OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Analysis of Secure Random Number Generation including Implementation and Testing of a FPGA True Random Number Generator

A Thesis
Presented to the Faculty of Computer Science and Mathematics
University of Applied Sciences Regensburg
Study Programme:
Computer Engineering

Bachelor Thesis

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science (B.Sc.)

Presented by:	Reilly Ertman
Student Number:	
Primary Supervising Professor:	
Secondary Supervising Professor:	
Submission Date:	

Abstract

This bachelor thesis defines, constructs, tests, interprets and discusses the quality of deterministic, physical and quantum random number generators and their entropy sources as the engine for random number generation, including their cryptographic weaknesses in the context of the 21st century. After the mathematical axioms of an ideal random generator are discussed, two trusted and widely-used generators are examined closely: the (1) Linux kernel entropy pool, which uses user IO interrupts and driver noise as entropy and (2) the Quantis PCIe, which utilises quantum mechanics to drive random number generation.

At the heart of this thesis, a cryptographically secure random number generator is implemented in VHDL-93 for a Xilinx FPGA mounted on a Spartan-3E development board. By chaining an odd number of inverters together, a ring oscillator is created. When several such oscillating rings of differing length exist in close proximity, a sufficiently chaotic phase noise results. Via a UART connection, the FPGA's output bitstream is captured and tested. In addition, output from Linux's kernel entropy pool and the Quantis PCIe quantum generator is likewise captured for testing.

A statistical null hypothesis is proposed, that all captured bitstreams are random. Using the National Institute of Standards and Technology's publication 800-22 as a guide for rigorous and adequate quality control, all bitstream output files are then tested for randomness. The results of the FPGA's ring oscillators show that the FPGA's output is as unpredictable as both Linux and quantum generators. However, the ring oscillators on the FPGA lack even basic safeguards against side channel attacks, rendering the device cryptographically insecure.

While quantum random number generators are not impervious to attacks, they appear to be less at risk. However, this topic remains largely understudied as the technology of quantum generators is rather new. The prevailing opinion within cybersecurity in the post-quantum era is that a random number generator's security lies in algorithmic complexity and manufacturer's safeguards preventing those algorithms from leaking.

Contents

List of Figures

List of Tables

I List of Abbreviations

FPGA	Field Programmable Gate Array
RNG	Random Number Generator
TRNG	True Random Number Generator
UART	Universal Asynchronous Receiver Transmitter
IO	Input/Output
PRNG	Pseudo Random Number Generator
QRNG	Quantum Random Number Generator
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
NIST	National Institute of Standards and Technology
CSRNG	Cryptographically secure random number generator
LSFR	Linear Shift Feedback Register
NRNG	Non-Deterministic Random Number Generator
DRNG	Deterministic Random Number Generator
RO	Ring Oscillator
MSB	Most Significant Bit
LSB	Least Significant Bit
AES	Advanced Encryption Standard
DES	Data Encryption Standard
HTE	Health Testing Environment
TX	Transmission
SCA	Side Channel Attack

1 Introduction

1.1 Motivation

The terms integrity, confidentiality, authenticity and privacy are key subjects within cybersecurity. When sending an email, browsing the internet, withdrawing money at an ATM, or any instance where information is digitised and travels through the web, there exists a fundamental risk to that data. The sent data may be leaked thus violating your privacy, a man-in-the-middle may sniff an email, thus violating the sender's confidentiality or the data may even be modified out-right, violating its integrity and authenticity.

Encryption is the solution to these problems. If done properly, encryption ensures that a message's contents and both sender/receiver's address have not been modified. At the heart of the engine driving encryption is a random number, which cannot be guessed. Here are some real-world use-cases of a Random Number Generator (RNG):

- Secret Key Generation: If viewed, an encrypted message looks like a nonsensical string of garbage letters/numbers. However, if the secret key is applied to that message, it is magically decrypted, and the true plain-text message can be read. Random values are used to create these secure keys for symmetrical ciphers via the Advanced Encryption Standard (AES) and Data Encryption Standard (DES), as well as for the Diffie-Hellman key exchange and for asymmetric key generation with RSA keys. [Bak22]
- Nonce Generation, used in challenge-response protocols, digital signatures and much more.
- Random Vector Initialisation: When a system resets, all internal vectors are reset to a value. If the reset value for vectors is always the same, the system is vulnerable to attacks. [Oh+23]
- Blinding: A common countermeasure to protect against timing attacks is to use random numbers to obfuscate data and keys.[Cha+10]

A cryptographically weak or faulty RNG jeopardises a message's integrity, and with so many individual parts, if one component is weak, the entire RNG will perform poorly: without countermeasures against a Side Channel Attack (SCA) or if the generator relies on a biased entropy source, the machine's output will therefore not be reliably random, putting the integrity of an otherwise robust encryption and authentication framework at risk. Therefore the need for an ideal RNG is essential within cybersecurity. [MM09] [Ruk+10] This ideal machine, that guarantees a message's integrity, is called a CSRNG.

Not all RNG are created equal however. They can be split into two categories: Deterministic Random Number Generator (DRNG) and Non-Deterministic Random Number Generator (NRNG). The former generates random values algorithmically and therefore deterministically, while the latter via observing unpredictable physical or quantum phenomenon. A NRNG that records physical processes within a noise source is a True Random Number Generator (TRNG), while a Quantum Random Number Generator (QRNG) observes on a quantum level. By recording these seemingly random events, *entropy* is captured. Faced with the task of random value generation, a significant hurdle arises: If computers are inherently deterministic and typically incapable of "thinking outside the box", how does a deterministic machine create chaos?

Note, this thesis uses the verbs capture, record, extract, generate and create for entropy interchangeably. Also, randomness, entropy, unpredictability, uncertainty and chaos are used interchangeably.

Also note that the topic of random number generation is extremely extensive. This thesis is meant as an introductory overview of the topic's core concepts.

1.2 Thesis Roadmap

This thesis explores various methods to record, manage, output and improve entropy for cryptographic purposes. The roadmap for this thesis is as follows: Chapter 2 begins with a philosophical exploration into the term *entropy* and eventually details the many essential moving parts of a working cryptographically secure random number generator. Chapter 3 examines two trusted and commonly used CSRNGs in depth and analyses their methodology for producing, managing and outputting random bits: Linux's native kernel RNG and the QRNG Quantis PCIe. The capstone of this thesis is in the fourth chapter, where a hardware TRNG is implemented on a Xilinx FPGA in Very High-Speed Integrated Circuit Hardware Description Language (VHDL). With just a few flip-flops and inverters, a fully functioning random generation machine is created. The next chapter prepares the reader to empirically answer the question whether a RNG is "good" or not, by introducing necessary mathematical axioms, terminology and tests. The test suite used in this thesis comes from the National Institute of Standards and Technology (NIST) and is viewed as the cutting-edge for statistical pattern recognition. In chapter 6, random output from 4 different CSRNGs are captured and tested for validation: (1) the Quantis PCIe, (2) the FPGA's TRNG and two Linux methods (3) /dev/urandom and (4) /dev/random. The results of the test are presented and interpreted.

For the final chapter, I examine the resilience of TRNGs and QRNGs against bad actors and so-called Side Channel Attacks. The pros and cons of either device are presented, and the evidence clearly favours one over the other as the optimal CSRNG.

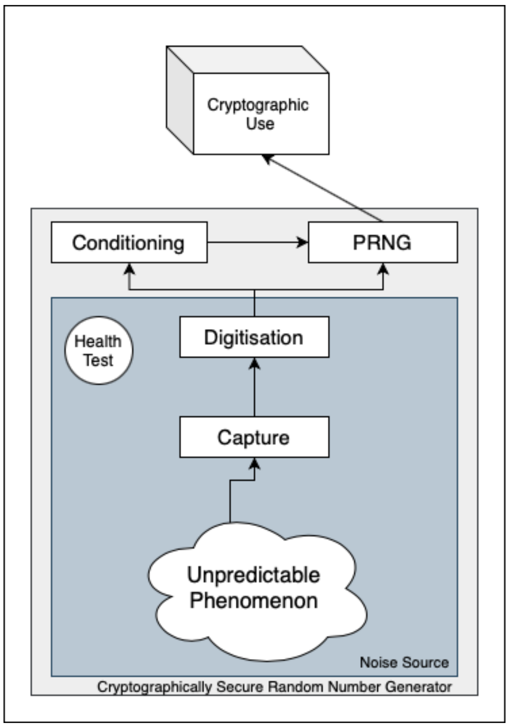


Figure 1: General Structure of a CSRNG

2 Building a RNG from the Ground Up

The goal of this thesis is to examine, test, show and discuss what makes a CSRNG effective. The term CSRNG is a catch-all term for any machine that guarantees its output is **absolutely unpredictable**, even against targeted attacks. [Nak18] These machines are realised as a mixture of software and hardware. They represent the gold standard for random value generation, as the values they generate are trusted and used as the bedrock for cryptographically secure encryption. No two CSRNG are identical; however, they generally follow the same structure. Figure 1 provides an overview for a generic CSRNG.

First and foremost, a CSRNG is a physical object that lives on a micro-controller, which observers unpredictable phenomenon from the physical world. They observe so-called "random" events within a noise source in order to extract entropy (more on entropy sources in chapter 2.4). The observation may be quantum, electrical, kinetic, seismic, barometric, radioactive, chemical in nature, etc. After recording the event either periodically or a-periodically, the resulting entropy is extracted and encoded into binary. A robust Health Testing Environment (HTE) is vital for start-up and run-time health checks to ensure proper entropy levels. Assuming healthy levels of entropy, the digitised information is then sent to the conditioner also called the entropy pool, where the entropy is managed. The conditioner has a number of important functions: It manages the internal state of the operating logic [RGX11], de-biases recorded events to further improve entropy (via for example a Neumann de-biaser), provides a container for the entropic values and also outputs a seed for the PRNG[Prn]. Finally, the conditioned information passes through a finite state machine or PRNG to stretch entropy, thereby improving data throughput.

Randomness is often conceptualized as a black box, where entropy goes in and random bits magically come out. This chapter aims to de-mystify the inner workings of a randomness generator by examining the many moving parts necessary for a robust CSRNG: starting from the philosophic understanding of entropy, to example entropy sources, and finally to how a CSRNG is realised in hardware and software.

2.1 Entropy as the Subject of a Random Number Generator

Entropy is a key concept in informatics, physics and chemistry - with use cases even in biology, economics, cosmology and beyond. [Mar17] Its definition differs depending on the scientific area. A valuable starting point to understanding this concept is through the physical science of thermal dynamics, where entropy was first defined by Rudolf Clausius in 1848. The second law of thermal dynamics states that an isolated system will always favour a state of equilibrium. During a system's move towards this equilibrium, energy is expended and this energy expenditure is irreversible. The change in an isolated system as it moves towards equilibrium is called **Entropy** (ΔS) [Sco03] and is expressed by the following integrated quotient, where an infinitesimally small and irreversible transfer of energy δQ_{rev} is divided by the absolute temperature of the system in Kelvin T, [Bla]:

$$\Delta S = \int \frac{\delta Q_{\text{rev}}}{T}$$

Simply put, entropy is the measure of the disorder, randomness, chaos or uncertainly of a system. [HS23] The entropy of an ordered system is low, while a disorderly system exhibits high entropy. [Weh78] The process of undoing entropy requires energy.

Consider figure 2, where a solid object exists in a heterogeneous system. As the object is heated, it turns into a liquid. Its individual molecules are now less rigidly structured. When more heat is added, the substance becomes gaseous and entropy increases further, as its molecules are now free-floating within the system. Disorder has thereby been increased. Even if the homogeneous system of the right image in figure 2 were frozen again, its configuration of molecule would be entirely different. Restoring this configuration would require energy. [Hel]

Understanding thermodynamic entropy, where disorder tends to increase over time in isolated systems, can provide an intuitive basis for understanding how, in information theory, higher entropy corresponds to greater unpredictability or lack of information.

