ECE 4180 - True Random Number Generator

Uday Goyat & Ethan Su

April 26, 2025

Abstract

This paper presents the implementation of a hardware-based True Random Number Generator (TRNG) using small-signal electronics. We discuss the design principles, entropy sources, randomness validation, and integration with a software API for number retrieval.

1 Introduction

Random number generators (RNGs) play a critical role in cryptography, simulations, and statistical modeling. While most computers use pseudorandom number generators (PRNGs), which are deterministic, True Random Number Generators (TRNGs) leverage physical processes to generate non-deterministic outputs. This paper explores our approach to implementing a TRNG using small-signal electronics.

The motivation of this lab is to provide a superior free API accessible through the internet for True RNGS (TRGNS) compared to the best free APIs available currently. As of right now, the best TRNG API on the web is available on https://www.random.org/clients/http/api/

There are two modes Manual and USB API Mode in our model. Within the manual mode, two Rotatory Pulse Generators coupled with a push button (for submission) can be used to generate any number between a minimum and maximum value, which is displayed on an LCD screen.

There is also a USB mode, which communicates the random numbers to a host device for a more developer-styled generation.

1.1 Throughput Analysis

We will utilize a USB connection with a 115,000 baud rate between our MBED LPC1768 and a machine hosting our server.

The **UART-based generator** operates at a baud rate of 115000 bits/second , where each 32 bit integer requires 10 bit per byte for transmission. The effective throughput is calculated as follows:

$$\frac{115,000 \text{ bits/sec}}{10 \text{ bits/byte}} = 11,500 \text{ bytes/sec} \tag{1}$$

$$\frac{11,500 \text{ bytes/sec}}{4 \text{ bytes/int}} = 2,875 \text{ ints/sec} \tag{2}$$

In contrast, **RANDOM.ORG's quota system** allows for 1,000,000 bit/day, which translates to:

$$\frac{1,000,000}{32} = 31,250 \text{ ints/day} \tag{3}$$

Even with the 200,000 bit daily top-up, the quota is exhausted within seconds at high request rates, requiring users to either wait for replenishment or purchase additional quota.

1.2 Sustained Performance Comparison

Table 1 summarizes the performance of both systems.

System	Max Ints/sec	Sustained Usage
115,000 baud rate	2,875	Yes
RANDOM.ORG	31,250 (burst)	No

Table 1: Comparison of UART-based and API-based random number generation

While RANDOM.ORG offers a **higher peak rate**, it is **limited by a strict quota** and requires **frequent backoffs**. In contrast, a **local UART generator provides a sustainable, continuous stream of random numbers** without external limitations.

2 Design and Methodology

Our TRNG leverages six independent analog noise sources connected to the MBED LPC1768 microcontroller. These noise sources originate from avalanche breakdown effects, ensuring non-deterministic entropy.

To stabilize the input, we implement a moving average filter on each analog signal. Each signal maintains a *lookback window* of recent samples, over which the mean is computed. We consider a signal to be **converged** once its distribution of measurements shows near-equal probabilities of values lying above or below the moving average, within a tolerance threshold of 1%.

Once convergence is achieved across all analog channels, random bits are extracted by comparing live analog readings against their respective moving averages. A positive deviation encodes a logical '1', and a negative deviation encodes a logical '0'. These individual bits are then concatenated to form a subset of the final random number.

Random number generation proceeds by accumulating these subsets iteratively, ensuring that sufficient entropy is gathered before returning a final output. This staged construction minimizes bias and maximizes uniform randomness quality.

The entire system is designed to be lightweight, non-blocking, and scalable to additional analog sources if needed.

3 Entropy Source

This TRNG leverages Zener diodes in avalanche breakdown, providing a robust source of entropy by exploiting the inherently unpredictable quantum noise generated during the cascaded ionization acceleration and collision upon reverse conduction of the Zener semiconductor PN junction.

This quantum noise, characterized by its randomness at the physical level, serves as a strong foundation for generating truly random signals. To further enhance entropy, the TRNG incorporates natural ambient electromagnetic interference (EMI), which introduces additional randomness from environmental electronic noise sources.

