

Semester Project

Thobias Høivik

Western Norway University of Applied Sciences
ELE201: Microcontrollers and Data Networks

Contents

1	Introduction	3
2	Theory	4
2.1	Randomness and Entropy	4
2.2	True vs. Pseudo Randomness	4
2.3	Hypothesis	4
2.4	Theoretical Modeling of Analog Noise	5
3	Implementation	7
3.1	Hardware Setup	7
3.2	Data Transmission	7
4	Results and Analysis	8
4.1	Results from the Raw Signal	8
4.2	Processing	9
4.3	Post-processing	11
4.4	Results After Post-Processing	14
5	Experimental Applications	15
5.1	Monte Carlo Estimate	15
6	Network Design	17
6.1	Example Network Topology	17
6.2	Transmission and Synchronization	17
7	Conclusion	18
7.1	Future Work	18

1 Introduction

In this project we use the STM32 microcontroller along with analog sensors to gather data that we analyze for statistical and informational randomness, with the goal of evaluating how feasible such a setup is as a hardware-based true random number generator (TRNG). The data will be sent via serial connection to a computer to analyze the quality of the data.

We will combine theory from information theory, probability and signal processing to assess the randomness of our data.

We will discuss:

- Theoretical background on randomness and entropy.
- Implementation and data transfer.
- Statistical analysis and randomness testing.
- Viability discussion and possible improvements.
- Network design.

2 Theory

2.1 Randomness and Entropy

Randomness can be defined both in a statistical and algorithmic sense. A sequence is considered random if it is unpredictable and lacks compressible structure.

Definition 2.1: Shannon Entropy

Given a discrete random variable X that takes values in χ with probability distribution $p : \chi \rightarrow [0, 1]$, its entropy is

$$H(X) = - \sum_{x \in \chi} p(x) \log_2 p(x).$$

(as introduced by Shannon [1] and discussed in [2, 4]). Entropy is a central measure in information theory pertaining to the randomness of data and it will be referenced extensively throughout this text. For our purposes of generating random sequences of bits, entropy becomes a surprisingly intuitive measure. In the case of a bit-string, it will have maximum entropy if and only if there is a 50% chance for every given bit in the string to be 0 or 1 and the proof only requires a bit of differential calculus. Thus we would want our source of random numbers to exhibit the maximum entropy possible for a bit-string of the given length.

2.2 True vs. Pseudo Randomness

A pseudo-random number generator (PRNG) produces deterministic sequences from an initial seed, while a true random number generator (TRNG) relies on physical entropy [3]. For hardware-based RNGs, ensuring unbiased and unpredictable output requires:

- High-quality analog entropy sources.
- Statistical post-processing.

2.3 Hypothesis

We believe that due to the inherit noise of electrical circuits that dominate at low frequencies, the least significant bits (LSBs) of our signal should be a source of high entropy. Then, with further post-processing we should get a high-quality source of random numbers which can be applied in processes which require randomness.

2.4 Theoretical Modeling of Analog Noise

To justify that the least significant bits of our ADC contains randomness, we model the analog sensor output as a combination of deterministic signal and noise:

$$V_{\text{ADC}}(t) = V_{\text{Signal}}(t) + V_{\text{Noise}}(t)$$

- V_{Signal} represents the slowly varying, predictable component (e.g. ambient light level in the case of photoresistor).
- V_{Noise} represents the unpredictable physical noise (thermal-, shot noise, quantization error, sensor-specific noise).

The noise, which is the sum of thermal-, shot-, and other electronic noise is typically modeled as a Gaussian random variable [5]:

$$V_{\text{Noise}}(t) \sim \mathcal{N}(0, \sigma_{\text{Noise}}^2)$$

The **ADC quantization** transforms the continuous voltage into discrete levels:

$$X = \text{ADC}(V_{\text{ADC}}) \in \{0, 1, \dots, 2^{12} - 1\}$$

The voltage corresponding to one LSB (least significant bit) step is

$$\Delta V = \frac{V_{\text{ref}}}{2^{12}}$$

for a 12-bit ADC with reference voltage V_{ref} .

