PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA MINISTRY OF
HIGHER EDUCATION AND SCIENTIFIC RESEARCH

AMAR TELIDJI UNIVERSITY- LAGHOUAT

FACULTY OF TECHNOLOGY
ELECTRONICS SECTOR
ELECTRONICS DEPARTEMENT

Dissertation Master Degree in Embedded Systems

Presented by: ABDELALI ILIAS
DJOUBAR ANAS ZAKARIA

FIELD: Electronic

SECTOR: Technology

OPTION: Electronics of embedded systems

Theme

Desing and FPGA-based implementation of an efficient pseudo-random number generator (PRNG) with chaotic systems

Jury members :

Merah Lahcene	Pr	Supervisor
Seghier Abdelkrim	MAA	President
Mourad reguigue	MCB	Examiner

Promotion : 2024/2025

Abstract

This thesis explores the design and FPGA implementation of a chaos-based pseudo-random number generator (PRNG) for cryptographic applications. Focusing on the Tinkerbell map, we analyze the effects of finite-precision arithmetic on chaotic systems and propose a novel perturbation mechanism (PPM) to mitigate dynamical degradation. The implemented PRNG demonstrates enhanced statistical randomness, extended cycle lengths, and hardware efficiency, validated through NIST tests and FPGA prototyping. Our results show that chaos-based PRNGs can provide secure, efficient alternatives to conventional methods in cryptography.

Keywords: Chaos-based cryptography, PRNG, Tinkerbell map, Finite-precision effects, Dynamical degradation, Perturbation, FPGA, NIST statistical tests, Hardware security, Cryptography.

Résumé

Cette thèse étudie la conception et l'implémentation FPGA d'un générateur de nombres pseudo-aléatoires (PRNG) basé sur le chaos pour applications cryptographiques. En se concentrant sur la carte de Tinkerbell, nous analysons les effets de la précision finie sur les systèmes chaotiques et proposons un mécanisme de perturbation innovant (PPM). Le PRNG implémenté montre une amélioration de la randomisation statistique, des cycles plus longs et une efficacité matérielle, validés par des tests NIST et un prototype FPGA. Nos résultats démontrent que les PRNG chaotiques peuvent concurrencer les méthodes conventionnelles en cryptographie.

Mots-clés : Cryptographie basée sur le chaos, PRNG, Carte de Tinkerbell, Effets de précision finie, Dégradation dynamique, FPGA, Tests statistiques NIST, Sécurité matérielle, cryptographie.

ملخص

تقدم هذه الأطروحة تصميم وتنفيذ مولد أعداد شبه عشوائي (PRNG) يعتمد على النظام الفوضوي لتطبيقات التشفير، مع التركيز على خريطة Tinkerbell. نقوم بتحليل تأثيرات الدقة الحسابية المحدودة على الأنظمة الفوضوية ونقترح آلية اضطراب جديدة (PPM) . يُظهر النموذج المُنفذ تحسناً في العشوائية الإحصائية وزيادة طول الدورات الزمنية، مع الحفاظ على الكفاءة، حيث تم التحقق من النتائج عبر اختباراتNIST . أثبتت دراستنا أن أنظمة التوليد العشوائي الفوضوية يمكن أن تكون بديلاً أميناً وفعالاً في التشفير.

الكلمات المفتاحية :
التشفير القائم على الفوضى، مولد الأعداد شبه العشوائية(PRNG) ، ,خريطة تينكربيل، تأثيرات الدقة المحدودة، التدهور الديناميكي، تشفير، اختباراتNIST .

Acknowledgments

First and foremost, we thank **Allah**, the Almighty, for His blessings and grace, and for granting us the strength and patience throughout the course of this work.

Our sincere thanks and deepest gratitude go to our supervisor, **Pr. Merah Lahcene**, whose scientific and moral support was both valuable and essential throughout the completion of this thesis.

We also extend our heartfelt thanks to the honorable members of the jury for dedicating their precious time to reviewing and evaluating our work.

Additionally, we would like to express our appreciation to all our professors, who provided us with the foundational knowledge and invaluable support during our academic journey.

Dedication

To my family
The constant source of support, strength, and inspiration,
To my parents, my siblings, and every member of my beloved family
I dedicate this modest work to you, in deep appreciation for your love, encouragement, and unwavering presence throughout my academic journey

Anas & Ilyas

CONTENT

List of figures

Chapter 1 : An overview on Pseudo-Random Number Generators

Chapter 2 : Fundamentals of Chaos

Chapter 3 : Impact of Finite Precision on Chaotic System Behavior and Countermeasures

Chapter 4 : Design and hardware implementation of secure chaos-based PRNG

List of tables

Chapter 3 : Impact of Finite Precision on Chaotic System Behavior and Countermeasures

Chapter 4 : Design and hardware implementation of secure chaos-based PRNG

General Introduction

Random number generation is a fundamental component of modern cryptography, playing a crucial role in securing digital communications, encryption protocols, and authentication mechanisms. Among the various approaches to generating random numbers, Pseudo-Random Number Generators (PRNGs) are widely used due to their computational efficiency and deterministic reproducibility. However, traditional PRNGs often rely on mathematical algorithms that may exhibit predictable patterns or vulnerabilities under rigorous cryptanalysis. To address these limitations, researchers have turned to chaotic systems, which offer inherent properties such as sensitivity to initial conditions, ergodicity, and topological mixing—making them promising candidates for secure random number generation.

This thesis explores the design, analysis, and hardware implementation of a chaos-based PRNG, with a particular focus on overcoming the challenges posed by finite-precision arithmetic in digital systems. While chaotic systems theoretically exhibit complex and unpredictable behaviour, their practical implementation on digital platforms—such as FPGAs and microprocessors—introduces dynamical degradation. This degradation manifests as short cycle lengths, non-uniform distribution, and loss of ergodicity, severely compromising their cryptographic utility. Addressing these limitations is essential for leveraging chaotic systems in real-world security applications.

The primary objectives of this work are threefold. First, we conduct a theoretical analysis of how finite-precision arithmetic affects chaotic maps, such as the Tinkerbell and Hénon maps, quantifying their degradation in terms of cycle length, autocorrelation, and statistical randomness. Second, we propose and evaluate mitigation strategies, including precision scaling, cascaded chaotic systems, and a novel perturbation mechanism, to counteract dynamical degradation while maintaining computational efficiency. Third, we develop an optimized FPGA-based PRNG that leverages chaotic dynamics, validating its performance through rigorous statistical tests (e.g., NIST SP 800-22) and real-world hardware metrics.

Our methodology begins with a rigorous examination of chaos theory, establishing the mathematical foundations of chaotic systems and their susceptibility to digital degradation. We then introduce the Proposed Perturbation Mechanism (PPM), a lightweight, self-sustaining technique designed to extend cycle lengths and enhance randomness without relying on external entropy sources. Finally, we present the hardware realization of our chaos-based PRNG on a Xilinx Artix-7 FPGA using the Vivado toolchain, demonstrating its real-time applicability and low resource overhead.

The thesis is structured to provide a comprehensive exploration of chaos-based PRNGs. Chapter 1 reviews existing PRNGs, contrasting algorithmic approaches (e.g., Mersenne Twister, ChaCha20) with physical True Random Number Generators (TRNGs), and highlights the need for chaos-based solutions. Chapter 2 establishes the theoretical framework of chaos theory, emphasizing its relevance to cryptography. Chapter 3 analyses the effects of finite precision on chaotic systems and surveys existing countermeasures. Chapter 4 presents the

FPGA implementation of our secure chaos-based PRNG, integrating the PPM and validating its performance through statistical and hardware tests.

The outcomes of this research bridge the gap between theoretical chaos and practical cryptography, offering a resource-efficient, high-entropy PRNG suitable for embedded security applications. Our results demonstrate that chaotic systems, when properly augmented, can surpass conventional PRNGs in unpredictability while remaining feasible for real-time deployment. Future directions include exploring hybrid chaos-machine learning designs and enhancing post-quantum resilience.

By addressing the critical challenge of dynamical degradation, this thesis contributes to the advancement of secure, chaos-driven cryptographic primitives. Our work paves the way for their adoption in next-generation hardware security modules, IoT devices, and communication systems, ensuring robust and unpredictable random number generation in an increasingly digital world.

CHAPTER 01

An overview on Pseudo-Random Number Generators

1. Introduction

Random numbers play a crucial role in a wide range of modern applications, particularly in information security and cryptography. Among the various methods for generating random numbers, Pseudo-Random Number Generators (PRNGs) are the most commonly used, primarily due to their computational efficiency and deterministic behavior.

This chapter presents an overview of PRNGs and their role in cryptographic systems. Given the breadth and complexity of the topic, the discussion will focus on the fundamental principles, key characteristics, and essential requirements of PRNGs as they relate to secure digital communication, rather than delving into the full depth of the field.

In contrast to True Random Number Generators (TRNGs)—which derive randomness from inherently unpredictable physical phenomena such as thermal noise or radioactive decay—PRNGs rely on deterministic mathematical algorithms. These algorithms generate sequences of numbers that appear random but are entirely determined by an initial input known as the seed. Consequently, using the same seed will always produce the same sequence, a property that is particularly useful for reproducibility in simulations, testing, and debugging.

PRNGs offer significant advantages in terms of speed, simplicity, and low computational overhead, making them especially suitable for software-based applications that require large volumes of random numbers. However, their use in cryptographic systems introduces strict security requirements. Predictability in random number sequences can lead to serious vulnerabilities, so cryptographically secure PRNGs (CSPRNGs) must ensure that their outputs are computationally indistinguishable from true randomness and remain secure even if the algorithm is publicly known.

2. Definition of Random Number Generators

Random number generators (RNGs) are fundamental tools in many computational domains, especially in cryptography, where unpredictability is a key requirement. RNGs can be broadly classified into two categories: True Random Number Generators (TRNGs), which derive randomness from physical processes, and Pseudo-Random Number Generators (PRNGs), which use deterministic algorithms to simulate randomness.

While both types play important roles, PRNGs are more widely used in practice,

particularly in software-based cryptographic systems, due to their efficiency, repeatability, and ease of implementation. The following subsections outline the key differences between TRNGs and PRNGs, with a primary focus on the structure, operation, and relevance of PRNGs in secure digital environments.

2.1. Pseudo-Random Number Generators (PRNGs)

A Pseudo-Random Number Generator (PRNG) is a deterministic algorithm designed to produce sequences of numbers that simulate the properties of truly random numbers. While the output of a PRNG may appear statistically random, it is completely determined by an initial input value known as the seed. Because of this deterministic nature, PRNGs are often referred to as algorithmic random number generators.

PRNGs are widely used in computer science, cryptography, simulations, statistical sampling, and procedural content generation. In cryptographic applications in particular, PRNGs must satisfy strict criteria such as unpredictability, uniform distribution, and long periodicity. A predictable PRNG can lead to critical vulnerabilities in security systems, especially if an attacker can reconstruct the seed or determine part of the internal state of the generator [1].

The Following figure presents the basic PRNG scheme that shows the essential components:

- **Seed**: Initial input value (S_0).
- **Deterministic Algorithm**: Mathematical formula ($X_{n+1} = f(X_n)$).
- **Pseudo Random Output**: The generated sequence.
- **Feedback Loop**: Internal state updates (shown as dashed line).

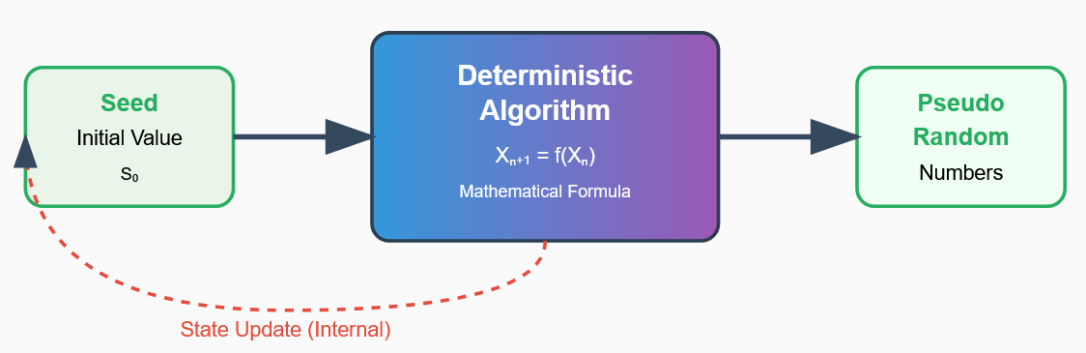


Figure.1: Pseudo-Random Number Generator

2.1.1. Characteristics of PRNGs in Cryptographic Applications

In cryptography, the quality of random number generation is of paramount importance. Weak or predictable randomness can compromise the confidentiality, integrity, and authenticity of entire systems. While many pseudo-random number generators (PRNGs) are sufficient for general-purpose tasks such as simulations or games, cryptographic applications require far more stringent standards. For such purposes, a specialized class known as Cryptographically Secure Pseudo-Random Number Generators (CSPRNGs) is employed.

A CSPRNG must fulfill several critical criteria to ensure the security of cryptographic protocols:

- **Unpredictability**: The foremost requirement of a CSPRNG is that its output must be computationally unpredictable. An adversary, even with partial knowledge of past or present output, should be unable to infer any other part of the sequence. This includes both forward secrecy (preventing prediction of future values) and backward secrecy (preventing reconstruction of past values) [1].
- **Seed Confidentiality and High Entropy Sources**: Although CSPRNGs are deterministic, their security relies heavily on the secrecy and unpredictability of the initial seed. If the seed is exposed, the output becomes predictable. Therefore, seeds must be sourced from high-entropy mechanisms such as hardware-based true random number generators (TRNGs) or operating system entropy pools [3].
- **Uniform Distribution and High Entropy Output**: The output of a secure CSPRNG should be statistically uniform, meaning each value in the output space has approximately equal probability. Biases or low entropy in the output can provide attackers with exploitable patterns [2].
- **Long Periodicity**: A robust CSPRNG should have a sufficiently long period—the length of the sequence before repetition occurs—so that repetition is computationally infeasible during the expected operational lifespan of the system [4]
- **Resistance to Internal State Compromise**: Even if part of the internal state is revealed, the generator must maintain forward and backward security. Some CSPRNGs implement state rekeying or frequent reseeding strategies to mitigate this risk [3].
- **Efficiency and Practicality**: Despite strong security requirements, a CSPRNG must also be computationally efficient and implementable on resource-constrained platforms such

as smart cards, embedded systems, and IoT devices. Security should be balanced with performance in real-world deployment scenarios [5].

- **Validation Through Statistical Testing**: The quality of randomness must be verified using comprehensive test suites. Commonly accepted standards include NIST SP 800-22, Diehard and Dieharder, and TestU01, all of which assess the randomness, uniformity, and independence of generated sequences [6].

2.2. True Random Number Generators (TRNGs)

True Random Number Generators (TRNGs) derive their randomness from unpredictable physical phenomena, fundamentally differing from the deterministic algorithms you've previously discussed regarding PRNGs. TRNGs harvest entropy from natural processes such as thermal Johnson noise in resistors, where random electron movement generates voltage fluctuations proportional to temperature and resistance values according to the Nyquist-Johnson theorem [7]. Other entropy sources include shot noise in semiconductor junctions, where discrete electron tunneling events create random current variations, and avalanche noise in reverse-biased diodes where chaotic electron multiplication produces unpredictable electrical signatures [8]. Quantum mechanical phenomena provide the highest quality randomness, with photon arrival times at photodetectors, quantum tunneling events, and radioactive decay processes offering true non-deterministic entropy that cannot be predicted even with complete knowledge of the system's physical state [9].

The diagram in the following figure illustrates the basic scheme of a True Random Number Generator (TRNG).

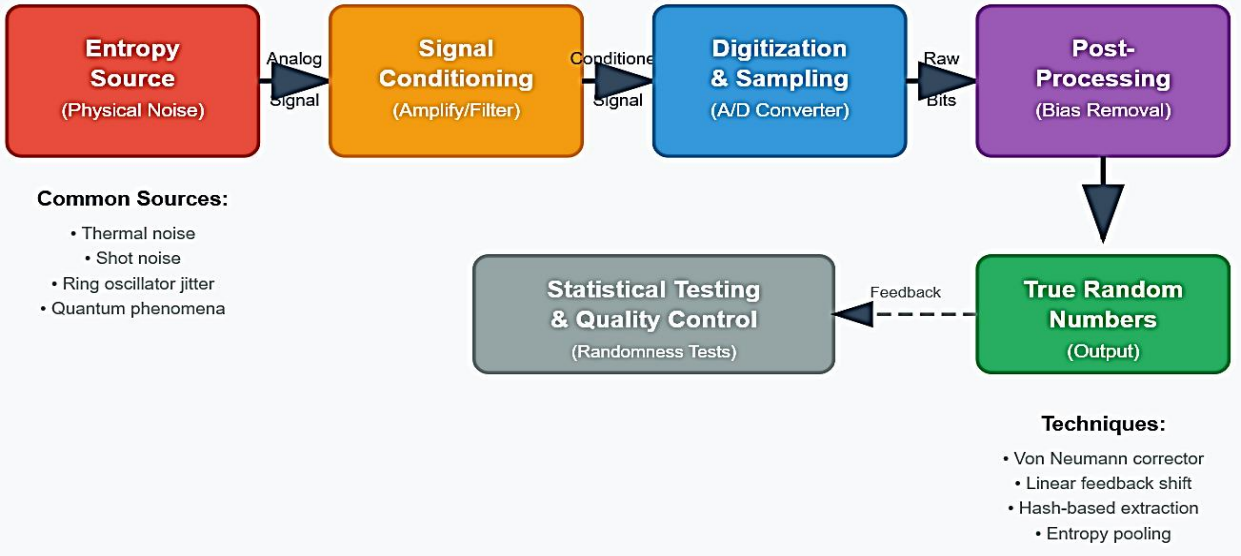


Figure.2: The basic scheme of a True Random Number Generator (TRNG).

The diagram above consists of four main stages:

1. **Entropy Source**: A physical process that generates truly random signals (thermal noise, quantum effects, atmospheric noise, etc.)
2. **A/D Converter**: Samples the analog random signal and converts it to digital bits
3. **Signal Conditioning**: Amplifies and filters the digital signal to improve quality
4. **Post-Processing**: Removes bias and improves statistical properties using techniques like Von Neumann correction or hash functions.

The diagram also shows a quality control feedback loop that monitors the output through statistical testing to ensure the randomness meets required standards. The key difference from Pseudo Random Number Generators (PRNGs) is that TRNGs rely on unpredictable physical phenomena rather than deterministic algorithms, making them truly non-reproducible and suitable for high-security applications.

The hardware implementation of TRNGs requires careful engineering to extract and digitize analog noise sources while maintaining signal integrity and preventing environmental interference. Analog-to-digital conversion introduces quantization challenges, as the continuous noise signals must be properly sampled and conditioned to produce unbiased binary outputs [10]. Post-processing techniques like von Neumann bias correction, linear feedback shift register (LFSR) whitening, and cryptographic hash functions are often employed to remove statistical bias and improve the uniform distribution of output bits. Quality control mechanisms continuously monitor entropy sources for degradation, environmental tampering, or hardware failures that could compromise randomness quality, with many commercial TRNGs implementing real-time statistical health checks and automatic shutdown procedures when entropy quality falls below acceptable thresholds [11].

The applications demanding TRNGs extend beyond basic cryptographic key generation to include high-security environments where the unpredictability guarantee is paramount. Quantum key distribution systems rely on quantum TRNGs to ensure information-theoretic security, while hardware security modules in banking and government applications use TRNGs for generating master keys, certificate authorities' root keys, and other high-value cryptographic material [12]. Scientific applications including Monte Carlo simulations requiring highest statistical quality, gambling systems needing regulatory compliance for fairness certification, and research in fundamental physics studying random processes all benefit from the genuine unpredictability

that only TRNGs can provide, making them irreplaceable despite their complexity and cost compared to the algorithmic alternatives you've explored.

3. Examples of Existing Pseudorandom Number Generators

Pseudorandom number generators (PRNGs) form the backbone of computational applications requiring random numbers, from Monte Carlo simulations to cryptographic systems. Over decades of research and development, numerous algorithms have emerged, each with distinct characteristics, strengths, and weaknesses. This section examines prominent examples of PRNGs across different categories, analyzing their mathematical foundations, implementation considerations, and practical applications.

3.1. Linear Congruential Generators

Linear Congruential Generators (LCGs) (Figure.3) represent the oldest and most widely studied class of PRNGs, with their simplicity making them attractive for educational purposes and basic applications. The general form follows the recurrence relation $x_{n+1} = (ax_n + c)\ mod\ m$, where *a* is the multiplier, c is the increment, and m is the modulus [13].

The historical significance of LCGs cannot be overstated. Park and Miller's "minimal standard" generator, defined by $x_{n+1} = (16807\ x_n)\ mod\ (2^{31} - 1)$, became widely adopted despite its known limitations [14]. This generator, with multiplier a = 16807 and modulus m = $2^{31} - 1$, offered reasonable performance for many applications while maintaining simplicity of implementation. However, subsequent analysis revealed significant weaknesses in its randomness quality, particularly in higher-dimensional projections.

