A Combined Design of 4-PLL-TRNG and 64-bit CDC-7-XPUF on a Zynq-7020 SoC

Oğuz Yayla [1†] and Yunus Emre Yılmaz [1,2*†]

[1*]Institute of Applied Mathematics, Middle East Technical University, 06800, Ankara, Turkey.
[2]Aselsan Inc., 06200, Ankara, Turkey.

*Corresponding author(s). E-mail(s): yeylmz@gmail.com;
Contributing authors: oguz@metu.edu.tr;
†These authors contributed equally to this work.

Abstract

True Random Number Generators (TRNGs) and Physically Unclonable Functions (PUFs) are critical hardware primitives for cryptographic systems, providing randomness and device-specific security. TRNGs require complete randomness, while PUFs rely on consistent, device-unique responses. In this work, both primitives are implemented on a System-on-Chip Field-Programmable Gate Array (SoC FPGA), leveraging the integrated Phase-Locked Loops (PLLs) for robust entropy generation in PLL-based TRNGs. A novel backtracking parameter selection algorithm for the TRNG implementation is employed, alongside a methodology to boost data generation rates without compromising entropy. The design is rigorously evaluated using the German BSI AIS-20/31 standards. For the PUF implementation, the Arbiter PUF, known for its lightweight design and key generation, is enhanced to resist machine learning attacks by implementing a 32-bit and a 64-bit component-differentially challenged XOR Arbiter PUF (CDC-XPUF). These designs are tested using standard PUF metrics, including uniformity, correctness, and uniqueness. Finally, a combined 4-PLL-TRNG and 64-bit CDC-XPUF design is introduced and evaluated for its suitability in Internet-of-Things (IoT) systems, demonstrating strong performance in both TRNG and PUF tests. The tests are conducted on the Xilinx Zynq 7020 SoC using a ZC702 evaluation board, confirming the effectiveness of this integrated approach for secure, low-resource applications like IoT.

Keywords: PLL-TRNG, CDC-XPUF, SoC, FPGA

1 Introduction

Random numbers play a fundamental role in cryptography, where they are used to generate confidential keys, padding data, initialization vectors, and nonces in challenge-response protocols. These random numbers also generate masks to mitigate side-channel attacks. Random Number Generators (RNGs), key cryptographic primitives, are designed to produce bit sequences with no discernible patterns, requiring both independence and uniform distribution. Among RNGs, there are two main types: True Random Number Generators (TRNGs) and Pseudo-Random Number Generators (PRNGs). Those who wish to gain more in-depth knowledge about random numbers can refer to [1] and [2] for further study. Recently, a

related hardware primitive, the Physically Unclonable Function (PUF), has emerged. PUFs are used for hardware authentication and generating device-specific keys, making them suitable for secure cryptographic systems. More detailed information about PUFs can be found in [3].

The circuitry for both PUFs and TRNGs demands minimal systematic mismatch to avoid bias. While both produce unpredictable outputs, the key distinction is that PUFs deliver consistent responses to the same challenge, whereas TRNGs generate new, random outputs each time they are run, as indicated in [4].

System-on-chip (SoC) Field-Programmable Gate Arrays (FPGAs), or SoCs, have gained significant popularity due to their ability to combine programmable logic with hard processor cores on a single semiconductor device. This integration offers several advantages, including higher integration density, reduced power consumption, smaller form factors, and improved communication bandwidth between the processor and FPGA. In this work, an SoC was chosen over a traditional FPGA, as the inclusion of hard processor cores in the SoC enables the execution of additional applications directly on the chip, eliminating the need for external hardware connections.

Phase-locked loops (PLLs) are essential components in both FPGAs and SoCs, functioning as feedback control systems that synchronize the phase of a locally generated signal with an input signal. These structures are particularly useful in the design of TRNGs, leveraging their analog properties to provide a source of unpredictable randomness [5]. Additionally, PLLs contribute to enhancing clock distribution and on-chip frequency synthesis, with isolated power sources in FPGAs further improving security. However, the limited number of available PLLs restricts their scalability for larger designs.

A key challenge in PLL-based TRNG design is the selection of optimal settings from a large configuration space. The selected parameters must ensure both adequate entropy generation and sufficient output bit rates, as described in the parameter selection methodology by [6].

Once random numbers are generated, it is essential to evaluate their quality thoroughly. Two prominent methodologies for this evaluation are the NIST SP 800-22 standard and the AIS-20/31 guideline, both of which provide rigorous frameworks for assessing the randomness and security of these numbers. The NIST SP 800-22 suite provides a pass or fail result for each test as it is explained in [7]. If a particular test fails, the randomness of the RNG may be called into question. In contrast, AIS-20/31 offers a more comprehensive evaluation framework aimed at determining the security levels of an RNG as it is described in [8]. For a TRNG, this evaluation not only considers the randomness of the output but also assesses the entropy source and operational security to ensure the generator's robustness from a security standpoint. In this work, by considering the properties of the AIS-20/31, the generated random numbers are evaluated using the AIS-20/31 [8] standard, a methodology set forth by the German Federal Office for Information Security (BSI), to assess their quality and ensure compliance with security standards. This evaluation confirms the improved performance and reliability of the proposed design.

