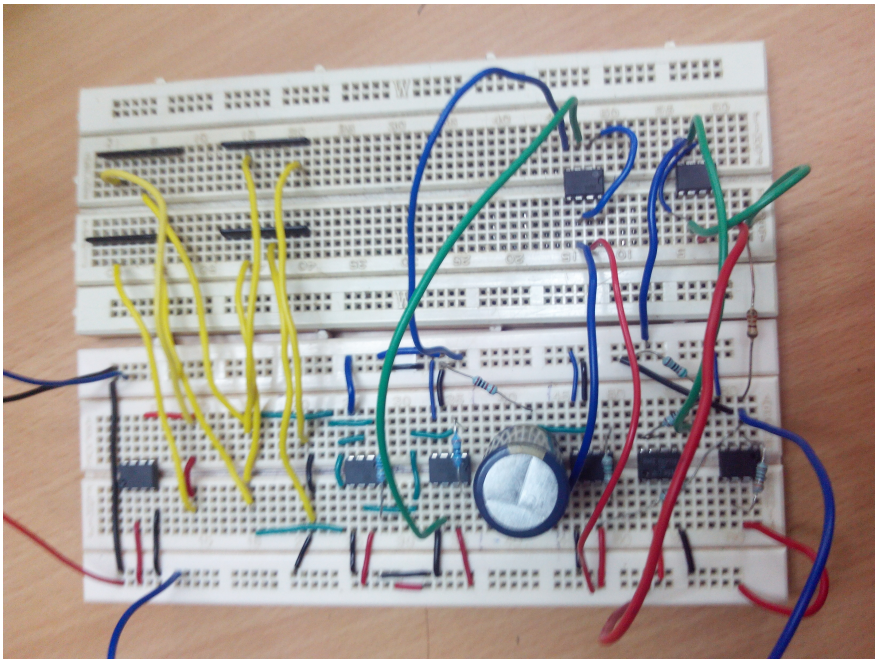

Figure :

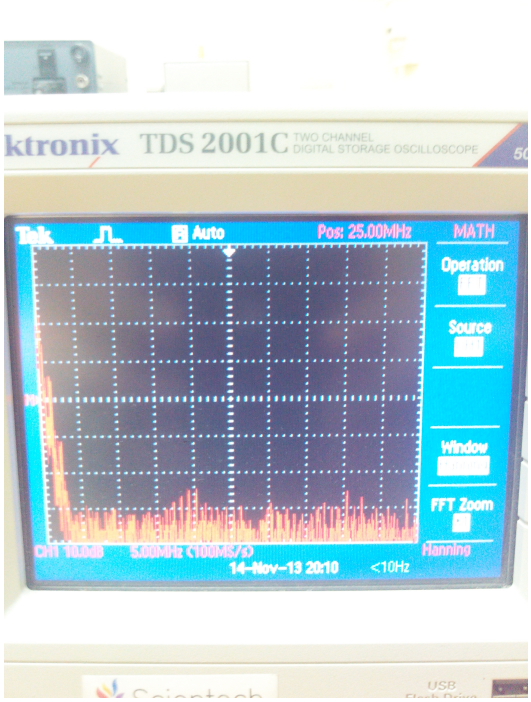

Figure : Fast Fourier Transform of noise data

5 Code for Fast Fourier Transform

```
\fontsize{6pt}
#define _USE_MATH_DEFINES
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>

#define PI M_PI
#define TWOPI (2.0*PI)

using namespace std;
int NFFT=0;
double *data;//FFT data size

void four1(double data[], int nn, int isign)//FFT
{
    int n, mmax, m, j, istep, i;
    double wtemp, wr, wpr, wpi, wi, theta;
    double tempr, tempi;

    n = nn << 1;
    j = 1;
    for (i = 1; i < n; i += 2) {
if (j > i) {
    tempr = data[j];     data[j] = data[i];     data[i] = tempr;
    tempr = data[j+1]; data[j+1] = data[i+1]; data[i+1] = tempr;
}
m = n >> 1;
while (m >= 2 && j > m) {
    j -= m;
    m >>= 1;
}
j += m;
```

```
    }
    mmax = 2;
    while (n > mmax) {
istep = 2*mmax;
theta = TWOPI/(isign*mmax);
wtemp = sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi = sin(theta);
wr = 1.0;
wi = 0.0;
for (m = 1; m < mmax; m += 2) {
    for (i = m; i <= n; i += istep) {
j =i + mmax;
tempr = wr*data[j]   - wi*data[j+1];
tempi = wr*data[j+1] + wi*data[j];
data[j]   = data[i]   - tempr;
data[j+1] = data[i+1] - tempi;
data[i] += tempr;
data[i+1] += tempi;
    }
    wr = (wtemp = wr)*wpr - wi*wpi + wr;
    wi = wi*wpr + wtemp*wpi + wi;
}
mmax = istep;
    }
}

int getData()//Get data from sample.txt
{
  int i=0;
  string line;
  ifstream myfile ("sample.txt");
  if (myfile.is_open())
  {
    while ( getline (myfile,line) )
    {
        i++;
```

```
    }
    myfile.close();
  }

  else
  {
          cout << "Unable to open file";
          return 0;
  }

  NFFT = (int)pow(2.0, ceil(log((double)i)/log(2.0)));
  data=new double[2*NFFT+1];
  i=0;
  myfile.open("sample.txt",ios::in);
  if (myfile.is_open())
  {
    while ( getline (myfile,line) )
    {
        data[2*i+1] = line[0]*1-48;
data[2*i+2] = 0.0;
i++;
    }
    myfile.close();
    while(i<NFFT)
    {
data[2*i+1] = 0.0;
data[2*i+2] = 0.0;
i++;

    }
  }

  else
  {
          cout << "Unable to open file";
          return 0;
  }
```

```
  return 1;

}

//data stores the Fourier transform . 2n+1 gives real. 2n+2 gives complex

int main()
{
    int a=getData();
    a=0;
    while(a<NFFT)
    {
    cout<<data[2*a+1]<<" "<<data[2*a+2]<<endl;
    a++;
    }
    four1(data, NFFT, 1);
        a=0;
    while(a<NFFT)
    {
    cout<<data[2*a+1]<<" "<<data[2*a+2]<<endl;
    a++;
    }
}
```

6 Work Distribution

The main components of the project were:

1. Analog Circuit design and connection to Arduino
2. Verification of randomness

All three of us contributed significantly to each component of the project but the circuit design was mostly handled by Anshul and Rishabh and Verification of randomness was mostly handled by Anushrut.