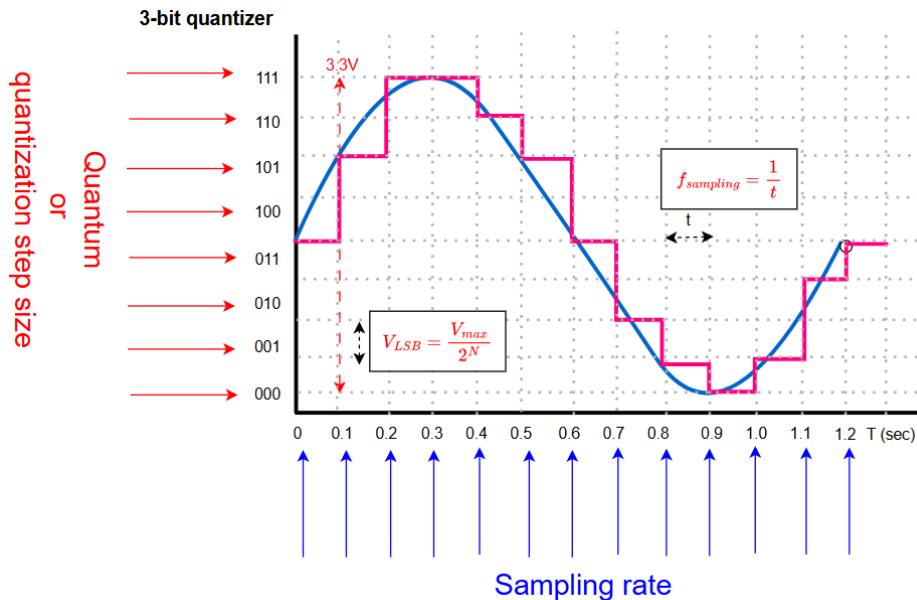


Figure 1: Quantization of analog signal [10].

The randomness of the extracted bits is measured by the Shannon Entropy. For a discrete random variable X with k possible outcomes, the maximum possible entropy $H_{\text{MAX}} = \log_2(k)$ bits.

We extract the 4 LSBs, which gives $2^4 = 16$ possible outcomes from $0000_b \rightarrow 1111_b$. The maximum entropy then is

$$H_{\text{MAX}} = 4 \text{ bits}$$

Maximum entropy is achieved if the probability of observing any of the 16 outcomes is uniform [2, 4]:

$$P(D_{\text{LSB}} = i) = \frac{1}{16} \text{ for } 0 \leq i \leq 15$$

To achieve a near-uniform distribution across the 16 LSB states, the standard deviation of the noise σ_{Noise} must be large enough to span multiple quantization steps.

Let $\Delta V_{4\text{-bit}}$ be the voltage corresponding to the 4 LSBs:

$$\Delta V_{4\text{-bit}} = 16\Delta V$$

Consider an input voltage V_{ADC} that falls within a $\Delta V_{4\text{-bit}}$ window. The probability that the resulting digital word D falls into a specific 4-bit LSB state i depends on the area under the Gaussian Probability Density function of the noise that falls into the corresponding voltage interval I_i .

The critical condition for near-maximum entropy is then:

$$\sigma_{\text{Noise}} \gg \Delta V$$

If the noise standard deviation is significantly larger than the LSB step size then the Gaussian noise distribution becomes "smeared" across multiple LSB intervals. Since the noise is zero-mean and σ_{Noise} is large, the probability of the total input V_{ADC} falling into any one LSB interval I_i is nearly equal to the probability of it falling into an adjacent $I_{i\pm 1}$.

3 Implementation

3.1 Hardware Setup

We will use the STM32f767zi microcontroller, connected to a photoresistor.

The microcontroller samples the analog voltage via the ADC and stores or streams the digital values over a serial or network interface. We will be sampling at 12-bits, keeping the 4 least significant bits as we believe these will be most susceptible to noise.

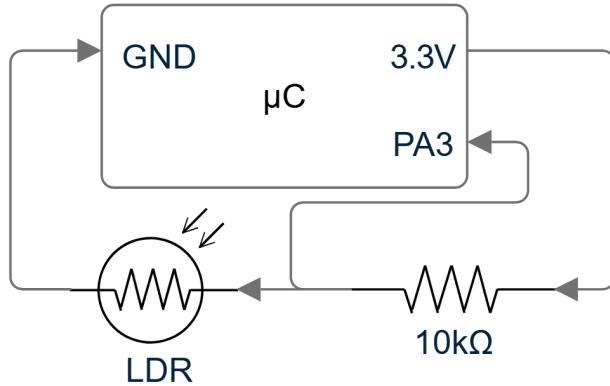


Figure 2: Diagram

3.2 Data Transmission

Data is transmitted from the STM32 to a computer for analysis. In our experimentation we use USART/Serial Connection using a micro-usb connection from the microcontroller to a computer.

4 Results and Analysis

4.1 Results from the Raw Signal

We begin by analyzing the apparent randomness of the raw signal.

In our experimentation we first use Python with the serial library to read the incoming data and matplotlib to plot the signal over time.