While PLL-based TRNGs offer high entropy, they often suffer from low data output rates. To address this, a novel TRNG design utilizing four PLLs is proposed in [9], significantly enhancing performance. We implement both the classical and our improved approach and present the implementation details of these and the successful AIS-20/31 test results in [9].

The Arbiter PUF, the first silicon-based physically unclonable function, efficiently generates numerous secret keys from input data while maintaining a lightweight structure, making it ideal for device authentication in resource-constrained environments such as IoT systems. However, its vulnerability to machine learning (ML) attacks underscores the need for enhanced designs to bolster security. To mitigate this weakness, various improvements have been made to arbiter PUF designs, specifically targeting resistance to ML attacks, and XOR PUFs were proposed in [10] to improve security against ML attacks.

In [11], we implemented a Component-Differentially Challenged XOR Arbiter PUF (CDC-XPUF), which is resistant to such attacks. Drawing on the designs proposed in [12] and [13], this implementation focuses on using 64-bit or longer challenges and at least seven parallel PUF

streams. Research has shown that these configurations are robust against advanced ML attacks [13].

Given their roles in cryptographic systems, TRNGs and PUFs have become essential components in modern IoT devices. When combined, these primitives can provide a strong root of trust for embedded devices. This integration not only enhances security but also optimizes resource usage, particularly on SoC or FPGA platforms. In this study, after implementing TRNGs and PUFs individually on an SoC in previous works [9] and [11], a combined design is proposed where the random numbers produced by the TRNG are used to generate six out of the seven challenges for the CDC-7-XPUF, excluding the main challenge. Our work presents two primary contributions:

- By combining the 4-PLL-TRNG and the 64-bit CDC-7-XPUF, a design functioning as both a TRNG and a PUF is developed, creating a structure that can serve as a hardware primitive in IoT systems. In this dual structure, the random numbers generated by the TRNG are used to create new challenges by XORing the main challenge in the PUF. Test scenarios are designed considering that both subsystems could operate simultaneously in real-time applications. Within these test scenarios, the tests applied to both the TRNG and the PUF are also applied to this dual structure. As a result of these tests, it is demonstrated that the new structure is secure for use.
- It is shown that the combined structure of 4-PLL-TRNG and 64-bit CDC-7-XPUF has an efficient structure in terms the resources of the SoC. Hence, it is a suitable and promising candidate, allocating the necessary design resources for the software and firmware.

The organization of the paper is as follows:

- In Section 2, preliminary information about TRNGs, PUFs, PLL-TRNG, and SoCs is presented.
- In Section 3, the details of PLL-TRNG and CDC-7-XPUF implementations are presented.
- In Section 4, we describe the implementation of the combined design of 4-PLL-TRNG and 64-bit CDC-7-XPUF in detail and subsequently present the results and comparisons with the discrete implementations of TRNG and PUF.
- In Section 5 The results of the combined implementations and their indications are discussed.
- In Section 6, we give the conclusion of our work and some future directions.

2 Background

2.1 Basics of PLL

Phase-locked loops (PLLs), which vary in number and properties, can be found in any FPGA or SoC. PLLs are feedback control systems that automatically adjust the phase of a locally generated signal to align with the phase of an input signal. A PLL is a circuit (as depicted in Fig. 1) that uses an input signal to synchronize a signal from an embedded oscillator on it. The grey blocks represent the analog components, which cannot be parameterized, whereas the M, N, and C integer division coefficients, depicted in white blocks, need to be configured. These coefficients are essential for calculating the output frequency of the PLL (f_{out}) from the reference frequency (f_{ref}), as described in Equation (1).

$$f_{out} = f_{ref} \times \frac{M}{N \times C} \tag{1}$$

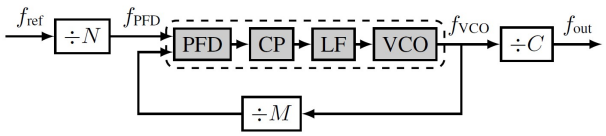

Fig. 1 Block diagram of a PLL (PFD: phase frequency detector, CP: charge pump, LF: loop filter, VCO: voltage-controlled oscillator) [6]

2.2 Random Bit Generation Principle of the PLL-TRNG

The working principle of the PLL-TRNG with one PLL, and also two PLL versions of PLL-TRNG, is presented in Fig. 2 and Fig. 3. The TRNG exploiting the jitter introduced by the PLL, which is presented in Fig. 2, was first proposed in [14]. The jittered clock signal clk_1 from the PLL is sampled by a D flip-flop (D-FF) using the reference clock signal clk_0. The 1-bit counter records the

number of samples that equal one. Due to the frequency relationship established by the PLL, a pattern with a period $T_Q = K_D \times T_0 = K_M \times T_1$ emerges at the flip-flop output. As a result, some samples are consistently 1 (shown as blue in Fig. 2 and these are 4^{th} and 7^{th} dots), some are always 0 (shown as green and these are 2^{nd} and 5^{th} dots), and others are random (shown as red and these are 1^{st}, 3^{rd}, 6^{th}, and 8^{th} dots). By applying the coherent sampling principle and rearranging the samples based on their positions, the waveform of one period of clk_1 can be reconstructed, as it is also described in [15].