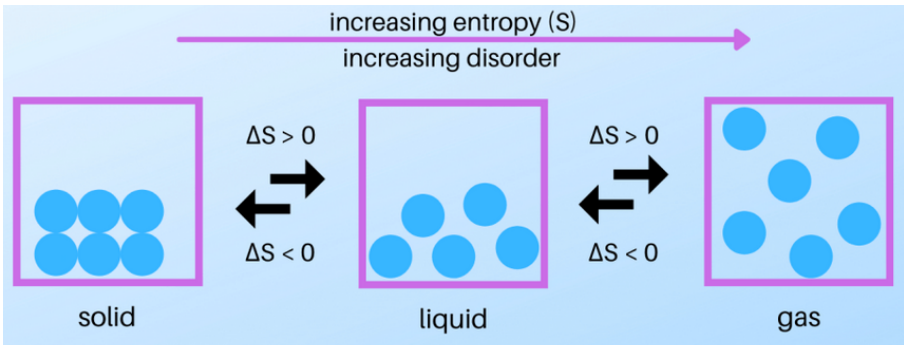


Figure 2: Entropy increases as system reaches equilibrium [Hel]

2.2 Understanding Shannon Entropy

Moving towards a theory of entropy within the context of information theory to use as the basis for an RNG, the exploration into probability starts with the premise that there exists an underlying force of uncertainty that permeates all things. This force drives a sequence of events, whose outcome is independently probabilistic and thus unpredictable.[Bel20] [CL16] Information theory attempts to describe the amount of unpredictability or deviation of an observed result from what was expected - via the term *surprise*. This concept *quantifies the information gained* from a certain outcome. More importantly, surprise also measures the evidence against a null hypothesis (more on null hypothesis and its use in statistical test in chapter 5.2), where P is the probability of seeing event X_i [Ell21]:

$$Surprise = \log_2(1/P(x_i)) = -\log_2(P(X_i))$$

Surprise is high when the probability of seeing a particular event is low and low when probability is high. Log base 2 is used in information theory, as a RNG outputs one of two values: 0s or 1s. Note that surprise is measured in bits.

Consider the theoretical example of a fair eight-sided die, where each outcome has an equal probability of 1/8, where P denotes the probability to see outcome X_i:

$$P(X_i) = \begin{cases} \frac{1}{8} & \text{if } x = 1, 2, 3, 4, 5, 6, 7, \text{ or } 8 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} Surprise &= -log_2(P(X_i)) = -log_2(1/8) \\ &= -(log_2(1) - log_2(8)) = -(0 - log_2(8)) \\ &= 3 \text{ bits of information gained from outcome} \end{aligned}$$

An 8-sided die has 8 possible outcomes, which can be encoded into 3 bits. Therefore, 3 bits of information are gained from each roll of the die. In the case of a fair 16-sided die, surprise increases per roll. With 16 different outcomes, the surprise is 4 bits per roll, as 16 values can be encoded into 4 bits.

$$\begin{aligned}
Surprise &= -log_2(P(X_i)) = -log_2(1/16) \\
&= -(log_2(1) - log_2(16)) = -(0 - log_2(16)) \\
&= 4 \text{ bits of information gained from outcome}
\end{aligned}$$

With a solid understanding of surprise, Shannon's famous equation for entropy from 1948 measures the uncertainty of a probability or the expected surprise from a distribution, where $H(X)$ denotes the Shannon Entropy of the random variable X. Log base 2 describes the binary option of either 1 or 0, and $P(X_i)$ is the likelihood of seeing outcome X_i:

$$Shannon\ Entropy = E(Surprise) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2(P(x_i)) = \sum_{i=1}^{n} \log_2(1/P(x_i)) \cdot P(x_i)$$

2.2.1 Calculating Shannon Entropy from a Binary Distribution

Consider a distribution with two uneven outcomes. Let A and B have the following probabilities, where for each run, the probability of seeing outcome A is 25% and the probability of outcome B is 75%:

$$P(A) = 0.25\ ;\ P(B) = 0.75$$

The Shannon entropy or average surprise of each outcome is calculated with the formula $Surprise = \log_2(1/P(x_i))$:

$$\begin{aligned}
Surprise(A) &= -log_2(0.25) = 2 \text{ bits} && (1) \\
Surprise(B) &= -log_2(0.75) \approx 0.311 \text{ bits} && (2) \\
Shannon\ Entropy &= E(Surprise) = (2 + 0.311)/2 && (3) \\
Shannon\ Entropy &= E(Surprise) \approx 0.811 \text{ bits} && (4)
\end{aligned}$$

This distribution's entropy per outcome is approximately 0.811 bits. This value represents the average amount of surprise we would expect from each experiment run. As a central goal of a RNG is to extract maximum entropy, let us tweak the distribution to mimic the flipping of an ideal coin, where either outcome is exactly 50% likely:

$$P(A) = 0.5 = P(B) = 0.5 = 50\%$$

The Shannon Entropy is calculated:

$$Surprise(A) = -log_2(0.50) = 1 \text{ bit} \tag{5}$$
$$Surprise(B) = -log_2(0.50) = 1 \text{ bit} \tag{6}$$
$$Shannon\ Entropy = E(Surprise) = (1+1)/2 \tag{7}$$
$$Shannon\ Entropy = E(Surprise) = 1 \text{ bit} \tag{8}$$

By tweaking the probabilities so that each outcome is equally likely, we have maximised the information gained or unpredictability. The goal for designing a RNG is thus finding an entropy source in the physical world, that provides an even distribution of outcomes like with an ideal fair coin.

2.3 Subjective and Objective Qualities of a "good" RNG

With a mathematical understanding of an ideal entropy source, let us review the statistical axioms necessary to realise an ideal source of chaos. As mentioned later in this thesis, there exist many test-suites that *objectively* test the quality of randomness. However, the *subjective* nature of randomness is often overlooked. A successful RNG must also pass this subjective test of unpredictability.

2.3.1 Unpredictability as a Subjective Quality

Random and pseudo-random numbers generated for cryptographic applications must be unpredictable, but how is unpredictability defined, and who defines it?

Informally, information theory describes unpredictability or chaos as the absence of a pattern or correlation. [Bri19] Said differently, uncertainty is an aversive state which may be resolved by identification.[MM98] Within an interdisciplinary branch of mathematics, *chaos theory* purports that within believed randomness of a chaotic system, patterns always exist. [Fou] They must simply be recognised. If every n-th bit in a bitstream is a logical 0 or if the pattern "0110" is observed at the beginning of every data packet, one would consider the data to be unpredictable. However, consider the following philosophical questions:

If a string of values is not observed, is it by default predictable, unpredictable, or neither? Or said differently, is every string of values unpredictable until software with enough statistical rigor finds a hidden pattern and therefore deems it unpredictable? According to the German Federal Office for Information Security, the needed sophistication of a RNG to generate values that appear unpredictable depends on the observer and his skill-set. [IS22] The more intelligent the observer, the more likely the string is to contain a pattern and therefore be predictable.

The first historical method of encryption offers a good example of the subjective nature of randomness: the Caesar cipher. The Roman emperor Caesar was faced with a problem:

He wished to send an important message to a friend; however, he cannot guarantee that the postman will not read the letter nor that the letter itself be stolen. His solution was to encrypt the message with a shared symmetrical secret key. Both parties agreed beforehand on a secret key of $K = 4$ for example. Caesar creates a message in plaintext as shown below. He then shifts the letter's value K amount to generate the ciphertext. The ciphertext is sent through the mail, and Caesar no longer has to worry that any unauthorised person will read its contents. Once his friend receives the letter, he undoes the symmetrical key encryption to decrypt the plaintext from the ciphertext.

$$\begin{aligned}&\text{Plain text:} \quad \text{a n d y o u t o o b r u t u s}\\ &\text{Cipher text: E R H C S Y X S S F V Y X Y W}\\ &\text{with key } K = 4\end{aligned} \tag{9}$$

The complexity of the Caesar cipher is astonishingly simple. The first references of this cipher date back to 100 BC, while the latest surviving records date to the 9th century. [Hol17] The world's first recorded encryption system lasted 900 years, yet in the 21st century, the Caesar cipher has no integrity, as it would be cracked in a few microseconds with any modern CPU. [Sin00]

The integrity of a cryptographic encryption system is therefore a dynamic metric, which increases or diminishes depending on the observer's ability to detect unpredictability. [CL16] While the gold standards of RSA or AES may be secure today, their algorithmically created digest may be deemed unpredictable and therefore insecure with future advances in technology.

2.3.2 Gaussian Distribution as an Objective Quality

The second key quality of a "good" RNG is a uniform Gaussian distribution. [KGW19] As discussed earlier, a truly random sequence of bits should mimic the outcomes of tossing an ideal coin, where either result is exactly 50%. An ideal distribution is described via Random variable $\mathrm{X_1}$, $\mathrm{X_2}$...$\mathrm{X_n}$ below:

$$X_i = \begin{cases} 0 & \text{with probability } \frac{1}{2} \\ 1 & \text{with probability } \frac{1}{2} \end{cases}$$
$$\text{with } X = \{0, 1\}$$

The probability that either outcome occurs 50% of the time, given an ideal coin:

$$P(X = 1 \mid \text{"Ideal Coin" }) = 0.5 = P(X = 0 \mid \text{"Ideal Coin" })$$

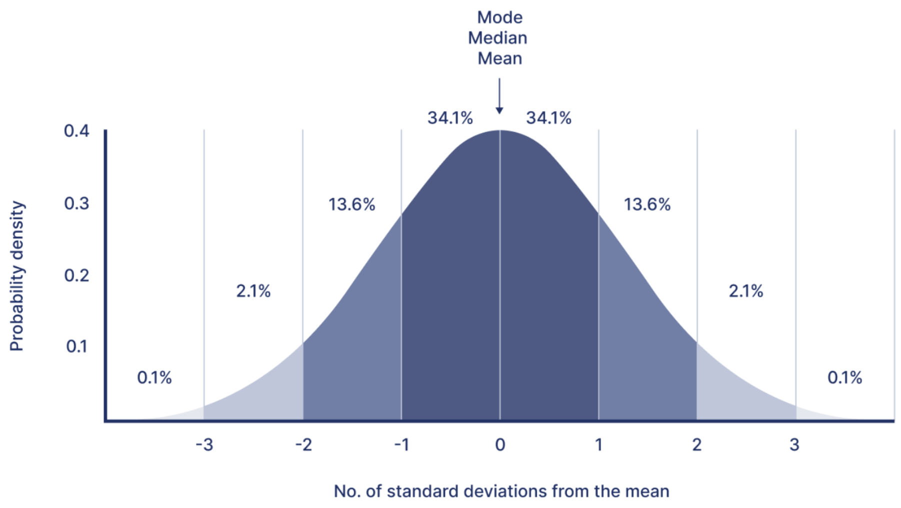


Figure 3: An ideal Gaussian Distribution [Bha]

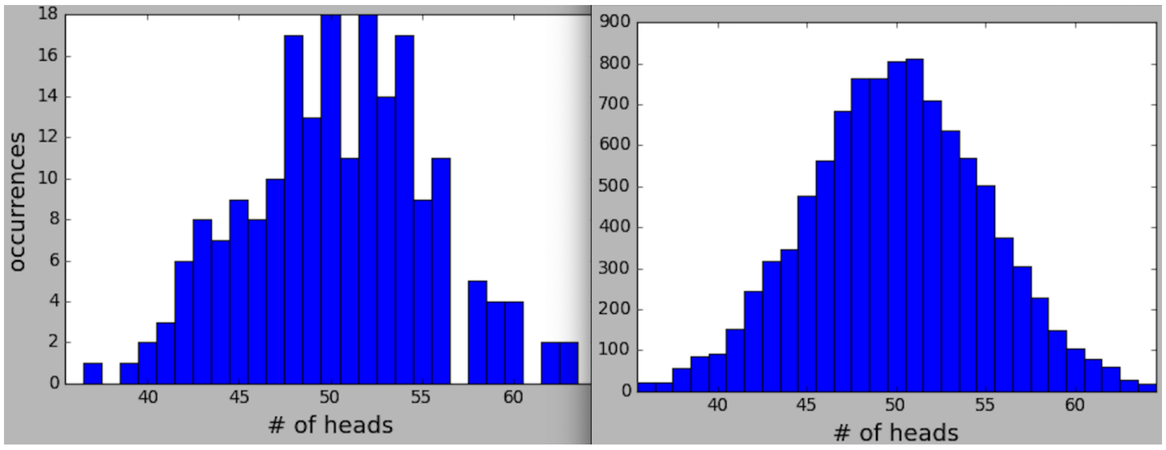


Figure 4: Increasing number of experiment runs

The bell-shaped curve in figure 3 denotes an ideal Gaussian distribution. Where 68% percent of the values occur within one standard deviation from the mean. Inside two standard deviations from the mean, 96% of the values are located, while extreme values over three standard deviations from the mean occur just 0.2% of the time. This curve describes the distribution of random variables, which an ideal RNG should create in the aggregate. [MR14] [WHG09] Given an experiment where we flip an ideal coin 100 times and repeat the experiment x times, we might expect the results to immediately and always adhere to the Gaussian distribution. However, as seen from figure 4, this is not the case.

