Semester Project

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

Contents

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, signal processing, and networking to assess both the quality of randomness and how such a system could integrate into a distributed architecture.

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 for scaling such systems.

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 $p : \chi \to [0, 1]$, 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 meassure in information theory pertaining to the randomness of data and it will be referenced extensively throughout this text.

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]:

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

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

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

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}.

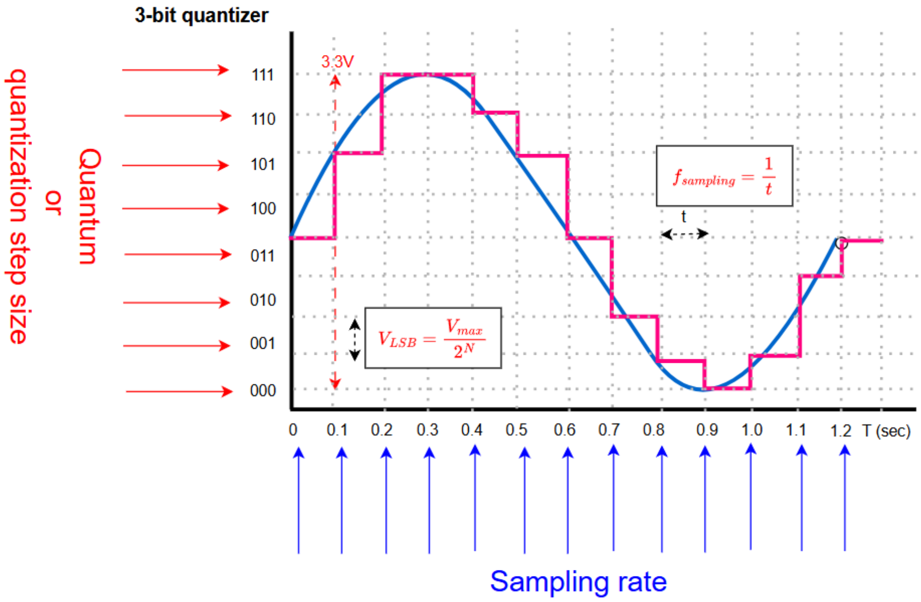

Figure 1: Quantization of analog signal.

The randomness of the extracted bits is meassured 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 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 $I_{i\pm1}$.

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.

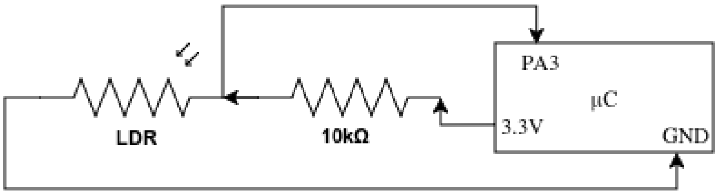

Figure 2: Diagram

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.