This work adopts a two PLL-TRNG architecture as a reference model due to its better performance characteristics. The incorporation of two PLLs significantly enhances design flexibility by expanding the practical operating ranges for critical parameters, K_M and K_D, consequently increasing attainable bit and entropy rates. Moreover, this configuration substantially reduces autocorrelation between output bits. While incurring increased implementation costs, these can often be mitigated through resource sharing with other system components, as proposed in [16].

In this two PLLs case, firstly, as it is stated in Fig. 3, we have the following equality

$$\frac{f_1}{f_0} = \frac{K_M}{K_D}, \tag{2}$$

where K_M and K_D are integer values representing frequency multiplication and division factors, depending on the configuration of PLLs. Each PLL has its multiplication and division factors. Moreover, they are related to K_M and K_D as:

$$K_M = K_{M_1} \cdot K_{D_0} \tag{3}$$
$$K_D = K_{M_0} \cdot K_{D_1} \tag{4}$$

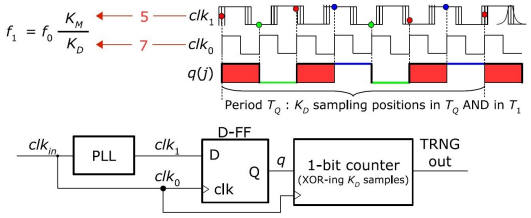

Fig. 2 Principle of the PLL-TRNG with one PLL [15]

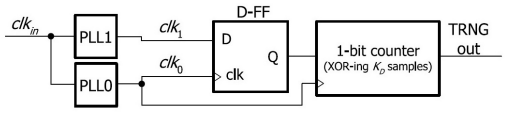

Fig. 3 PLL-TRNG with two PLLs Configuration [15]

The output (Q) of D-FF in the left part of Fig. 2 has a pseudo-random pattern with a certain period. After XORing that pattern in the decimator or 1-bit counter, the bit rate of the PLL-TRNG is defined as follows:

$$R = \frac{f_0}{K_D} = \frac{f_1}{K_M} \tag{5}$$

The entropy rate per bit at generator output depends on the parameters of the jitter and on the parameters of the generator, which are characterized by its sensitivity to the jitter:

$$S = \Delta^{-1} = f_0 \cdot K_M = f_1 \cdot K_D \tag{6}$$

The design of PLL-TRNG relies on choosing appropriate PLL multiplication and division factors. However, selecting these factors can be challenging due to the physical constraints of the PLL, such as the maximum and minimum values of N, M, C, and the input, output, PFD, and VCO frequency range. Consequently, determining these values is an optimization problem, and our solution to this problem is explained in Section 3.1.2 for Zynq 7020 SoC values listed in Table 1.

Table 1 The ranges of possible values for the PLL parameters and frequencies for Zynq-7000 SoC [17], [18]

Parameters	**Xilinx Zynq-7000**	
	Min	**Max**
f_{ref} (MHz)	19	800
P_{VCO}	1	1
M	2	64
N	1	56
C	1	128
$f_{PFD}(MHz)$	19	450
$f_{VCO}(MHz)$	800	1600
f_{out} (MHz)	6.25	464

2.3 Basics of PUFs

PUF extracts entropy from the physical characteristics of an integrated circuit (IC). Each chip exhibits variations due to the inherent unpredictability in the manufacturing process. PUFs harness static entropy from the fluctuations in the manufacturing process. Once the chip is fabricated, the disparities in the manufacturing process become consolidated and undergo minimal changes throughout the chip's lifespan. Consequently, this form of entropy is termed static entropy as it is stated in [19]. Basically, a PUF generates a sequence (response) of the unique signature by input initial states (challenge), so-called challenge-response pairs (CRPs). Each PUF can be represented as a black box, $R = f(C)$, as illustrated in Fig. 4, where the $f()$ is secret.

Fig. 4 Generic PUF model [19]

In the literature, there are various types of PUFs, and they can be classified with respect to their entropy sources and their CRPs [20]. In this research, an intrinsic and delay-based strong PUF, named Arbiter PUF, is implemented. It is important to note that the Arbiter PUF is a strong PUF. A Strong PUF can generate a vast number of challenge-response pairs (CRPs), making it impractical to read all possible CRPs within a reasonable timeframe. This property makes them suitable for applications requiring high security due to their extensive challenge-response space.

2.4 Types of Arbiter PUFs

2.4.1 Basic Arbiter PUF

An Arbiter Physical Unclonable Function (APUF), which was first defined in [21], is a robust PUF relying on delay, featuring a race condition between two symmetrical digital paths. In each delay stage, two multiplexers (MUXes) are incorporated, and their operation is governed by challenges (C_0 C_{n-1}).

Upon activation, the APUF initiates its operation with a trigger signal. This signal traverses two paths determined by a pre-input challenge, ultimately reaching an arbiter. The arbiter then determines which of the two paths is faster in generating the binary response that aligns with the black-box model ($R = f(C)$), as it is illustrated in Fig. 4, where C is the challenge and R is the response.