With the help of Matlab, an ideal coin was flipped 100 times. After each experiment, the number of heads was recorded on the x-axis. The y-axis depicts the number of times a certain result was observed. In the graph on the left, the experiment was carried out 200 times. On the right side, the same exact experiment was conducted 10,000 times. The graph on the left is irregular and far away from a symmetrical Gaussian distribution. For example, the bin of the left graph in figure 4, where # of heads = 57, there are no occurrences. The bin where # of heads = 51 is substantially smaller than for the bin # of heads = 50. If the number of experiment runs is not increased, one might believe the gambler's fallacy, where occurrences seem like dependent events, when they are in fact independent. [Sta] In other words, the outcome of any prior toss does not influence subsequent ones. [XH14] After increasing the number of experiment runs to 10,000, we see the Gaussian distribution forming on the right side of figure 4.

This observation confirms the *central limit theory*, that for a given array of experiments X_1, X_2, $..X_n$ with expected values $E(X_1)$, $E(X_2)$, $..E(X_n)$, then the limit as n approaches infinity is indeed the Gaussian standard normal distribution. [MR14]

2.4 Entropy Sources and the Quality of Randomness

As mentioned in chapter 2.2, a qualified, un-biased entropy source drives a trusted and secure RNG. Said differently, the quality of entropy dictates the quality of randomness in the bit-stream. The lower the average entropy per bit, the higher the bias and the more uneven the Gaussian distribution. [Cao+22] Therefore choosing the right source for chaos is absolutely crucial. Common examples of entropy sources in modern RNGs include:

- Phase noise from ring oscillation, where entropy is extracted from the noise when signals actively propagate rings of inverters in close proximity with each other. This example is the focus of the next chapter, where ring oscillators on an FPGA are implemented and tested for statistical rigor.
- Shot noise, where entropy is extracted by counting the individual photos received with a sensor from a light pulse. [WHG09].
- Time stamps between observations of decaying radioactive particles. [Lib]
- Thermal noise, also known as Johnson-Nyquist noise, is the natural variation in voltage that occurs spontaneously in a transistor when in a state of thermodynamic equilibrium. These variations stem from the thermal energy of electric charges within the transistor. [AD13]

Note that all of these examples of chaos are non-deterministic, and can be divided into two categories: physical and quantum. Shot noise and decay time of radioactive particles rely on quantum mechanics, while both thermal and phase noise rely on physical mechanics to create entropy. In this subsection, we will discuss these two categories of entropy sources.

2.4.1 Physical Noise Sources

If the essence of entropy is the internal chaos of a system, then the physical world is a good place to start. [Cao+22] Physical random number generators or TRNGs make use of a wide range of entropy sources: metastability, thermal noise, ring oscillation, etc. This type of generator requires a capture mechanism, which samples the physical world within a noise source after an event or specific amount of time. After capturing the signal, it is digitised to binary with a simple ADC.

Note that entropy extracted from physical processes always features a bias, reducing the entropy per bit. Therefore, the use of a conditioner that de-biases entropy is vital to a well-functioning TRNG (more on this in chapter 2.5). [Cao+22]

2.4.2 Quantum Noise Sources

While TRNGs[1] observe physical phenomenon, which present inherent bias, QRNGs capture entropy observed through quantum processes, which are inherently unpredictable via the property of particle superposition. As sources of quantum entropy inherently lack bias, QRNGs offer a clear advantage over TRNGs in this regard. [RK20]

In classical physics, the state of a system is typically described by a set of parameters, and at any given time, the system is considered to exist in a single definite state. This deterministic view of the world means that if a complete set of parameters (for example position, velocity) is known of a system at a particular moment, a particle's behavior can be calculated or "simulated" at any future state via long-standing constants defined within physics, such as the speed of light in a vacuum $c = 3.00 * 10^8$, the gravitational constant $G = 6.674 * 10^-11$, the elementary charge $e = 1.602 * 10^-19$, to name a few examples. [Zho+17] The realm of physics provides a mathematical framework to describe, calculate and predict the behavior of physical particles.

The quantum world differs from the physical world in this regard, as a quantum particle may exist in two or more distinct states simultaneously. This multi-state condition is called superposition. A famous experiment, which vividly illustrates this concept is the double-slit experiment.

First performed by Thomas Young in 1801 to prove that light behaves indeed like a wave and not a particle, the double-slit experiment was later retrofitted to examine the behavior of electrons. The now famous experiment goes as such: As seen in figure 5, a source emits subatomic particles (like photons, electrons or neutrons) one at a time towards a barrier with two slits. The barrier with two slits functions as a filter for the single-particle source. The particles either hit the barrier and do not pass or they are aimed correctly to pass either slit 1 or slit 2. Behind the barrier, a detector d either observes and records where the particles land or it does not. [KH23]

When detector d attempts to observe whether the particle passed through slit 1 or 2, the interference pattern disappears and the particles behave more like individual particles. Thus

[1]Note that some papers define QRNGs as a TRNG. I do not. I define a TRNG as any generator that measures exclusively physical processes to extract entropy. QRNGs observe quantum processes.

the act of measuring forces the particle into a state, (as the electron must have passed through either slit 1 or slit 2). The particle thus looses its superposition and assumes a state. [Zho+17]

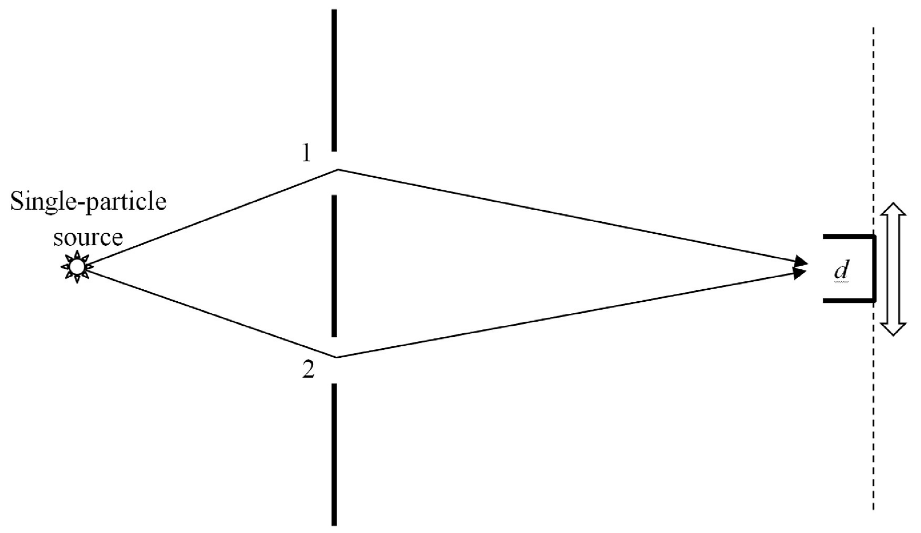


Figure 5: Thomas Young's double slit experiment [RK20]

When the particles are not observed, in other words, when there is no attempt to determine which of the two slits the particle passed through, the particle act like a wave. Meaning the particles behave as if they are in a superposition of states, simultaneously passing through both slits. [RK20]

The experiment provides profound insight into the nature of quantum particles and the important role that *observations* play in the quantum world. The principle of superposition leads to the concept that the future state of a quantum system cannot be precisely predicted or determined in advance, and the outcome can be expressed probabilistically via a Gaussian distribution. The inherent unpredictability of quantum processes is a fundamental aspect of nature, and it distinguishes quantum from physical entropy sources with a clear use case as an ideal bias-free engine for random number generation. [KH23]

2.4.3 Examples of Quantum Entropy Sources

There is theoretically no limit to the number of potential quantum entropy sources. Schemes of various quantum random sources have already been realised and demonstrated. One such generator extracts entropy by splitting a photon on a beam splitter, at a generation rate of 1 bit per 11 ns or 1 Mbit/s [Jen+00], or from the random distribution of timestamps between a photon detector recording individual photons [MXW05], or from counting the amount of photos received from a light pulse (also called shot-noise). [WHG09]. Another device generates entropy from the phase noise of lasers [Guo+10]. These are just a few examples of many possible sources to derive entropy via quantum mechanics.

2.5 Managing and Outputting Entropy

After examining viable entropy sources to drive a random generation machine, we will now turn our focus and see how a CSRNG manages and processes captured entropy internally. Using figure 1 as a roadmap for this section, we will first examine the (1) conditioner, which stores, counts and de-biases entropy, the (2) health test, which monitors and provides procedures to repair entropy and (3) the PRNG, which stretches entropy to increase output.

This section of the bachelor thesis describes what happens in a CSRNG after an entropic event has been recorded from a noise source. An ADC digitises the analog signal, and the now digital information is taken to the conditioner or global entropy pool. The entropy pool is typically a FIFO structure.

2.5.1 The Conditioner

Section 2.2. demonstrated that an entropy rate of 1 bit of information per bit is ideal, yet any bias in the noise source will quickly reduce this value. As all physical noise sources possess an inherent bias, the conditioner is a vital component within a CSRNG. The conditioner is a deterministic function which increases the entropy per bit to an ideal maximum of 1. [Tur+18] There exist various methods for achieving this. Later in this thesis, I implement a Von Neumann randomness extractor to artificially raise the entropy level per bit.

2.5.2 Health Test

Another vital function of a CSRNG is a robust Health Test Environment HTE. The amount of entropy in an entropy pool (entropy per bit) at any given point in time is dynamic and also quantifiable by calculating the average surprise of a distribution. This value should be constantly monitored to ensure system integrity. An RNG that does not monitor its health will very likely output non-random values. The American National Institute of Standards and Technology (NIST) even requires manufactures of RNGs used to encrypt sensitive governmental data to implement detection mechanisms that monitor this value. [Tur+18] This framework of checks to monitor the level of entropy in a system is referred to as a HTE while an instance of a test is a health test. Health tests are expected to raise alarm in three cases:

- when a significant decrease of average entropy per bit within the entropy pool is detected. This occurs when more values are output than are input.
- when the noise source fails, which occurs when the sensor fails, for example.
- when the RNG's hardware itself crashes or malfunctions. [Tur+18]

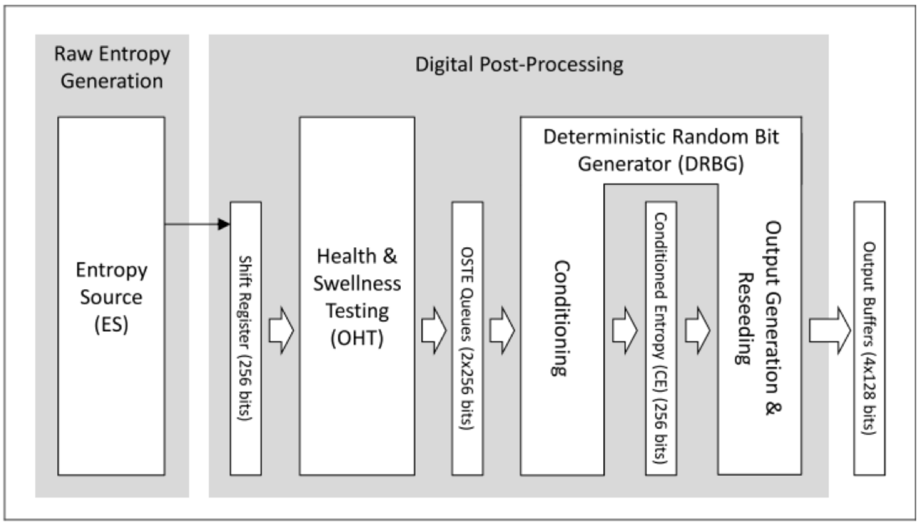

Raw Entropy Generation
Digital Post-Processing
Entropy Source (ES)
Shift Register (256 bits)
Health & Swellness Testing (OHT)
OSTE Queues (2x256 bits)
Deterministic Random Bit Generator (DRBG)
Conditioning
Conditioned Entropy (CE) (256 bits)
Output Generation & Reseeding
Output Buffers (4x128 bits)

Figure 6: Architecture of example HTE [AD13]

To fulfill these requirements, two categories of health tests are necessary: startup tests and continuous tests. [Ruk+10] The former occurs at startup before the RNG beings producing output. An example of a startup test is waiting several clock cycles for old values in the PRNG's linear feedback shift register (LSFR) to shift out before introducing new values. Linux systems offer a good example of continuous or run-time health tests: The entropy pool is assigned a counter to track entropy levels and blocks output until entropy levels raise above a certain threshold. More on this in chapter 3.1. [IS22]

In his paper, Design and Testing of Random Number Generators, Daniele Antonioli provides a concrete implementation of a run-time health test shown in figure 6. [AD13] After raw entropy from a noise source is collected into 256 bit chunks, they are then passed to the HTE for evaluation. Each 256-bit sample is considered healthy if the number of times each pattern appears, denoted with n, falls within certain bounds, as shown in the table below:

Bit Pattern	**Bounds per 256-bit sample**
0000	$1 < n_1 < 16$
0001	$1 < n_2 < 16$
0010	$1 < n_3 < 16$
0011	$1 < n_4 < 16$
0100	$1 < n_5 < 16$
0101	$1 < n_6 < 16$
0110	$1 < n_7 < 16$
0111	$1 < n_8 < 16$
etc..	etc..