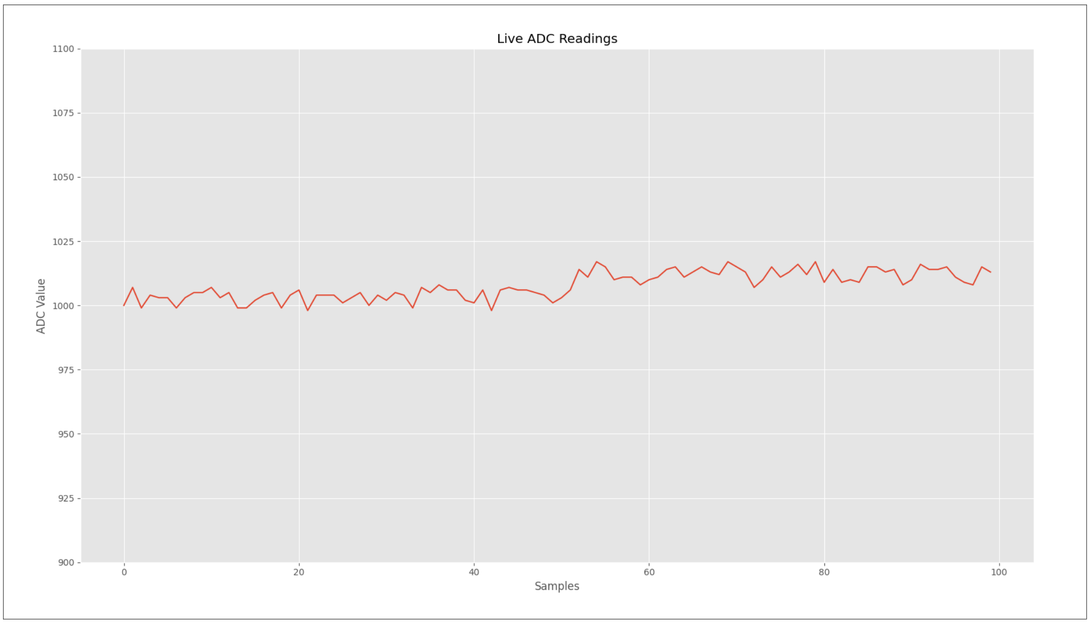

Figure 3: Plot of the raw signal over time

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 ceirtainly 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.

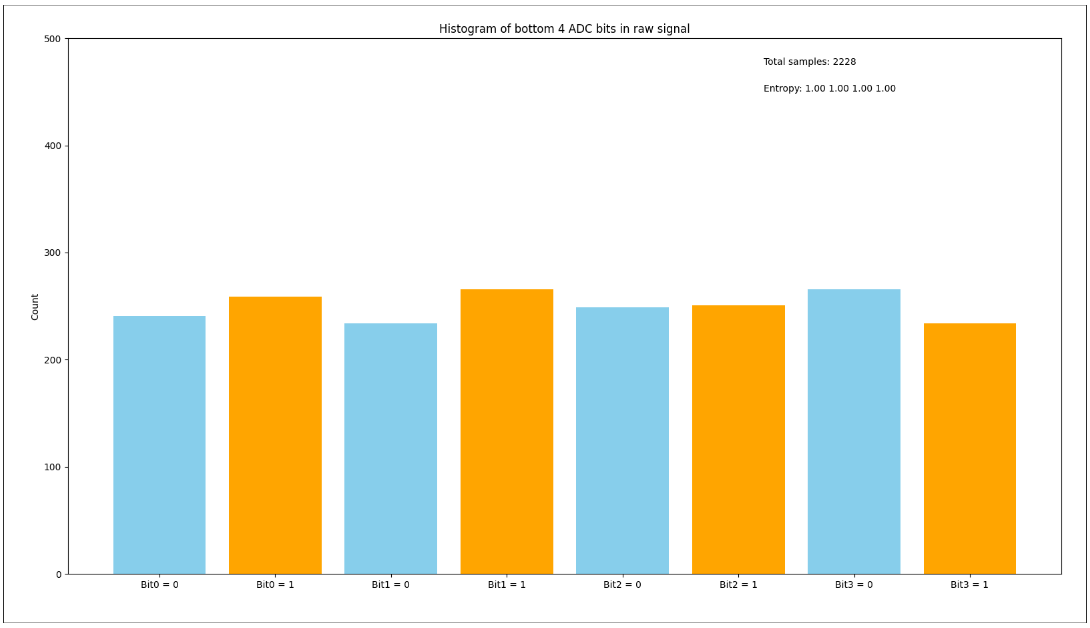

Figure 4: Histogram of 4 LSBs with photoresistor covered

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 $H_{\text{MAX}} = 4$, meaning that the 4 LSBs exibit 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 exibit 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.