2.4.2 XOR Arbiter PUF (XOR-PUF)

Due to the limited resistance of arbiter PUFs against machine learning modeling attacks, a new PUF design was introduced in [10]. This new design incorporates a non-linear XOR gate into multiple arbiter PUFs to generate the final response and is referred to as the XOR arbiter PUF. An n-XOR-PUF consists of n-component arbiter PUFs (also known as streams or sub-challenges), wherein the responses from all n-component arbiter PUFs are XORed together at the XOR gate to produce a single-bit response. It is important to note that all component arbiter PUFs in an XOR-PUF are supplied with the same challenge bits as described in [13]. The approach of generating all challenges using the same bitstream, as also argued in [12], has proven insufficient for achieving the desired resistance to machine learning attacks. Consequently, this has led to the development of a new PUF model, CDC-XPUF, which will be detailed in the following section.

2.4.3 Component-Differentially Challenged XOR-PUF (CDC-XPUF)

Component-differentially challenged XOR-PUF (CDC-XPUF), which was introduced in [22] and [23], and XOR-PUF share a similar architecture, comprising multiple arbiter PUF components and XOR gates. The key distinction between CDC-XPUF and XOR-PUF lies in the challenge inputs: each component arbiter PUF in a CDC-XPUF receives different challenge inputs, whereas all component arbiter PUFs in an XOR-PUF receive the same challenges [13]. Fig. 5 illustrates the structure of CDC-XPUF.

Studies such as [24], [25], [26], and [23] highlight that varying the challenges applied to XOR-PUF components can help mitigate vulnerability to machine learning (ML) attacks. Specifically, CDC-XPUFs demonstrate substantial improvements in security as the number of component PUFs increases. Notably, 64-bit CDC-XPUFs

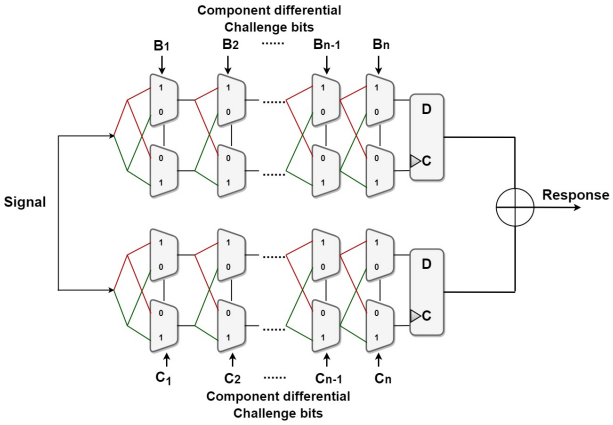

Fig. 5 A CDC-XPUF with 2 streams and n bits of each stream [13]

with seven components have been shown to be entirely resistant to the two ML attack methods tested in [13]. This outcome is highly promising for the IoT security sector, as many CDC-XPUFs, particularly those featuring 64-bit challenges with seven or more components, remain secure against advanced ML attacks. The experimental results in [13] effectively redefine the boundary between secure and insecure regions within the PUF circuit parameter space, providing key insights for PUF manufacturers and IoT security developers to refine protocols in CDC-XPUF-based applications and mitigate potential risks.

In CDC-XPUFs, in order to generate different challenge bits, a pseudorandom number generator (PRNG) structure is proposed in [13] as follows:

$$C_{n+1} = (a * C_n + g) \mod m, \tag{7}$$

where C is the sequence of the generated random number, a is a multiplier, g is a given constant, and $m = 2^K$, where K is the number of stages.

2.5 Evaluation Metrics of TRNGs and PUFs

2.5.1 Evaluation Criteria of TRNGs

The BSI test suites of AIS-20/31 [8] are highly regarded for evaluating the quality of RNGs due to their widespread adoption and recognition as reliable and effective tools. Hence, in this work, for the evaluation metrics of TRNG, AIS-20/31 is chosen. The summary of the tests found in AIS-20/31 under Procedure A and Procedure B in [8], along with brief explanations based on the standard used for cryptographic evaluation of RNGs:

Procedure A in AIS-20/31 Tests: Statistical Testing for Random Number Generators

This procedure focuses on statistical randomness of generated sequences and includes the following key tests:

Test T1 - Monobit Test:

Purpose: Evaluates the balance of 1s and 0s in the binary output of the RNG.
Explanation: Ensures that roughly half of the bits are 1s and half are 0s, which is expected from a random sequence.

Test T2 - Poker Test:

Purpose: Tests the frequency of different bit patterns (like a poker hand).
Explanation: The goal is to assess the uniform distribution of small groups of bits (e.g., 4 bits) in the output. A non-uniform distribution would indicate potential bias or non-randomness.

Test T3 - Runs Test:

Purpose: Evaluates the length and frequency of consecutive sequences of identical bits (runs of 0s or 1s).
Explanation: This test checks if the runs of 0s and 1s appear with the expected frequency and length, as expected in a random sequence.

Test T4 - Long Runs Test:

Purpose: Detects any overly long sequences of identical bits.
Explanation: If the RNG produces unusually long runs of 0s or 1s, this could indicate non-random behavior, which the test aims to capture.

Test T5 - Autocorrelation Test:

Purpose: Measures the correlation between bits in the sequence at various spacings.
Explanation: Ensures that the sequence is not predictable and that the occurrence of one bit does not depend on earlier bits.