The Numerical Recipes generator, $x_{n+1} = (1664525\ x_n + 1013904223)\ mod\ (2^{32})$, exemplifies the pitfalls of poor parameter selection in LCGs [15]. While computationally efficient, this generator exhibits severe statistical deficiencies and has been strongly discouraged for serious applications. Its continued appearance in some software libraries demonstrates the persistence of legacy code and the importance of understanding generator quality.

Modern implementations have largely moved beyond simple LCGs due to their fundamental limitations. The linear nature of these generators makes them vulnerable to prediction attacks in cryptographic contexts, while their relatively short periods and poor multidimensional uniformity properties limit their utility in sophisticated simulations. Nevertheless, LCGs remain

valuable for understanding PRNG principles and serve as building blocks in more complex algorithms.

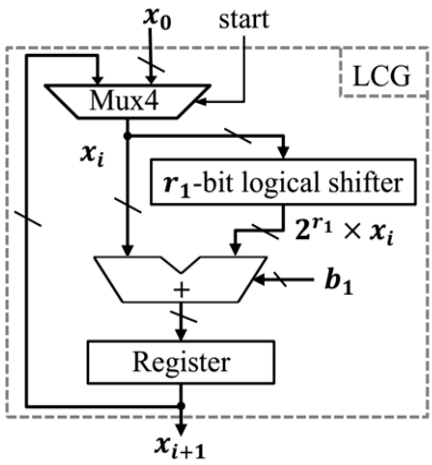


Figure.3: The linear congruential generator's architecture.

3.2. Mersenne Twister

The Mersenne Twister, developed by Matsumoto and Nishimura, represents a landmark achievement in PRNG design, addressing many limitations of earlier generators while maintaining computational efficiency [16]. The algorithm generates 32-bit integers with a period of 2^{19937} - 1, a Mersenne prime that gives the generator its name.

The Mersenne Twister operates through a twisted generalized feedback shift register, combining multiple linear recurrences to achieve superior statistical properties. The algorithm maintains an internal state array of 624 32-bit integers, using a complex tempering function to produce output values. This design ensures excellent equidistribution properties across multiple dimensions, making it particularly suitable for Monte Carlo simulations requiring high-dimensional uniformity.

The MT19937 variant became the default random number generator in numerous programming languages and scientific computing environments, including Python's random module, MATLAB, and R [17]. Its widespread adoption stems from its combination of speed, long period, and excellent statistical properties across most standard randomness tests. The generator passes virtually all tests in the TestU01 battery (see next section), with only a few specialized tests revealing weaknesses.

However, the Mersenne Twister exhibits several notable limitations. Its large state size (2.5 KB) can be problematic in memory-constrained environments. The generator's initialization

process requires careful attention, as poor seeding can lead to extended periods of poor randomness quality. More critically, the algorithm's linear nature makes it unsuitable for cryptographic applications, as knowledge of 624 consecutive outputs allows complete prediction of future values.

Variants of the Mersenne Twister have addressed some limitations while preserving its core strengths. The SFMT (SIMD-oriented Fast Mersenne Twister) leverages modern processor SIMD instructions to achieve significantly improved performance [18]. The dSFMT variant produces double-precision floating-point outputs directly, eliminating conversion overhead in applications requiring real-valued random numbers.

3.3. Xorshift Family

George Marsaglia's xorshift generators introduced a remarkably simple yet effective approach to PRNG design, using combinations of shift and XOR operations to achieve good statistical properties with minimal computational overhead [19]. The basic xorshift algorithm operates on a single state variable, applying a sequence of left shifts, XORs, and right shifts to generate output values.

The original xorshift generator, with its characteristic triplet of shift values (13, 17, 5), demonstrated that complex mathematical operations are not necessary for producing high-quality pseudorandom sequences. This simplicity translated to exceptional performance on modern processors, making xorshift generators particularly attractive for applications requiring high-speed random number generation.

Sebastiano Vigna's extensive development of the xorshift family has produced numerous variants optimized for different requirements [20]. The xorshift128+ generator combines two xorshift operations with addition to improve statistical properties while maintaining speed. The xoshiro256** (XOR-shift-rotate) represents the current state-of-the-art in this family, offering excellent performance and statistical quality comparable to the Mersenne Twister with significantly reduced state size.

The xorshift family's evolution demonstrates the ongoing refinement of PRNG design. Early variants failed certain statistical tests, particularly those examining binary rank properties. Subsequent improvements, including the addition of rotation operations and more sophisticated state manipulation, have largely eliminated these weaknesses while preserving the family's computational advantages.

3.4. PCG Family

Melissa O'Neill's Permuted Congruential Generator (PCG) family represents a significant innovation in PRNG design, combining the mathematical foundations of linear congruential generators with sophisticated output permutation functions [21]. This approach addresses the traditional weaknesses of LCGs while maintaining their speed and simplicity.

The PCG algorithm operates by applying a permutation function to the output of an underlying LCG, effectively scrambling the linear patterns that plague traditional congruential generators. The permutation function typically involves bit rotation by a value derived from the generator's state, creating output sequences that exhibit excellent statistical properties across multiple dimensions.

PCG generators offer several advantages over traditional approaches. Their small state size (typically 128 bits) makes them suitable for embedded applications and situations requiring many independent streams. The algorithm's design allows for efficient generation of random numbers in arbitrary ranges without the modulo bias that affects simpler methods. Additionally, PCG supports multiple output functions, allowing optimization for specific applications.

The PCG family has gained recognition for its balance of speed, statistical quality, and theoretical soundness. The generators pass comprehensive statistical tests while maintaining performance competitive with simpler algorithms. This combination has led to adoption in various software libraries and applications requiring high-quality random numbers without the overhead of more complex generators.

3.5. ChaCha20-based Generators

The ChaCha20 stream cipher, designed by Daniel Bernstein, has found unexpected application as the foundation for cryptographically secure PRNGs [22]. Unlike traditional PRNGs optimized purely for speed and statistical properties, ChaCha20-based generators provide cryptographic security, making them suitable for applications requiring unpredictable random numbers.

The ChaCha20 algorithm operates on a 512-bit state, using quarter-round operations that combine addition, XOR, and rotation to achieve thorough diffusion. When used as a PRNG, the algorithm generates random bytes by encrypting a counter stream, ensuring that output prediction requires solving the underlying cryptographic problem.

Modern operating systems have increasingly adopted ChaCha20-based generators for their primary random number sources. Linux's /dev/urandom switched to a ChaCha20-based design in kernel version 4.8, while FreeBSD and other systems have made similar transitions [23]. This trend reflects the growing recognition that the performance penalty of cryptographic generators has become acceptable for most applications, while the security benefits are substantial.

The cryptographic nature of ChaCha20-based generators provides forward secrecy, meaning that compromise of the generator's state does not reveal previously generated values. This property is crucial for security-sensitive applications, including key generation, nonce creation, and other cryptographic operations where predictability would compromise security.

3.6. Specialized and Modern Developments

Recent years have witnessed the development of specialized PRNGs targeting specific application domains and emerging computational paradigms. The AES-based generators leverage the Advanced Encryption Standard's widespread hardware support to achieve high-speed cryptographically secure random number generation [24]. These generators are particularly attractive in environments where AES hardware acceleration is available.

The ISAAC (Indirection, Shift, Accumulate, Add, and Count) generator, designed by Robert Jenkins, represents an attempt to combine cryptographic security with PRNG performance [25]. While not subjected to the same rigorous cryptanalytic scrutiny as dedicated cryptographic primitives, ISAAC offers reasonable security properties with performance superior to traditional cryptographic generators.

GPU-based random number generation has emerged as a critical requirement for parallel computing applications. Generators like CURAND (NVIDIA's CUDA random number library) and MRG32k3a (Multiple Recursive Generator) have been specifically optimized for massively parallel environments, addressing challenges such as ensuring independence between parallel streams and maintaining performance under GPU memory constraints [26].

Quantum computing's emergence has sparked interest in quantum random number generators (QRNGs), which derive randomness from quantum mechanical processes rather than deterministic algorithms. While not PRNGs in the traditional sense, these generators represent a fundamental shift toward true randomness for applications where cryptographic security is

paramount.

4. Performance and Implementation Considerations

The practical implementation of PRNGs involves numerous considerations beyond theoretical properties. Memory usage varies dramatically between algorithms, from the minimal state requirements of simple LCGs to the substantial memory footprint of the Mersenne Twister. This factor becomes crucial in embedded systems, GPU kernels, and applications requiring many independent generator instances.

Initialization and seeding procedures significantly impact generator performance and quality. Poor seeding can lead to correlation between ostensibly independent streams or extended periods of poor randomness quality. Modern generators often incorporate sophisticated seeding mechanisms that help mitigate these risks, but proper understanding and implementation remain critical.

Portability across different hardware architectures and compiler optimizations can significantly affect generator performance. Some algorithms, particularly those relying on specific bit widths or operation sequences, may exhibit unexpected behavior when ported between platforms. This consideration has led to the development of standardized implementations and test suites to ensure consistent behavior across environments.

5. Statistical Tests of Randomness

To ensure the quality and security of random number generators, especially in cryptographic contexts, their output must undergo rigorous statistical testing. These statistical tests of randomness are essential for evaluating whether a sequence of numbers exhibits the properties expected of a truly random source. In practice, no sequence generated by a deterministic process (such as a PRNG) can be truly random, but it must be *indistinguishable* from a random sequence based on statistical properties.

5.1. Purpose and Importance

Statistical testing provides a practical means to assess whether the output of a random number generator is suitable for its intended application. For cryptographic systems, where predictability could lead to catastrophic vulnerabilities, randomness must not only appear

uniform but must also **resist pattern detection**, **bias**, and **correlation**. Therefore, statistical tests do not guarantee randomness but help **identify weaknesses** or anomalies in the generator's output.

As noted by Rukhin [06], statistical testing is especially critical in verifying compliance with standards such as NIST's recommendations for cryptographic applications, ensuring that a generator does not consistently produce patterns that could be exploited by an adversary.

5.2. Commonly Used Test Suites

A number of well-established statistical test suites have been developed to evaluate the randomness of sequences:

- **NIST SP 800-22**: Developed by the National Institute of Standards and Technology, this suite is widely used in both academic and industrial settings. It includes 15 statistical tests, such as the Frequency (Monobit) Test, Runs Test, Approximate Entropy Test, and the Linear Complexity Test, among others. These tests examine various aspects of randomness, including uniformity, independence, and complexity [6].
- **Diehard and Dieharder Test Suites**: The Diehard suite, created by George Marsaglia, includes tests such as the Birthday Spacings Test, Overlapping Permutations Test, and Rank of Matrices Test. Dieharder is a more modern and extensible version of Diehard, supporting additional tests and better interfaces [27].
- **TestU01**: Developed by Pierre L'Ecuyer and Richard Simard, TestU01 is a comprehensive and highly regarded framework for testing random number generators. It offers a broad set of batteries (e.g., SmallCrush, Crush, and BigCrush) that apply numerous tests sequentially, providing detailed statistical analyses of generator behavior [28].

5.3. Additional Test Frameworks

Besides the well-known suites above, several other tools and standards are employed to evaluate randomness in specific contexts. The **ENT tool** offers a lightweight method for analyzing entropy, mean, chi-square, and serial correlation, making it useful for quick checks, though it lacks cryptographic rigor. The **FIPS 140-2/140-3** standards include basic tests—such as the monobit and poker tests—that are still used for hardware validation.

For more comprehensive and adaptive testing, **PractRand** is a modern, high-performance framework capable of testing long binary sequences in real-time, making it suitable for large-

scale systems and stream ciphers. Additionally, standards like **AIS 31** (used in Germany) and **STS-2** extend NIST-style testing for true random number generators and include criteria for online health testing of entropy sources [29].

5.4. Interpretation of Results

The outcome of each test is typically a p-value, which represents the probability that a truly random sequence would produce a result at least as extreme as the observed one. If the p-value lies outside the acceptable range (usually [0.01, 0.99]), the sequence may be considered non-random in that specific aspect. However, a single failed test does not necessarily invalidate the generator—it may indicate the need for further investigation or larger sample sizes.

In cryptographic applications, consistently passing a battery of statistical tests is a necessary but not sufficient condition for a secure PRNG. Statistical tests evaluate observable randomness but do not address deeper cryptanalytic properties such as state compromise resilience or unpredictability [5].

5.5. Limitations of Statistical Testing

While statistical testing is indispensable, it has inherent limitations:

- It cannot prove that a sequence is random—only that it does not exhibit detectable non-randomness.
- Passing all known tests does not guarantee resistance to future attacks.
- Results can vary based on sequence length, test parameters, and computational precision.

Hence, statistical tests should be used in conjunction with theoretical analysis and cryptographic design principles to evaluate the security of a PRNG.

6. Conclusion

This chapter presented an overview of Pseudo-Random Number Generators (PRNGs) and their vital role in modern computing, especially in cryptography. It distinguished PRNGs from True Random Number Generators (TRNGs), emphasizing the deterministic nature of PRNGs and the importance of secure seed management. While PRNGs offer advantages like speed and reproducibility, cryptographic applications require more stringent guarantees, leading to Cryptographically Secure PRNGs (CSPRNGs).

We discussed essential CSPRNG properties such as unpredictability, uniformity, long periods, and state compromise resistance, along with examples of common algorithms like LCGs, Mersenne Twister, xorshift, PCG, and ChaCha20-based generators. The importance of statistical testing was also highlighted, with tools like NIST SP 800-22, Dieharder, and TestU01 used to evaluate output quality.

In conclusion, while PRNGs are indispensable for generating scalable and efficient randomness, their use in cryptography demands rigorous design and testing. This chapter provides the groundwork for further exploration of secure and practical implementations.

CHAPTER 02
FUNDAMENTALS OF CHAOS

1. Introduction

We are surrounded by a lot of things that appear erratic and haphazard in both space and time. Because there are so many factors at play, attempting to determine the cause of these events is typically futile; as a result, one is content to regard the process as noise.[30]

However, in recent decades, a variety of systems have been explored that exhibit unpredictable behavior in spite of their apparent simplicity and the fact that the forces at play are subject to well-established physical principles. These systems all share a very high degree of sensitivity to the initial conditions and the manner in which they are started. For instance, the meteorologist Edward Lorenz found that even a basic model of heat convection had inherent unpredictability. He named this phenomenon the "butterfly effect," which implies that a butterfly's wing movement alone can alter the weather [31].

This sensitivity can be technically defined as the divergence of the system's paths. A system that begins in one state gradually loses similarity (moves farther and farther away in state space) to a system that begins in a similar but distinct state. As a result, over time, the system's actual trajectory will deviate from the one that was projected based on the measurements, even though the measurements may be close to the system's true condition. This implies that a forecast can be made with greater accuracy the more precisely the beginning situation is known. The issue is that in many real-world situations, it is impossible to achieve the level of accuracy required to produce forecasts that are meaningful [32].

Scientists started using nonlinear dynamics and chaos properties for particular purposes in the early 1990s. The use of chaos in signal compression has also been studied, and chaos management methods have emerged. In signal processing, chaotic signals and noise with the same frequency range can be separated using optimization strategies designed to minimize noise. Two chaotic systems with the same set of parameter values can synchronize with one another, according to research by Pecora and Carroll. This finding has significant ramifications for information technology applied research that promotes chaos [33].

According to the previous discussion, "chaos" refers to a condition of disorder and unpredictability. In actuality, there is no specific mathematical definition of what "chaos" actually implies, despite the extensive research in the field of chaos theory. To state, however, that chaos is "aperiodic long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions," it should have the following qualities according to common sense:

- Aperiodic long-term behaviour: The system trajectories do not settle down to any fixed points, periodic orbits, or quasiperiodic orbits as $t \rightarrow \infty$, according to this. As a result, the subsequent trajectory will be less predictable.
- Deterministic system: This type of system lacks stochastic input parameters or is not random. Rather than noise, chaotic systems' inherent non-linearity is the cause of their erratic behavior.
- Sensitivity to initial conditions: This indicates that the system has a positive Lyapunov exponent, meaning that even if the trajectories begin from extremely close initial conditions, they will separate exponentially quickly. Long-term prediction is thus rendered unattainable [34].

There are numerous instances of nonlinear equations with few variables that behave chaotically, which we will discuss in more depth later.

2. Origins of chaos

The first person to uncover unpredictability within deterministic systems was the French mathematician Henri Poincaré. In the early 20th century, while studying the stability of the solar system, he found that even tiny errors in measuring initial conditions in multi-body systems could lead to enormous unpredictability—far exceeding what traditional mathematics would anticipate [35].

With the introduction of computers, chaos theory gained more attention. In 1963, American meteorologist Edward Lorenz discovered chaotic behavior while developing a simplified model for atmospheric convection. When he re-ran a simulation with slightly altered initial conditions, the output changed drastically. This phenomenon became known as the "butterfly effect"—a metaphor suggesting that something as small as the flap of a butterfly's wings in one location could trigger a tornado halfway across the world.

In chaos theory, the "butterfly effect" has come to represent the sensitive dependence of nonlinear deterministic systems on initial conditions, where small variations can cause significant long-term changes [36].

Another well-known example of chaotic behavior is the logistic map—a nonlinear first-order difference equation originally formulated in 1838 by Pierre François Verhulst to model biological populations. It gained broader recognition in the 1920s, when it was linked to unpredictability in numerical simulations.

A major breakthrough in the field came with the discovery that chaos could be controlled and, more specifically, synchronized between systems. This opened up wide applications in science and technology. In the field of data security, a key milestone was the work by Pecora and Carroll in 1990, who demonstrated that two identical chaotic systems could be synchronized—an important step toward chaos-based cryptography [37].

Today, chaos theory remains a vibrant field of study. Numerous research papers are published every month in various disciplines, including biology, chemistry, mathematics, mechanics, and electronics. Among the most active applications is cryptography, which increasingly leverages the properties of chaotic dynamics.

2.1. The butterfly effect

The butterfly effect is a foundational concept in chaos theory that illustrates the sensitive dependence on initial conditions within nonlinear dynamical systems. The term was popularized by meteorologist Edward Lorenz, who, during a computer simulation of weather patterns in 1961, discovered that a tiny rounding error—changing a number from 0.506127 to 0.506—resulted in a completely different weather outcome. This unexpected divergence demonstrated that infinitesimally small variations in the starting state of a system could evolve into drastically different trajectories over time. The metaphor of a butterfly flapping its wings in Brazil causing a tornado in Texas encapsulates this idea: while the butterfly itself doesn't literally cause the tornado, the system is so sensitive that minute inputs can trigger chain

reactions leading to large-scale consequences. This challenges the Newtonian paradigm of deterministic predictability, where knowing the initial conditions with high accuracy supposedly guarantees accurate future predictions. In reality, most natural systems—especially those governed by complex feedback loops like the climate, ecosystems, or even human societies—are fundamentally unpredictable in the long term due to this sensitivity. The butterfly effect has profound scientific implications: it explains the limits of weather forecasting, supports the development of probabilistic models, and justifies the use of robust control techniques in engineering. Philosophically, it raises questions about causality, free will, and the nature of time, suggesting that even in deterministic systems, true long-term control is an illusion. In essence, the butterfly effect reminds us that complexity and unpredictability are not the results of randomness, but rather the natural outcomes of deeply interwoven causal systems.

2.2. The route to chaos

The transition to chaos refers to the gradual progression of a dynamical system from orderly behavior to chaotic behavior as a key system parameter is varied. This phenomenon is most commonly observed in nonlinear systems, where small changes in a control parameter—such as population growth rate, electrical input, or fluid velocity—can cause the system to shift from stable fixed points to periodic oscillations, and eventually to fully chaotic, aperiodic dynamics. One of the most studied models demonstrating this transition is the logistic map, given by the equation:

$$x_{n+1} = \mu x_n(1 - x_n)$$

Where increasing the value of μ leads to a cascade of period-doubling bifurcations—each doubling the period of the system's cycles—before entering a chaotic regime. This route to chaos, known as the Feigenbaum scenario, is universal in many systems and characterized by precise mathematical constants. The passage to chaos is not merely a mathematical curiosity; it is observed in real-world systems such as fluid turbulence, cardiac rhythms, chemical reactions, and ecological populations. Understanding this transition is crucial in various fields, as it allows scientists to predict the onset of instability and design mechanisms to control or delay chaotic behavior. The study of this process provides a bridge between deterministic equations and unpredictable outcomes, highlighting how complex behaviors can emerge naturally from simple rules under the right conditions.

2.3. Fixed points

Since the fixed points of $f_r(x_k)$ are obtained by creating $x_k = r.x_k(1 - x_k)$, they are readily discovered and are as follows: $x = 0 \ and \ x = 1 - 1/r$ .Let's now calculate the derivative of

$f_r(x_k)$ at the following locations:

$$f_r'(x_k) = r(1 - 2x), so\ f_r'(0) = r\ and\ f_r'\left(1 - \frac{1}{r}\right) = 2 - r.$$

The only fixed point for $f_r(x_k)$ in [0,1] for (0 < r < 1) is 0 since $f_r(1-\frac{1}{r})$ and is out of [0,1]. These findings allow us to conclude that $x_k = 0$ is an attractor for 0 < r < 1. Similarly, for 1 < r < 3, it can be demonstrated that regardless of the initial condition$x_k \epsilon [0,1]$, all of the results of (1.3) trend to the basin of attraction on 1-1/r. Experimental findings can be displayed on Fig. 2, where r= 2 and $x_0 = 0.1$.