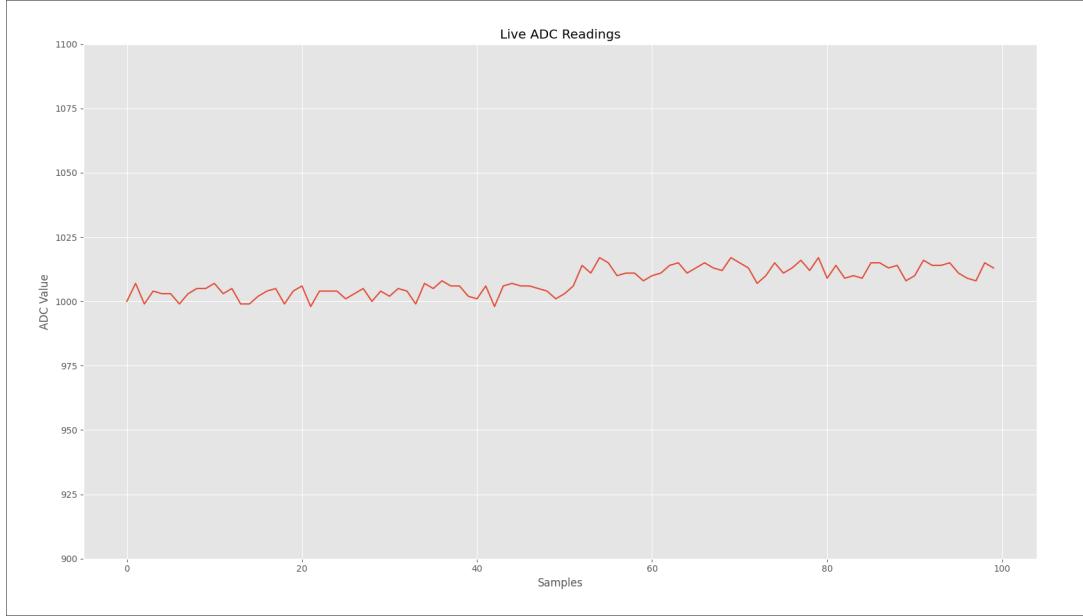


Figure 3: Raw signal plot showing small fluctuations.

As we can see in this plot, while the signal remains relatively stable, reflecting the stable light level we were testing against, there are definite micro-jitters present. This is a good sign and is almost certainly due to the noise which we predicted would dominate at small levels.

We do have to take into account the fact that the micro-jitters might be due to slight fluctuations of the ambient light-level in our tests. To mitigate this possibility we will from this time forward be testing with the photoresistor covered with electrical tape.

Now we look at the observed probabilities pertaining to the 4 LSBs. We want a roughly equal probability that a given bit in the 4 LSBs is 0 or 1.

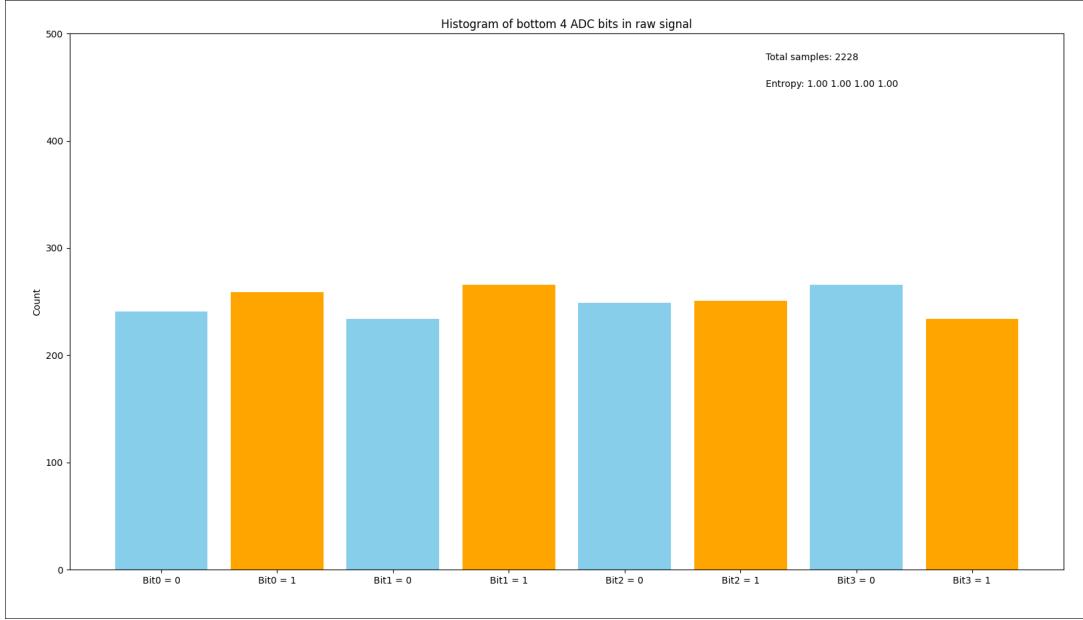


Figure 4: 4 LSB histogram showing exhibiting entropy.

As we can see there appears to be a roughly equal chance that a given bit in the 4 LSBs is 0 or 1, pointing towards high entropy in the LSBs.

Furthermore, we can see that each bit has an entropy of 1 and the sum of the 4 LSBs entropies is 4. As we determined earlier the maximum entropy $H_{MAX} = 4$, meaning that the 4 LSBs exhibit maximum entropy.

4.2 Processing

Our goal is to transform the raw data stream (which is **biased** and **predictable**) into a shorter, statistically perfect random stream. So far we've already done some processing in discarding all, but the 4 LSBs.