Procedure B in AIS-20/31 Tests: Entropy and Stochastic Model Evaluation

This procedure emphasizes evaluating the RNG's entropy source and its model to ensure unpredictability. The focus is less on statistical randomness and more on the inherent unpredictability of the generated bits.

Test T6 - Uniform Distribution Test:

Purpose: This test evaluates whether the output of the random number generator (RNG) follows a uniform distribution.
Explanation: The RNG's output should be uniformly distributed, meaning each possible output value should have an equal probability of occurring. If certain values are more or less frequent, it would indicate a bias, which would compromise the randomness and unpredictability of the RNG. The Uniform Distribution Test checks for this by analyzing the distribution of the generated random numbers.

Test T7 - Test for Homogeneity:

Purpose: This test evaluates whether the output from different sections or time periods of the RNG behaves in a similar (homogeneous) manner.
Explanation: The homogeneity test checks if the random numbers produced by the RNG are consistent over time. It ensures that the quality of randomness doesn't fluctuate between different runs or time periods. If the output from various sections shows significant differences, it could indicate a problem with the RNG's stability or the entropy source, potentially introducing weaknesses in cryptographic applications.

Test T8 - Entropy Estimation Test:

Purpose: To measure the amount of entropy in the output of the RNG, ensuring sufficient randomness.
Explanation: This test calculates the entropy (often min-entropy) of the generated random numbers. Entropy estimation assesses the unpredictability of the sequence by examining how difficult it is to predict the most likely outcome. It ensures that the randomness generated by the RNG has a high degree of unpredictability, which is crucial for cryptographic security. A lower-than-expected entropy value could indicate predictability, thus compromising the security of the random numbers.

Entropy is a very important parameter in evaluating randomness. Hence, how it is evaluated must examined carefully. The entropy values produced by Procedure B of the BSI suite are estimations of the min-entropy H_{min}, which is the most conservative measure of unpredictability, calculated as the negative logarithm of the probability of the most likely outcome. Depending on the application of the implemented entropy source module, another related metric, Shannon entropy H_S, may be required. Shannon entropy can be derived from min-entropy using the following formula:

$$H_S = -2^{-H_{min}} \cdot \log_2\left(2^{-H_{min}}\right) - \left(1 - 2^{-H_{min}}\right) \cdot \log_2\left(1 - 2^{-H_{min}}\right) \quad (8)$$

In addition to these, the BSI specifies that the confidence level of the results is 99.87% [8].

2.5.2 Evaluation Metrics of PUFs

This section outlines a set of PUF characteristics to evaluate the suitability of a PUF design for security applications. Certain statistical properties, such as stability, correctness, diffuseness, uniformity, and uniqueness, can be empirically demonstrated through silicon-based experimentation. Other attributes, including the security vulnerability of PUFs, require computational analysis for thorough assessment.

The first section explains how implemented PUFs are not vulnerable to machine learning (ML) attacks. In the subsequent sections, the evaluation criteria studied and constructed by either Hori et al. [27] or Maiti et al. [28] are explained. They are grouped with respect to three different properties of the responses, and these groups are listed below and explained in detail in the following sections. The metrics in the first and the second groups evaluate the responses of the same PUFs, although the metrics in the third group evaluate how the responses vary between different devices.

The quality of random numbers is pivotal in cryptography, necessitating a thorough evaluation of their properties. While Hori et al. [27] defines

the randomness metric, Maiti et al. [28] defines the uniformity metric. In this work, we think that the uniformity metric is more suitable to use. Because, although randomness in Hori et al. [27] indicates that randomness is evaluated, only some kind of uniformity is evaluated as in [28]. This choice can be understood better by the explanation in *Fingerprint Property* part below. In addition to these, as indicated in [29], in general, how to determine the exact entropy of the PUF responses is another very important open research problem. Consequently, for the PUF implementation, only the uniformity and diffuseness metrics are used to evaluate entropy.

Resistance to Machine Learning (ML) Attacks

PUFs are considered secure due to their inherently unclonable architecture. However, several successful studies have demonstrated that PUFs can be mathematically cloned using the additive delay model, see for instance [30]. Additionally, if adversaries gain access to a sufficient number of silicon CRPs, PUFs may become susceptible to machine learning attacks, as explained in [31], [32], [33], [34]. Therefore, it is imperative for users to ensure that PUFs are resistant to all forms of attacks before deploying them in practical applications.