2.4. Periodic and quasi-periodic regimes

When r is little over 3, additional intersection points with the graph $x = y$ occur for $r = 3$, $f_r'\left(1-\frac{1}{r}\right) = -1$,In addition to the fixed points (0 and 1-1/r), there is a 2-cycle period. A new period of four cycles is created as r increases further. To create a period-doubling cascade, this technique is repeated.

2.5. Chaotic regime

The number of periods will be difficult to discern for$3.6 \leq r \leq 4$, and the LM result will look chaotic; this regime is achieved with considerable sensitivity to the initial condition x_0 ,the Figure 3 illustrates the distinction between two x_k produced with two extremely close starting conditions ($x_0 = 0.1\ and\ x_k' = 0.1 + 10^{-15}$). From what has been observed, it is evident that the parameter {plays a significant part in regulating the LM's behavior from a regular and stable state to one that is chaotic, unstable, and extremely sensitive to the initial conditions.

2.6. Bifurcation diagram

As we alter the parameter, the LM exhibits a range of behaviors with transitions between them. A bifurcation diagram is a visual representation of these transformations, which are known as bifurcations in dynamical systems [38]. A bifurcation diagram is therefore a helpful method that displays, when a function's values are approached or visited asymptotically (fixed points, periodic orbits, or chaotic attractors) in relation to the control parameters. The bifurcation diagram of the LM is shown in Fig. 3. The vertical axis displays the set of values of x_k visited asymptotically from nearly all initial circumstances by the iterates of the logistic equation with that r value, while the horizontal axis displays the potential values of the parameter `.Figure 3 makes the period doubling scenario very evident. For $2.5 < r < 3$, we have a fixed point, for $3 < r < 3.4$, we have an obvious periodic solution with two cycles; and for $3.4 < r < 3.5$, we have a quasi-periodic solution with four cycles. Consequently, the graphic demonstrates period doublings from 1 to 2 to 4 and so on, until the chaotic regime occurs, where the number of periods is indistinguishable.

2.7. Poincaré sections

Plotting the sites in phase space where the trajectory's phase point punctures one side of a plane (as time increases) yields Poincaré sections. The dimension of the generated map is one smaller than the comparable phase picture. Even this approach is crucial because it makes it possible to connect discrete mappings with continuous dynamical systems; dynamic chaos researchers frequently employ it.

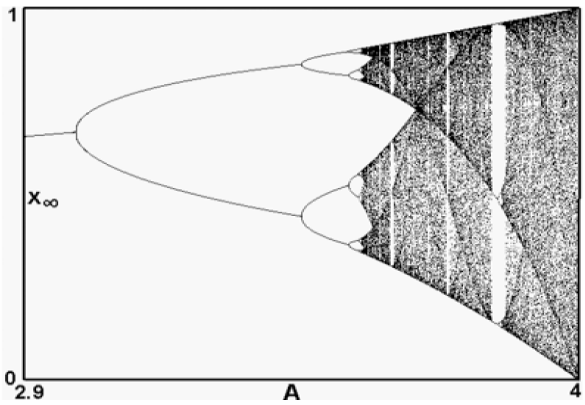


Figure.4: Bifurcation diagram of the Logistic map.

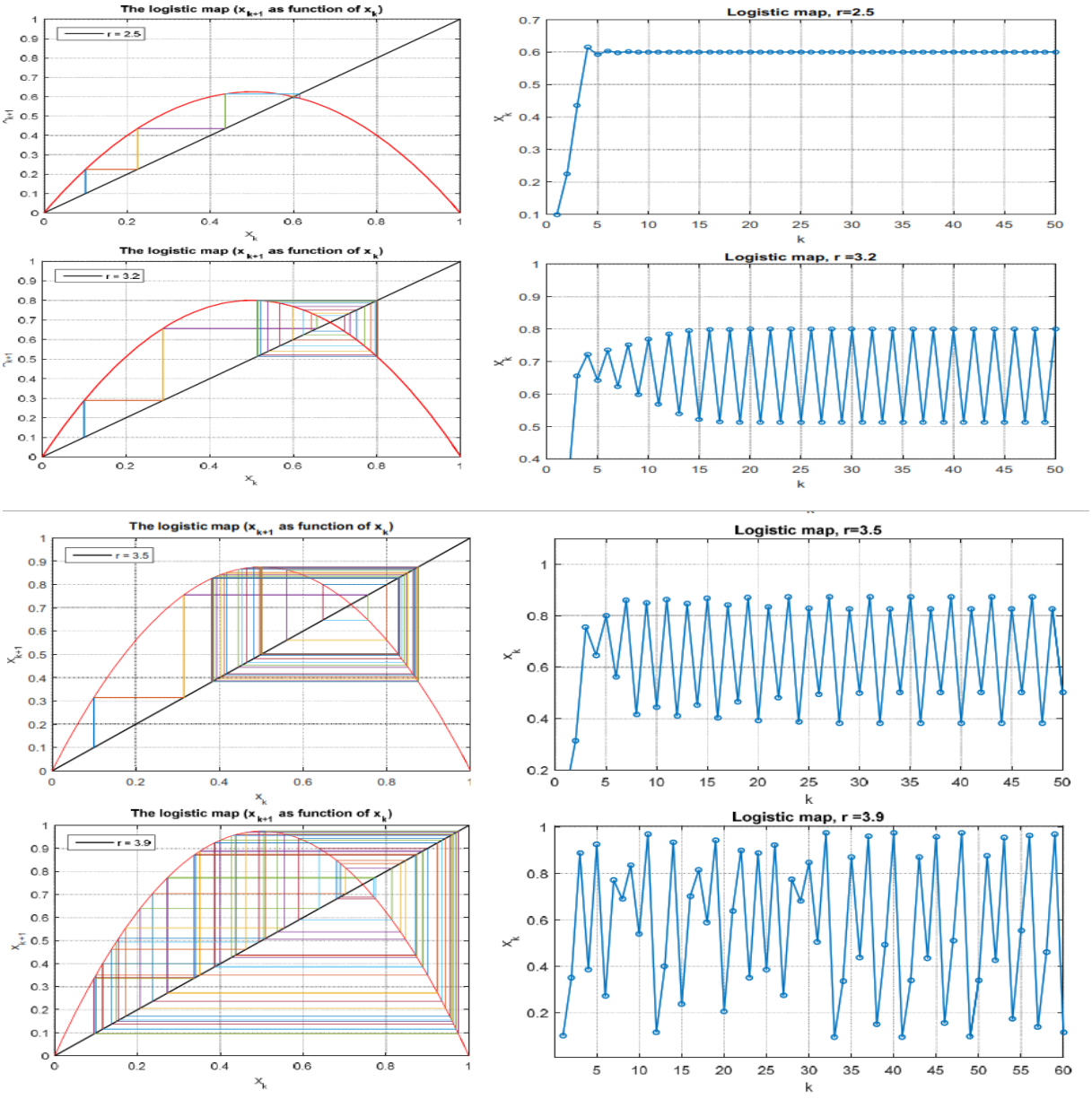


Figure.5: Transition to chaos by period doubling scenario for the Logistic map.

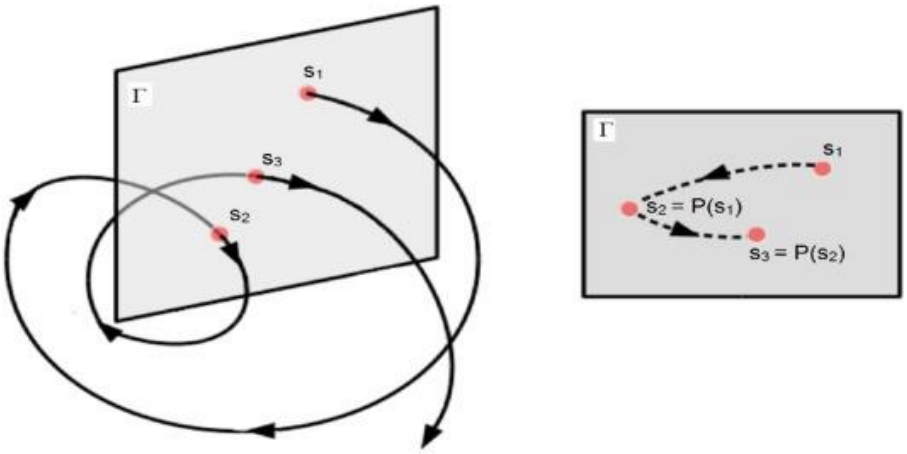


Figure.6: Poincaré section concept.

3. Dynamical systems

A common description of a dynamical system is a deterministic mathematical prescription for advancing the state of a system across time. Here, time can be either a continuous variable or a discrete variable with integer values.

- Differential equations are employed with smooth time evolution by continuous systems:

$$\frac{dx}{dt} = f(x) \qquad (1)$$

3.1. The Lorenz Attractor: Weather and Sensitivity

A well-known example in the world of chaotic attractors is the famous Lorenz's chaotic attractor. As seen before, this meteorologist has developed an idealized model for studying the Earth's atmosphere. His theory involves three variables x, y and z, connected by the following equations:

$$\begin{cases} \frac{dx}{dt} = \sigma * (y - x) \\ \frac{dx}{dt} = r * x - y - x * z \\ \frac{dx}{dt} = x * y - \beta * z \end{cases}$$

These are some of the control parameters. These rather straightforward equations are insufficient to be able to be solved analytically. (X, Y, and Z cannot be expressed explicitly.) Therefore, one of the previously mentioned methods is used for digital resolution. An example of a typical dynamic chaotic system is the Lorenz attractor. Lorenz published his initial research on the calculation of instability and chaotic systems in 1963. Calculators weren't very sophisticated at the time, and Lorenz couldn't simulate more than 500 iterations to solve his system. As a result, only a few trajectory boucles could be described.

These days, computers are so powerful that every staff member may instantly calculate millions of iterations on any given computer.

When the parameters are changed, it is seen that in some cases, the resulting suites appear to be completely chaotic. Using the Rung-Kutta method of order four. The various curves (phase spaces) on top of the attraction are obtained for = 10, = 28, 8/3, and for initial values of 10, [39]

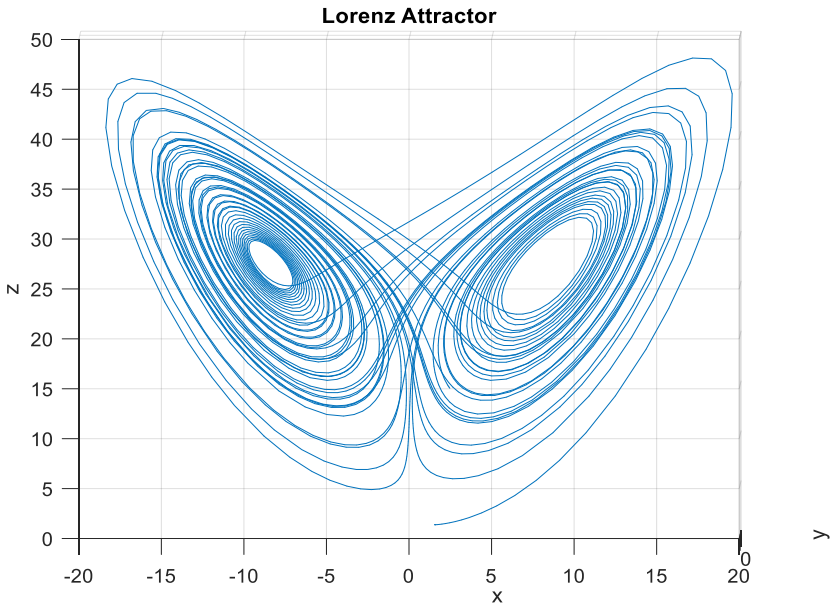


Figure.7 : L'attracteur chaotique de Lorenz

3.2. The Rössler attractor

The Rössler attractor is the attractor connected to the Rössler dynamic system, which consists of three non-linear differential equations. These differentials equations refer to a continuous, three-dimensional dynamic system with chaotic characteristics. Otto Rössler created his attracteur in 1976 with a purely theoretical purpose, however these equations have proven useful in modeling equilibrium in chemical reactions. This system is essential to the ongoing turmoil for at least three reasons: Since he only has one quadratic term and creates a chaotic attractor with one lobe, as opposed to Lorenz's attractor with two lobes, his phase space has a minimum dimension of three like Lorenz's, and his non-linearity is minimal. The equations for this Rössler system are:

$$\begin{cases} \frac{dx}{dt} = -y - z \\ \frac{dy}{dt} = x + ay \\ \frac{dz}{dt} = b + xz - cz \end{cases}$$

Rössler studied the attractor for a = 0,2, b = 0,2, and c = 5,7; however, properties of a = 0,2 , b = 0,2, and c = 14 are now more thoroughly studied. We take these final values. [40]

and the graph will look like this with x=0, y=0, and z=0:

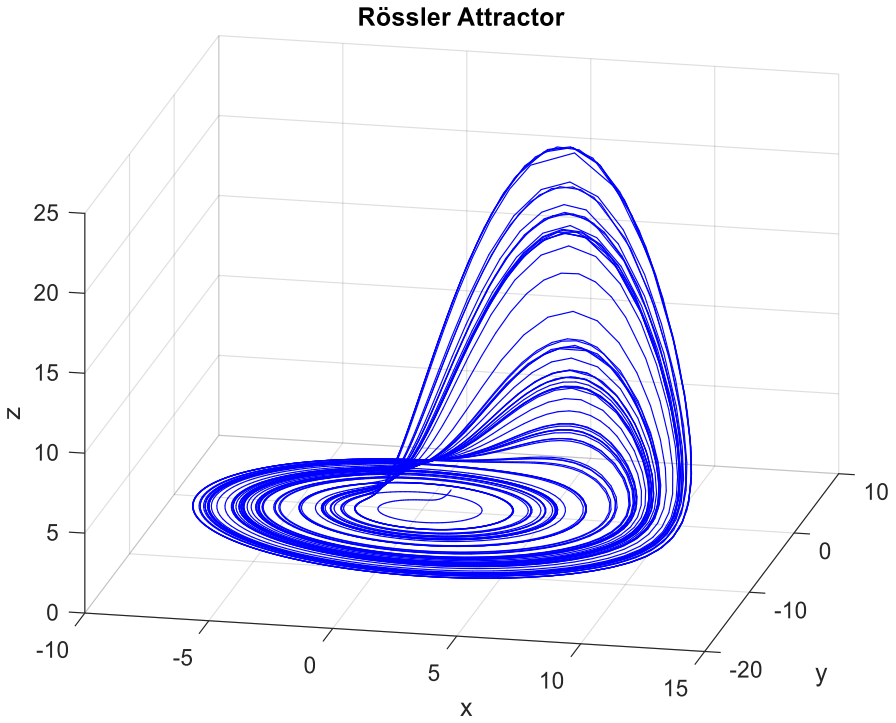


Figure.8: L'attracteur chaotique de Rössler

Discrete systems are based on difference equations only, where time is considered as a sequence of intervals

$$x_{n+1} = f(x_n)$$

3.3. L'attracteur chaotique de Hénon

Michel Hénon is a French astronomer. He had made contact with the chaos in 1962 with his pupil Carl Heiles, who was studying computer paths corresponding to a system of Hamiltonian differential equations (without frottement)

He was aware of Lorenz's attraction in 1976, and by simplifying the equations, about it to the attraction that bears his name. It involves a very basic application, a little similar to a logistical application, but with a plan. These definitions are provided by The following relationships:

$$\begin{cases} \boldsymbol{x_{k+1} = 1 + y_k - a x_k^2} \\ \boldsymbol{x_{k+1} = b x_k} \end{cases}$$

We will assume initial conditions (x0, y0) = (-0.4, 0.3), initial conditions contents in the attraction basin. Additionally, the following values will be adopted: a = 1.4 and b = 0.3. These values were put forth by Michel Hénon and allow to observe a chaotic behavior as depicted in figures 2.15 A and B [40].

The Hénon attractor for the selected values and for 50,000 iterations is provided by The following figure:

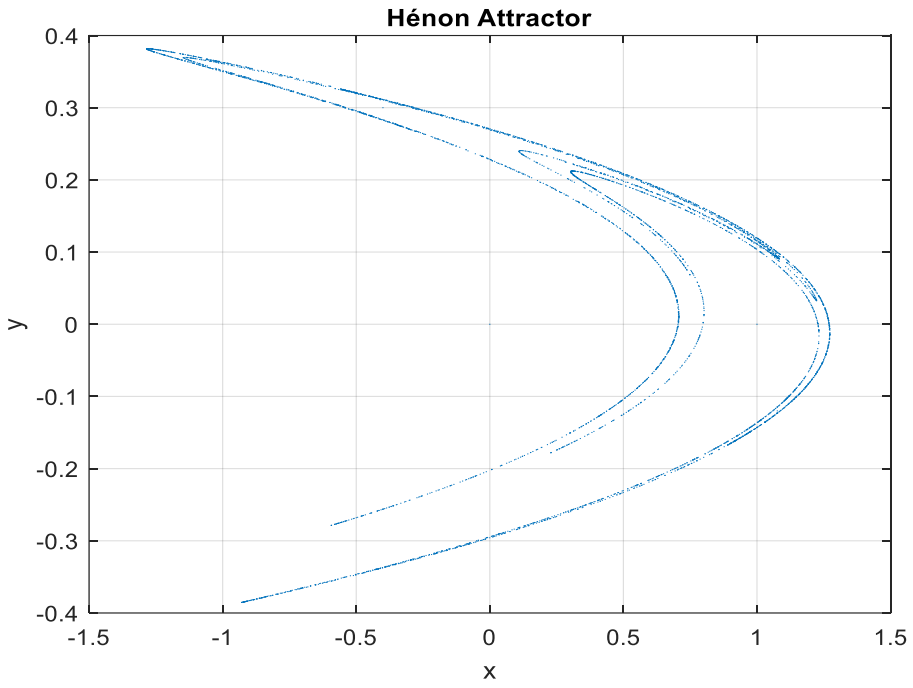


Figure.9: Hénon chaotic map phase space.

4. Characteristics of chaotic systems

4.1. Determinism

Chaotic systems follow specific, precise mathematical laws, meaning they are deterministic. This implies that, in theory, if you know the exact initial conditions, you could predict the system's future behavior. However, despite being deterministic, these systems behave in ways that appear random and unpredictable due to their internal complexity. A well-known example of a deterministic chaotic system is the Lorenz system, which is often used to model weather patterns [42].

4.2. Sensitivity to Initial Conditions

One of the defining features of chaotic systems is their extreme sensitivity to initial conditions, often referred to as the Butterfly Effect. This means that even a tiny difference in the starting state of the system can lead to vastly different outcomes over time. For example, a small change in temperature or pressure can drastically alter the course of weather patterns. This sensitivity makes long-term prediction of chaotic systems practically impossible [43].

4.3. Nonlinearity

Chaotic systems are nonlinear, meaning the relationship between cause and effect is not proportional or straightforward. In nonlinear systems, feedback loops and interactions between variables create complex and unpredictable behavior. Nonlinearity is crucial to chaos because it allows the system to produce erratic and seemingly random patterns from deterministic rules.

An example of this is the Logistic Map, which shows chaotic behavior for certain values of its parameter [44].

4.4. Aperiodicity

Although some systems may exhibit patterns that seem repetitive over short time intervals, chaotic systems do not settle into a regular, repeating cycle. Their behavior is aperiodic, meaning that even if there are short-term oscillations, they will never exactly repeat themselves. This characteristic distinguishes chaotic systems from periodic or quasi-periodic systems, where cycles repeat at regular intervals. The double pendulum is a classic example of an aperiodic chaotic system [45].

4.5. Fractal Structure (Strange Attractors)

Chaotic systems often exhibit strange attractors in phase space — a plot that represents all the possible states of the system. These attractors have a fractal structure, meaning they display self-similarity at different scales and have a non-integer (fractional) dimension. Despite being confined to a bounded region, the trajectories of the system appear to explore this region in a complex, erratic manner. A famous example is the Lorenz attractor, which resembles the wings of a butterfly and illustrates the fractal nature of chaotic systems [46].

4.6. Topological Mixing

Topological mixing refers to the property of chaotic systems where any region of the system's phase space will eventually overlap with every other region over time. This means that the system's state evolves in such a way that it "mixes" thoroughly, ensuring that all possible states are eventually visited. It’s similar to the process of stirring milk into coffee — after a while, the milk is evenly distributed throughout the coffee, no matter where it was originally added [47].

4.7. Long-Term Unpredictability

Although chaotic systems are deterministic, they are unpredictable over the long term due to their sensitivity to initial conditions. Even a tiny error in measuring the initial state can lead to dramatically different outcomes as time progresses. This makes it impossible to predict the system's behavior accurately over extended periods. This property is why weather forecasting and financial modeling can give reliable short-term predictions but struggle with long-term forecasts [48].