As discussed previously, the 4 LSBs are a good source of entropy, but using 4 bits at a time is not sensible as it is relatively easy to predict given the small number of combinations at only 16.

Now, since we have observed that each individual bit of the LSBs exhibit a maximum entropy (1), then taking 64 samples and appending each by bit-shifting 4 bits to the left should give us a 256 bit number with maximum entropy.

In this section we will examine 4 central metrics in determining the quality of the 256-bit number.

1. **Entropy.** Entropy is still important and we want an entropy as close to 1 as possible in this case.
2. **Bit frequency.** We examine how many 0s vs. 1s there are in a given bit-string. A 50/50 distribution is obviously desireable.
3. **Autocorrelation.** The correlation between consecutive bits is important and we want to be as close to 0 as possible, indicating no correlation/pattern in consecutive bits.
4. **Runs.** Runs are the number of "blocks" of like bits. For example 0111110_b contains 3 blocks, the first 0, the consecutive 1s and the final 0. A run of 256 is bad because it indicates alternating

0s and 1s. 1 is also very bad as it indicates that the string only has 0s or 1s. Hence we want a run-count somewhere in the middle of these extremes.

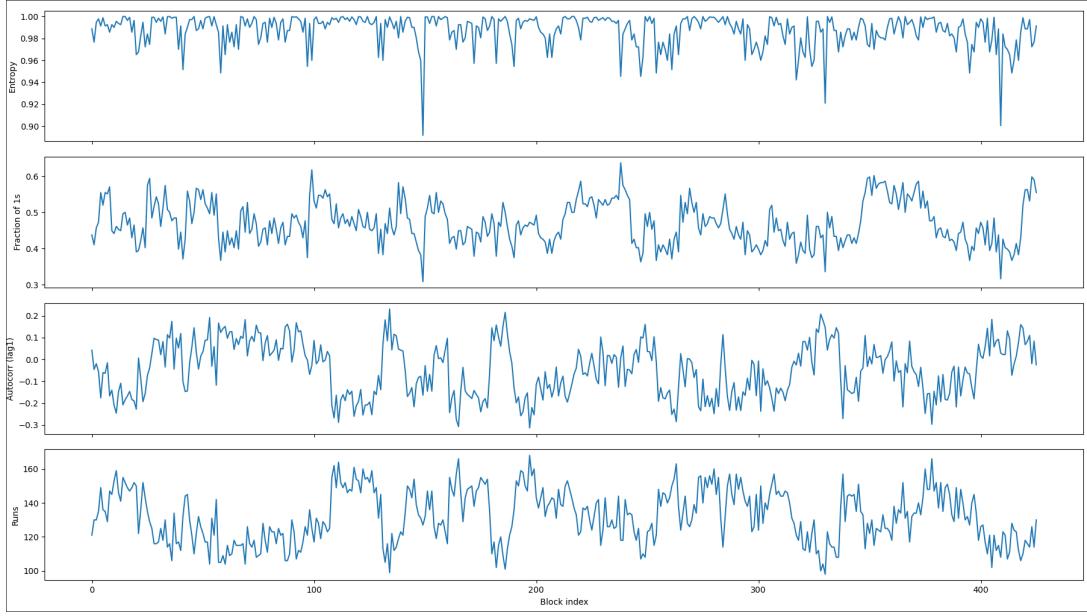


Figure 5: Readings showing good results from 256-bit numbers.

Our readings show that our 256-bit numbers tend to hover around the desired values, never reaching any disconcerting values during the test that produced this plot. During this test, 434 numbers were analyzed with the following averages:

1. Average Entropy: 0.987
2. Average Bit Frequency: 0.472
3. Average Autocorrelation: -0.045
4. Average Runs: 131.990

These results are promising, but we want to employ a more stringent test to analyze the **RNG**-quality of our 256-bit blocks. The employed test suite enforces a rigorous hypothesis testing framework, where the **null hypothesis** H_0 posits the bitstream as being perfectly random, and results are accepted only if the calculated P-value is greater than the significance level $\alpha = 0.01$. Specifically, the **Monobit Frequency Test** assesses the uniformity of 0s and 1s by calculating the Z -score,

$$Z = \frac{|S_n - n/2|}{\sqrt{n/4}}$$

where S_n is the sum of ones in the $n = 256$ bit sequence, and deriving the P-value via the complementary error function, $P_1 = \text{erfc}(Z/\sqrt{2})$. The **Runs Test** evaluates the occurrence of contiguous identical bits by calculating the observed number of runs V_n and comparing it against the expected mean μ and variance σ^2 , again yielding a Z -score

$$Z = \frac{|V_n - \mu|}{\sqrt{\sigma^2}}$$

for the P-value P_2 . Finally, the **Serial Overlapping Blocks Test** serves as a strong proxy for Approximate Entropy by using a χ^2 goodness-of-fit statistic

$$\chi^2 = \sum_{i=1}^{16} \frac{(C_i - E)^2}{E}$$

across all $2^m = 16$ possible $m = 4$ bit blocks, with the P-value P_3 determined by the Chi-squared survival function $\chi^2.sf(\chi^2, 2^m - 1)$. The block is certified as *random* if and only if $P_1 \geq \alpha \wedge P_2 \geq \alpha \wedge P_3 \geq \alpha$, providing a composite measure highly resistant to statistical bias.