The study in [13], a comprehensive evaluation of the security of CDC-XPUFs against advanced ML attack methods, utilizing problem-specific parameter values, was conducted to assess the robustness of CDC-XPUFs. Compared to previously reported findings, their study uncovered vulnerabilities in the CDC-XPUF with PUF circuit parameter configurations that were previously not considered insecure. Specifically, they successfully compromised 64-bit CDC-6-XPUFs using approximately 100 million simulated CRPs, and 64-bit CDC-5-XPUFs with 4.5 million simulated CRPs or 2.5 million silicon CRPs. Additionally, they managed to break 128-bit CDC-5-XPUFs with 40 million simulated CRPs, instances that had previously been considered resistant to any existing ML attack methods. Notably, the method in [13] was able to break 64-bit CDC-4-XPUFs using only around 80,000 CRPs, significantly fewer than those used in earlier studies. On the other hand, it also demonstrates that the security of CDC-XPUFs improves substantially as the number of component PUFs increases, with 64-bit CDC-XPUFs featuring seven components proving entirely resilient to the two ML attack methods employed. This finding is particularly encouraging for the IoT security community, as many CDC-XPUFs remain secure, especially those with 64-bit or longer challenges and seven or more component PUFs, which are resistant to the most advanced ML attack methods developed to date. Consequently, the experimental attack study in [13] redefines the boundary between secure and insecure regions within the PUF circuit parameter space, offering valuable insights to PUF manufacturers and IoT security developers for refining the protocols of CDC-XPUF-based applications and mitigating potential risks.

Reliability of Responses From the Same PUFs

PUF responses must be reliable and trusted in real-world applications. A PUF is considered reliable if it consistently generates the same response when the same challenge is applied to the same device. Several factors can affect the reliability of these responses, particularly changes in the operating environment. These factors include, but are not limited to, ambient temperature, humidity, the junction temperature of the circuit, power supply voltage, and circuit aging.

In this work, the environmental variances listed above have not been changed. We have worked at an ambient room temperature of approximately 27^{o}C, stable humidity, and stable core voltage of Zynq-7020 SoC. In terms of the reliability of responses from the same PUFs, steadiness, and correctness are examined in this section.

Steadiness

Steadiness is a reliability metric that is defined by Hori et al. [27]. When generating the same responses multiple times on the same device, it is expected that all responses must be identical. Steadiness indicates how stably a PUF outputs the same responses to the same challenge sets. The steadiness result is 1 if there are no changes in the

responses that were recorded during the experiment. Steadiness can be calculated as follows:

$$S = 1 + \frac{1}{N_c}\sum_{k=1}^{N_c} \log_2 max\{\frac{\sum_{j=1}^{N_a} b_{k,j}}{N_a}, 1 - \frac{\sum_{j=1}^{N_a} b_{k,j}}{N_a}\}, \quad (9)$$

where N_c denotes the number of different challenges used, N_a denotes the number of times each challenge is applied, and $b_{k,j}$ denotes the j-th response among all N_a responses to the k-th challenge in the set of all N_c challenges. The stable CRPs that pass the steadiness test are known as "Correct ID". See [12] for details.

Correctness

This metric is defined by Hori et al. [27] and is almost the same metric as reliability, which is defined by Maiti et al. [28]. The only difference between their equations is the normalization factor. Correctness is normalized by the maximum value of the Fractional Hamming Distance of the responses, while reliability is normalized by the average. Hence, we only computed the correctness value and ignored the reliability. The ideal value of the correctness is 1, which can be calculated as follows:

$$C = 1 - \frac{2}{N_c \times N_a}\sum_{k=1}^{N_c}\sum_{j=1}^{N_a}(b_k \oplus b_{k,j}), \quad (10)$$

where b_k is the "Correct ID". This "Correct ID" is determined by the majority voting of all of the giving responses for the input challenge. N_c is the number of challenges in the dataset. $b_{k,j}$ is the response of the j-th response in the set of all N_a responses to the k-th challenge.

Entropy of Responses From the Same PUFs

A PUF is considered uniform if it generates an equal distribution of zeros and ones in response to a set of challenges. This characteristic is particularly desirable in block and stream cipher processes, as repeated patterns in secret keys are deemed detrimental. In terms of entropy, Hori et al. [27] introduced the diffuseness metric, while Maiti et al. [28] proposed the uniformity metric. Given the close resemblance between Hori's [27] randomness metric and Maiti's [28] uniformity metric, only the uniformity metric is assessed in this context.

Diffuseness

The diffuseness metric, introduced by Hori et al. [27], is an intra-chip metric that assesses the variability of a PUF's responses to different challenges. A PUF is considered to exhibit diffuseness if it produces distinct responses for distinct challenges; for instance, the response to a specific challenge X should differ from the responses generated by other challenges. Diffuseness is quantified by calculating the fractional Hamming distance between the responses produced by the same device in response to a set of challenges. The diffuseness can be computed using the following formula:

$$D = \frac{4}{K^2 \times L}\sum_{l=1}^{L}\sum_{i=1}^{K-1}\sum_{j=i+1}^{K}(b_{i,l} \oplus b_{j,l}), \quad (11)$$

where L is the responses' length, counted in bits, and K is the number of such multi-bit responses used in the experimental study.