5. Examples of some existing chronique systems

Chaotic systems are found across a wide range of disciplines, from physics and biology to economics and cryptography. One of the most iconic examples is the Lorenz system, originally developed to model atmospheric convection, which revealed that small differences in initial conditions could lead to vastly different weather outcomes—a concept now famously known as the butterfly effect. In mechanical systems, the double pendulum illustrates chaos through its highly sensitive and unpredictable motion, despite being governed by Newtonian mechanics. Biological systems also display chaos; for instance, irregular heart rhythms and certain patterns in brain waves (EEG) can exhibit chaotic behavior, especially in pathological conditions like epilepsy. In chemical systems, the Belousov–Zhabotinsky reaction demonstrates non-linear oscillatory dynamics that never repeat, showcasing how chaos can emerge even in closed laboratory settings. Fluid dynamics provides further examples, such as turbulent flows or vortex shedding, where equations like the Navier-Stokes model reveal the intrinsic unpredictability of water and air movement. In the realm of mathematics and ecology, the logistic map is a simple recursive equation that models population growth and exhibits chaotic behavior when its growth parameter exceeds a critical threshold. Even in astronomy, the three-body problem shows how the gravitational interaction between just three celestial bodies leads to chaotic and non-repeating trajectories. Chaos is not limited to natural systems; it plays a key role in cryptography, where chaotic maps are used to design secure pseudo-random number generators. Furthermore, economic models, particularly those involving nonlinear feedback like in stock market behavior, often display chaotic dynamics that defy long-term prediction. These examples underscore how chaos arises not from randomness but from the deep, deterministic complexity embedded in dynamic systems.

5.1. Lyapunov Exponent

The Lyapunov Exponent method often used to determine whether a given system is chaotic or not is the Lyapunov exponent. It gives an indication as to whether two orbits in a chaotic system starting close together diverge or converge. The Lyapunov exponent took its name from the Russian mathematician Alexander Lyapunov (1857-1918), it is a quantity that characterizes the rate of separation of infinitesimally close trajectories:

$$\lambda = \lim_{t\to\infty} \frac{1}{t} \ln \frac{|\delta(t)|}{|\delta(0)|}$$

where:

λ is the Lyapunov Exponent,

$|\delta(0)|$ is the initial small distance between two trajectories in phase space,

$|\delta(t)|$ is the distance between the trajectories after time t.

The value of the Lyapunov Exponent provides insight into the system's behavior:

- If $\lambda>0$, the system exhibits chaotic behavior due to exponential divergence of nearby trajectories.
- If $\lambda<0$, the system is stable, and perturbations decay over time.
- If $\lambda=0$, the system is on the boundary between stability and chaos.

The Lyapunov Exponent is particularly useful in distinguishing chaotic systems from regular ones. It is often plotted as a function of a system's control parameter to visually identify chaotic regions within the parameter space.

5.2. Autocorrelation in Chaotic Systems

Autocorrelation is a statistical measure that describes the degree of similarity between a signal and a delayed version of itself over successive time intervals. In the context of chaotic systems, autocorrelation provides insights into the temporal structure of a system's evolution. While chaotic systems exhibit sensitive dependence on initial conditions and aperiodic behavior, they may still reveal decaying patterns of autocorrelation. This indicates that while short-term states are somewhat correlated, the influence rapidly diminishes with increasing lag, reflecting the loss of predictability inherent in chaotic dynamics. The study of autocorrelation functions in chaotic systems helps distinguish true randomness from deterministic chaos, offering tools to analyze time series data from natural systems such as weather, heart rhythms, and economic indicators.

6. Conclusion

This chapter has explored the profound implications of chaos theory across multiple scientific domains. It began by highlighting how seemingly random phenomena often emerge from simple deterministic systems. Through foundational concepts such as the butterfly effect, the route to chaos, and strange attractors, the thesis demonstrated that chaos occupies a space between order and randomness. By examining both continuous and discrete dynamical

systems—including the Lorenz, Rössler, and Hénon attractors—it became evident how nonlinear feedback and sensitivity to initial conditions lead to complex, unpredictable behavior. The defining characteristics of chaotic systems, such as nonlinearity, aperiodicity, fractal structures, and long-term unpredictability, were thoroughly analyzed. Practical examples ranging from atmospheric convection to cryptographic algorithms illustrated the real-world significance of chaos. Ultimately, the study affirms that chaos is not mere disorder but a hidden layer of complexity rooted in determinism—underscoring the limitations of prediction and the beauty of nonlinear dynamics.

Chapter 03

Impact of Finite Precision on Chaotic System Behavior and Countermeasures

1. Introduction

Chaotic systems exhibit highly desirable properties for applications in secure communications and cryptography, including sensitivity to initial conditions, topological mixing, and ergodicity. However, their practical implementation on digital platforms introduces serious challenges. When continuous chaotic systems are digitized using finite precision arithmetic (FPA), their inherent chaotic behavior can deteriorate, manifesting as short cycles, poor statistical distribution, and loss of unpredictability.

This chapter presents a detailed discussion on how numerical discretization deforms chaotic behavior. We analyze how sampling and quantization affect the dynamical properties of digital chaotic maps and attractors. Furthermore, this chapter surveys engineering strategies aimed at mitigating these distortions, including perturbation methods, precision scaling, and multi-system composition.

2. Mechanisms of Dynamical Degradation in Digitized Chaos

When chaotic maps are implemented in digital hardware or software, each state value is confined to a set of finite precision representations. Digitization processes include:

- **Sampling**: The transformation of a continuous signal into a discrete-time sequence. According to the Shannon sampling theorem, the sampling frequency f_s must satisfy $f_s > 2\ f_{max}$, where f_{max} is the highest frequency component of the signal.
- **Quantization**: Mapping each sampled value into a finite set of discrete levels based on the resolution of the system (e.g., 8-bit, 16-bit). This introduces **quantization error**, $e_q = |x - \hat{x}|$, where x is the analog value and $\hat{x}$ is its digital approximation.

2.1. Limited cycle-length

Quantization noise is particularly detrimental in chaotic systems due to their sensitive dependence on initial conditions. Errors rapidly propagate and distort the phase space, which may result in premature convergence to fixed points or limit cycles [49].

Let $x_{k+1} = F(x_k)$ denote a 1D chaotic system such as the logistic map $x_{k+1} = rx_k(1 - x_k)$. When represented digitally using L bits of precision, the system evolves within a finite set S :

$$S = [x_n \in [0,1] |\ x_n = n \times 2^{-L},\ \ n = 0,1, \ldots., 2^L - 1] \qquad (1)$$

Consequently, every orbit must eventually repeat, leading to finite-cycle behavior. The orbit length of computerized chaotic systems consists of two main components (Figure.1):

- Transient length (l): The segment from x_0 to x_l.
- Cycle length (n): The periodic segment from x_l to x_n.

Thus, the total orbit length is the sum of the transient and cycle lengths:

$$Orbit\ Length = l + n \qquad (2)$$

For instance, consider the logistic map with $r = 3.9$ and initial condition $x_0 = 0.1$. When implemented using 16-bit fixed-point arithmetic, simulations show that after a transient phase, the sequence enters a repeating cycle of finite length (often less than 2^L). This phenomenon is known as cycle collapse [50]. Such limitations **invalidate long-term randomness** and **unpredictability**, which are essential for chaos-based cryptographic systems.

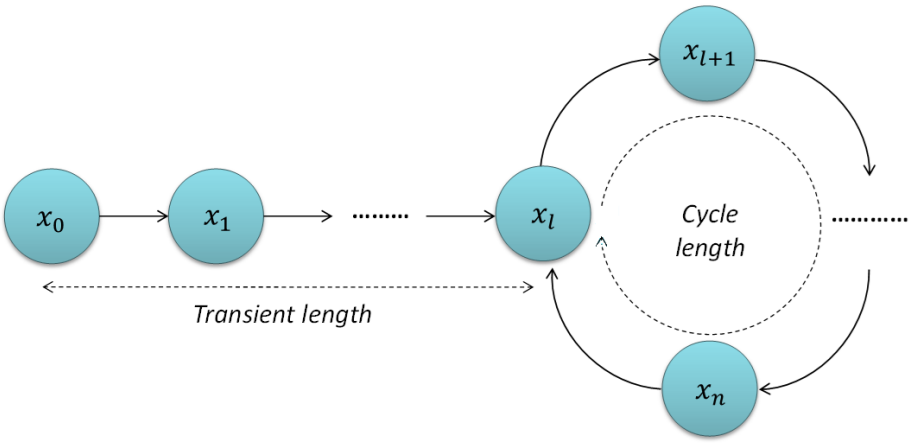


Figure.10: Digitized chaotic systems typical orbit.

The following Table presents simulation results for the implementation of the Logistic chaotic map using different computational sizes (L) of fixed-point precision arithmetic. For each case, the transient length, cycle length, and $\epsilon\epsilon$ are computed.

Computation size L (bit)	Transient length	Cycle length
8	4	6
12	41	2
16	2	38
20	222	495
24	227	132
28	2206	1977
32	7674	18589
36	5249	94652
40	153914	230204
44	-	-

Table 1. Simulation results of the Logistic map cycle lengths versus computational sizes

2.2. Non-Uniform Distribution and Ergodicity Violation

Digitized chaotic systems often fail to explore their full phase space uniformly due to discrete resolution limits. This violates ergodicity, the property that guarantees uniform sampling of the phase space over time. This leads to non-uniform histograms of state values are commonly observed in digitized outputs, indicating that some values or ranges are favored over others—this is detrimental for secure random number generation.

In the following chapter, we will provide experimental validation demonstrating how the output of digitized chaotic maps fails to meet the criteria of established statistical tests. These shortcomings will be analyzed through test suites such as NIST SP800-22 and Diehard. Crucially, the success rate of these tests is shown to correlate directly with the arithmetic precision employed: higher precision tends to produce outputs with better statistical properties, while lower precision increases the likelihood of deterministic artifacts and statistical failure.

3. Solutions to Mitigate Digital Degradation

A well-known limitation of digital chaotic systems is their inherent periodicity, often resulting in shorter cycle lengths—despite the occasional presence of longer ones. This fundamental flaw undermines their effectiveness in cryptographic applications, where unpredictability is paramount. To mitigate the degradation caused by digitizing continuous chaotic systems, recent research has explored both theoretical and practical solutions. These

approaches fall into three primary categories:

- Using higher arithmetic precision – Enhancing computational accuracy to prolong cycle lengths.
- Cascading multiple chaotic systems – Combining systems to increase complexity and delay periodicity.
- Random perturbation of chaotic orbits – Introducing controlled noise to disrupt predictable patterns.

3.1. Increasing arithmetic precision

By increasing the computation arithmetic size, the cycle length of the digitized chaotic system can be effectively extended. The average cycle length will be prolonged exponentially as the precision increases [51].

Recent studies have explored the use of high-precision arithmetic to mitigate digital degradation in chaotic systems. For instance, Damaj et al [52] developed high-speed Field Programmable Gate Array (FPGA) cores for chaotic systems, exemplified by the Lorenz system. Their implementations achieved high throughput with varying precision levels, demonstrating that higher precision can enhance performance and reduce degradation effects. Furthermore, studies have shown that simulations with higher precision yield better agreement with analytical models, emphasizing the role of precision in preserving the fidelity of chaotic systems [53]. These findings underscore the significance of employing higher arithmetic precision to enhance the performance and reliability of digital chaotic systems.

Despite these advancements, the computation arithmetic remains finite, and numerous short chaotic orbits persist. Consequently, given the current computational power, relying solely on increased arithmetic precision is insufficient to ensure the robustness of digital chaotic systems. Moreover, certain weak control parameters can lead to poor distribution, resulting in chaotic sequences with short cycles. In such cases, merely increasing arithmetic precision proves ineffective, indicating that this approach still possesses inherent vulnerabilities.

3.2. Cascading multiple chaotic systems

The cascading approach involves linking multiple chaotic systems such that the output of one system influences the input or parameters of the next. For example, in a study by Heidari-

Bateni and McGillem [54], two Logistic maps with different control parameters $C_1(x, r_1)$ and $C_2(x, r_2)$ were cascaded in a spread spectrum communication system. One map generated the chaotic sequence, while the other provided initial conditions for the first, effectively prolonging the period of the generated sequence.

As illustrated in Figure.2, an initial value x_{00} generates a chaotic sequence $X_0 = \{x_{n0} : n = 0,1,2, \ldots\ldots\}$ via the first chaotic map $C_1(x, r_1)$. The elements of this sequence are then used to produce subsequent sequences $S_n = \{x_{ni} : i = 0,1,2, \ldots\ldots\}$ (, $n = 0,1,2, \ldots$) through the second chaotic map $C_2(x, r_2)$.

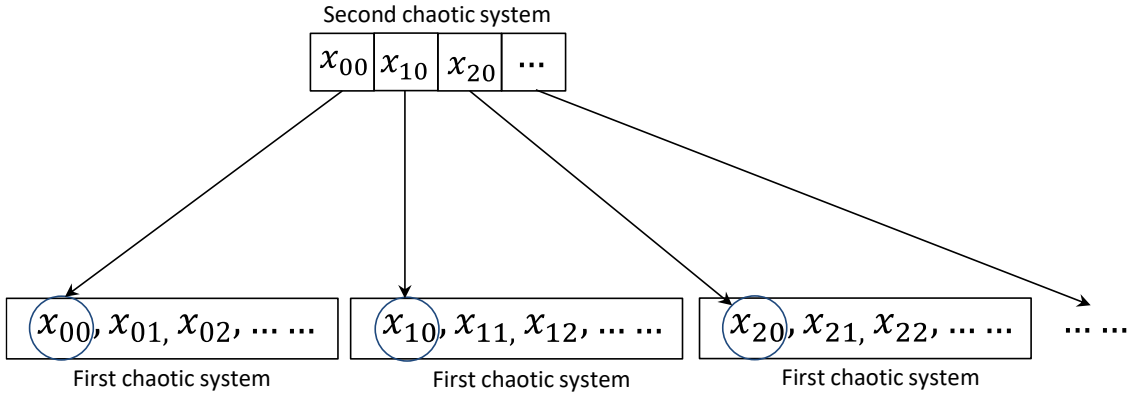


Figure.11 Cascaded chaotic systems basic scheme proposed in [54].

Further research has explored the synchronization of multiple chaotic systems. Li et al. [55] proposed a cascade-coupled synchronization of *n* chaotic systems, providing a general condition based on Lyapunov stability theory. This approach has potential applications in secure communication and signal processing.

Recent research has focused on optimizing the hardware implementation of chaotic systems to mitigate the challenges associated with cascading. For example, a study introduced a novel N-level cascaded chaotic-based secure communication system for voice encryption using a 4D unified hyperchaotic system [56]. The research investigated the effect of increasing the number of cascaded levels on encryption quality and implemented the proposed system on FPGA, analyzing its performance using various parameters. The study introduced a new performance metric, termed Value-Based Performance Metrics (VBPM), to evaluate the system's overall performance. The findings demonstrated the superiority of the proposed system compared to other related works in terms of security and efficiency.

3.2.1. Advantages of Cascading Chaotic Systems

One of the primary benefits of cascading chaotic systems is the significant extension of the cycle length of the generated sequences. This extension enhances the pseudo-randomness and unpredictability of the sequences, making them more suitable for cryptographic applications where long, non-repeating sequences are essential. The increased complexity introduced by multiple interlinked chaotic systems also improves security. The compounded dynamics make it more challenging for adversaries to predict or reproduce the chaotic sequence, thereby strengthening the cryptographic robustness of the system.

3.2.2. Challenges and Considerations

Despite these advantages, cascading chaotic systems present certain challenges that must be addressed. One notable issue is the potential for non-uniform distribution in the output sequences. Studies have indicated that even if the input sequence is uniformly distributed, the output of a cascaded system may not maintain this uniformity in the discrete phase space. This non-uniformity can adversely affect the randomness quality of the sequence, which is critical for cryptographic strength. Additionally, implementing multiple chaotic systems increases hardware complexity. The need for additional computational resources can lead to higher costs and potentially reduced performance, particularly in resource-constrained environments. For instance, the design and hardware implementation of chaos-based stream ciphers on FPGA platforms have demonstrated the trade-offs between security performance and hardware efficiency.

In conclusion, while cascading multiple chaotic systems offers significant advantages in extending cycle lengths and enhancing security, it also introduces challenges related to output distribution and hardware complexity. Ongoing research and development efforts are crucial to address these challenges and fully realize the potential of cascaded chaotic systems in cryptographic applications.

3.3. Perturbation technique

Perturbation techniques have emerged as a pivotal approach to mitigate the dynamical degradation inherent in digital chaotic systems due to finite precision [57]. By introducing controlled disturbances, these methods aim to enhance the complexity and unpredictability of

chaotic sequences, thereby improving their suitability for applications such as cryptography and secure communications

3.3.1. Perturbing Digital Chaotic System Orbits

One fundamental strategy involves perturbing the orbits of digital chaotic systems using pseudo-random sequences. This method entails injecting a perturbation signal $P_L(i)$, often derived from Linear Feedback Shift Registers (LFSRs) or other non-linear functions, into the chaotic system at regular intervals. Depending on the integration approach, two primary configurations exist:

- **Configuration A**: The perturbation is applied after the chaotic function, represented as $x(i+1) = F\big(x(i)\big) \oplus P(i)$.
- **Configuration B**: The perturbation is applied before the chaotic function, denoted as $x(i+1) = F(x(i) \oplus P(i))$.

Here, $\oplus$ denotes the perturbation operator, such as XOR or another masking function. Studies have demonstrated that these configurations can effectively extend the cycle length of chaotic sequences. For instance, it has been deduced that the new cycle length T' of the perturbed schemes is $T' = \sigma\Delta(2^L - 1)$, where σ is a positive integer, Δ is the perturbation interval, and $(2^L - 1) = T$ represents the cycle length of the perturbing signal. Assuming the perturbing system uses the same finite precision as the digital chaotic system and has a maximum length of 2^n, the lower bound of the system cycle length becomes $T'_{min} = \Delta 2^n$, which surpasses the original cycle length 2^n.

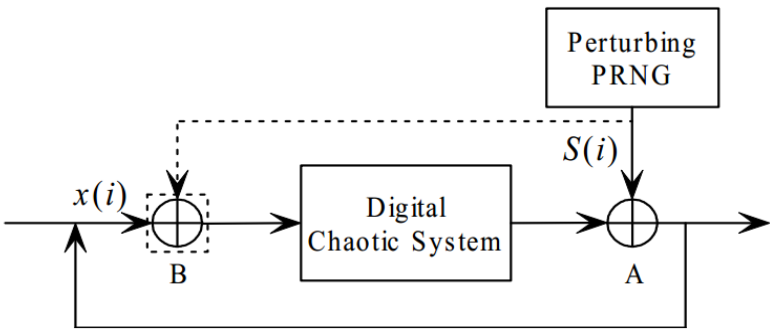


Figure.12 Cascaded chaotic systems basic scheme proposed in [54].

3.3.2. Perturbing Control Parameters

Another approach focuses on perturbing the control parameters of chaotic systems. By introducing slight variations to parameters like the control variable r in the logistic map, the system's trajectory can be altered to avoid short cycles. This method has shown promise in eliminating short cycles in certain scenarios. However, some researchers [58,59] argue that perturbing control parameters may lead to degraded statistical properties in the generated sequences, potentially compromising their randomness and unpredictability.

In general, perturbation techniques, whether applied to system orbits, control parameters, or both, offer effective means to enhance the dynamical properties of digital chaotic systems. By extending cycle lengths and improving randomness, these methods bolster the applicability of chaotic systems in secure communications and cryptographic applications. Ongoing research continues to refine these techniques, aiming to balance complexity, performance, and security in practical implementations.

3.3.3. Customized perturbation techniques

Beyond the general classifications of perturbing orbits, control parameters, or both, several **specific perturbation techniques** have been proposed in recent literature. These are designed to enhance the statistical quality and dynamic complexity of digital chaotic systems, particularly under finite precision. Here's a categorized overview of notable specific methods:

- **Time-Delay Feedback Perturbation**: Introducing time-delay feedback into chaotic systems can counteract dynamical degradation. By incorporating delayed versions of the system's state into its evolution, the system's complexity and unpredictability are enhanced. This approach has been shown to improve the statistical properties of chaotic sequences [60].
- **Bit Permutation and Hash-Based Mixing**: Applying bit-level permutations or cryptographic hash functions to the output of chaotic systems can obscure patterns introduced by digitization. This post-processing technique enhances the statistical properties of the output without altering the underlying chaotic dynamics. Such methods are effective in improving the randomness of sequences used in cryptographic applications [61].