Table 1: Example health bounds for 256-bit samples.

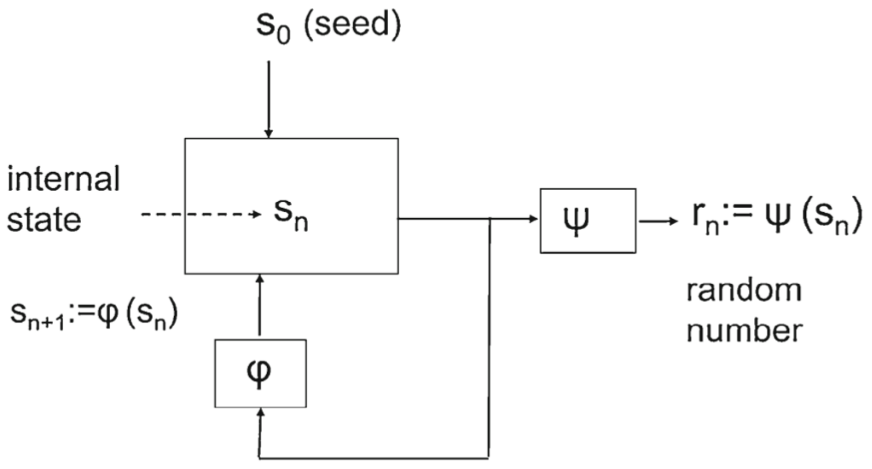


Figure 7: Structure of a PRNG [Prn]

Note these values serve as an example. The HTE unit tracks the most recent 256 samples. The data is considered to be healthy if all n fall within the allowed boundary. Note that the HTE unit is not intended as a measure of entropy. It offers a simple check whether the entropy is badly broken, by watching if a pattern infinitely repeats. These tests are typically simple and lack extensive rigor, as they are run-time tests implemented in digital logic, which are resource intensive, expensive and slow. [Tes20]

2.5.3 Pseudo-Random Number Generator

As seen in figure 1, the PRNG is the last module of a CSRNG. Its two purposes are to obfuscate the data and stretch entropy, thereby increasing bit through-put.

A PRNG or deterministic random number generator (DRNG) is purely *algorithmic*. For a given input X, it will always return output Y; PRNGs are usually realised as a LSFR, which is said to be cryptographically secure if and only if both seed and the PRNG's architecture are secret. As seen in figure 7, PRNGs take in seed $s0$ and some subset ϕ of output ψ as inputs while in the internal state s_n. Random number r_n is then computed from ϕ and $s0$. PRNGs offer many advantages. They are low cost, require little to no dedicated hardware, implementations can be done in software and they can increase through-put considerably. [AD13] However, there are disadvantages. For pure PRNGs, the output is completely determined by the seed and ϕ. If an attacker knows a seed and the device's internal architecture, he can calculate any future value, until a new seed in introduced. Therefore, the device's internal architecture must be kept secret. [Prn] Chapter 4.4 will go more in-depth on PRNGs.

3 Examples of Modern CSRNGs

In this chapter, we will take a closer look of two relevant and trusted CSRNGs on the market used for encryption: Linux's native RNG and the Quantis PCIe.

3.1 Linux Kernel Entropy Pool

The Linux operating system offers a cryptographically secure RNG, which is integrated directly within the kernel. While written entirely in software, it is not deterministic. As seen in figure 8, the Linux-RNG can be broken down into three sections. First entropy is captured, then the pool of sampled entries is managed/processed. Finally the random values appear in the bitstream after the PRNG algorithm.

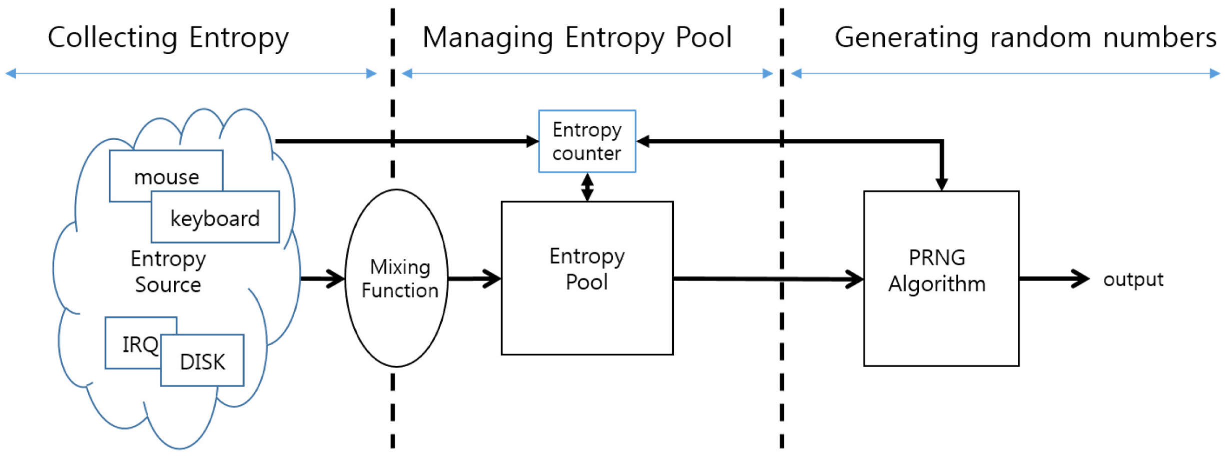


Figure 8: Architecture of Linux's RNG [YK18]

Linux systems provide a unique approach to collecting entropy. On any personal computer, the user inputs commands via keyboard, mouse, usd-stick, monitor, etc., to interface with the computer. According to the German Federal Office for Information Security, the Linux's kernel utilizes these Input/Output (IO) events as sources of noise to supply input to its entropy pool.[IS22] After an entropic event is recorded, the event time and value are injected into the entropy pool. In addition to listening for IO events, the Linux-RNG also periodically polls device drivers. [com]

3.1.1 /dev/urandom vs. /dev/random

Two methods exist for extracting entropy from the Linux-RNG: /dev/random and /dev/urandom. [Lin] The simple difference between these two methods is that /dev/urandom features a health test.

In figure 8, the Linux-RNG features an entropy counter. The value is incremented when a new entropic value is introduced and decrements when a value is output. Should this counter reach a critical threshold, /dev/urandom waits to output until this counter reaches a healthy value again. This is one example of many possible implementations of a run-time health test. [YK18] The other method dev/random does not monitor the entropy pool. If the pool is empty, i.e. no entropy is available, values are still output. This may be dangerous as ”random” bits will still be generated with no guarantee that these numbers are in fact random. After leaving the entropy pool, data then moves to the PRNG, where it will be stretched before appearing in the bitstream. [Mue] For this thesis, a simple shell script was written to create 4 MB of data, where both /dev/urandom and /dev/random functions are called. The results from the NIST test suite are found in chapter 6.

3.2 Quantis PCIe

Another example of an industry-relevant high-end CSRNG is the Quantis QRNG PCIe, which exploits elementary quantum optic processes, that are fundamentally probabilistic, to produce true randomness. The Quantis PCIe features a range of attractive qualities, including:

- consistent 4Mb/s random output
- both start-up and run-time health tests
- cryptographic certification as set forth by NIST 800-90B [Tur+18]
- uses-cases in financial transaction, scientific modeling, machine learning and encryption.

Most notably is its added resistance to attack or manipulation via a side channel, as thermal noise of the hardware itself contributes to less than 1% of range manipulation. As mentioned later in chapter 7, any CSRNG is subject to manipulation via its electrical components. [MM09] For example, the air temperature or phase noise of a circuit affects the speed/oscillation of signals, which in turn affect the numbers generated by the RNG. The Quantis PCIe is very resilient to being manipulated in these ways. [SE]

Figure 9 denotes how the Quantis PCIe generates random values via the intrinsic randomness of subatomic particles. A photon source emits photons towards a semi-transparent mirror. Behind the mirror is a sensor. If the photon reflects off the mirror (thus not registered by the sensor), a 0 is recorded. If the photon passes through the mirror, the sensor registers the photon, and a 1 is recorded. [idq]

The process of using the Quantis PCIe to generate statistical data for testing was simple. I installed the PCIe board into the computer and used Quantis’ native Windows 10 program to select the data size to generate.

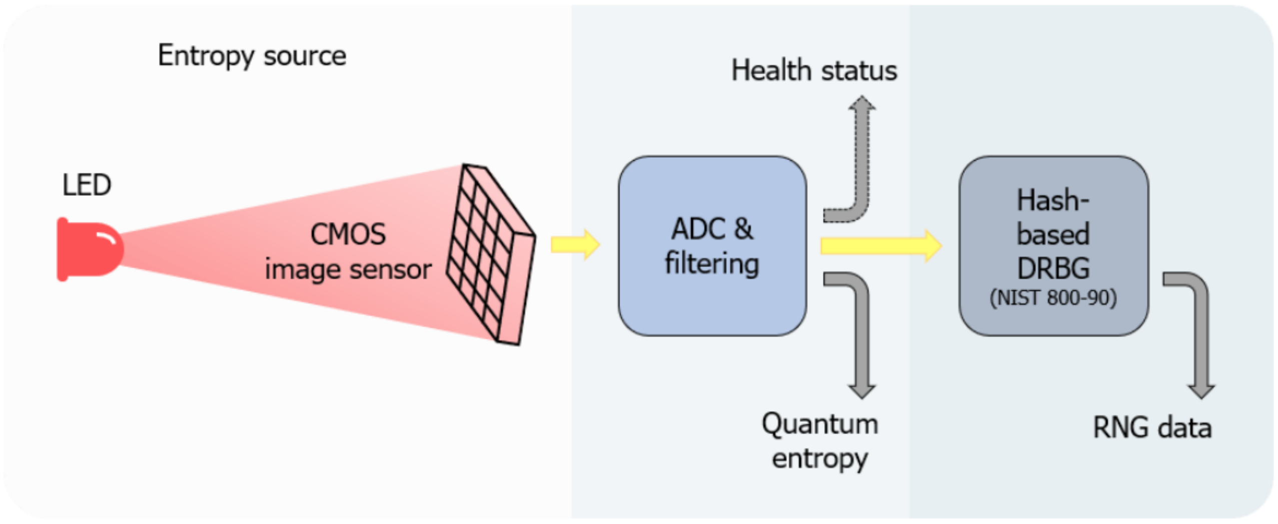


Figure 9: Entropy Source and Architecture of Quantis RNG [idq]

4 Designing and Implementing a CSRNG on an FPGA

With an understanding of entropy, its sources, probabilistic axioms and the building blocks to design an RNG, its time to roll up the sleeves and finally make one. A CSRNG using the phase noise from free-propagating ring oscillators in close proximity as the physical entropy source was synthesised on a Xilinx XC3S1600E FPGA on a Spartan-3E development board in VHDL-93. The TRNG.vhd VHDL module exists on a single clock domain and can be broken down into three subsections:

- Entropy generation via ring oscillation
- Neumann de-biaser to remove inherent physical entropy bias
- a PRNG via LSFR to improve output speed

4.1 Control Flow

Both de-biasing and LSFR processes are controlled via vld_in and vld_out signals. When a high vld_in signal is received for a single clock cycle, execution begins. Upon process completion, the vld_out signal is held high for a single clock cycle and then fed into the next process. The signal vld_in also plays a crucial role in initialising the TRNG. Before the TRNG begins generating a new value, all shift registers must be empty, to ensure a neutral reset state.

To ensure a clean reset-state, an incoming vld_in is sent through a reset shift register. Once the reset value passes through the reset shift register, enough clock cycles have passed to ensure all other shift registers have been cleared of values. This requirement constitutes a startup health check mentioned in chapter 2.5.2.