In our statistical testing, we apply the **Z-test** rather than the **T-test**. This choice is justified because our bitstreams contain a large number of independent samples ($n = 256$ or more), which allows us to assume that the sampling distribution of the mean is approximately normal by the Central Limit Theorem. The population variance is known (under H_0) to be $\sigma^2 = 0.25$ for Bernoulli($p = 0.5$) data, so a Z-test is the appropriate parametric test. In contrast, a T-test would be necessary only if σ^2 were unknown or estimated from small samples. Our approach incorporates a few of the many tests employed by the NIST SP 800-22 test suite [11].

With this approach we quickly discover the limitations of using only the raw signal of our LDR to create our numbers.

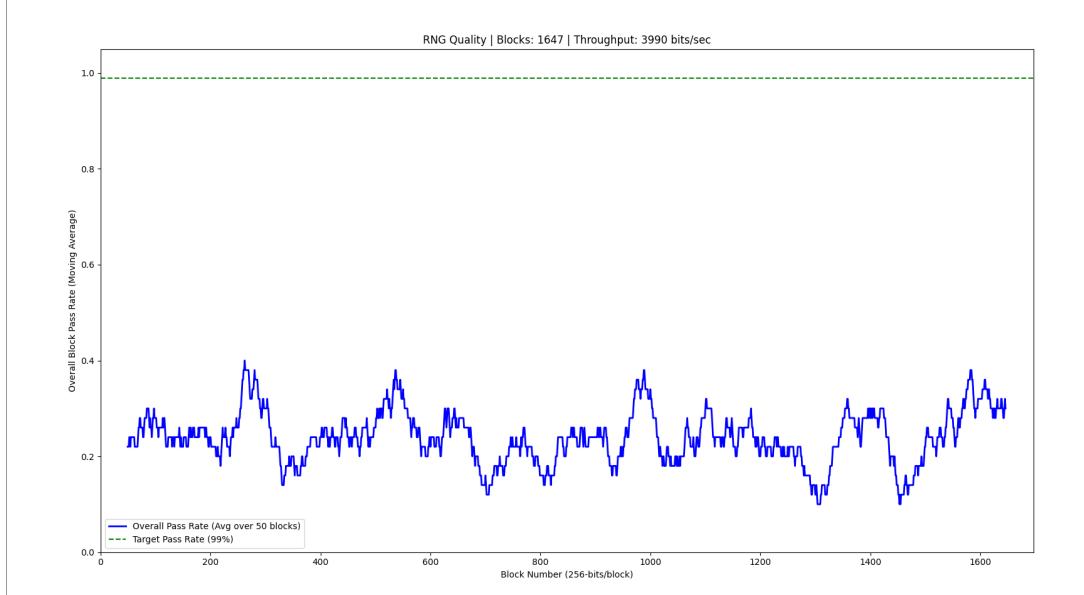


Figure 6: Test suite results showing high fail-rate.

It is clear, from the fact that less than half of our 256-bit numbers pass all our tests, that some more involved processing is required to get high-enough-quality random numbers.

After testing over 10,000 blocks we find that we pass the bit-frequency test 79.2% of the time, runs is passed 9.36% of the time and the serial overlapping test is passed 45.11% of the time. This is far below our desired result and suggests that more involved processing is required to get our data up to higher quality.

4.3 Post-processing

To try to improve our random-numbers we will implement and test a number of different post-processing techniques.

1. **XOR Whitening.** XOR Whitening is a simple linear post-processing technique designed to remove bias and short-range correlations in binary sequences.

Our raw 4 LSB data contains high entropy, but likely exhibits a small DC bias due to the observed bit frequency and minor autocorrelation. These imperfections can reduce randomness quality in cryptographic contexts. By XOR-ing adjacent bits or consecutive samples we can break

local correlations and push the bit frequency closer to 0.5, which is a requirement in our more stringent test suite.

Given a bit sequence x_0, \dots, x_{n-1} , we compute a whitened sequence y_i as

$$y_i = x_i \oplus x_{i-1}, \quad (i \geq 1)$$

This is equivalent to applying a discrete derivative in \mathbb{F}_2 [6]. Theoretically, if x_i and x_{i-1} have any non-zero bias or dependence, the XOR removes the common component.

The resulting sequence has

$$P(y_i = 1) = 2p(1 - p)$$

where $p = P(x_i = 1)$.

If $p \neq 0.5$, this transformation drives $P(y_i = 1)$ closer to 0.5.

XOR Whitening is computationally trivial and thus very suitable for real-time usage, not reducing the bit-rate significantly.