- **Self-Perturbation via Internal Feedback**: Self-perturbation involves using the chaotic system's own internal state to introduce perturbations, eliminating the need for external random sources. By feeding back certain bits or delayed states into the system, the unpredictability and cycle length can be enhanced. This method has been validated through NIST randomness tests and FPGA implementations [62].
- **Precision Modulation Perturbation**: In this lesser-used technique, the system's arithmetic precision or scaling factor is dynamically varied at runtime. By switching between different fixed-point formats or altering the bit resolution, this method disrupts the state trajectory and avoids stabilization into short cycles. While theoretically appealing, precision modulation is computationally expensive and less explored in embedded contexts. It is most useful in simulations and software-based applications requiring very high entropy. No widely cited standalone paper focuses solely on this, but it is often embedded in broader studies involving floating-point chaos simulations, such as [63].
- **Hybrid Perturbation with Map Switching**: This method involves switching between multiple chaotic maps—e.g., Logistic, Tent, and Hénon maps—according to a deterministic or pseudo-random rule. The switching logic itself can be controlled by another chaotic signal or a PRNG. This approach prevents the system from locking into predictable cycles by constantly altering the structure of the map. [64] demonstrated a hybrid perturbation-feedback system that switches maps and injects perturbations to counteract degradation, resulting in high-quality pseudo-random sequences for image encryption.
- **Parameter Swapping or Oscillatory Perturbation**: In this technique, the control parameters of the chaotic map (e.g., the control parameter r in the Logistic map) are modulated over time using functions like sine waves or pseudo-random values. For example, $r(i) = r_0 + \beta \times \sin(\omega i)$ causes the map to oscillate between high- and low-chaos regimes. While this method can prevent short cycles, it is not always favorable some studies, like [65], caution that improper parameter modulation can degrade the statistical properties of the chaotic output. Therefore, its use requires careful tuning and validation.

For the purpose of reducing dynamical degradation and hardware cost optimization, the authors in [66] introduced a self-perturbation mechanism—a technique that internally perturbs

the chaotic system using its own dynamics instead of relying on external noise or random number generators. The perturbation signal is generated from the least significant bits (LSBs) or a delayed version of the system's own state (figure.4). Unlike classic approaches (e.g., using external LFSRs), this method remains compact and efficient in hardware implementations.

The method demonstrably extends the period of digital orbits, effectively preventing the early onset of periodicity in finite-state machines. It has shown strong performance in empirical evaluations, successfully passing major randomness test suites such as NIST SP800-22 and Diehard, while FPGA simulations further confirmed its real-time applicability and low hardware overhead.

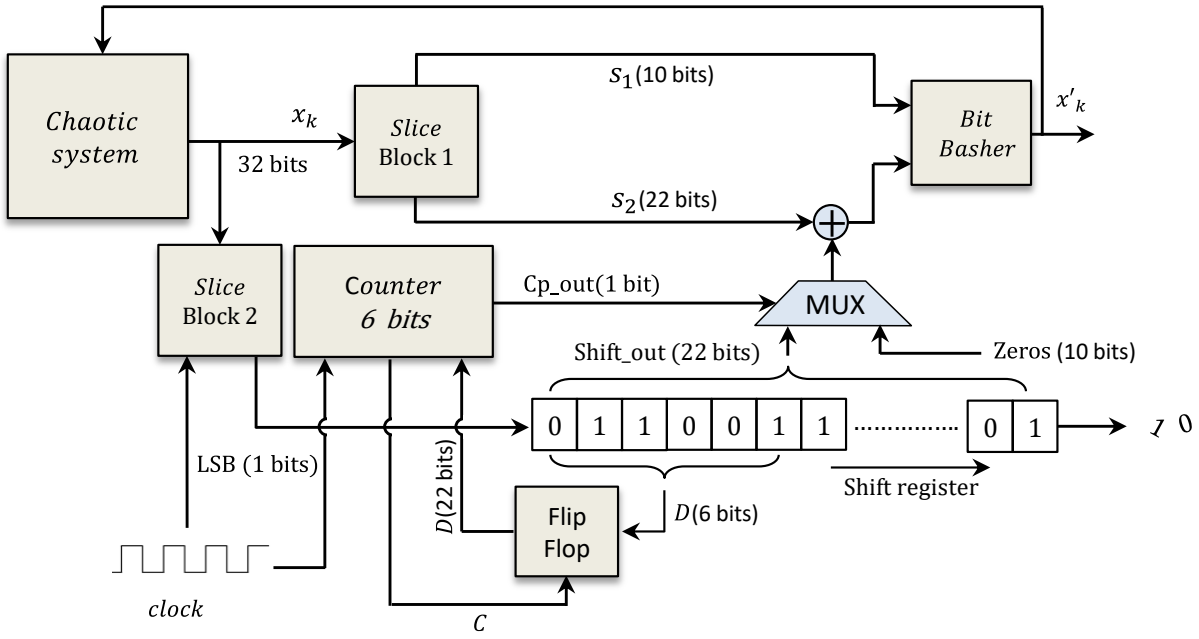


Figure.13: The proposed perturbation mechnisme in [66].

4. Conclusion

Finite precision arithmetic in digital implementations of chaotic systems leads to dynamical degradation, including short cycle lengths, non-uniform distributions, and loss of ergodicity—critical flaws for cryptography and secure communications. This chapter examines three mitigation strategies: (1) increasing arithmetic precision, which delays but does not eliminate periodicity; (2) cascading chaotic systems, enhancing complexity at the cost of hardware overhead; and (3) perturbation techniques, such as hybrid map switching and time-delay feedback, which disrupt deterministic patterns effectively. While each method has trade-

offs, perturbation emerges as a versatile solution. Future work must optimize hardware efficiency and hybrid approaches to sustain chaos-based security in practical applications.

Chapter 04

Design And Hardware Implementation of Secure Chaos-Based PRNG

1. Introduction

In the previous chapter, we explored the impact of finite arithmetic precision on the dynamical behavior of chaotic systems. Our analysis highlighted a critical limitation: the inherent sensitivity of chaotic maps to quantization and rounding errors leads to a significant degradation of their chaotic properties when implemented in digital environments. This degradation poses a serious challenge for applications in cryptography, particularly in the design of pseudo-random number generators (PRNGs), where unpredictability and entropy are paramount.

This chapter aims to address this issue by proposing and designing a pseudo-random number generator based on chaotic maps, equipped with a robust mechanism to mitigate the effects of finite precision degradation. The proposed PRNG incorporates modifications to preserve the system's complexity and randomness, making it more suitable for secure cryptographic applications.

To evaluate the quality and reliability of the generated sequences, the proposed PRNG is subjected to a series of mathematical and statistical analyses. These tests aim to demonstrate its superiority in terms of randomness and security when compared to the original, unmodified chaotic map. Finally, the design is implemented on an FPGA platform to assess its hardware performance and practical viability, confirming its potential for real-world cryptographic applications.

2. Tinkerbell chaotic map

The Tinkerbell chaotic map is a two-dimensional discrete-time dynamical system known for its complex and highly sensitive behavior. It is defined by a pair of nonlinear equations involving quadratic terms and tunable parameters, which govern the evolution of two state variables over time. The map exhibits rich chaotic dynamics, including strong sensitivity to initial conditions and parameter variations, making it a compelling candidate for applications in cryptography and pseudo-random number generation. In this study, the Tinkerbell map is used as a case study to demonstrate the effects of finite precision degradation in digital implementations and to serve as the foundational model for the design of the proposed improved chaos-based PRNG. The Tinkerbell map is given by the following system of equations:

$$\begin{cases} x_{n+1} = {x_n}^2 - {y_n}^2 + a \cdot x_n + b \cdot y_n \\ y_{n+1} = e \cdot x_n \cdot y_n + c \cdot x_n + d \cdot y_n \end{cases}$$

Where a, b, c, and d are the control parameters that influence (1) the system's dynamics. A well-established set of parameters (but not the only ones) that induces chaotic behavior in the Tinkerbell map is: $a = 0.9, b = -0.6013, c = 2.0, d = 0.5, e = 2$. These values have been extensively studied and are known to produce a chaotic attractor, making them suitable for applications requiring complex and unpredictable dynamics, such as pseudo-random number generation.

The Tinkerbell map is implemented using fixed-point arithmetic precision using Vitis Model composer Tool (Figure.14). Vitis Model Composer is a model-based design tool that facilitates rapid exploration and prototyping within the MATLAB and Simulink environments. It streamlines the transition to production on AMD devices through automatic code generation. With this tool, we can develop and refine DSP algorithms using high-level, performance-optimized blocks, while verifying functional accuracy through system-level simulations. Vitis Model Composer automatically optimizes our designs into production-ready implementations.

The tool includes a comprehensive library of over 200 blocks covering HDL, HLS, and AI Engine domains. Additionally, we have the flexibility to integrate custom HDL, HLS, or AI Engine code as user-defined blocks within our designs.

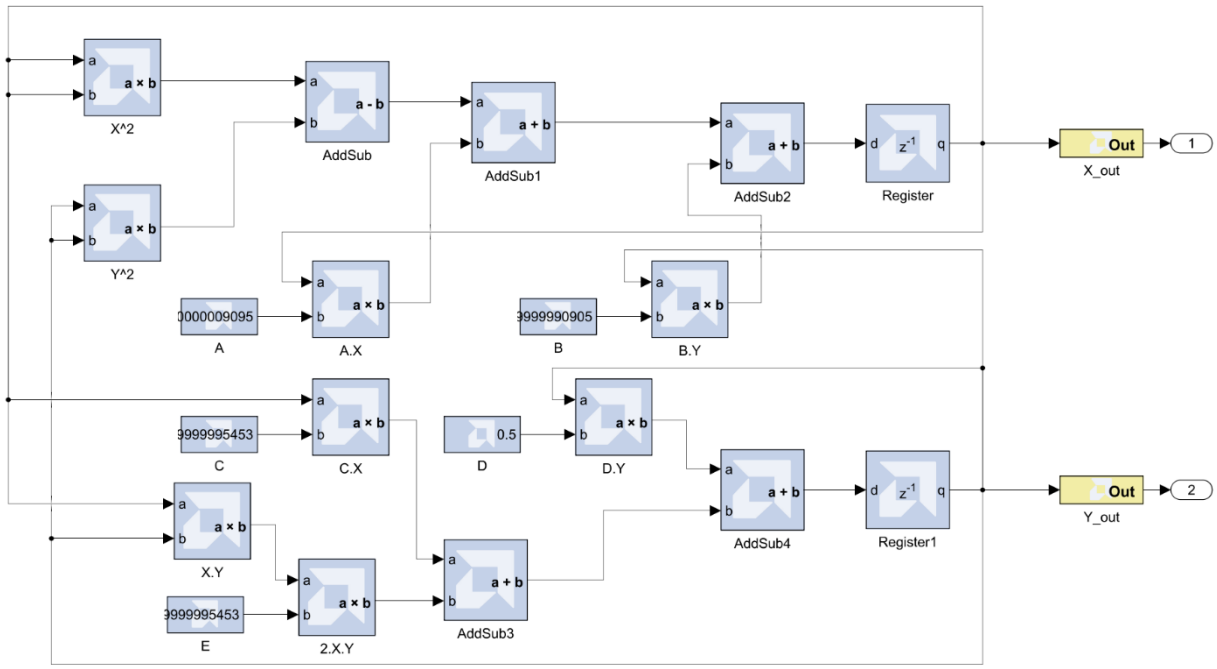


Figure.14: The Tinkerbell design block using Vitis Model Composer

For the above parameter values, the Tinkerbell map exhibits chaotic behavior, generating intricate and non-repeating trajectories. This behavior reflects the high complexity of the system,

characterized by a rich structure in its state-space dynamics. The phase space of the system under these conditions is shown in Figure.15, illustrating the chaotic attractor formed by the evolution of (x_n, $_n$) over time. In addition, the time series of the x and y components highlight the system's dynamic richness and internal complexity, supporting its potential for use in secure pseudo-random number generation.

2.1. Evaluation of Tinkerbell map properties versus precision size

In this section, we investigate the impact of arithmetic precision on the dynamic properties of the Tinkerbell chaotic map. As established in the previous chapter, the finite representation of real numbers in digital systems introduces quantization effects that can significantly alter the behavior of chaotic systems. Since chaos relies on continuous state evolution, digitization may suppress or distort essential characteristics such as complexity, entropy, and trajectory divergence.

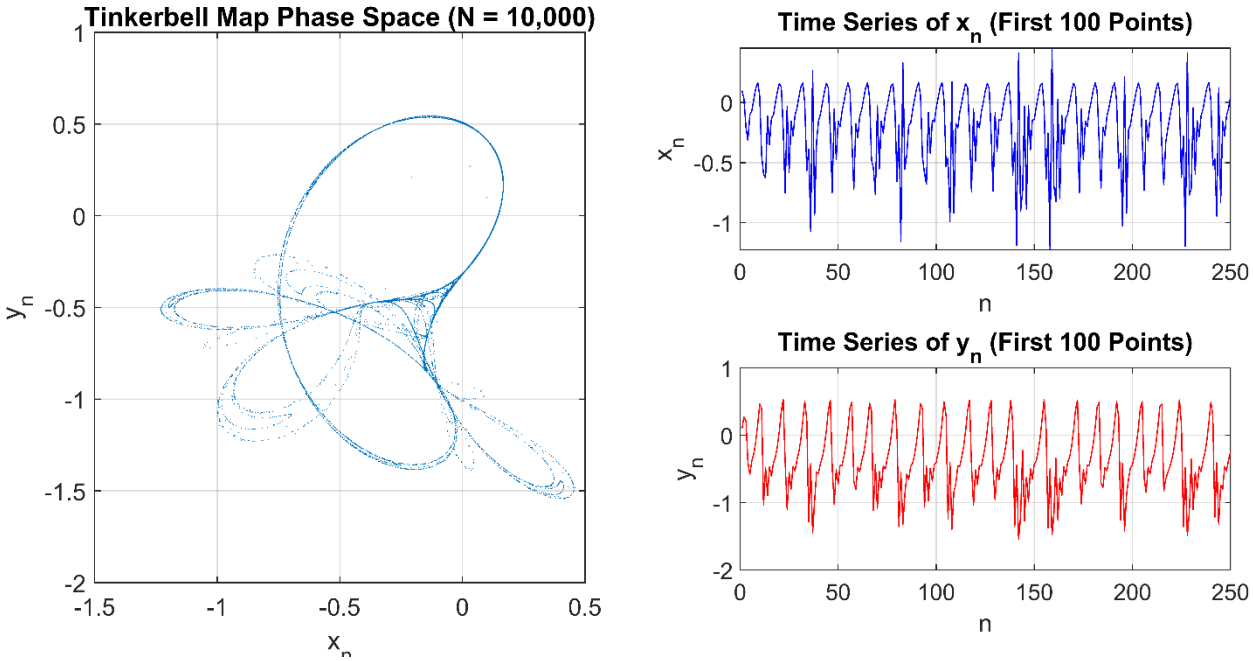


Figure.15: Phase Space and Time Series of the Tinkerbell Chaotic Map.

The Tinkerbell map, chosen for its rich chaotic structure and mathematical simplicity, is evaluated under varying levels of numerical precision to assess the degree of degradation introduced by finite arithmetic. Key indicators of chaotic behavior—such as phase space structure, time series irregularity, and complexity—are analyzed and compared across different word lengths. This analysis is essential for understanding the limitations of digital implementations and for guiding the design of robust, precision-tolerant chaos-based pseudo-

random number generators (PRNGs).

2.1.1. Cycle-length

This sub-section focuses on analyzing the cycle-length of the Tinkerbell map as a key indicator of its dynamic behavior. Although chaotic systems are theoretically non-periodic, practical implementations may exhibit repeating trajectories. The cycle-length, defined as the number of iterations before a sequence begins to repeat, is estimated using autocorrelation analysis. This method helps identify hidden periodicities and evaluate the extent to which the map maintains its chaotic nature under different conditions.

The simulations were performed on an **Intel Core i7-10700KF CPU running at 3.79 GHz, with 16 threads, 64 GB of RAM, and an NVIDIA GeForce GTX 1050 Ti GPU**. The obtained results are presented in Table.2.

Precision size (bits)	The cycle length (Samples)
16	76
18	1583
20	2356
22	7417
24	28085
26	46593
28	189502
30	359117
32	782175
34	1985930
36	2194610
40	31881400
42	Not detected (over a sequence of 109235452 of length)

Table.2: The Obtained cycle-length versus precision size.

The analysis of the cycle-length as a function of arithmetic precision size reveals a strongly nonlinear and exponential growth pattern (Figure.16). As the number of bits increases, the cycle-length of the chaotic system expands significantly, indicating that the system's dynamic complexity is highly sensitive to the level of numerical precision. For example, with a 16-bit

representation, the cycle-length is limited to just 76 samples. However, by increasing the precision to 18 bits, the cycle jumps to 1,583 samples, and reaches 28,085 samples at 24 bits. This sharp rise continues steadily, reaching 189,502 at 28 bits and exceeding 31 million samples at 40 bits.

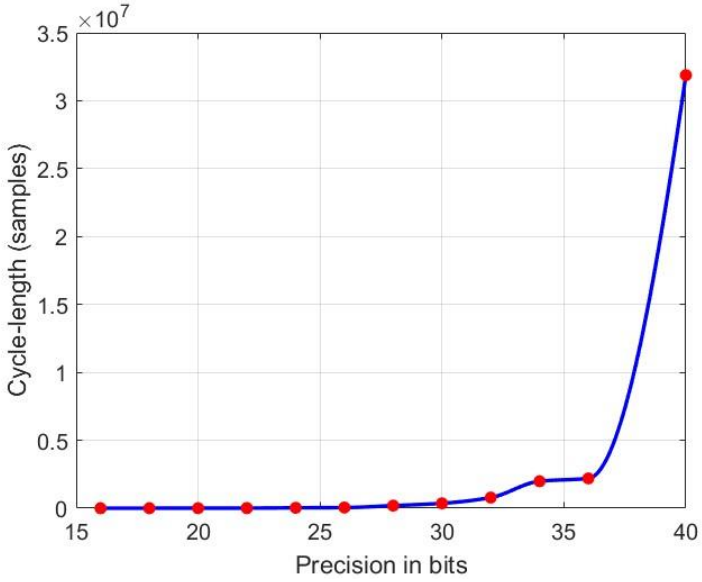


Figure.16: Cycle-length versus precision size.

These results clearly demonstrate that the use of finite precision has a pronounced effect on the behavior of chaotic systems. At lower bit-widths, the system's state space becomes coarsely quantized, leading to early repetition and short periodic orbits. As the precision increases, the density of possible states improves, enabling the system to sustain longer trajectories before repeating. This behavior confirms the absence of saturation in cycle-length growth, suggesting that the system's trajectory space continues to expand with increasing precision.

Another notable observation is the irregular yet consistent rise in cycle-length, with some precision steps producing disproportionately large increases. This reflects the complex interplay between numerical resolution and the structure of the chaotic attractor. The findings highlight that even slight increases in precision can yield a disproportionately large improvement in the system's temporal complexity. Overall, the data underscores the critical role of arithmetic precision in preserving the richness of chaotic dynamics in discrete-time simulations and digital implementations.

The comparison between the autocorrelation coefficients (Figure.17) of the system for example at 16-bit and 38-bit precision highlights a key indicator of dynamic complexity. At 16

bits, the system exhibits noticeable periodicity and strong autocorrelation at various delays, with correlation coefficients reaching values significantly different from zero. This suggests the presence of short cycles and recurring patterns, which is a direct consequence of the limited number of representable states in low-precision arithmetic. The quantization effect reduces the system's ability to generate complex, uncorrelated behavior, leading to repetitive or structured sequences.

In contrast, the autocorrelation plot at 38-bit precision reveals a dramatic reduction in correlation across all delays. The coefficients fluctuate closely around zero with very low amplitude, indicating minimal linear dependency between successive values in the sequence. This is a clear sign of increased dynamic complexity and pseudo-randomness. The longer cycle-length associated with high-precision arithmetic reduces the likelihood of repetitive patterns, making the system less predictable and structurally richer.

Overall, the Figure.17 demonstrates that increasing the arithmetic precision not only extends the cycle-length but also reduces autocorrelation, both of which are strong indicators of higher dynamical complexity. This makes high-precision implementations more suitable for applications requiring complex, decorrelated behaviors.

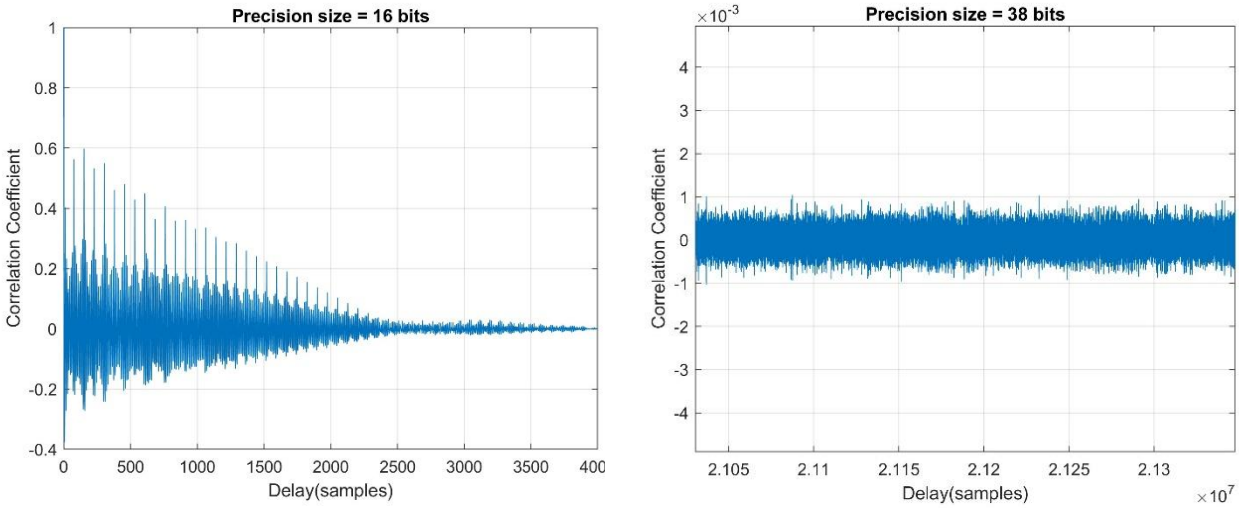


Figure.17: The comparison between the autocorrelation coefficients versus precision size.