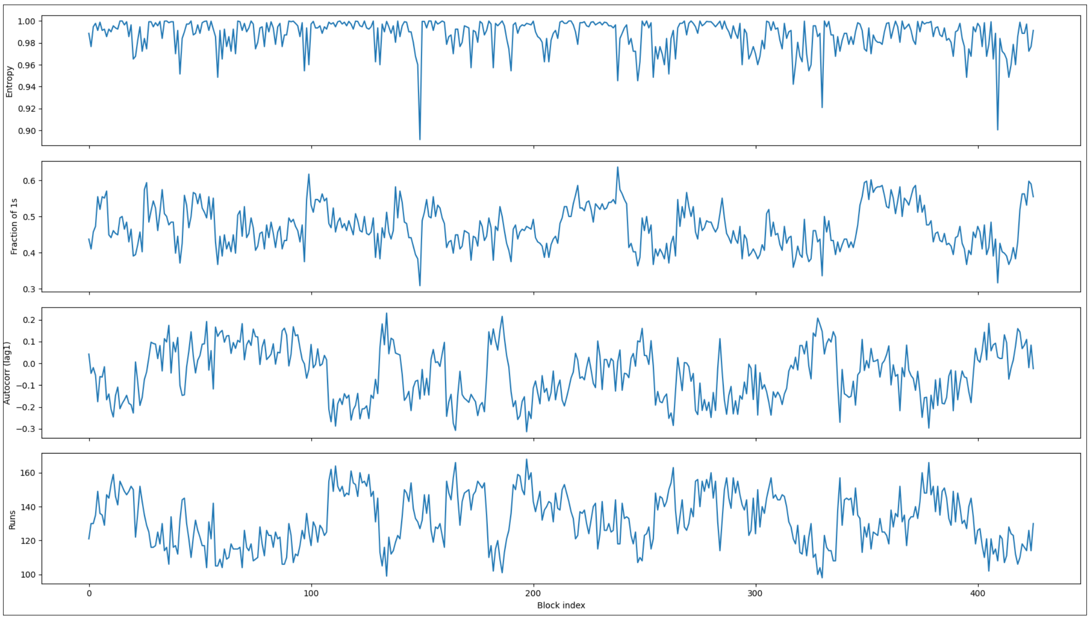

Figure 5: Readings of various metrics in 256-bit number.

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,

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

, 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

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

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.

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

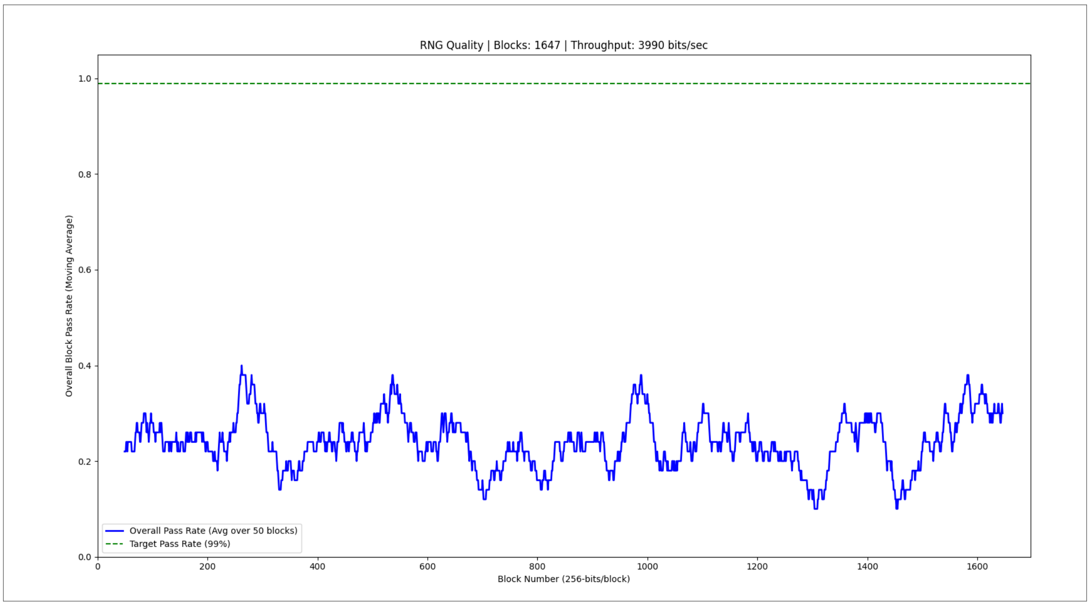

Figure 6: Results from our test suite

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.

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 exibits 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 $x_0, \ldots, x_{n-1}$, 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:

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

   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 $T \subseteq \{0, \ldots, m-1\}$ 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 $2^8 - 1$.

   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 Network Design

A possible setup:

- Each microcontroller is connected to a local router.
- A dedicated subnet for sensor nodes (e.g., 192.168.10.0/24).
- Central analysis server on a separate subnet.

4.5 Subnetting and Addressing

We show how to subnet the network efficiently:

- Example: dividing a /24 into four /26 subnets.
- Assigning IP ranges for sensors, analysis nodes, and administration.

4.6 Data Security and Transmission Integrity

We briefly discuss:

- Packet integrity verification (CRC or checksum).
- Optional encryption for data in transit.
- Synchronization and time-stamping for accurate sampling.

5 Conclusion

We summarize:Fundamentals of Precision ADC Noise Analysis

- Theoretical feasibility of analog-sensor-based TRNGs.
- Experimental results and limitations.
- Potential for distributed entropy networks.

5.1 Future Work

- Hardware whitening circuits and amplification.
- FPGA or ASIC implementations.
- Scaling to larger networks and entropy pooling.

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`.