Uniformity

The uniformity, which was introduced by Maiti et al. [28], of a PUF measures the degree of zeros and ones in the produced responses. Its ideal value is 0.5. The uniformity can be calculated as follows:

$$U = \frac{1}{N_r}\sum_{i=1}^{N_r} b_i, \quad (12)$$

where N_r is the response length in a set, and bi is the i-th response bit. The randomness metric, defined by Hori et al. [27], is not used for the evaluation since it is very similar to the uniformity. In order to make this statement more clear, the equations to calculate the randomness are provided below:

$$H = -\log_2 max(p, 1-p), \quad (13)$$

where p is the frequency of '1' in the response set given by:

$$p = \frac{1}{N_r} \sum_{i=1}^{N_r} b_i \tag{14}$$

where N_r is the response length in a set, and bi is the i-th response bit. It is obvious that the Equations (12) and (14) are nearly the same. These two equations define the same thing actually, and it is the uniformity of the responses. Hori et al. [27] claim that taking this uniformity and using them in (13) calculates the randomness. The approach presented by Hori et al. [27] is not suitable for accurately calculating randomness. Equation (13) can only provide information regarding the percentage distribution of 0s and 1s, which is already captured in the uniformity metric proposed by Maiti et al. [28] in Equation (12). Thus, using this equation does not contribute to a deeper understanding of randomness beyond what uniformity already indicates. As stated in [29], how to determine the exact entropy of the PUF responses is another very important open research problem. Hence, in order to evaluate entropy, we use the uniformity metric introduced by Maiti et al. [28].

Fingerprint Property

In this paper, we only consider the uniqueness metric as the fingerprint property. PUF uniqueness measures the variation in outputs across different devices, serving as a key metric for determining a device's ability to uniquely identify itself, which is critical for secure authentication.

Uniqueness

The uniqueness was introduced by Maiti et al. [28], and it can be calculated using a Hamming Distance between two devices' responses. The calculation is as follows:

$$U_k = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{HD(ID_i, ID_j)}{L} \tag{15}$$

where ID_i and ID_j are two L-bit responses of a PUF installed on two different chips (the i-th and j-th chip) to the k-th challenge repeatedly applied L times. The ideal value of the Maiti's uniqueness [28] is 0.5. In addition to these metrics, the resource utilization rate is a metric for both TRNGs and PUFs.

2.5.3 The Resource Utilization Ratio of the SoC

Although it is not included in any standard evaluation method, the resource utilization rate within the SoC or FPGA has also been a key evaluation metric in our study. This is because our goal is to minimize the resource usage of the hardware primitives we utilize, ensuring that there is still space available for other designs that will be implemented for additional applications within the SoC or FPGA. This metric has been applied for both the TRNGs and PUFs designs.

3 Implementation Details

In this section, the implementation details of the combined TRNG and PUF are presented. Before explaining the combined design, the TRNG and PUF parts are explained in a detailed way.

3.1 4-PLL-TRNG Implementation

The internal structure and the process of determining the PLL-TRNG parameters are explained in the following sections.

3.1.1 Internal Structure of 4-PLL-TRNG Configuration

A primary limitation of PLL-TRNGs is their comparatively low output data rate. To address this constraint, this work proposes a methodology to enhance output capacity by leveraging additional PLLs available within the SoC. The Zynq 7020 SoC, featuring four PLLs, represents the upper bound for this implementation. The implemented PLL-TRNG with four PLLs, named 4-PLL-TRNG, with a specific interconnection is shown in Fig. 6. In this configuration, two PLLs are used as reference clocks, while two PLLs are used as jittered clocks.

3.1.2 Determining PLL-TRNG Parameters

In our combined design, as the parameter search algorithm, the backtracking algorithm in [6] is

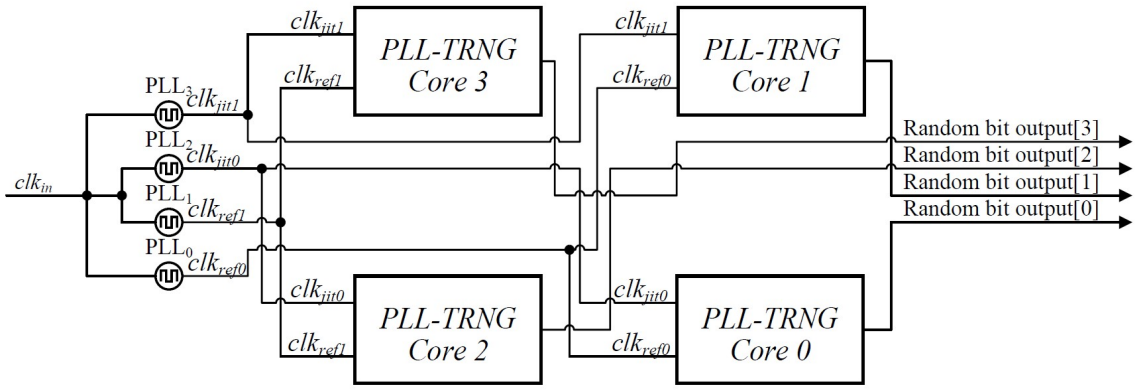

Fig. 6 Implemented 4-PLL-TRNG Configuration

selected. Given a set of variables explained in Section 2.1 and Section 2.2 and constraints listed in Table 1, this backtracking method iteratively investigates potential solutions. Unlike a brute-force approach, it promptly eliminates any variable values that fail to meet a constraint, then backtracks to explore other possible values until all valid solutions are identified. The algorithm detailed in [6] involves determining the PLL-TRNG parameters that comply with both physical constraints and application requirements.

The code in the backtracking is open-source shared in [35]. Hence, we can modify it for Zynq 7020 SoC parameters provided in Fig. 1. Table 1 presenting the range of PLL values of Zynq-7020 SoC is prepared using [18]. However, the maximum value of f_{out} is not determined by PLL parameters, it is determined by BUFG properties. BUFG must be used in the SoC design, and hence, it restricts f_{out} value for the search algorithm. That maximum value can also be found in [18].