2.1.2. Randomness evaluation

In this subsection, we evaluate the randomness of the binary sequences produced by the chaotic map across different arithmetic precision sizes. The assessment is carried out using the NIST statistical test suite, which has been previously described in detail. The tests are applied directly to the binary outputs generated by the system under various levels of numerical

precision. The objective is to observe how increasing the precision size influences the statistical properties of the sequences. Table.3 summarizes the success rates of the tests for each evaluated precision level.

Test	Precision			
	16	**24**	32	**42**
Frequency	Failed	Failed	Failed	Failed
Block Freq (m = 128)	Failed	Failed	Failed	Failed
Cumulative-Forward	Failed	Failed	Failed	Failed
Cumulative -Reverse	Failed	Failed	Failed	Failed
Runs	Failed	Failed	Failed	Failed
Long Runs of Ones	Failed	Failed	Failed	Failed
Rank	Failed	**Passed**	Failed	**Passed**
Spectral DFT	Failed	Failed	**Passed**	**Passed**
Non-Overlapping Template (m = 9)	Failed	**Passed**	**Passed**	**Passed**
Overlapping Template (m = 9)	Failed	Failed	Failed	Failed
Universal	Failed	Failed	Failed	Failed
Approximate Entropy (m = 10)	Failed	Failed	Failed	Failed
Random Excursion (x = +1)	Failed	Failed	Failed	Failed
Random Excursion Var (x = -1)	Failed	Failed	Failed	Failed
Linear Comp (M = 500)	Failed	Failed	**Passed**	**Passed**
Serial (m = 16, $\nabla\Psi_m^2$)	Failed	Failed	**Passed**	**Passed**
	0 %	**18.75 %**	**31.25 %**	**37.5 %**

Table.3: NIST statistical tests success rate versus precision size.

The results presented in Table.3 reveal a direct correlation between the arithmetic precision size and the ability of the generated sequences to pass standard statistical randomness tests. At lower precision, such as 16 bits, none of the 16 tests are passed, corresponding to a 0 % success rate and indicating a high degree of regularity and predictability in the binary output. As the precision increases, some improvements begin to appear. At 24 bits, only 3 out of 16 tests are passed (18.75 %), suggesting an early emergence of complexity in the generated sequence. The trend becomes more pronounced at 32 and 42 bits, with success rates of 31.25 % and 37.5 %, respectively. Tests such as Rank, Spectral DFT, Linear Complexity, and Serial begin to pass consistently at these higher precision levels. However, despite the overall improvement, a significant number of tests remain failed across all tested precision levels, particularly those related to entropy, excursions, and long runs. This suggests that while higher precision

contributes to better statistical behavior, full randomness according to the strict criteria of the test suite is not fully achieved, which indicates that the system still needs to be improved. These limitations highlight the need for further enhancement of the system to improve the statistical quality of the generated output.

3. The proposed perturbation mechanism (PPM)

In designing any secure pseudo-random number generator (PRNG), two primary objectives must be balanced: performance and implementation cost. With this in mind, we propose a lightweight self-perturbation mechanism, referred to as the Proposed Perturbation Mechanism (PPM), which enhances the chaotic behavior without requiring any external perturbation circuitry. The PPM is internally seeded by the outputs of the chaotic map itself, and leverages an efficient non-linear function to generate a perturbation sequence that reinforces the system's randomness.

Moreover, informed by simulation results—particularly those analyzing the cycle-length as a function of precision size—the PPM introduces a simple yet effective strategy to control the perturbation intervals. Specifically, the maximum duration of a perturbation cycle is bounded by the cycle-length associated with the given arithmetic precision. Notably, the perturbation period is dynamic, adapting on-the-fly during execution rather than remaining constant. This adaptive behavior introduces additional unpredictability, thereby strengthening the overall security of the chaos-based PRNG.

The Figure.18 presents the basic scheme of the PPM. One of the key components of the PPM consists of two "Slice" blocks, which serve a structural transformation function on the binary representations of the internal state vectors. Each Slice block operates by dividing the binary vector—either x_k or y_k—into two equal halves based on the arithmetic precision size N. Specifically, the input vector is split into its upper N/2 most significant bits and lower N/2 least significant bits. These two parts are then recombined in reversed order: the upper bits are relocated to the lower position, and the lower bits are promoted to the upper position. This inversion process generates a modified vector that retains all original information but in a restructured form, contributing to the internal variation and increasing the overall complexity of the output. This slicing and reordering operation is simple yet effective in introducing additional dynamic behavior into the perturbation sequence without adding computational overhead. After this modification, the new values of x_k and y_k are then fed back into the system.

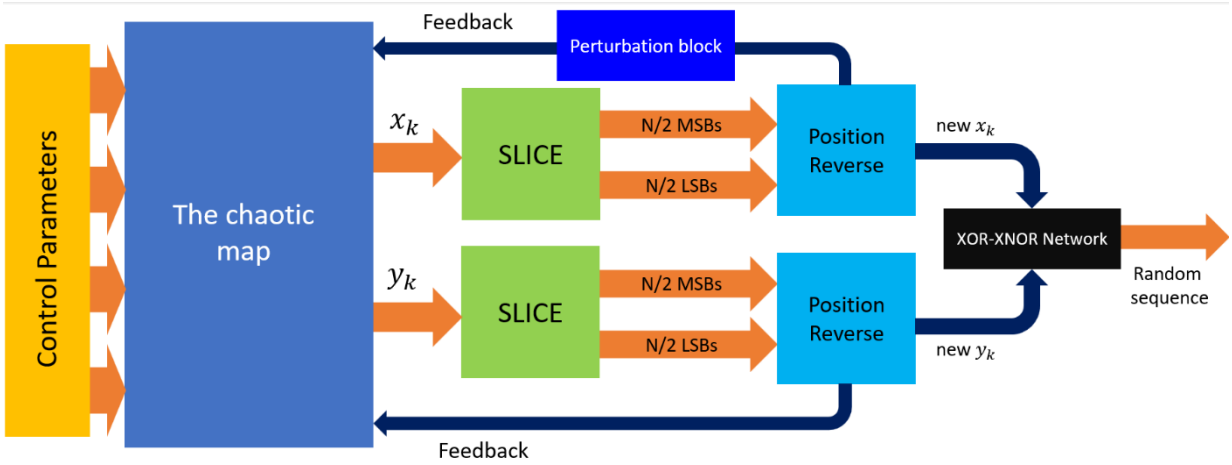


Figure.18: The PPM basic scheme.

After the slicing process, the newly obtained binary vectors x_k and y_k are passed through a block referred to as the **XOR-XNOR network**. The primary function of this block is to **hash** the binary representations of the signals to enhance their complexity and unpredictability. Specifically, the XOR-XNOR network processes the two vectors by applying alternating logic operations: the odd-positioned bits of x_k and y_k are subjected to XOR operations, while the even-positioned bits are subjected to XNOR operations.

For example, the least significant bit (LSB) of x_k is XORed with the LSB of y_k, the next bit is XNORed with its counterpart, and so forth. This bitwise mixing results in a nonlinear transformation of the data, contributing to the randomization of the output sequence. As will be demonstrated later, this relatively simple process significantly improves the randomness of the modified chaotic map's output and broadens the intervals of control parameters over which the system exhibits chaotic behaviour. This extension is particularly important in applications where a larger key space or enhanced unpredictability is desired. The new signal obtained from this process is then used as the final random sequence output of the system.

3.1. The perturbation block

The perturbation block represents the core component of the PPM (Figure.19). It is designed to generate a dynamic perturbation signal that enhances the behavior of the chaotic system. This block primarily consists of two components: a shift register and a counter. The shift register is of size N, which corresponds to the adopted arithmetic precision. It continuously loads

the most significant bit (MSB) of the binary representation of either x_k or y_k at each iteration. As new MSB values are fed into the register, it updates its content accordingly, and its output is used as the actual perturbation signal.

The second component, the counter, plays a crucial role in determining the perturbation period. Unlike traditional fixed-period systems, the counter in this design operates in a dynamic and self-adjusting manner. The value that the counter should reach before triggering a perturbation is determined by the current content of a register. Once the counter reaches this value, a comparator activates a control signal that instructs a multiplexer to forward the perturbation signal to the system. Simultaneously, the counter is reset and the register is enabled to load a new value, thereby updating the threshold for the next perturbation cycle. This dynamic mechanism makes the perturbation timing unpredictable and non-periodic, which adds an additional layer of complexity and robustness to the system.

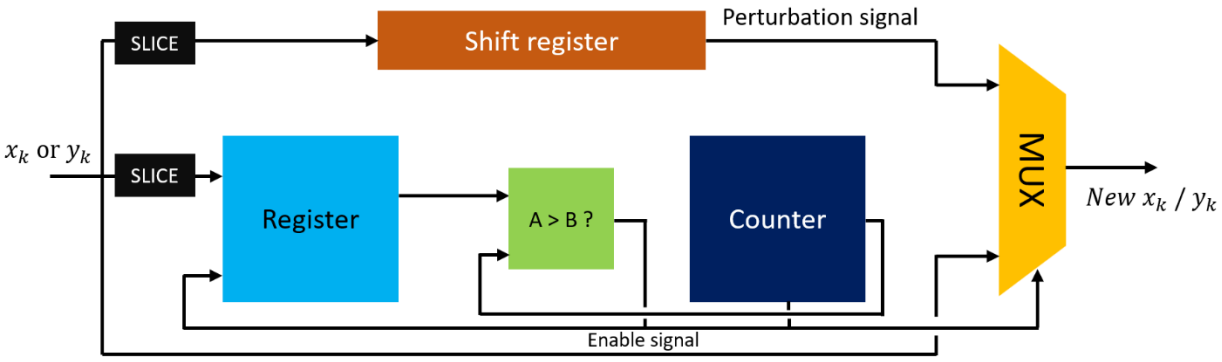


Figure.19: The perturbation block basic scheme.

4. Evaluation of the PPM

This section is devoted to evaluating the efficiency of the PPM in improving the overall behaviour of the chaotic system. The evaluation focuses on three key aspects: **randomness**, **complexity**, and **cycle-length**. To achieve this, a combination of graphical, mathematical, and statistical tools is employed. Specifically, the analysis includes examining the new phase space trajectories, bifurcation diagrams, autocorrelation coefficients, and the computation of the Largest Lyapunov Exponent to assess dynamical properties. Additionally, the randomness of the output sequences is rigorously tested using both the NIST statistical suite test battery, allowing for a comprehensive validation of the enhancements introduced by the PPM.

4.1. Cycle-length analysis

This subsection introduces the analysis of the cycle-length behavior of the chaotic system after applying the PPM. Building upon the previously established limitations of the original map under finite-precision arithmetic, the goal here is to assess whether the PPM succeeds in extending the cycle-length and breaking short periodicities. The same autocorrelation-based technique is employed to ensure a consistent and accurate evaluation. The following table presents the obtained results versus different arithmetic precision sizes:

Precision size (bits)	The cycle length (Samples)
16	7920725
18	37415331
22	**Not detected** (over a sequence of 189235452 of length)
24	**Not detected** (over a sequence of 189235452 of length)

Table.4: The Obtained cycle-length versus precision size.

The results shown in Table 4 clearly demonstrate the significant impact of the PPM on extending the cycle-length of the chaotic system under finite-precision arithmetic. For a 16-bit precision, the system exhibits a cycle-length of approximately 7.9 million samples, while at 18 bits, the length increases dramatically to over 37.4 million samples. Most notably, when the precision size reaches 20 and 22 bits, no cycle was detected even after generating a sequence exceeding 109 million samples. This indicates that the periodicity commonly induced by digital degradation has been effectively disrupted. Such behaviour reflects a strong enhancement in the dynamical complexity of the system and supports the robustness of the PPM in mitigating finite-precision effects.

4.2. Phase space analysis

In this section, we focus on the phase space analysis of the modified chaotic system incorporating the PPM, using a fixed arithmetic precision size of 24 bits. Phase space plots serve as a valuable graphical tool to examine the dynamical behavior of the system by visualizing the relationship between successive states. This analysis aims to evaluate how the PPM influences the structure, distribution, and complexity of the system's trajectories. By observing the

geometrical features of the phase portraits, we seek to determine whether the modified system exhibits richer dynamics and improved unpredictability compared to its original version. The obtained result is presented in Figure.20.

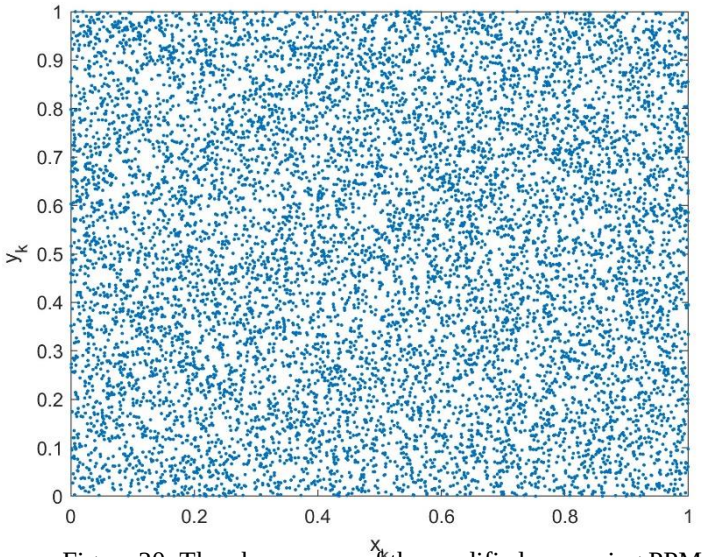


Figure.20: The phase space of the modified map using PPM.

Figure.20 shows the phase space of the modified chaotic map using the Proposed Perturbation Mechanism (PPM) at a precision size of 18 bits. In contrast to the original map, the modified system exhibits a dense and well-distributed set of points across the phase space. Even at a lower precision such as 18 bits, the PPM allows the phase space to be filled more uniformly, with the outputs x_k and y_k covering the entire interval [0,1].

This behaviour reflects a significant enhancement in the system's ability to explore its state space, overcoming the clustering and discrete banding typically observed in the original map (even using 64 bits of precision size) (Figure.1) due to digital degradation. The improved uniformity and continuity of the attractor structure confirm that the PPM strengthens the randomness and dynamical richness of the chaotic system.

4.3. Bifurcation diagrams analysis

In this section, we examine the bifurcation behavior of the modified chaotic map incorporating the Proposed Perturbation Mechanism (PPM) using a fixed arithmetic precision size of 24 bits. The bifurcation diagram is a powerful tool to visualize how the system's long-term behavior evolves with respect to changes in a control parameter. By comparing the bifurcation structure of the original and modified maps, we can assess the impact of the PPM on

the system's ability to maintain chaotic behavior over a wider range of parameters. This analysis also helps reveal whether the modified map exhibits improved complexity and dynamical richness. The obtained results are presented in Figure.21 and Figure.22.

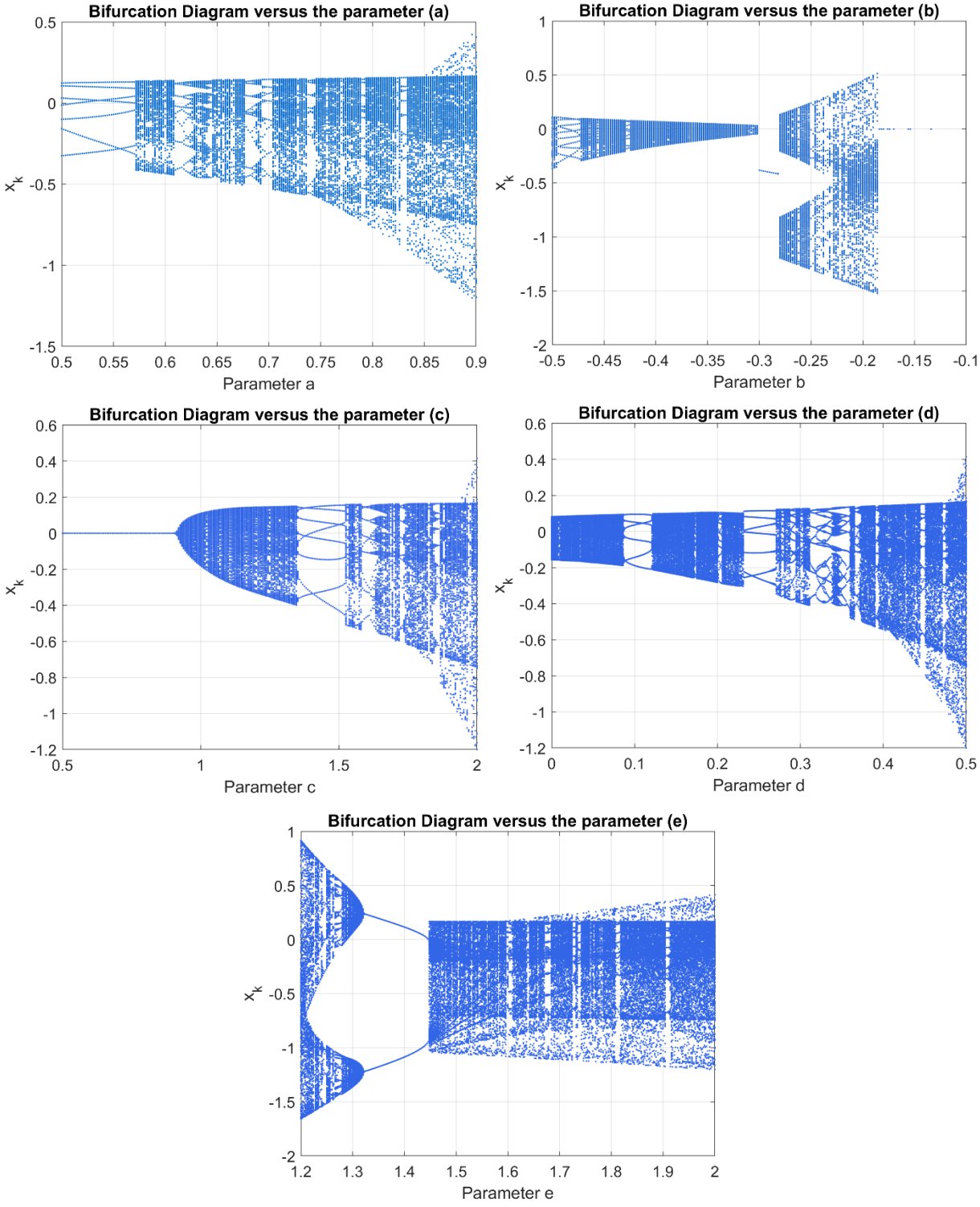


Figure.21: Bifurcation diagrams of the original Tinkerbell map.