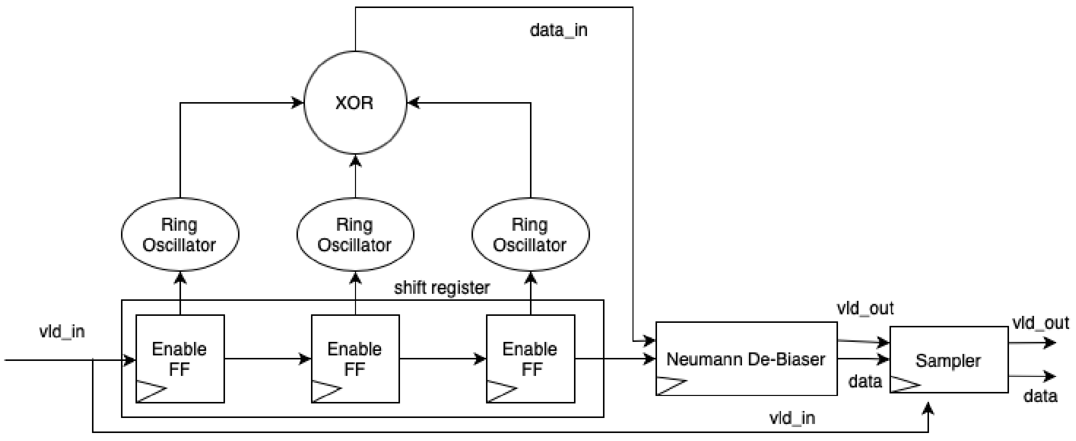


Figure 10: Architecture of FPGA RO CSRNG

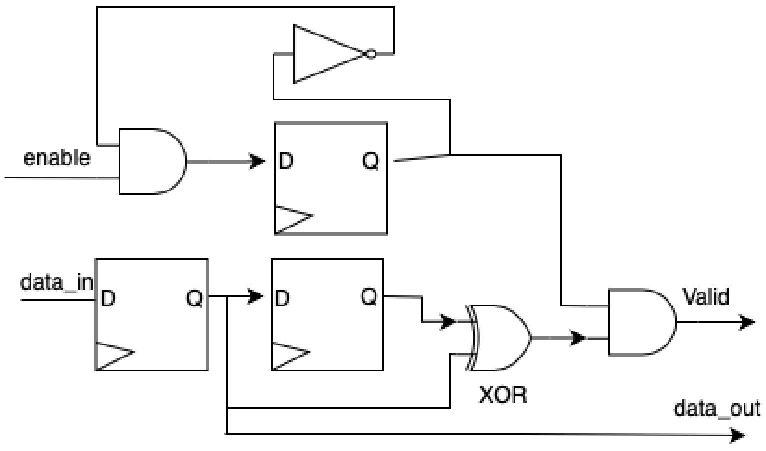


Figure 11: Von-Neumann Debiaser improves entropy [RYD20]

4.2 Entropy Generation

The TRNG.vhd module may be instantiated with any number of ROs, but each must have an odd number of inverters in sequence, where the last inverter connects to the first. Thus the signal changes polarity every time it propagates around the RO. This propagation frequency is determined by the circuit's thermal and shot noise of the board's digital-logic electrical components on the FPGA chip. [BFV10] In addition, when several ROs exist in close proximity, the signals propagating each RO interfere with each other. This interfering phase noise slightly increases or decreases the propagation frequency, thereby adding entropy. [Mur+19]

The state of the last inverter in the RO is captured on the rising-edge by the synchronous clk_in signal. As seen in figure 10, each RO passes through a single XOR gate before entering the conditioner. Once the vld_in exits the final RO enable flip-flop, the Neumann de-biaser process begins.

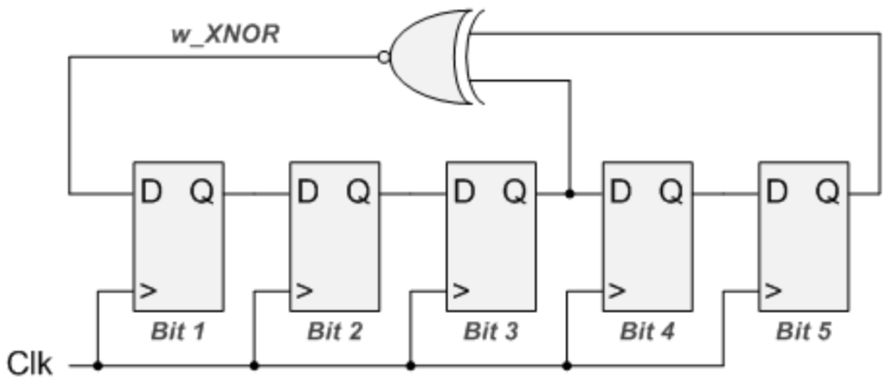


Figure 12: 5th Order LSFR

4.3 Conditioning with a Von-Neumann De-Biaser

A Von-Neumann bias-extractor or conditioner is a commonly-used technique to de-bias a bit sequence. This method repairs entropy from biased noise sources without knowing the exact bias value. [RYD20] At the heart of the process, every two incoming bits are examined for rising and falling edges via a XOR gate.

A rising edge (”01”) and a falling edge (”10”) both output ’1’. [Pam+18] Obviously, the de-biasing unit requires two data bits to make a decision, meaning it operates every other clock cycle. If no edge is detected (”00” or ”11”), the valid signal stays low, and the PRNG process does not begin. Once an edge is detected, a valid bit is sent to the PRNG to begin execution.

4.4 PRNG to Stretch Entropy

At the heart of the PRNG function is a LSFR. When a vld_in is received and the process begins, the CSRNG must run enough clock cycles to ensure all values have exited its shift registers, so that the process begins in a reset state. Within cryptography, a linear shift feedback register is a primitive type of PRNG, which is purely deterministic. If one has information about the PRNG’s internal state and specific implementation, any previous output can be reverse-engineered.

A LSFR features x number of single-bit storage elements, called flip-flops, which all reside in the same clock domain. The output of the first flip-flop in a chain is fed into the second, the output of the second flip-flop is fed into the third, etc. Finally, the XOR-ed output of the last flip-flop and another flip-flop is used as output for the first flip-flop. While figure 12 shows the output of the XOR gate being fed back into the chain, in the context of a CSRNG, this output would appear on the bitstream.

An advantage of the PRNG is that it can output data on every clock cycle. As they are deterministic, PRNGs have low entropy and are thus cryptographically insecure. Yet their processing speed is very high, with a speed of gigabits per second. TRNGs and QRNGs, on the other hand, are high in entropy and thus generally cryptographically secure yet have much slower output speeds. Therefore, PRNGs have a clear use-case. [Prn] Every 10 clock cycles or so, the TRNG injects a bit of entropy into the LSFR, and the LSFR will output data until entropy is low again. Then it will again receive an entropy injection. Thus, a CSRNG's output speed is improved with a PRNG at the end of the chain.

4.5 UART Module to Capture

Data leaving a CSRNG must be delivered in a ready format for consumption. For this raw data to be transmitted over a wire, it must be packaged in a protocol, so that the receiver can disseminate it. Therefore, a Universal Asynchronous Receiver Transmitter (UART) Transmission (TX) module was written in VHDL-93 and synthesised on the Xilinx FPGA to transmit serial data. The Spartan 3E development board's RS323 port was connected to a PC terminal. With the help of a run-time Python Serial Port Extension called Pyserial, the incoming UART data was unpackaged and decoded into binary. [Lie] Bitstream files for FPGA, QRNG and both Linux methods can be found in the thumb drive included with this Bachelor's Thesis. This data file will be later tested via the NIST 800-22 testing suite.

In order to provide raw random data for the widest range of cryptographic uses possible, special attention was given to the UART module. Once the UART TX module successfully sends a data packet out of the FPGA, the UART TX outputs a vld_out signal to inform the CSRNG to generate a new value. This power-saving features generates data on a need-by basis. Also the TX module is completely configurable. The baud rate can be set to the desired speed, as well as (1) the number of data bits per packet, (2) the type of parity, (3) number of stop bits and also (4) system clock speed is fully configurable at compile-time. Below is a snippet of the UART's interface options in VHDL:

```vhdl
entity uart_tx is
  generic(
    gi_baud_rate          : integer range 9600 to 115200     := 115200;
    gi_num_data_bits      : integer range 5 to 9             := 8;
    gi_num_parity_bits    : integer range 0 to 1             := 0;
    -- options are "none, even, odd, mark, space
    gs_parity_type        : string                           := "none";
    gi_clk_freq           : integer range 10 to 500000000    := 50000000;
    gi_num_stop_bits      : integer range 1 to 2             := 1);
```

Listing 1: uartTX.vhd

5 Statistically Testing Captured Data

Randomness is a probabilistic attribute, meaning that the characteristics of a random sequence can be defined and articulated through probability. The expected results of statistical tests, when conducted on a genuinely random sequence can be expressed in probabilistic terms. Numerous statistical tests already exist, each rigorously evaluating the presence or absence of a "pattern." Three of the most relevant and trusted test suites include:

- **NIST 800-22**: In 2010, the American National Institute of Standards and Technology (NIST) published NIST 800-22, a comprehensive set of fifteen tests that represent the cutting-edge in RNG testing. These tests play a crucial role in the selection of the Advanced Encryption Standard (AES) cipher by the Federal Information Processing Standard (FIPS) in 2001. [Ruk+10]
- **diehard**: Developed by the mathematician G. Marsaglia from the University of Florida State. This test suite contains 12 tests. Originally developed in 1990, diehard may be outdated in the 21st century.
- **dieharder**: The dieharder suite includes tests from both NIST 800-22 and diehard. dieharder expects a much larger set of data (above 4 GB). For this thesis, I am capturing data via UART. Therefore, capturing gigabytes of data on my PC via UART takes forever and is unrealistic.

5.1 Statistical Foundation for Testing

Bitstreams have been gathered from 4 sources: (1) oscillating rings on an FPGA, (2) Quantis PCIe, (3) non-blocking Linux /dev/random method and (4) the blocking Linux /dev/urandom method. The 4 captured data files shall be put through the NIST 800-22 test suite. However, before proceeding with testing, a solid understanding of some statistical fundamentals is necessary:

- **Hypothesis Testing**: In order to test the captured bitstreams, two hypotheses are made and used as the basis for testing. Within this bachelor's thesis, *the null hypothesis H_o claims that the 4 bitstreams under test are random.* If the results of the test suite do not disprove the null hypothesis, it is accepted as true. [MM19] If the bitstreams fail any test, H_o is false and the alternative hypothesis H_a (that the bitstreams are indeed not random) is proven true. If the data is indeed random, the correct conclusion is to accept H_o and to reject H_a.

- **False Positive or Type I Error**: If the data under test is genuinely random, then a decision to mistakenly reject the null hypothesis (that the sequence is not random) will probabilistically occur a small percentage of the time, resulting in a Type I error. Said differently, the Type I Error represents the likelihood for a truly random RNG to produce

True Situation	**Conclusion**	**Conclusion**
	Accept H_o	Accept H_a (Reject H_o)
Data is Random (H_o is True)	No Error	Type I Error
Data is not Random (H_a is True)	Type II Error	No Error

Table 2: Hypothetical Conclusion Table [Ruk+10]

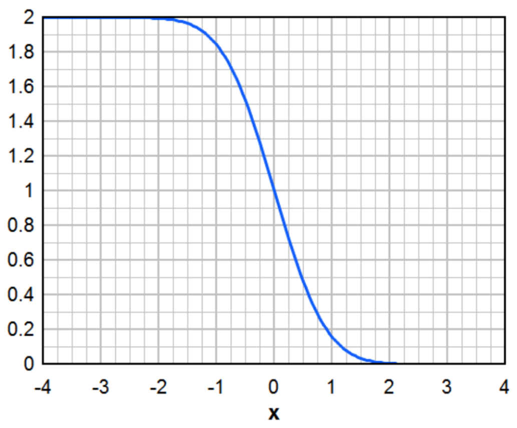


Figure 13: Returns approximation of erfc(x) = 1 - erf(x) [And98]

a bit sequence that is deemed to have originated from a faulty one. The likelihood for a Type I Error is denoted by alpha α and is referred to as the **level of significance** of the test. The value of α is usually set in the range of [0.001, 0.01]. For this thesis, α is set to 0.01.

- **False Negative or Type II Error**: Conversely, when the data is genuinely non-random, falsely accepting the null hypothesis (that the data is random) is called a Type II Error, denoted by the symbol ß. In other words, the Type II Error conveys the probability for a faulty RNG to be mistaken as "good".[Ruk+10]
- **P-Value**: According to NIST 800-22, the P-value summarizes the strength of the test results against the null hypothesis. For P-value = 1, the sequence appears to be perfectly random; while for P-value = 0, the given sequence is not random. Note that the output of each of NIST's 15 tests is a P-Value.
  If $P-Value \geq \alpha$, then the null hypothesis is accepted; while, if $P-Value < \alpha$, then the null hypothesis is rejected. If α is set to 0.001, one sequence in every 1000 will be falsely rejected. For a $P-value < 0.001$, a sequence would be considered to be non- random with a confidence of 99.9%, as seen in the formula below [AD13]:

$$if\ (P-value \leq \alpha) \rightarrow H_o \text{ is rejected with a confidence of } 1-\alpha$$

$$if\ (P-value > \alpha) \rightarrow H_o \text{ is accepted with a confidence of } 1-\alpha$$

- **ERFC**: Special functions required to run NIST 800-22 tests include the complementary error function (erfc). This function describes the likelihood of a extreme occurrence outside of three standard deviations σ from the mean of the Gaussian standard distribution. [And98] Note that this function lies outside the scope of this bachelor's thesis. As the complementary error function is already included in the math.h file provided by NIST 800-22 and nearly all NIST tests require the ERFC function, one only needs a very basic understanding of the function in order to plug values in. The ERFC is plotted in figure 13.