2. **Von Neumann Corrector.** The Von Neumann corrector is a classical post-processing algorithm that eliminates bias without assuming any specific noise model [7]. It was originally proposed by John von Neumann in 1951 as a way to extract unbiased bits from biased coin flips.

The algorithm processes the bitstream in non-overlapping pairs:

$$(x_0, x_1), (x_2, x_3), \dots$$

and outputs:

$$y_i = \begin{cases} 0 & \text{if } (x_{2i}, x_{2i+1}) = (0, 1) \\ 1 & \text{if } (x_{2i}, x_{2i+1}) = (1, 0) \\ \text{discard} & \text{if } (x_{2i}, x_{2i+1}) \in \{(0, 0), (1, 1)\} \end{cases}$$

The idea is that if the input bits are independent, the pairs (01) and (10) occur with equal probability even if the marginal distribution of x_i is biased. Hence, the output is guaranteed to be unbiased, though at the cost of a variable bit-rate reduction.

This method is particularly valuable when the bias is strong or unknown, but independence between successive bits can be assumed. Its simplicity and provable bias removal make it a standard benchmark in physical random number generator literature.

3. **Linear Feedback Shift Register (LFSR) Mixing.** A Linear Feedback Shift Register (LFSR) is a widely used structure for generating and transforming binary sequences with good statistical properties [8]. In our context, we employ a short LFSR as a *whitening filter* that linearly mixes consecutive raw bits to reduce correlations and spectral peaks [9].

An LFSR consists of a register of length m and a feedback function defined over \mathbb{F}_2 :

$$s_{n+m} = \bigoplus_{i \in T} s_{n+i},$$

where $T \subseteq \{0, \dots, m-1\}$ defines the feedback taps. For example, a primitive polynomial such as $x^8 + x^6 + x^5 + x^4 + 1$ yields a maximal-length LFSR of period $2^8 - 1$. For our purpose we use the primitive polynomial $x^{256} + x^{10} + x^5 + 1$.

To whiten the input sequence x_i , we XOR it into the feedback path:

$$s_{n+m} = x_n \oplus \bigoplus_{i \in T} s_{n+i}.$$

This injects entropy from the measured signal into a well-distributed linear recurrence, effectively smoothing out local correlations and enhancing bit balance.

Unlike XOR Whitening, LFSR mixing has memory and thus acts as a linear filter over multiple past samples. The resulting bitstream exhibits flatter spectra and higher effective entropy, while retaining full bit-rate.

4.4 Results After Post-Processing

Here we present the results from testing our data after post-processing. The tests we're performed on at least 1000 256-bit numbers after the respective post-processing techniques.

	Monobit freq. Pass Rate	Runs Pass Rate	Serial Overlap Pass Rate
XOR	92.45%	12.34%	57.58%
Von Neumann	64.71%	10.78%	36.27%
LFSR	98.95%	22.77%	94.66%
LFSR → XOR	98.51%	24.19%	93.70%
LFSR → Von Neumann	99.98%	20.69%	94.25%
LFSR → Von Neu. → XOR	99.36%	22.53%	95.28%

Table 1: Pass Rates for processed numbers.

As we can see our post-processing techniques help bring up the pass rates, especially when it comes to the Monobit Frequency test and Serial Overlap test. However we see that the Runs pass rate remains relatively low. Now, this is more of a problem for the purpose of cryptography, but for the purpose of pure random number generation it is less of an immediate issue.

5 Experimental Applications

Now that we have a reasonably high quality source of random numbers we can demonstrate a few applications of these numbers.

5.1 Monte Carlo Estimate

The Monte Carlo method uses statistical sampling to determine a deterministic result and is one of the most effective ways to visually and numerically test the quality of a random number source. The process involves generating a large number of random points $(x, y) \in [0, 1]^2$.

Given that

$$\frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\frac{1}{4}\pi r^2}{r^2} = \frac{\pi}{4}$$

we can estimate π by generating N random points in the unit square, count the number of points in the circle, equivalently count the number of points satisfying

$$x^2 + y^2 \leq 1$$

calculating π as

$$\pi \approx 4 \frac{N_c}{N}$$

where N_c denotes the number of points that lie in the inscribed circle.

This process will converge to π as we generate more points. For this process to work we need a uniform and unbiased random number generator since if our RNG is biased, say towards generating points at $(0, 0)$, this process will not converge towards π . Furthermore our underlying random process must be independent such that there is low correlation between a given samples and preceding samples.

For the sake of time we use the LFSR+XOR post-processing as it is the most time-efficient processing method amongst our top performers.