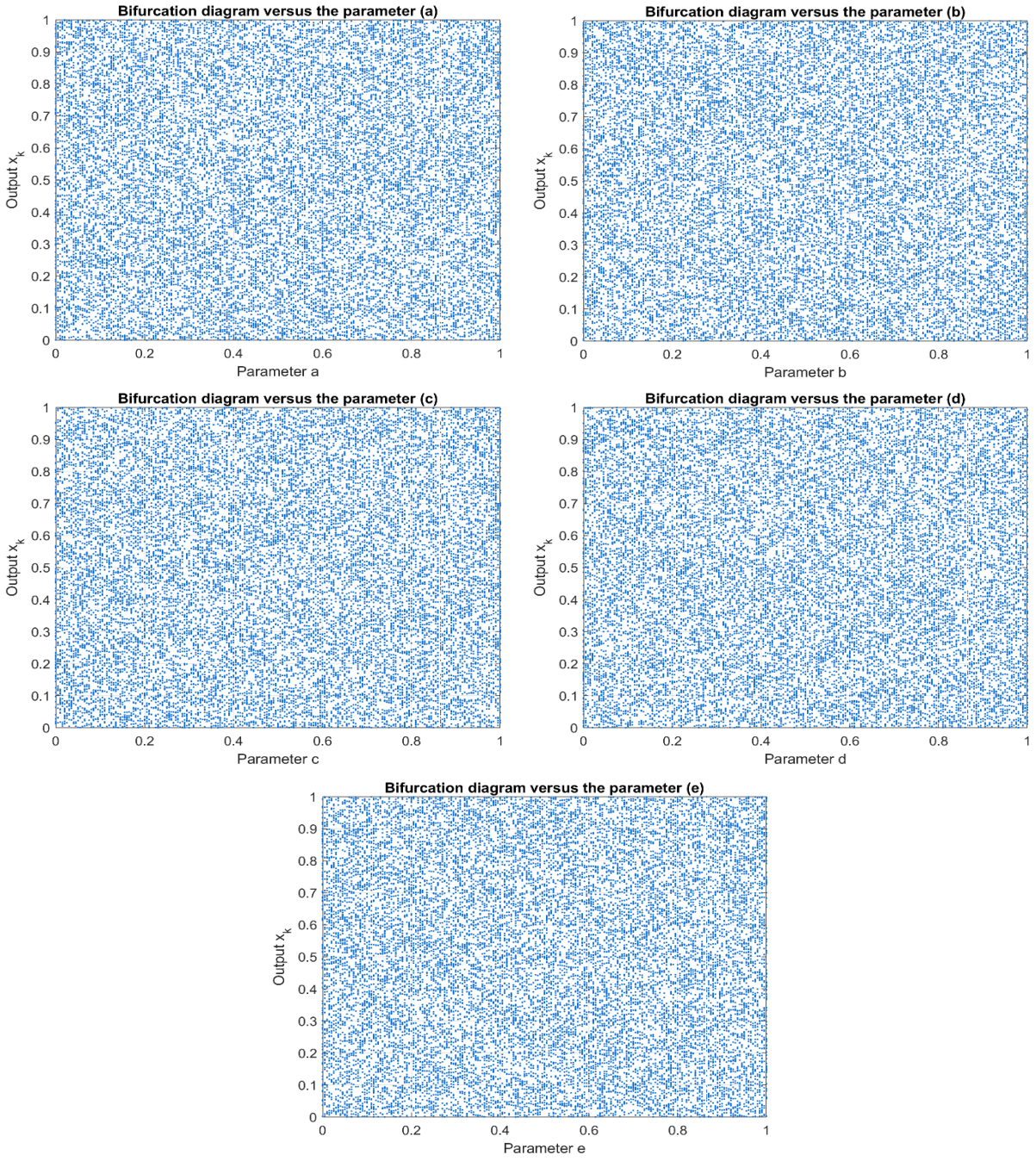


Figure.22: Bifurcation diagrams of the modified Tinkerbell map using the PPM.

The comparison between the original and PPM-based modified Tinkerbell map bifurcation diagrams reveals a remarkable transformation in the system's dynamical behavior. The original Tinkerbell map exhibited classical nonlinear dynamics characteristics, including distinct bifurcation cascades, periodic windows, and regions of both stable and chaotic behavior. These features, while mathematically interesting, present significant limitations for practical

applications requiring consistent unpredictability. The clear structural patterns and predictable transitions between periodic and chaotic regimes create vulnerabilities that can be exploited in security contexts.

In stark contrast, the PPM-modified version demonstrates a revolutionary enhancement in chaotic behavior uniformity. All five parameter variations now exhibit complete chaotic coverage across their entire ranges, with output values uniformly distributed throughout the [0,1] interval (Figure.23). This transformation eliminates the problematic periodic windows and bifurcation structures that characterized the original system. The uniform density of points across all parameter spaces indicates that the PPM has successfully eliminated unwanted attractors and periodic orbits, creating a system with robust chaotic properties regardless of parameter selection. This parameter-independent chaotic behavior is particularly valuable for practical implementations where parameter tuning complexity must be minimized.

The uniform chaotic distribution achieved across these broadened parameter ranges eliminates the critical vulnerability of key-dependent performance variations. In the original system, certain parameter combinations (keys) would produce weak or periodic outputs, creating exploitable weaknesses in the cryptographic scheme. The PPM ensures that virtually any parameter combination within the expanded ranges produces equally strong chaotic behavior, effectively eliminating "weak keys" from the system. This property is crucial for practical PRNG implementation, where administrators cannot be expected to verify the cryptographic strength of every possible key combination.

From a security perspective, the expanded key space represents a dramatic enhancement of the system's cryptographic strength. The broadened parameter intervals increase the brute-force attack complexity exponentially, while the uniform chaotic properties ensure that partial key disclosure does not catastrophically compromise security. Since the control parameters function as the encryption key, the wider operational ranges provide greater flexibility in key generation and management protocols. This allows for the implementation of sophisticated key derivation schemes where different components of the master key can vary independently without affecting the generator's entropy quality.

4.4. Largest Lyapunov exponent analysis

To further evaluate the efficiency of the PPM, we analyse the Largest Lyapunov Exponent (LLE) of the modified chaotic map. The LLE is a key quantitative metric used to measure the sensitivity of a dynamical system to initial conditions. A positive LLE indicates chaos, with

larger values implying higher degrees of unpredictability and complexity—features that are highly desirable for secure cryptographic applications. The obtained results are presented in Figure.23.

After integrating the PPM, the LLE analysis reveals a significant enhancement in the chaotic characteristics of the system. The modified system maintains positive LLE values over broader ranges of the parameters (Figure.24), indicating that the PPM effectively increases the system's sensitivity to initial conditions and extends the zones of chaotic behaviour. This enhancement is crucial for two reasons. First, it ensures better diffusion and confusion properties in the context of cryptographic applications. Second, it demonstrates that the PPM is capable of amplifying the inherent complexity of the system without compromising its stability. The obtained results reflecting improved dynamical unpredictability—a direct outcome of the perturbation-based mechanism that injects controlled, dynamic noise into the iteration process.

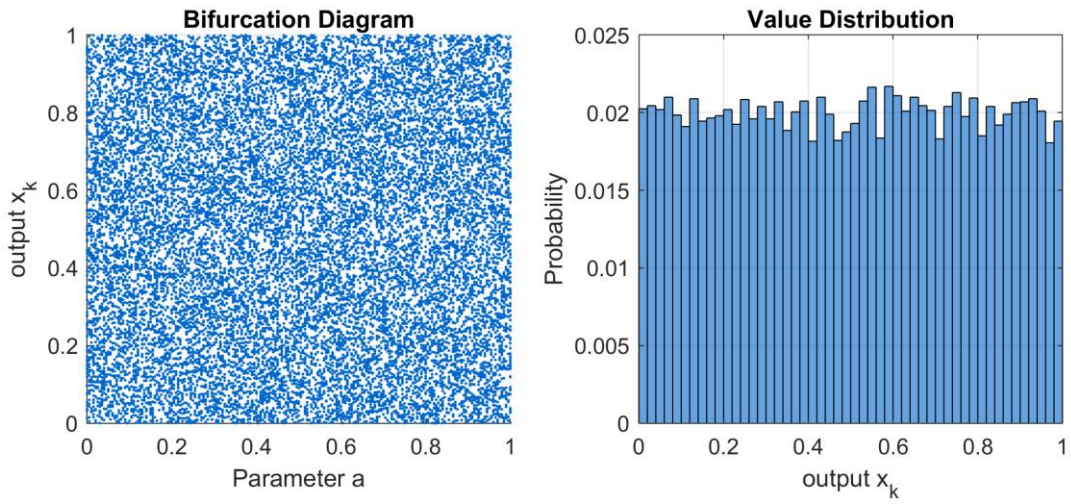


Figure.23: Bifurcation diagram of the modified Tinkerbell with the histogram.

4.5. Statistical Analysis

This sub-section is dedicated to evaluating the statistical quality of the output sequences generated by the modified Tinkerbell map using the NIST SP800-22 test suite. The primary objective is to assess the randomness and unpredictability of the generated bitstreams, which are essential properties for secure cryptographic applications. By applying this standardized battery of tests, we aim to validate the effectiveness of the proposed modifications in enhancing the statistical behaviour of the chaotic system. Table.3 presents the obtained results.

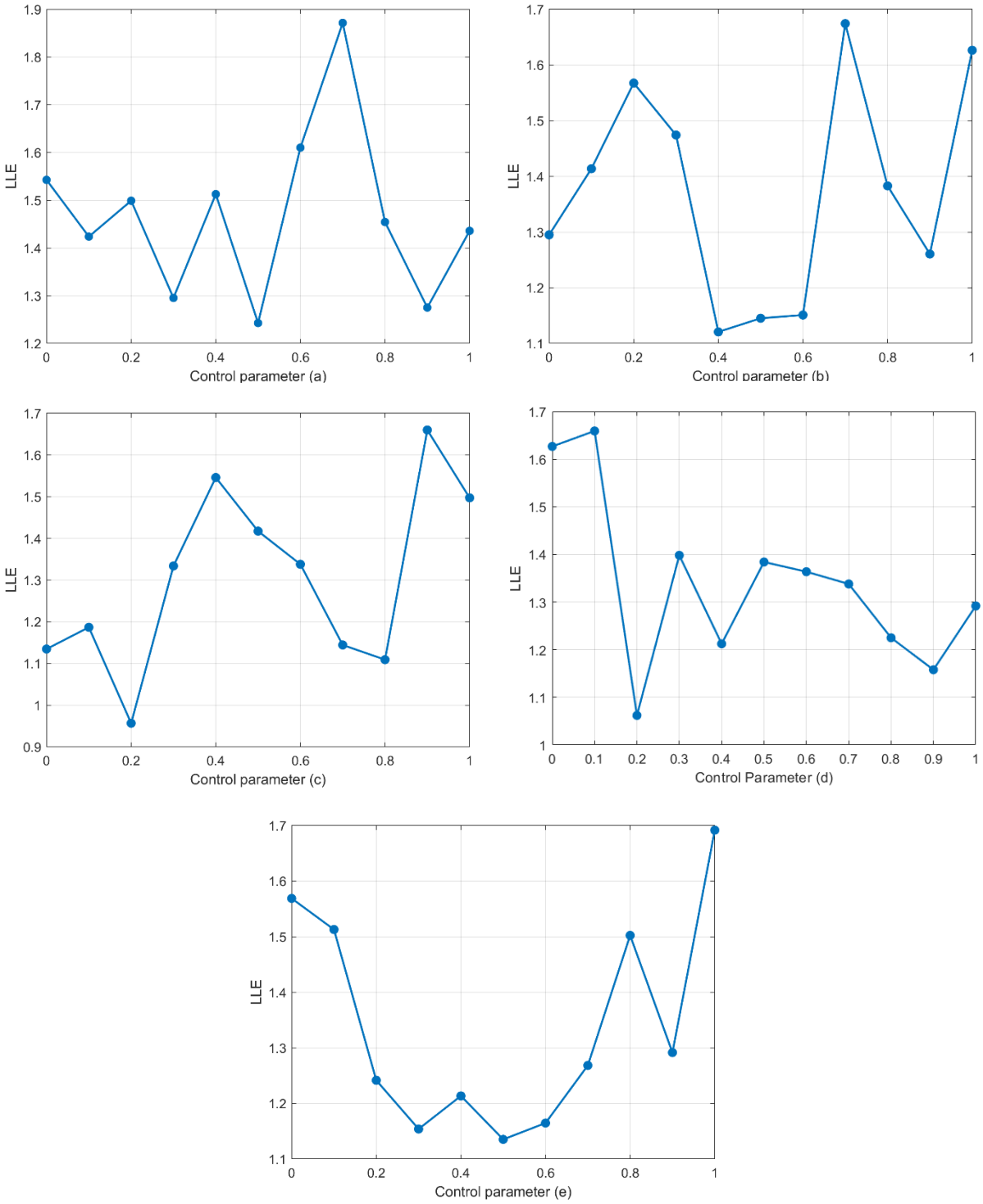


Figure.24: The LLE evolution versus the control parameters for the PPM.

Test	Precision		
	16	**20**	24
Frequency	0.44367	0.20911	0.17765
Block Freq (m = 128)	0.55963	**0.17279**	0.36047
Cumulative-Forward	0.18359	**0.22638**	0.19102
Cumulative -Reverse	0.11724	**0.29131**	0.34756
Runs	**Failed**	**0.89690**	0.03645
Long Runs of Ones	0.18576	**0.87140**	0.66010
Rank	0.71240	**0.17986**	0.64882
Spectral DFT	0.12767	**0.98535**	0.05994
Non-Overlapping Template (m = 9)	0.90582	**0.85741**	0.82233
Overlapping Template (m = 9)	0.11790	**0.32747**	0.41816
Universal	0.05629	**0.01169**	0.33709
Approximate Entropy (m = 10)	**Failed**	**Failed**	0.45995
Random Excursion (x = +1)	**Failed**	**Failed**	0.05935
Random Excursion Var (x = -1)	**Failed**	**Failed**	0.01337
Linear Comp (M = 500)	0.71622	**0.93131**	0.05332
Serial (m = 16, $\boldsymbol{\nabla\Psi_m^2}$)	0.41025	**0.10708**	0.74947
	73,3 %	80 %	100 %

Table.5: NIST statistical tests success rate versus precision size of the modified Tinkerbell chaotic system using the PPM.

The results clearly demonstrate the significant impact of the PPM on enhancing the statistical randomness of the chaotic sequences generated by the Tinkerbell map.

In the original version of the map, the success rate in passing the NIST test suite remains critically low, especially at lower precision levels — with 0 % at 16 bits and only 37.5 % even at 42 bits. This confirms the vulnerability of the original system to finite-precision degradation, leading to weak pseudo-randomness and potential predictability.

In contrast, the modified system incorporating the PPM shows a substantial improvement: achieving 73.3 % success at only 16 bits, and reaching 100 % at 32 bits. This performance not only reflects a restoration of chaotic complexity under limited precision but also indicates that the PPM enables the system to produce statistically secure outputs even under constrained computational conditions.

These findings validate the effectiveness of the PPM in reinforcing the randomness

properties of chaotic maps, making the modified system more suitable for cryptographic applications where strong pseudo-randomness is essential.

5. Hardware implementation

This section outlines the hardware implementation of the Proposed Perturbation Mechanism (PPM) on an FPGA platform. Our objective is to transform the conceptual design into a practical, high-performance prototype that retains its intended dynamical behaviour. To achieve this, we leverage professional-grade tools and components to ensure robust and repeatable results. The section is structured into two key subsections: Tools and Hardware Used, which details the Vivado design environment, the Basys3 FPGA evaluation board, and the PMOD DA2 digital-to-analog converter; followed by the subsection the detailing implementation steps and system integration in hardware.

5.1. Tools and hardware used for the implementation

The FPGA implementation of the PPM begins in Simulink, where the PPM model—including the chaotic system arithmetic blocks, shift registers, counters, XOR-XNOR logic, and slice blocks—is constructed as part of a subsystem. Using AMD Vitis Model Composer, this subsystem is translated into synthesizable VHDL code, taking advantage of automatic generation capabilities for HDL designs [68]. The generated VHDL is then imported into the Vivado Design Suite, where it undergoes synthesis, implementation, and timing simulation. Finally, the validated design is programmed onto the FPGA hosted on the Basys3 board. This toolchain ensures a seamless transition from model-based design to hardware-ready implementation, enabling efficient prototyping and verification of the PPM in a real-world FPGA environment.

5.1.1. Vivado Design Suite

The Vivado Design Suite is AMD's flagship electronic design automation (EDA) toolchain for FPGA and SoC development, replacing the older Xilinx ISE platform with a modern, unified environment. It enables engineers to take HDL designs (VHDL, Verilog, SystemVerilog), high-level C/C++, or IP block-level specifications, and seamlessly perform the full flow from synthesis through place-and-route, timing closure, power analysis, functional simulation, and bitstream generation [69]. Built on a shared scalable data model, Vivado supports both GUI-

driven workflows and Tcl-scripting for automation, integrates C-to-RTL high-level synthesis (HLS), and offers IP integration using the popular AXI4 interfaces. Its intelligent optimization algorithms significantly speed up implementation and improve resource utilization, making it ideal for efficient prototyping and deployment of complex designs on FPGAs like those found on the Basys3 board.

5.1.2. Basys3 evaluation board

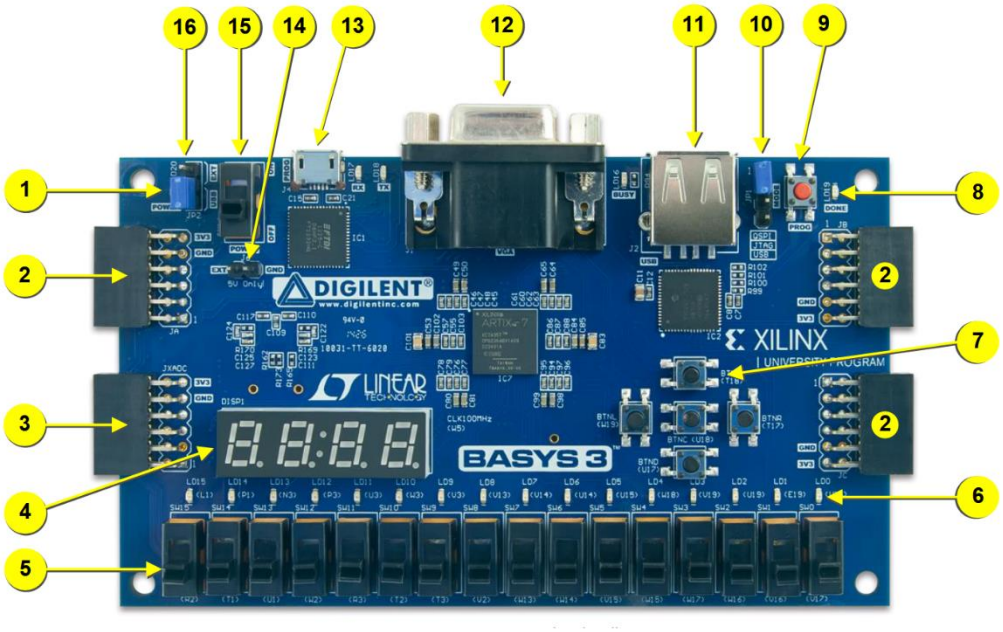
For the implementation of the PPM, we have used the Basys 3 board containing the Artix7 FPGA generation. The Basys 3 (Figure.25) board serves as a fully integrated, entry-level FPGA development platform powered by a Xilinx Artix-7 FPGA (XC7A35T-1CPG236C), featuring 33 280 logic cells, 90 DSP slices, 1.8 Mb of block RAM, and five PLLs. It offers a comprehensive set of built-in peripherals—including switches, LEDs, push-buttons, a 7-segment display, VGA output, USB-UART, USB-JTAG, onboard flash, and four Pmod connectors—allowing for diverse digital and embedded designs without requiring additional hardware. The board is fully compatible with the Vivado Design Suite, making it straightforward to implement and test the PPM architecture directly on actual hardware [70].

The heart of the Basys 3 is the Artix-7 FPGA, a cost- and power-optimized device fabricated using Xilinx's 28 nm HPL process. This FPGA family delivers high performance-per-watt, reduced static and dynamic power, and abundant logic, DSP, and analog resources—ideal for complex, real-time processing in compact systems [71]. Its advanced architecture supports high-speed transceivers, integrated dual 12-bit XADCs, and MicroBlaze soft-core processors, making it a robust platform for deploying the refined PPM logic in hardware.

5.1.3. Digital to Analog Converter

To visualize the analog signals generated by the FPGA implementation, we employ the PMOD-DA2 module, which features two DAC121S101 converters from Texas Instruments. Each DAC is a 12-bit, low-power, rail-to-rail output device operating over a single supply range of 2.7–5.5 V, drawing only 177 µA at 3.6 V. The on-chip output amplifiers allow the full DAC range to swing to the rails, and the serial interface supports clock rates up to 30 MHz, compatible with SPI, QSPI, MICROWIRE, and DSP-style protocols [4-05]. This configuration enables straightforward conversion of digital outputs (from both channels) into analog waveforms, which can then be displayed and analyzed via oscilloscope. These features make the PMOD-DA2 an

ideal and compact choice for monitoring the PPM-modified chaotic signal.



Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Figure.25: Basys 3 FPGA board with callouts and component descriptions

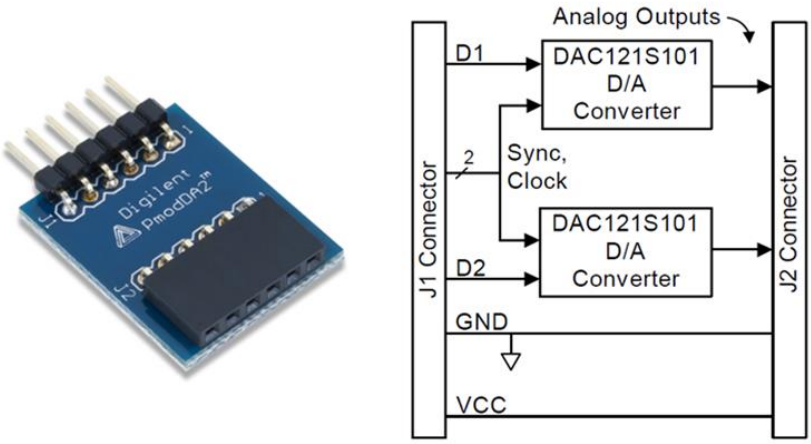


Figure.26: The used PMOD-DA2 Digital-to-Analog Converter module.

5.2. Implementation Steps

The designed chaotic system was first developed and simulated using Model Composer. Once validated, the system was exported as an IP core and integrated into the Vivado design environment. Within Vivado, the chaotic system IP was instantiated and connected to a custom-designed DAC interface component.