5.2 NIST 800-22 Test Suite

This chapter examines 8 of the 15 NIST tests. [Ruk+10] The output of each test is a P-Value that is compared to a chosen alpha $\alpha = 0.01$. If the P-Value is greater than α, then the null hypothesis is accepted.

5.2.1 Frequency Test

The most straight-forward test for chaos is the frequency test, as this test is considered to be a filter for NIST's other tests: If a bitstream fails this test, it will very likely fail the proceeding tests. The test counts the amount of 0s and 1s in the data. Everytime a 1 appears, a counter is incremented. When a 0 is counted, the counter decrements. [AD13]

$$Sum = \sum_{i=1}^{n} a_n(i)$$

$$a_n(i) = \begin{cases} 1 & \text{if } a(i) = 1, \\ -1 & \text{if } a(i) = 0 \end{cases}$$

5.2.2 Frequency Test within a Block

The second test in the NIST suite is the frequency test within a block. This test is identical to the previous one but with one addition: The bitstream under test is broken up into N blocks. The N blocks of data are then individually put through the frequency test.

5.2.3 The Runs Test

The runs test looks for the longest run of identical bits. If the length of the longest run of 1s or 0s is either above or below a certain threshold, then the test fails. NIST defines the threshold t as $t = 2 \ / \ \sqrt{n}$, where n equals the absolute sum of the block (see frequency test) and is explained in figure 14. If the proportion of 1s in a sequence is less than the constant t, the test fails. If the t condition is met, then the prerequisites are met for the test to actually begin.

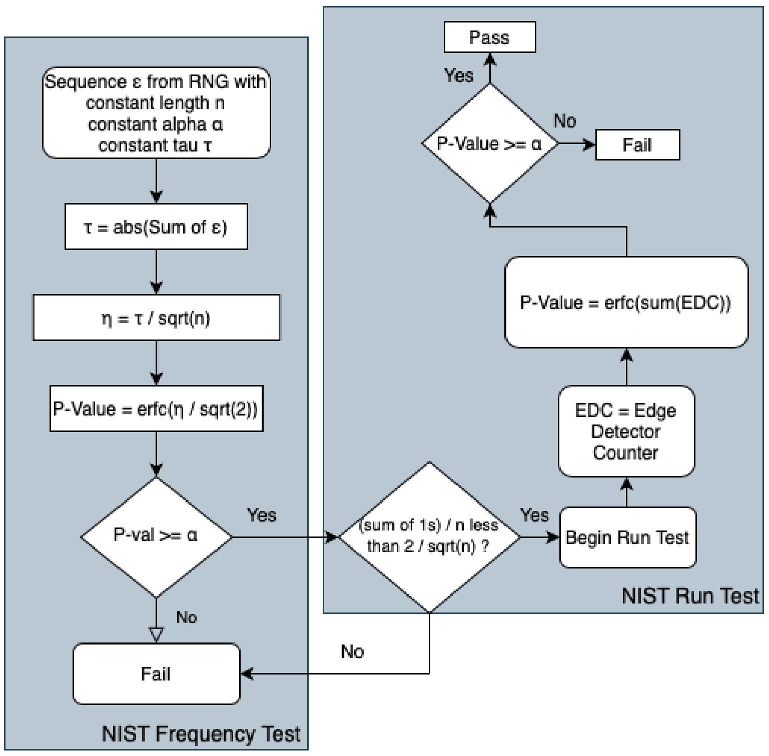

Sequence ε from RNG with constant length n constant alpha α constant tau τ
τ = abs(Sum of ε)
η = τ / sqrt(n)
P-Value = erfc(η / sqrt(2))
P-val >= α
Yes
No
Fail
NIST Frequency Test
(sum of 1s) / n less than 2 / sqrt(n) ?
Yes
No
Begin Run Test
EDC = Edge Detector Counter
P-Value = erfc(sum(EDC))
P-Value >= α
Yes
Pass
No
Fail
NIST Run Test

Figure 14: Decision Tree for Frequency and Run Test [Ruk+10]

The block under test is then scanned again, and each time the sequence of bits flips, the edge detector counter (EDC) is incremented. The sum of the EDC is then used as the argument in the ERFC function. The resulting P-value is compared to a. If the P-value is larger than alpha, the test passes.

5.2.4 Test for the Longest Run of Ones in a Block

This test is identical to the last, except that it breaks down a sequence ϵ into M blocks of x bits per block. After dividing the blocks into sections of equal bits, the remaining bits are ignored.

5.2.5 Binary Matrix Rank Test

The test centers on examining the rank of disconnected sub-matrices derived from the complete sequence. Its objective is to assess the presence of linear dependence within specific, fixed-length sub-strings. It's worth noting that this test is also featured in the DIEHARDER suite.

5.2.6 Non-overlapping Template Matching Test

This test examines and counts the frequency of a target string of bits, thereby detecting non-ideal generators which output repeating patterns. The test is described where,

S = The original data sequence under test.

S_L = The length of the data sequence

$M = S_L/N$ = The length of an individual block of bits.

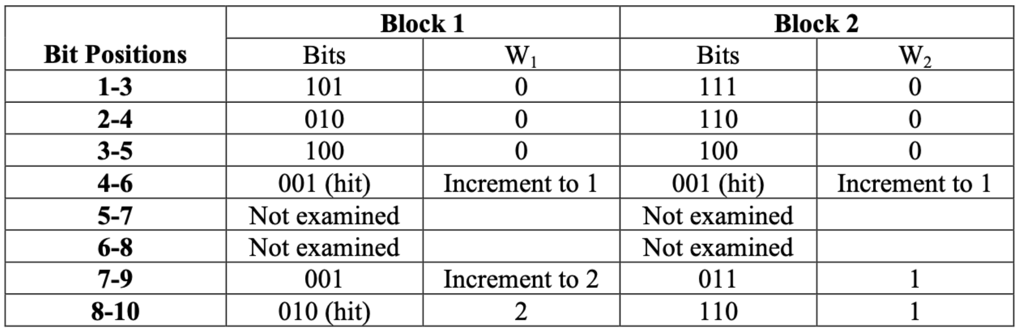
N = The number of independent blocks, each of size M.

B = The specific target string under examination.

m = The length in bits of the target string.

Given a target bitstream, start at the Most Significant Bit (MSB), and look for the target string B of size m. If a match occurs, increment the counter and move m bits to the right. If a match is not found, move just 1 bit to the right and search again.

Consider the example shown in figure 15, where $S = 10100100101110010110$ and $S_L = 20$. If $N = 2$ and $M = 10$, then block 1 is 1010010010 and block 2 is 1110010110. An m-bit window is created as each block is searched from MSB to Least Significant Bit (LSB). If a match occurs, the counter increases and moves one spot to the left. If $m = 3$ and the target string is $B = 001$, the non-overlapping template matching test would go as follows:



	Block 1		**Block 2**	
Bit Positions	Bits	W_1	Bits	W_2
1-3	101	0	111	0
2-4	010	0	110	0
3-5	100	0	100	0
4-6	001 (hit)	Increment to 1	001 (hit)	Increment to 1
5-7	Not examined		Not examined	
6-8	Not examined		Not examined	
7-9	001	Increment to 2	011	1
8-10	010 (hit)	2	110	1

Figure 15: Decision Tree for Frequency and Run Test [Ruk+10]

Then calculate the expected value and variance:

$$\mu = (M - m + 1)/2^m$$

$$variance = M * (1/2^m - (2m - 1)/2(2 * m)$$

Note that a P-Value is calculated for the frequency of each bit pattern and individually compared again alpha α.

5.2.7 Overlapping Template Matching Test

This test also measures the frequency that a specific sequence appears in the data. While the previous test moves m-bits to the right after a hit, the m-bit window of this test always moves just 1 bit to the right.

5.2.8 Cumulative Sum Test

The aim of this test is to target start-up or shutdown errors in the RNG, by counting the proportion of 1s and 0s at the beginning and end of the data sequence. More specifically, the test determines whether the cumulative sum of a partial sequence is too large or small relative to the expected behavior of a truly random sequence. A mode is selected ($mode$ = {forwards, backwards}). Forwards counts from MSB to LSB (from beginning towards end of sequence) and backwards counts from LSB to MSB (backwards from end of the sequence).

Let sequence $s = 1011010111$, sequence length $S_L = 10$, $mode =$ forwards, where $S_k =$ the partial sum of s at index k. The partial sums for S_k of the bit string are as follows:

$$\begin{aligned}
s &= 1, (-1), 1, 1, (-1), 1, (-1), 1, 1, 1 \\
S_1 &= 1 \\
S_2 &= 1 + (-1) = 0 \\
S_3 &= 1 + (-1) + 1 = 1 \\
S_4 &= 1 + (-1) + 1 + 1 = 2 \\
S_5 &= 1 + (-1) + 1 + 1 + (-1) = 1 \\
S_6 &= 1 + (-1) + 1 + 1 + (-1) + 1 = 2 \\
S_7 &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) = 1 \\
S_8 &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 = 2 \\
S_9 &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 = 3 \\
S_{10} &= 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + 1 + 1 = 4 \\
Largest\ S_k &= z = 4 = S_{10}
\end{aligned}$$

The largest value for S_k is 4, therefore z = 4. [Ruk+10] Plug the values into figure 16 to receive the P-Value:

$$\text{Compute } P\text{-}value = 1 - \sum_{k=\left(\frac{-n}{z}+1\right)/4}^{\left(\frac{n}{z}-1\right)/4} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}}\right)\right] +$$

$$\sum_{k=\left(\frac{-n}{z}-3\right)/4}^{\left(\frac{n}{z}-1\right)/4} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right)\right]$$

Figure 16: Calculate P-Value for Cumulative Sum Test [Ruk+10]

The standard normal distribution ϕ is defined as follows:

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2}\, dt$$

If the P-value is greater than alpha, the test pasts. Large values z indicate there are too many ones or too many zeros at the beginning or end of the sequence. Small values indicate that ones and zeros are intermixed too evenly.

6 Results and Interpretation

Now that we know how the tests operate, let us finally test the data. Four megabits of data from all four CSRNGs (QRNG, FPGA TRNG, dev/urandom and dev/random) were captured and run through the NIST test suite. For tests requiring M amount of blocks, $M = 10$ was selected. The significance level alpha α was set to 0.01. The P-Value for all tests exceeded α, meaning all tests passed, and the null hypothesis H_o is accepted with a confidence of $1 - \alpha = 99.9\%$. Note that I was unable to run the random excursion, random excursion variant and Mauerer's tests due to time constraints on the thesis's deadline. These results are not included. The table below shows results for the FPGA's TRNG:[2]

Test	**Pass Fail**	**P-Value**
Frequency Test	Pass	0.739918
Freq Test within a Block	Pass	0.213309
Runs Test	Pass	0.213309
Longest Run of Ones in a Block	Pass	0.350485
Binary Matrix Rank	Pass	0.534146
Non-overlapping Template Matching	Pass	0.534146
Overlapping Template Matching	Pass	0.213309
Fourier Transform Test	Pass	0.534146
Linear Complexity Test	Pass	0.350485
Serial Test Sum	Pass	0.534146
Approx. Entropy Test	Pass	0.122325
Cumulative Sums	Pass	0.7399

Table 3: Null hypothesis for FPGA RO is accepted.

To provide a visual yet anecdotal representation of the data, 9.8KB were captured from the QRNG, TRNG and both blocking /dev/random and non-blocking /dev/urandom Linux-RNGs. As seen in figures 17 through 20, each byte value was converted to a grey-scale pixel, with the

[2]The data captured from captured from the QRNG and Linux-RNGs also passed the NIST tests. The results can be found on the thumb-drive, that was submitted along with this thesis.

dimensions 70 x 140 pixels. Looking at these visual representations, the eye fails to notice a pattern in either four grey-scale images, indicating that the output is sufficiently chaotic. While using a visual representation of the data is not conclusive evidence to confirm this thesis's null hypothesis, that a sequence is truly random, it is interesting to see nonetheless.