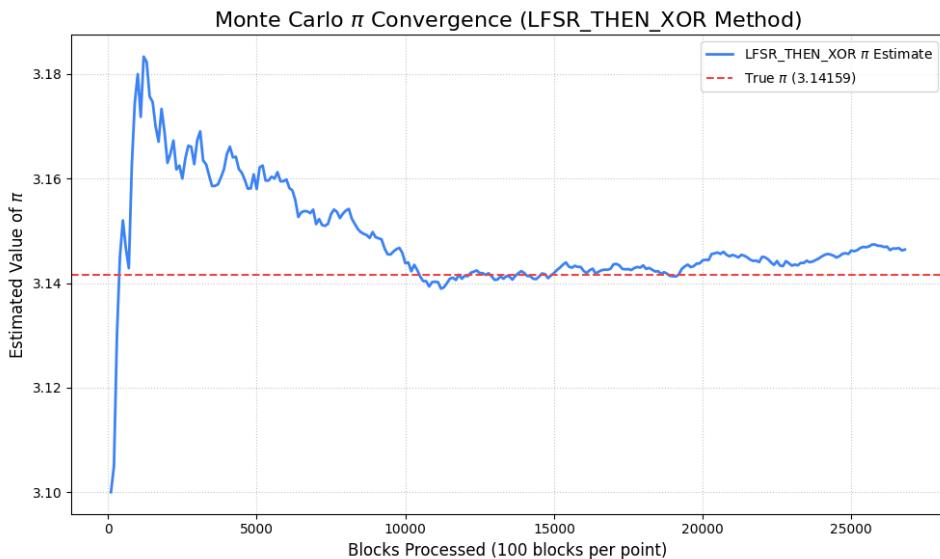


Figure 7: Pi estimate converging quickly to low error.

Through our experimentation we find that our π -estimate converges reasonably quickly when compared to software-based pRNGs. While our estimate diverges a little after around 20000 samples we achieved a estimated value of 3.1467... after only 107520 points generated, yielding an error of around 0.005. The real discrepancy when compared with traditional methods is that this simulation took 1724.98 seconds where a traditional approach might achieve a similar number of points in a few seconds.

6 Network Design

In practice, a hardware RNG based on microcontrollers can be deployed as a distributed network of entropy sources. Each node contributes independent random samples to a central entropy pool. This architecture is attractive because it improves fault tolerance and aggregate entropy through diversity of physical sources (e.g. temperature, light, or electrical noise variations across devices).

6.1 Example Network Topology

We consider a small local network consisting of:

- 1 Central server (entropy collector and analysis node).
- 4 STM32 RNG sensor nodes.
- 1 router connecting them in a star topology.

The network is built on the private IPv4 block 192.168.10.0/24. To organize traffic and reduce broadcast load, we divide it into smaller subnets using /26 masks (each supporting up to 62 hosts).

Subnet	Address Range	Purpose
192.168.10.0/26	192.168.10.1 – 192.168.10.62	Sensor nodes (RNG units)
192.168.10.64/26	192.168.10.65 – 192.168.10.126	Central server + analysis tools
192.168.10.128/26	192.168.10.129 – 192.168.10.190	Administrative access / test devices
192.168.10.192/26	192.168.10.193 – 192.168.10.254	Reserved for future expansion

Table 2: Example subnetting scheme.

Each STM32 node communicates with the central server using UDP packets to minimize transmission overhead. The payloads consist of 256-bit entropy blocks accompanied by a timestamp and CRC checksum:

[HEADER | TIMESTAMP | 256-bit DATA | CRC16]

6.2 Transmission and Synchronization

UDP transmission ensures low-latency updates, while the server performs redundancy checks by verifying integrity.

A small excerpt of the configuration might look like this:

```
Server IP: 192.168.10.70
Sensor_1 IP: 192.168.10.10
Sensor_2 IP: 192.168.10.11
...
UDP Port: 4000
```

This modular layout allows the system to scale by simply allocating new subnets or extending address ranges. In larger deployments, several microcontroller clusters could feed entropy into a regional collector, forming a multi-layer “entropy network” for research or security systems.

7 Conclusion

This project demonstrates the theoretical and experimental feasibility of using analog noise from sensors, specifically a photoresistor connected to an STM32 microcontroller, as a source of physical entropy for a hardware true random number generator (TRNG). By isolating and sampling the least significant bits of ADC readings, we captured the stochastic component of electronic noise and quantified its entropy. Statistical and information-theoretic analysis confirmed that while the raw data exhibited measurable randomness, post-processing was essential to achieve high-quality output.

Through XOR Whitening, Von Neumann correction, and LFSR-based mixing, we significantly improved bit balance and independence, with the combined LFSR→Von Neumann→XOR pipeline producing randomness that passes most stringent tests. The Monte Carlo simulation of π demonstrated that the generated random numbers behave statistically as expected, albeit with performance constraints due to serial data transfer rates.

A simple but scalable network architecture was also proposed, showing how multiple microcontroller nodes could contribute entropy to a central pool over standard IP networks. This opens the door to distributed entropy networks and low-cost, scalable random number generation systems.

7.1 Future Work

This project is ripe with various future ideas and hypotheses to test. For one we may wish to gather more statistically significant data, running more numbers through our various test suites.

To achieve this we can address data generation speed, tackling the problem of being able to generate and process more numbers in a given time interval than we are at the moment. To do this there are many optimizations we can make, like improving our testing environments or using top-of-the-line software like NIST SP 800-22 or Dieharder. Post-processing steps like LFSR can be done in hardware to lessen computational strain.