Temperature variations also play a role, subtly influencing semiconductor behavior and adding unpredictability to the noise profile.

Additionally, ADC jitter—minute, non-deterministic variations in sampling intervals—can be harnessed to inject further randomness during digitization.

By combining these diverse and independent entropy sources—quantum noise from avalanche breakdown, ambient EMI, thermal fluctuations, and ADC jitter—the TRNG achieves a high-quality, unpredictable output suitable for cryptographic applications and other use cases requiring strong randomness.

4 USB UART API Communication

To provide an accessible True Random Number Generator (TRNG) service, we implemented a USB UART-based API that allows external programs to request random integers directly from the MBED LPC1768 microcontroller.

The embedded C++ code defines a `USB_API_Comm()` function, which listens over the UART interface for incoming commands. A request consists of two integers, `num,max_val`, indicating the number of random integers desired and the maximum bound for each number. Upon receiving and validating the request, the microcontroller generates the required random numbers using hardware entropy, ensuring each output falls within the specified range via rejection sampling.

Each response from the MBED is sent line-by-line in the format:

```
index,value
```

Once all requested random numbers are transmitted, the MBED signals completion with a `DONE` message.

On the client side, we implemented a Python interface leveraging the `pyserial` library. The `MBEDInterface` class wraps the serial communication, handling message formatting, response parsing, and optional base conversion (binary, octal, decimal, hexadecimal). Key validations, such as ensuring the requested number does not exceed microcontroller constraints and that the specified range is within 64-bit limits, are also performed by the client.

This lightweight USB UART API enables external systems to access high-throughput true random numbers without the overhead of networking stacks or external dependencies, maintaining minimal latency and simple interoperability across platforms.

5 Summary

What does the project do?

- The project acts as a general-purpose True Random Number Generator, using small signal electronics as a source of entropy for randomness.

What is the purpose of each peripheral?

- **5V Adapter:** Provides power to entire device
- **Small Signal Electronics:** The hardware source of entropy which leverages the quantum effects of Zener diodes in avalanche breakdown operation (one for each analogIn)
- **HS-422 Servo:** Rotates the small signal electronics to randomize ambient electromagnetic interference, further ensuring randomness
- **Max RPG Encoder:** Hardware UI for user to select max value (press to traverse the digits and dial to set each digit)
- **Min RPG Encoder:** Hardware UI for user to select min value (press to traverse the digits and dial to set each digit)
- **Button:** Hardware UI for user to generate the random number given the set min-max values
- **μLCD144G2:** Hardware UI for user to visualize device mode, calibration status, min-max values, and number generation
- **Host Device:** User device connected to mbed for more versatile generation features for development

Were there any issues with the project?

- Limited number of analogIn pins meant only 6 samples/bits per cycle. For our goal of ranges up to 64-bit integers, this hardware limitation requires 11 cycles of latency ((64 // 6) + 1 = 11). If we were to have much more hardware (64 inputs), we could have instantaneous generation at our max range.
- mbed ADC speed and resolution were limited for the high frequency noise. Ideally, would have several hundred MHz sampling.
- mbed stack size limited lookback window for computing moving averages for each channel

How is the project different from similar ones?

- Refer to 1.1 and 1.2

What else would you add?

- More complex ADCs for greater resolution (small LSB), faster sampling, and more channels.
- ADC gain stage to match peak-to-peak changes in inputs to ADC V_{ref}
- More memory for a longer lookback window for computing moving averages for each channel

6 Conclusion

In conclusion, this hardware-based True Random Number Generator (TRNG) demonstrates the effectiveness of leveraging physical phenomena to produce high-quality randomness. By harnessing inherent unpredictabilities such as electronic noise, quantum effects, and environmental variations, the TRNG provides a reliable and robust source of entropy that cannot be replicated by deterministic algorithms.

This makes it particularly well-suited for applications requiring strong security, such as cryptography, secure communications, and stochastic simulations.

The design highlights the advantages of hardware-based randomness over pseudo-random methods, ensuring true unpredictability and resilience against potential attacks. Overall, this TRNG showcases a practical and efficient solution for generating truly random numbers, reinforcing the critical role of hardware entropy sources in modern secure systems.