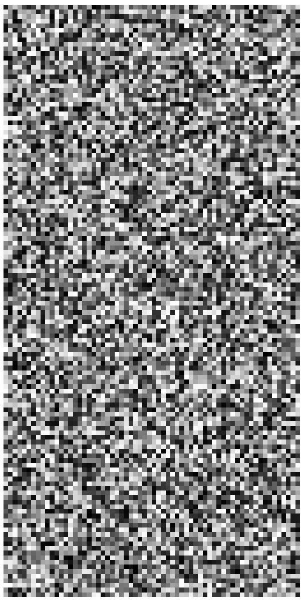


Figure 17: QRNG

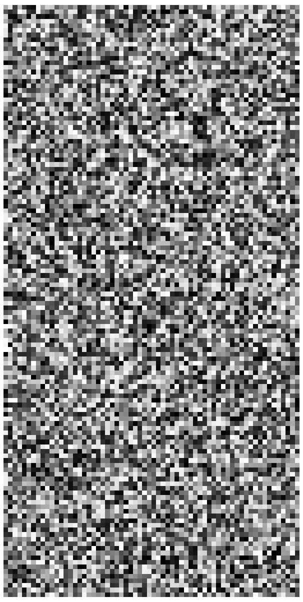


Figure 18: TRNG

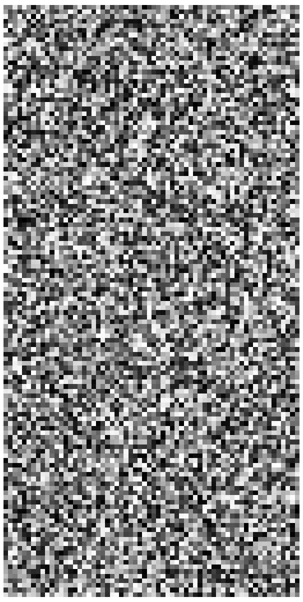


Figure 19: urandom

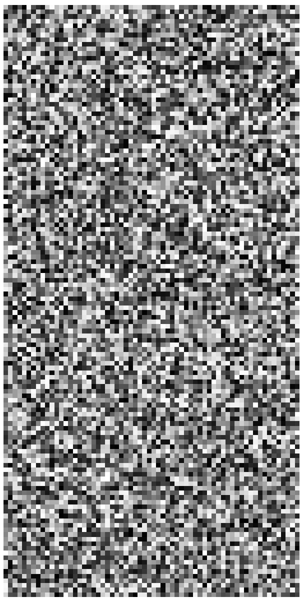


Figure 20: random

7 Discussion

While all four CSRNGs were validated by the NIST test results, the devices themselves were not put under any stress testing. Throughout testing, an ideal scenario was assumed, where the devices themselves were not manipulated or interacted with at all. This assumption is unrealistic. RNGs in real-world applications have people who install, repair and monitor them. The device itself may be in close proximity to an electromagnetic field, occasionally experience electrostatic discharge or have high-wattage signals flowing close by. Such a device lives in an ecosystem of variables that affect its performance. Moreover, an RNG itself consists of an array of microscopic electrical components that require a constant yet precise flow of electricity to function. This vector of fragile components is referred to as the device's side channel. [Cha+10]

7.1 Side Channel Attacks

A RNG's side channel can be exploited in a targeted attack. These attacks exploit physical vulnerabilities within the hardware implementation instead of its mathematical algorithm. [CCA22] A side channel attack or SCA can be done by introducing spikes in the power supply, emitting electromagnetic signals, altering the RNG's operating temperature beyond the recommended range, etc. TRNGs, which rely on physical entropy sources, are especially prone to SCAs. [Abd+15]

7.1.1 Side Channel Birthday Attack

A concrete example of a side channel attack is illustrated by Marketto in his paper. [MM09] Due to governmental regulations, many ATMs in the UK use the same CSRNG to generate random secret keys to encrypt sensitive customer information, like credit card number, transaction ID, PIN, etc. The entropy source for the CSRNG in the ATMs utilise ring oscillation. For cash withdrawals, an ATMs picks a number from four billion possible values. By introducing a signal into the CSRNG's power supply, Marketto reduces the effective range of randomly generated values from 4 billion or 2^{32} down to 2^8. With only 2^8 or 255 or n possible random key values, he uses the heuristics of the birthday paradox to brute force the key within $\sqrt{n}$ guesses.

The birthday paradox states that for every 23 people in a room, there is a 50% chance that two individuals share a birthday. If person 1 has his birthday on a random day, the probability for person 2's birthday to be on a different day is $\frac{364}{365}$, while person 3 has a $\frac{363}{365}$ percent chance to not share a birthday with either person 1 or 2 - as shown in the formula below:

$$P(A) = \frac{365}{365} - (\frac{365-1}{365} * \frac{365-2}{365} * \frac{365-3}{365} * ... * \frac{365-23}{365}) \approx 0.5 = 50\%$$

The same intuitive approach can be transferred to the situation with the ATM. Consider a fraudulent ATM that was modified to duplicate a transaction $\sqrt{255}$ or 16 times. Now 16 keys exist, within a range of 255 possible values. After 16 attempts to guess the key, there is a 50% chance to crack it, thereby jeopardising the customer's sensitive financial information:

$$P(\text{correct guess on first attempt}) = 1 - (\frac{256-16}{256})^1 \approx 0.06 = 6\%$$

$$P(\text{correct guess on 16-th attempt}) = 1 - (\frac{256-16}{256})^{16} \approx 0.50 = 50\%$$

Once the key is found, the customer's credit card number, PIN and personal information are compromised. Marketto goes on to illustrate the effects of this SCA on the ATM's TRNG in figure 21. All three images depict the bitstream under varying degrees of stress in the grey-scale image of 70 x 140 bits. Each white pixel represents a binary 0, while black pixels represent a binary 1. When the power supply is not manipulated, the RNG seems to output random values; however, when signals of either 1.82 and 1.93 MHz are injected, the RNG's output becomes quite predictable. The TRNG used in Marketto's experiment to encrypt sensitive financial data therefore seems insecure.

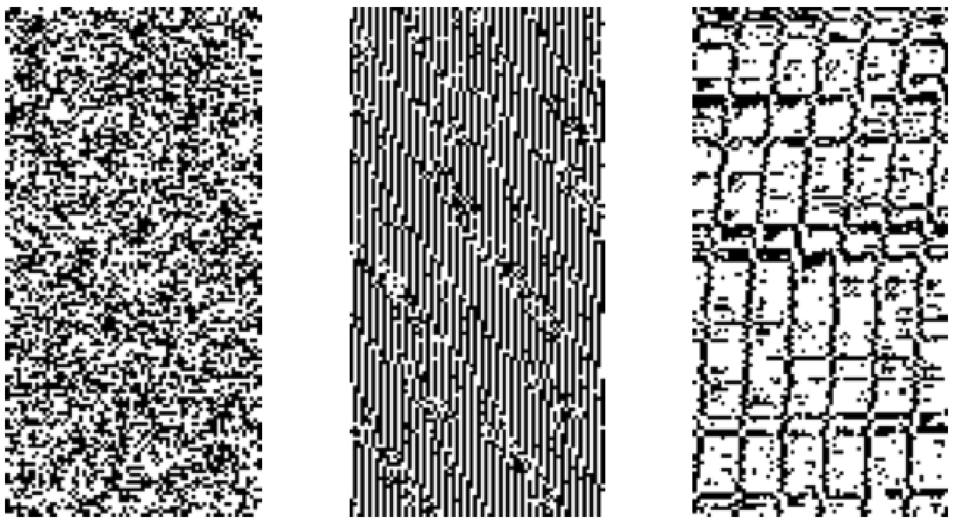

(a) No injection, 70×140 bits (b) 1.822880 MHz injection, 70×140 bits (c) 1.929629 MHz injection, 73×146 bits

Figure 21: Targeted attack via side-channel reduces RNG's integrity [MM09]

7.2 The Vulnerability of Oscillating Ring to SCAs

While a CSRNG's risk to a SCA depends upon its entropy source, perhaps no entropy source is at greater risk of manipulation than for oscillating rings. Commonly used for their ease of integration, and low power/area requirements, their primary drawback is that they tend to synchronise with other signals, such as the power supply, or external periodic signals. [BFV10]

An experiment was carried out to see the effect that a foreign signal had on an RO. As seen in figure 22, a single RO_{orig} runs at it's natural frequency of 296 MHz. A foreign $RO_{foreign}$ propagating at frequency f_{pert} is then instantiated in close proximity to RO_{orig}. The frequency of $RO_{foreign}$ is slowly raised from 285 to 300 MHz at steps of 0.025 MHz. [Mur+19] RO_{orig}'s frequency changes as the frequency of $RO_{foreign}$ approaches its own. Once $RO_{foreign}$ reaches 287 MHz, RO_{orig} deviates from its natural frequency, locks to $RO_{foreign}$ and stays synchronised until the frequency of $RO_{foreign}$ raises above its natural frequency.

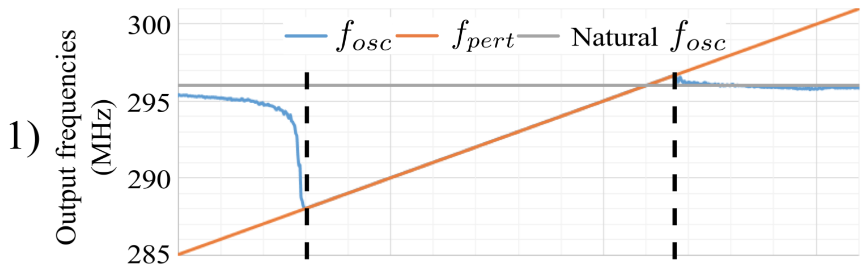


Figure 22: RO locks [Mur+19]

RO_{orig} clearly synchronises to a foreign signal, resulting in a side-channel attack and thereby causing the system to malfunction. This issue is particularly troublesome in data security applications, as it poses a threat to the cryptographic security of generated keys. In such instances, the generated keys become deterministic, resulting in predictability and a breach in the system's integrity. [ZS21]

7.3 Improving FPGA's CSRNG to Protect against Attacks

Thinking back to the TRNG implemented on the FPGA in chapter 4, design improvements are necessary to increase resistance to SCAs. While most improvements fall outside the scope of this bachelor thesis, two simple changes can be made: (1) instantiating only odd numbers of ROs and (2) ensuring no two ROs are the same length. ROs of differing length oscillate at different frequencies, thereby preventing individual ROs from locking or synchronising. The following code snippet demonstrates this optimisation:

```
entropy_source:
for i in 0 to NUM_of_ROs-1 generate
  ring_oscillator_inst : ring_oscillator
  generic map (
        Num_of_Inv  => Num_of_Inv_RO + 2*i) -- Ring oscillators of
            different length
```

Listing 2: uartTX.vhd

In the generics section of the VHDL code, default values can be set for objects upon instantiation. By making the following addition, NUM_INV => NUM_INV_START + **2*i**, each additionally instantiated ring oscillator will feature two more inverters than the previous.

The key point to communicate is that TRNGs are vulnerable to manipulation via their inherent side-channel. Countermeasures must constantly be taken to reduce this effect.

7.3.1 Side Channel Attacks in QRNGs

The discussion of QRNG's weakness to side-channel attacks is quite new within the realm of cybersecurity, as the first academic paper I could find on this subject appeared in 2016 (and the paper even incorrectly states that QRNGs are immune to side-channel attacks). [MYC16] Side channel attacks in the quantum world first became a point of interest for many researchers after the 2016 National Institute of Standards and Technology (NIST) called for post-quantum algorithms resistant to quantum computer attacks. [CCA22] The potential for side channel attacks on quantum generators is smaller than for TRNGs, as there are not yet any documented instances of attacks in real-world scenarios. The current security of post-quantum algorithms lies in algorithm complexity. Due to the novelty of many post-quantum cryptographic schemes, many side-channel vulnerabilities have not yet been evaluated.

7.4 Security of Physical RNG vs QRNG

The question arises, whether to rely on physical or quantum entropy sources to provide cryptographic security. There are two reasons why the future is quantum: (1) QRNGs do not need a conditioner or de-biaser, as entropy extracted through quantum mechanics are inherently unpredictable. This means that a QRNG features one less component that can be exploited via a SCA. Also, (2) QRNGs are still a young technology. Their potential for manipulation remains largely understudied, meaning a toolkit to exploit these devices does not yet exist.

8 Conclusion

In this thesis, I introduced the building blocks of a cryptographically secure random number generator and build one from scratch: From the theory of randomness in thermal dynamics and Shannon Entropy, to subjective and objective qualities of a RNG, to example noise sources, to the individual digital-logic modules of a well-functioning CSRNG. Chapter 3 puts this foundational knowledge to work by examining two modern randomness generators in action. As the capstone of this thesis, a CSRNG is implemented on an Xilinx XC3S1600E FPGA for the Spartan 3E development board. It uses the phase noise of free propagating oscillating rings in close proximity as the entropy source. The output bitstream is captured and tested via the NIST 800-22 test suite. The results validate the machine. Although the TRNG passed all validation tests, a side channel attack would easily compromise its integrity. Therefore, countermeasures are vital to prevent the ease of SCAs. Furthermore, quantum entropy sources seem superior to physical ones, as they appear to be more resilient against SCAs.