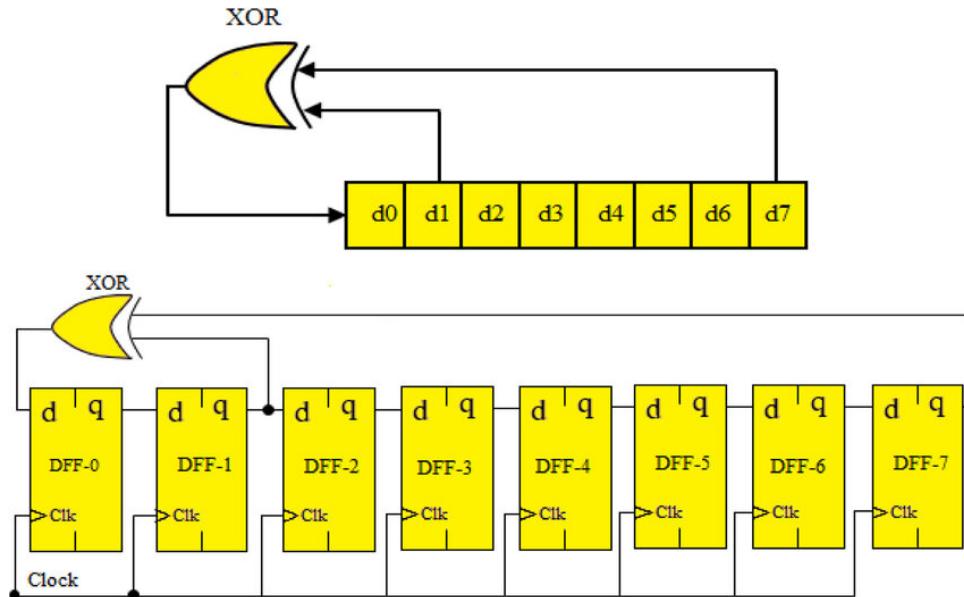


Figure 8: LFSR Wiring diagram.

We can elucidate and justify in a more rigorous way, why we are justified in believing that electrical noise is a source of high-entropy and thereby a good source for random number generation. By

doing more stringent testing we can address the application of cryptography, where more stringent requirements for our data may need to be met.

In addressing many of these issues, and showing that we have a source of randomness valuable for cryptographic applications, we could address cost and deployment complexity for use by private and public interests.

References

- [1] C. E. Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal, vol. 27, pp. 379–423, 623–656, 1948.
- [2] R. M. Gray, *Entropy and Information Theory*, Springer, 1990.
- [3] Cerberus Security, *Hardware Random Number Generators*, 2020.
https://cerberus-laboratories.com/blog/random_number_generators/
- [4] Statistics How To, *Shannon Entropy Definition*,
<https://www.statisticshowto.com/shannon-entropy/>
- [5] G. Gundersen, *Random Noise and the Central Limit Theorem*, Blog post, 01 February 2019.
<https://gregorygundersen.com/blog/2019/02/01/clt/>
- [6] F. Bagnoli, “Boolean derivatives and computation of cellular automata,” *Dipartimento di Matematica Applicata, Università di Firenze, Firenze, I-50139, Italy; also INFM and INFN, sezione di Firenze*, 1999. Available: <https://arxiv.org/pdf/cond-mat/9912353.pdf>.
- [7] Yuval Peres, “Iterating von Neumann’s Procedure for Extracting Random Bits,” *Annals of Statistics*, vol. 20(1), 1992.
- [8] Analog Devices, “Random number generation using LFSR,” Design Note, 200X.
- [9] “Linear-feedback shift register,” Wikipedia. Available:
https://en.wikipedia.org/wiki/Linear-feedback_shift_register.
- [10] *Online: HVL - ELE201 Microcontrollers and Data Networks Website*,
<https://fjnn.github.io/hvl-ele201/lectures/15-adc>, Latest access: 03/11/2025
- [11] A. Rukhin et al., *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22, Rev. 1a, Apr. 2010. [Online]. Available: <https://doi.org/10.6028/NIST.SP.800-22r1a>
- [12] Figure 8: *LFSR Feedback connection diagram (from Devari et al.), 2024, reproduced under CC BY-NC-ND 4.0 license*. Source: Devari, A., Kumari, A., Kuchal, P., & Verma, C., (2024). *Sequential Logic circuit gold codes for electronics and communication technologies. MethodsX*, 12(1), 102602. <https://doi.org/10.1016/j.mex.2024.102602>

Appendix A: Repository and Code Access

All source code, scripts, and datasets used in this project are available at the following public repository:

GitHub Repository: <https://github.com/ThobiasKH/ELE201-semesterproject>

The contents of the repository include, but are not limited to:

- STM32 firmware source code.
- Python scripts for serial communication and data analysis.
- Statistical test suite and Monte Carlo demonstration code.