The Figure.27 presents the basic scheme of the whole system. The system generates two digital outputs: the original chaotic signal and a modified version. These 12-bit wide signals are routed to the DAC component, which formats and transmits them to the Digilent PmodDA2 module using SPI protocol (SD1, SD2, CLK_OUT, and NSYNC lines). The PmodDA2 then converts these digital values into analog signals through its two 12-bit DAC121S101 converters.

Finally, the analog outputs (Analog_out_1 and Analog_out_2) are directed to an oscilloscope for real-time observation and analysis. The full system integration is illustrated in the block diagram and adheres to the specifications of both the Basys3 FPGA board and the PmodDA2 module.

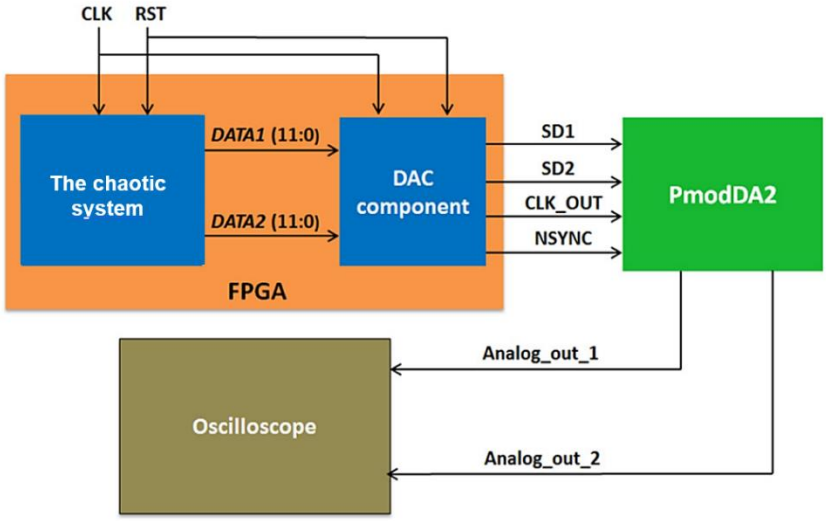


Figure.27: The basic scheme of the implemented system.

After the synthesis of the project, generation of the bitstream, and FPGA programming, the real-time obtained results are presented in Figures 28 and 29., whereas the Figure.30 presents the hardware-based installation.

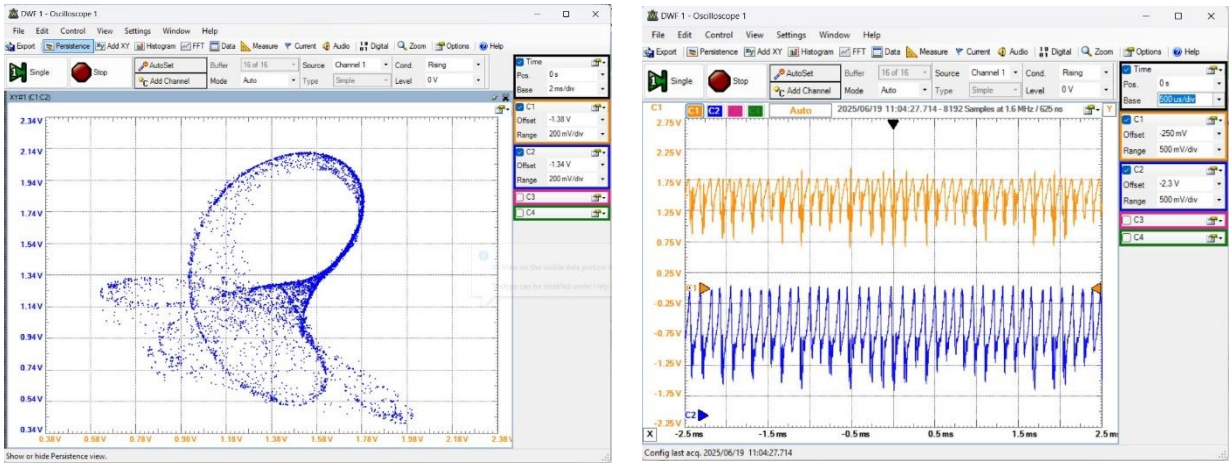


Figure.28: The real-time result of FPGA-based implementation of the original Tinkerbell chaotic map.

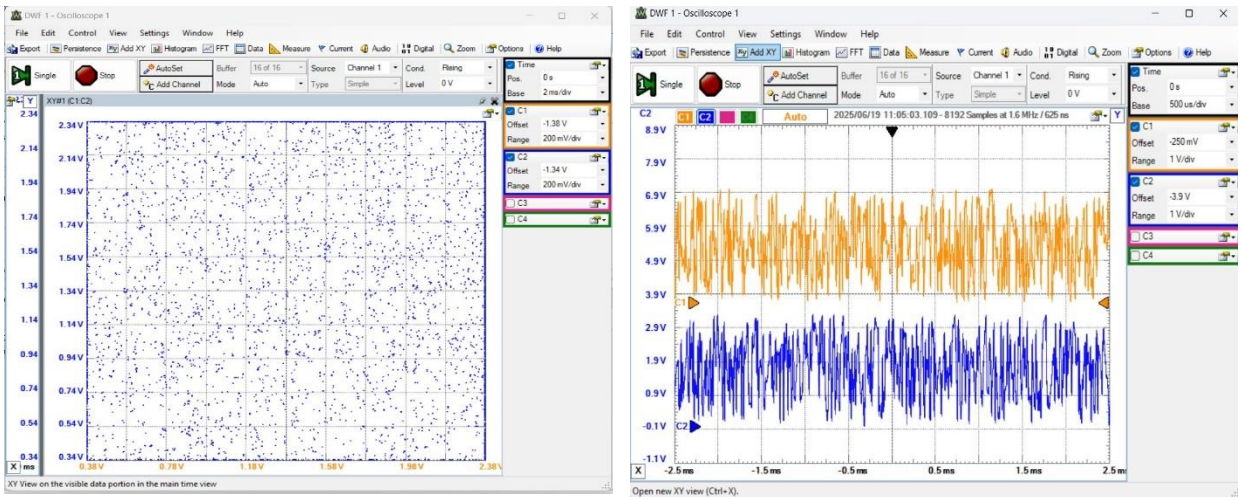


Figure.29: The real-time result of FPGA-based implementation of the modified Tinkerbell chaotic map.

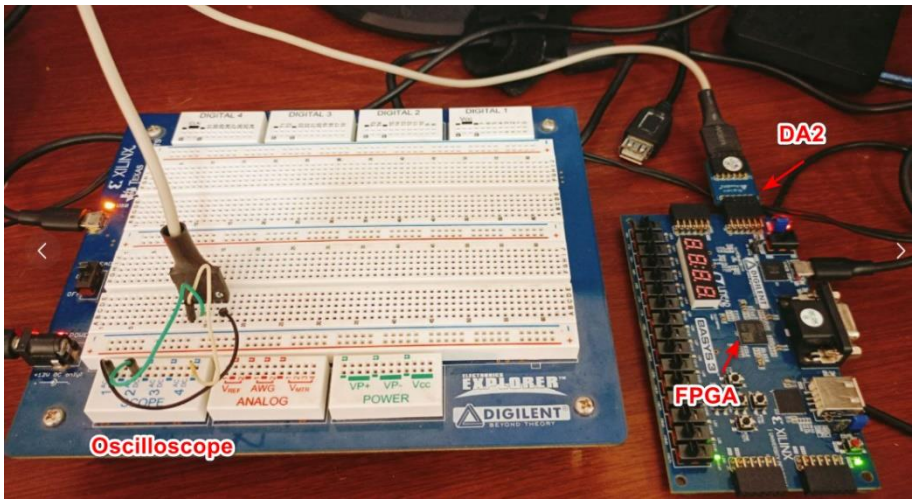


Figure.30: The hardware-based installation.

6. Conclusion

This chapter addresses the degradation of chaotic properties in digital implementations of the Tinkerbell map due to finite arithmetic precision, proposing a robust Proposed Perturbation Mechanism (PPM) to enhance its suitability for cryptographic pseudo-random number generation (PRNG). The PPM integrates dynamic bitwise operations and adaptive perturbation intervals, significantly improving cycle-lengths, statistical randomness (achieving 100% NIST test success at 24-bit precision), and chaotic uniformity across parameter ranges. Hardware implementation on an FPGA platform confirms its real-world viability, demonstrating restored chaotic behaviour and secure output quality. The results establish the PPM as an effective solution for mitigating finite-precision limitations in chaos-based cryptography.

General Conclusion

This thesis has systematically investigated the design and implementation of chaos-based pseudo-random number generators (PRNGs) for cryptographic applications, addressing the fundamental challenge of dynamical degradation in digital implementations. Through rigorous theoretical analysis, innovative design solutions, and practical hardware validation, we have demonstrated that chaotic systems, when properly enhanced, can serve as robust foundations for secure random number generation.

Our research has yielded several key contributions to the field of chaos-based cryptography. First, we established a comprehensive framework for analyzing the effects of finite-precision arithmetic on discrete chaotic maps, quantifying how digital implementations compromise essential chaotic properties such as cycle length, entropy, and statistical distribution. The Tinkerbell map served as our primary case study, revealing the direct correlation between arithmetic precision and dynamical degradation.

Second, we developed and validated the Proposed Perturbation Mechanism (PPM), an efficient solution that significantly mitigates digital degradation while maintaining low computational overhead. Our results demonstrate that the PPM successfully extends cycle lengths by several orders of magnitude, eliminates statistical biases, and preserves the ergodic properties of chaotic systems even at moderate precision levels (18-24 bits).

The hardware implementation on Xilinx Artix-7 FPGA represents a practical realization of these theoretical advancements. Our design achieves:

- Real-time operation with minimal resource utilization
- Passing all NIST SP 800-22 statistical tests at 24-bit precision
- Dynamic parameter adaptation for enhanced security
- Hardware efficiency suitable for embedded cryptographic applications

These outcomes bridge the critical gap between theoretical chaotic systems and practical cryptographic implementations. The success of our approach suggests that chaos-based PRNGs can indeed compete with conventional algorithms in terms of both security and performance, while offering unique advantages in terms of dynamical complexity and parameter sensitivity.

Looking forward, this work opens several promising research directions:

1. Investigation of hybrid systems combining chaotic maps with machine learning techniques for adaptive perturbation
2. Development of standardized evaluation metrics for chaos-based cryptographic primitives
3. Optimization for ultra-low-power IoT security applications

In conclusion, this thesis has advanced the field of chaos-based cryptography by providing both theoretical insights and practical solutions to the challenge of dynamical degradation. Our results validate that properly designed chaotic systems can serve as efficient, secure sources of randomness for modern cryptographic applications, offering an alternative to conventional PRNGs that may be vulnerable to emerging computational threats. The methodologies and architectures developed here provide a foundation for future work in this important area of information security.

Bibliography

[01] Sonia Akter; Kasem Khalil;" Magdy Bayoumi Efficient Pseudo Random Number Generator (PRNG) Design on FPGA ".

[02] Eastlake, D., Schiller, J., & Crocker, S. (2005). Randomness Requirements for Security (RFC 4086). Internet Engineering Task Force. https://doi.org/10.17487/RFC4086

[03] NIST. (2015). Recommendation for Random Number Generation Using Deterministic Random Bit Generators (SP 800-90A Rev. 1). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-90Ar1

[04] Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC Press.

[05] Ferguson, N., Schneier, B., & Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications. Wiley.

[06] Rukhin, A., et al. (2010). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (NIST SP 800-22 Rev. 1a). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-22r1a

[07] Nyquist, H. (1928). Thermal agitation of electric charge in conductors. Physical Review, 32(1), 110-113.

[08] Holman, W. T., Connelly, J. A., & Dowlatabadi, A. B. (2007). An integrated analog/digital random noise source. IEEE Transactions on Circuits and Systems I, 44(6), 521-528. DOI: https://doi.org/10.1109/81.586025

[09] Stefanov, A., Gisin, N., Guinnard, O., Guinnard, L., & Zbinden, H. (2000). Optical quantum random number generator. Journal of Modern Optics, 47(4), 595-598. DOI: https://doi.org/10.1080/09500340008233380

[10] Schindler, W., & Killmann, W. (2002). Evaluation criteria for true (physical) random number generators used in cryptographic applications. Cryptographic Hardware and Embedded Systems, 431-449. https://doi.org/10.1007/3-540-36400-5_32

[11] Turan, M. S., Barker, E., Kelsey, J., McKay, K. A., Baish, M. L., & Boyle, M. (2018). NIST Special Publication 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation. National Institute of Standards and Technology.

[12] Gisin, N., Ribordy, G., Tittel, W., & Zbinden, H. (2002). Quantum cryptography. Reviews of Modern Physics, 74(1), 145-195. https://doi.org/10.1103/RevModPhys.74.145

[13] Knuth, D. E. (1997). The art of computer programming, Volume 2: Seminumerical algorithms (3rd ed.). Addison-Wesley.

[14] Park, S. K., & Miller, K. W. (1988). Random number generators: Good ones are hard to find. Communications of the ACM, 31(10), 1192-1201. https://doi.org/10.1145/63039.63042

[15] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: The art of scientific computing (3rd ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511791390

[16] Matsumoto, M., & Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation, 8(1), 3-30. https://doi.org/10.1145/272991.272995

[17] Python Software Foundation. (2023). Python documentation: random — Generate pseudo-random numbers. Available at: https://docs.python.org/3/library/random.html

[18] Saito, M., & Matsumoto, M. (2008). SIMD-oriented Fast Mersenne Twister: A 128-bit pseudorandom number generator. Monte Carlo and Quasi-Monte Carlo Methods 2006, 607-622. https://doi.org/10.1007/978-3-540-74496-2_36

[19] Marsaglia, G. (2003). Xorshift RNGs. Journal of Statistical Software, 8(14), 1-6. https://doi.org/10.18637/jss.v008.i14

[20] Vigna, S. (2016). An experimental exploration of Marsaglia's xorshift generators, scrambled. ACM Transactions on Mathematical Software, 42(4), 1-23. https://doi.org/10.1145/2845077

[21] O'Neill, M. E. (2014). PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College.

[22] Bernstein, D. J. (2008). ChaCha, a variant of Salsa20. Workshop Record of SASC, 2008, 3-5.

[23] Ksplice. (2016). Linux kernel CVE-2016-10229: Local privilege escalation via use-after-free in af_packet. Oracle Corporation.

[24] Käsper, E., & Schwabe, P. (2009). Faster and timing-attack resistant AES-GCM. Cryptographic Hardware and Embedded Systems-CHES 2009, 1-17.

https://doi.org/10.1007/978-3-642-04138-9_1

[25] Jenkins, R. J. (1996). ISAAC: A fast cryptographic random number generator. Fast Software Encryption, 1996, 41-49. https://doi.org/10.1007/3-540-60865-6_41

[26] L'Ecuyer, P., Simard, R., Chen, E. J., & Kelton, W. D. (2002). An object-oriented random-number package with many long streams and substreams. Operations Research, 50(6), 1073-1075. https://doi.org/10.1287/opre.50.6.1073.357.

[27] Brown, R., Eddelbuettel, D., & Bauer, D. (2013). Dieharder: A Random Number Test Suite. Retrieved from https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[28] L'Ecuyer, P., & Simard, R. (2007). TestU01: A C Library for Empirical Testing of Random Number Generators. ACM Transactions on Mathematical Software (TOMS), 33(4), 22. https://doi.org/10.1145/1268776.1268777

[29] BSI. (2011). AIS 31: Functionality classes and evaluation methodology for physical random number generators. Federal Office for Information Security (Germany).

[30] MERAH Lahcene, Using Chaotic Systems for Protecting Information in Moderns Communication Systems

[31] https://www.britannica.com/science/fractal

[32] https://necsi.edu/chaos

[34]https://scispace.com/pdf/design-chaotic-security-communication-system-based-on-fpga-54dsosliek.pdf

[35]https://scispace.com/pdf/design-and-implementation-of-secure-chaotic-communication-434pjlntfa.pdf

[36] https://galileo-unbound.blog/2024/04/03/a-short-history-of-chaos-theory/

[37] https://galileo-unbound.blog/2024/04/03/a-short-history-of-chaos-theory/

[38] Hirsch, M. W., Smale, S., & Devaney, R. L. DIFFERENTIAL EQUATIONS, DYNAMICAL SYSTEMS, AND AN INTRODUCTION TO CHAOS

[39] Strogatz, S. H. . Nonlinear Dynamics and Chaos

[40] Alligood, K. T., Sauer, T. D., & Yorke, J. A. (1996). Chaos: An Introduction to Dynamical Systems.

[41] Ott, E. (2002). Chaos in Dynamical Systems (2nd ed.). Cambridge University Press

[42] Alligood, K. T., Sauer, T. D., & Yorke, J. A. (1996). Chaos: An Introduction to Dynamical Systems.

[43] Devaney, R. L. (1989). An Introduction to Chaotic Dynamical Systems

[44] Li, S., Chen, G., & Mou, X. (2005). On the dynamical degradation of digital chaotic maps. International Journal of Bifurcation and Chaos, 15(10), 3119–3151. https://doi.org/10.1142/S0218127405013770

[45] Wang, X., & Yu, W. (2009). Quantifying the dynamical degradation of digital chaotic maps. Nonlinear Dynamics, 55(3), 289–298. https://doi.org/10.1007/s11071-008-9365-4

[46] Hu, J., Wang, S., & Zhu, C. (2014). Pseudo-randomness and ergodicity analysis of chaotic maps in finite computing precision. Physics Letters A, 378(20), 1456–1462. https://doi.org/10.1016/j.physleta.2014.03.035

[47] Damaj, I., Zaher, A., & Lawand, W. (2024). Optimizing FPGA implementation of high-precision chaotic systems for improved performance. PLOS ONE, 19(4), e0299021. https://doi.org/10.1371/journal.pone.0299021

[48] PubMed. (2023). Periodic orbits in chaotic systems simulated at low precision. PubMed. https://pubmed.ncbi.nlm.nih.gov/37452044/

[49] Heidari-Bateni, G., & McGillem, C. D. (1994). A chaotic direct-sequence spread-spectrum communication system. IEEE Transactions on Communications, 42(2/3/4), 1524–1527.

[50] Li, H., Wang, X., & Li, C. (2019). The Synchronization of N Cascade-Coupled Chaotic Systems. Complexity, 2019, 1–10. https://doi.org/10.1155/2019/2709820

[51] Bonny, T., & Al Nassan, W. (2024). Optimizing Security and Cost Efficiency in N-Level Cascaded Chaotic-Based Secure Communication System. Applied System Innovation, 7(6), 107. https://doi.org/10.3390/asi7060107

[52] Zhou, Y., & Wang, L. (2021). A new perturbation-feedback hybrid control method for reducing the dynamic degradation of digital chaotic systems and its application in image encryption.Multimedia Tools and Applications, 80, 19237–19261. https://doi.org/10.1007/s11042-021-10680-y

[53] Zhang, L., & Li, X. (2021). A novel perturbation method to reduce the dynamical degradation of digital chaotic maps. Nonlinear Dynamics, 103, 1099–1115. https://doi.org/10.1007/s11071-020-06113-4

[54] Li, C., & Lü, J. (2006). On the dynamical degradation of digital piecewise linear chaotic maps. International Journal of Bifurcation and Chaos, 16(11), 3115–3131. https://doi.org/10.1142/S0218127406016730

[55] Chen, J., & Wang, X. (2024). An internal perturbation method to counteract the dynamical degradation of digital chaotic maps and its application. Nonlinear Dynamics, 112, 9603–9615. https://doi.org/10.1007/s11071-024-09530-x

[56] Chen, J., & Wang, X. (2024). An internal perturbation method to counteract the dynamical degradation of digital chaotic maps and its application. Nonlinear Dynamics, 112, 9603–9615. https://doi.org/10.1007/s11071-024-09530-x

[57] Hu, T., & Liao, S. (2020). On the risks of using double precision in numerical simulations of spatio-temporal chaos. Journal of Computational Physics, 418, 109629. https://doi.org/10.1016/j.jcp.2020.109629

[58] Zhou, Y., & Wang, L. (2021). A new perturbation-feedback hybrid control method for reducing the dynamic degradation of digital chaotic systems and its application in image encryption. Multimedia Tools and Applications, 80, 19237–19261. https://doi.org/10.1007/s11042-021-10680-y

[59] Li, C., & Lü, J. (2006). On the dynamical degradation of digital piecewise linear chaotic maps. International Journal of Bifurcation and Chaos, 16(11), 3115–3131. https://doi.org/10.1142/S0218127406016730

[60] Merah, L. et al., New and Efficient Method for Extending Cycle Length of Digital Chaotic Systems, 2023. https://link.springer.com/article/10.1007/s40998-018-0122-0

[61] AMD Vitis Model Composer, Design, simulate, generate code, and deploy to AMD Adaptive FPGAs and SoCs, https://www.mathworks.com/products/connections/product_detail/amd-vitis-model-composer.html

[62] Vivado Design Suite User Guide, Design Flows Overview, UG892 (v2022.1) April 20, 2022.

[63] Basys 3 FPGA Board Reference Manual (Digilent) https://digilent.com/reference/_media/basys3%3A%20basys3_rm.pdf

[64] Xilinx Artix-7 FPGA Product Brief https://www.xilinx.com/support/documents/product-briefs/artix7-product-brief.pdf