In the post-quantum world, the definition of ”security” is rapidly changing. Along with it, the bar necessary to ensure cryptographic security is rising. The first recorded encryption algorithm, the Caesar cipher, took 900 years to be cracked. Today, it would take a mere microsecond to do the same. As the sophistication of encryption tools increases, so likewise do the tools for decryption. And as these advances in technology allow for ever-increasing algorithmic and pattern-detecting scrutiny, sequences once thought unpredictable are now rendered predictable. Randomness appears to be an ideal and not a definite state. ”Random enough” (in the context of a present moment’s technological sophistication) is now the only realistic goal.

Bibliography

[Abd+15] Abdalla et al. “Robust Pseudo-Random Number Generators with input Secure Against Side-Channel Attacks”. In: *Applied Cryptography and Network Security* (2015). URL: `https://doi.org/10.1007/978-3-319-28166-7_31`.

[AD13] Antonioli and Daniele. “Design and Testing of Random Number Generators (RNG)”. `http://www.lulu.com/shop/daniele-antonioli/design-and-testing-of-rng/ebook/product-20965725.html`. MA thesis. Italy and US: University of Bologna and University of Massachusetts Amherst, 2013.

[And98] Larry Andrews. “Special functions of mathematics for engineers”. In: *SPIE Press* (1998).

[Bak22] Anubhab Baksi. *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. Springer Singapore, 2022.

[Bel20] Edward Beltrami. *What is Random?* Springer, 2020.

[BFV10] Bernard, Fischer, and Valtchanov. “True-Randomness and Pseudo-Randomness in Ring Oscillator-Based True Random Number Generators”. In: *ReConFig09* (2010).

[Bri19] Encyclopedia Britannica. *Encyclopedia Britannica. Chaos Theory*. Encyclopedia Britannica, 2019.

[Cao+22] Yuan Cao et al. “Entropy Sources Based on Silicon Chips: True Random Number Generator and Physical Unclonable Function”. In: *Entropy* 24.11 (2022). ISSN: 1099-4300. URL: `https://www.mdpi.com/1099-4300/24/11/1566`.

[CCA22] Chowdhury, Covic, and Acharya. “Physical security in the post-quantum era”. In: *J Cryptogr Eng* (2022). URL: `https://doi.org/10.1007/s13389-021-00255-w`.

[Cha+10] Chari1 et al. “Designing a Side Channel Resistant Random Number Generator”. In: *Springer* (2010).

[CL16] Calude and Longo. “Classical, quantum and biological randomness as relative unpredictability”. In: *Natural Computing* (2016).

[Ell21] David Ellerman. *New Foundations for Information Theory*. Springer Cham, 2021.

[Guo+10] Guo et al. “Truly random number generation based on measurement of phase noise of a laser”. In: *Phys. Rev. E* 81 (5 2010), p. 051137. DOI: `10.1103/PhysRevE.81.051137`. URL: `https://link.aps.org/doi/10.1103/PhysRevE.81.051137`.

[Hol17] Joshua Holden. *he Mathematics of Secrets: Cryptography from Caesar Ciphers to Digital Encryption*. Princeton University Press, 2017.

[HS23] Uwe Hohm and Christoph Schiller. “Testing the Minimum System Entropy and the Quantum of Entropy”. In: *Entropy* 25.11 (2023). ISSN: 1099-4300. URL: `https://www.mdpi.com/1099-4300/25/11/1511`.

[IS22] Federal Office for Information Security. “Documentation and Analysis of the Linux Random Number Generator”. In: *Federal Office for Information Security* (2022). URL: https://www.bsi.bund.de/DE/Service-Navi/Publikationen/Studien/LinuxRNG/linuxrng.html.

[Jen+00] Jennewein et al. “A fast and compact quantum random number generator”. In: *Review of Scientific Instruments* 71.4 (Apr. 2000), pp. 1675–1680. ISSN: 0034-6748. DOI: 10.1063/1.1150518. eprint: https://pubs.aip.org/aip/rsi/article-pdf/71/4/1675/8813861/1675_1_online.pdf. URL: https://doi.org/10.1063/1.1150518.

[KGW19] Michael Kolonko, Feng Gu, and Zijun Wu. “Improving the statistical quality of random number generators by applying a simple ratio transformation”. In: *Mathematics and Computers in Simulation* (2019).

[KH23] M. J. Kazemi and V. Hosseinzadeh. “Detection statistics in a double-double-slit experiment”. In: *Phys. Rev. A* 107 (1 2023), p. 012223. DOI: 10.1103/PhysRevA.107.012223. URL: https://link.aps.org/doi/10.1103/PhysRevA.107.012223.

[Mar17] Amos Maritan. “Entropy and Its Applications across Disciplines”. In: *entropy* (2017).

[MM09] Markettos and Moore. “The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators”. In: *Cryptographic Hardware and Embedded Systems* (2009).

[MM19] David Moore and George McCabe. *Introduction to the Practice of Statistics.* W.H. Freeman and Co, 2019.

[MM98] Mullin and Mogg. “Dimensions of subjective uncertainty in social identification and minimal intergroup discrimination”. In: *Br J Soc Psychol* (1998).

[MR14] Montgomery and Runger. *Applied Statistics and Probability for Engineers(6th ed.)* Wiley, 2014.

[Mur+19] Ugo Mureddu et al. “Experimental Study of Locking Phenomena on Oscillating Rings Implemented in Logic Devices”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.7 (2019), pp. 2560–2571. DOI: 10.1109/TCSI.2019.2900017.

[MXW05] Hai-Qiang Ma, Yuejian Xie, and Ling-An Wu. “Random number generation based on the time of arrival of single photons”. In: *Appl. Opt.* 44.36 (2005), pp. 7760–7763. DOI: 10.1364/AO.44.007760. URL: https://opg.optica.org/ao/abstract.cfm?URI=ao-44-36-7760.

[MYC16] Ma, Yuan, and Cao. “Quantum random number generation.” In: *npj Quantum Inf* (2016). URL: https://doi.org/10.1038/npjqi.2016.21.

[Nak18] Svetlin Nakov. *Practical Cryptography.* MIT license, 2018.

[Oh+23] Mi-Kyung Oh et al. “Implementation and characterization of flash-based hardware security primitives for cryptographic key generation”. In: *ETRI Journal* 45.2 (2023), pp. 346–357. DOI: `https://doi.org/10.4218/etrij.2021-0455`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.4218/etrij.2021-0455`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.4218/etrij.2021-0455`.

[Pam+18] Venkata Rajesh Pamula et al. “A 65-nm CMOS 3.2-to-86 Mb/s 2.58 pJ/bit Highly Digital True-Random-Number Generator With Integrated De-Correlation and Bias Correction”. In: *IEEE Solid-State Circuits Letters* 1.12 (2018), pp. 237–240. DOI: `10.1109/LSSC.2019.2896777`.

[Prn] “Cryptographically secure random number generator with chaotic additional input”. In: *Nonlinear Dynamics* (2014).

[RGX11] Gregory Rose, Alexander Gauntman, and Lu Xiao. “Cryptographically Secure Pseudo-Random Number Generator”. In: *United States Patent* (2011).

[RK20] Sergey Rashkovskiy and Andrei Khrennikov. “Psychological ‘double-slit experiment’ in decision making: Quantum versus classical”. In: *Biosystems* 195 (2020), p. 104171. ISSN: 0303-2647. DOI: `https://doi.org/10.1016/j.biosystems.2020.104171`. URL: `https://www.sciencedirect.com/science/article/pii/S0303264720300708`.

[Ruk+10] Rukhin et al. “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”. In: *National Institute of Standards and Technology* (2010).

[RYD20] Rozic, Yang, and Dehaene. “Iterating Von Neumann’s Post-Processing under Hardware Constraints”. In: *ESAT/MICAS, KU Leuven and IMEC* (2020).

[Sco03] David Scott. “The arrow of time”. In: *International Journal of Hydrogen Energy* (2003).

[Sin00] Simon Singh. *The Code Book*. Anchor, 2000.

[Tes20] Teseleanu. “Random Number Generators Can Be Fooled to Behave Badly”. In: (2020).

[Tur+18] Turan et al. “Recommendation for the Entropy Sources Used for Random Bit Generation”. In: *NIST Special Publication* (2018).

[Weh78] Alfred Wehrl. “General properties of entropy”. In: *Rev. Mod. Phys.* 50 (2 1978), pp. 221–260. DOI: `10.1103/RevModPhys.50.221`. URL: `https://link.aps.org/doi/10.1103/RevModPhys.50.221`.

[WHG09] Wei, Hong, and Guo. “Bias-free true random-number generator”. In: *Opt. Lett.* 34.12 (2009), pp. 1876–1878. DOI: `10.1364/OL.34.001876`. URL: `https://opg.optica.org/ol/abstract.cfm?URI=ol-34-12-1876`.

[XH14] Juemin Xu and Nigel Harvey. “Carry on winning: The gamblers’ fallacy creates hot hand effects in online gambling”. In: *Cognition* (2014). URL: `https://www.sciencedirect.com/science/article/pii/S0010027714000031`.

[YK18] Yongjin Yeom and Ju-Sung Kang. “Probability distributions for the Linux entropy estimator”. In: *Discrete Applied Mathematics* 241 (2018). Cryptography and Future Security, pp. 87–99. ISSN: 0166-218X. DOI: `https://doi.org/10.1016/j.dam.2016.07.019`. URL: `https://www.sciencedirect.com/science/article/pii/S0166218X16303390`.

[Zho+17] Zhi-Yuan Zhou et al. “Quantum twisted double-slits experiments: confirming wavefunctions’ physical reality”. In: *Science Bulletin* 62.17 (2017), pp. 1185–1192. ISSN: 2095-9273. DOI: `https://doi.org/10.1016/j.scib.2017.08.024`. URL: `https://www.sciencedirect.com/science/article/pii/S2095927317304358`.

[ZS21] Zhiwen Zhang and Tao Su. “Behavioral Analysis and Immunity Design of the RO-Based TRNG under Electromagnetic Interference”. In: *Electronics* 10.11 (2021). ISSN: 2079-9292. DOI: `10.3390/electronics10111347`. URL: `https://www.mdpi.com/2079-9292/10/11/1347`.

List of References

[Bha] Pritha Bhandari. *Normal Distribution | Examples, Formulas, Uses*. Gauss Standard Distribution. URL: https://www.scribbr.com/statistics/normal-distribution/.

[Bla] Blackburn. *Law of Thermodynamics*. URL: https://www.oxfordreference.com/view/10.1093/acref/9780199541430.001.0001/acref-9780199541430-e-3091.

[com] Kernel development community. *The Linux Kernel documentation*. Linux Random Number Generation1. URL: https://docs.kernel.org/#the-linux-kernel-documentation.

[Fou] Fractal Foundation. *What is Chaos Theory?* Chaos. URL: https://fractalfoundation.org/resources/what-is-chaos-theory/.

[Hel] Anne Helmenstine. *What Is Entropy? Definition and Examples*. Thermal Entropy. URL: https://sciencenotes.org/what-is-entropy-definition-and-examples/.

[idq] idquantique. *Quantis QRNG USB*. URL: https://www.idquantique.com/random-number-generation/products/quantis-random-number-generator/.

[Lib] LibreTexts. *The Concept of Entropy*. Radioactive Decay Entropy. URL: https://chem.libretexts.org/Courses/Mount_Royal_University/Chem_1202/Unit_7%3A_Principles_of_Thermodynamics/7.1%3A_The_Concept_of_Entropy.

[Lie] Christ Liechti. *Pyserial 3.5*. Pyserial. URL: https://pypi.org/project/pyserial/.

[Lin] Tom Linux. *Myths about /dev/urandom*. Linux inner workings. URL: https://www.2uo.de/myths-about-urandom/.

[Mue] Stephan Mueller. *Linux Random Number Generator -- a new approach to the Linux /dev/random*. Linux Random Number Generation. URL: http://www.chronox.de/lrng.html.

[SE] Quantic SE. *Quantis QRNG PCIe-40M PCIe-240M*. Quantis inner workings. URL: https://marketing.idquantique.com/acton/attachment/11868/f-9a58efcf-5169-45ba-8a1c-ceea3761581f/1/-/-/-/-/Quantis%20QRNG%20PCIe%20New%20Generation_Brochure.pdf.

[Sta] Tom Stafford. *Why we gamble like monkeys*. Gambler's Fallacy. URL: https://www.bbc.com/future/article/20150127-why-we-gamble-like-monkeys.