After determining the parameters for our implementation, the algorithm results are ordered with respect to three different configurations. Those are the maximum bit rate (max. R), the maximum sensitivity to jitter (max. S), and the maximum $R \cdot S$ value as the optimization between max. S and max. R. After obtaining the candidate results for three different configurations of the PLL-TRNG implementation, those results must be tested with one more criterion. The sampling process of the jittered clock with the reference clock is illustrated in Fig. 2. In order to obtain random numbers at the output of this PLL-TRNG, at least one sample is required to be affected by the jitter. This necessitates that the distance between any edge of clk_0 and its corresponding edge on clk_1 must be less than Δ. This condition is met if the following condition holds, as given in [36] and [37]:

$$\sigma_{jit} > max(\Delta T_{min}), \tag{16}$$

where σ_{jit} is the standard deviation of the jitter at the output of the PLL, and $max(\Delta T_{min})$ is the largest distance between the two closest edges of clk_0 and clk_1. This can be computed as given below, see [36] and [37]:

$$\begin{aligned} max(\Delta T_{min}) &= \frac{T_{clk_0}}{4K_M} gcd(2K_M, K_D) \\ &= \frac{T_{clk_1}}{4K_D} gcd(2K_M, K_D), \end{aligned} \tag{17}$$

where gcd is the greatest common divisor of two integers.

Upon executing the backtracking algorithm and obtaining results for the selected SoC, the maximum value of $max(\Delta T_{min})$ can be determined. However, accurately measuring or estimating σ_{jit} presents significant challenges. At this juncture, the estimation tool named *Clocking Wizard* in Vivado 2019.1 can be utilized. This tool provides an estimation of the jitter at the PLL's output clock, given the PLL parameters. Consequently, the results from the backtracking

algorithm are first examined, and max. R, max. S and the max. $R \cdot S$ are identified. These three candidates are then evaluated against Equation (16). Candidates failing to satisfy the equation are discarded, and alternative candidates from the backtracking results are considered.

The results of the search algorithms are listed in Table 2. As it can be seen, all the selected configurations satisfy Equation (16).

3.2 64-bit CDC-7-XPUF Implementation

In our combined design, 64-bit CDC-7-XPUF is implemented due to the considerations about the ML attacks. Details about these considerations are presented in Section *Resistance to Machine Learning (ML) Attacks*. In the version presented in the literature, the challenges except the main challenge are derived from the main challenge using PRNG, whose equation is presented in Equation (7). However, this method is changed in the combined design and the details of it are presented in Section 3.3.

For the statistical characteristics CRPs, we generated up to 16,000 (challenges) $\times$ 32 (iterations) $\times$ 128 (response length) $\times$ 3 (Zynq 7020 SoCs) CRPs out of each design. The repetition of the CRPs is needed to study the statistical characteristics and investigate related metrics such as correctness and steadiness. The CRPs were captured at an ambient temperature of approximately 27°C, and the core voltage was set to 1.0V. The ambient temperature does not reflect the temperature of the chip, which has changed as long as the experiments continue.

3.3 Implementation of the Combined Design 4-PLL-TRNG and CDC-7-XPUF

4-PLL-TRNG is a version of PLL-TRNG prepared with four discrete PLLs. The detailed illustration of this version is shown in Fig. 6. There are three different configurations of PLL-TRNGs in this work, as explained in Section 3.1.2. Those are the maximum bit rate (max. R), the maximum sensitivity to jitter (max. S), and the maximum $R \cdot S$ value as the optimization between max. S and max. R. In the combined designs, we choose max. R to work with and use the values in Table 2 for PLL configuration.

In the PUF part, in order to differentiate the challenges, a new approach is applied instead of PRNGs. The results in [38] demonstrate that PUFs utilizing fix-point-free permutations exhibit a level of resistance to machine learning attacks that is nearly equivalent to that of pseudorandom input transformations, which [38] asserts to be the most robust approach in mitigating such adversarial techniques. Furthermore, the proposed design incurs minimal hardware overhead, as it solely involves a fixed routing mechanism for challenge bits to the individual arbiter chains. This fix-point-free transformation is a one-to-one and onto (or a bijective) function. In our work, we use another bijective and random transformation to generate other challenges from the main challenge. In that method, we apply the XOR operation to the main challenge with the random numbers, which is the output of the 4-PLL-TRNG. As it is in [38], we expect resistance to ML attacks.

The block diagram of this combined design is shown in Fig. 7. CDC-7-XPUF needs random numbers provided from 4-PLL-TRNG in order to generate challenges. Hence, the 4-PLL-TRNG must first be run, and random numbers must be obtained. For this application, we have six 64-bit challenges in addition to one 64-bit main challenge. Hence, we use $6 \times 64 = 384$ bit of random numbers. Additionally, two discrete BRAMs connected to these systems are used to allow the two subsystems to operate separately. However, since there is only one processor on the PS side, the BRAMs are transferred to the PC via UART and USB as random numbers or responses fill them. Since the processor can only deal with one BRAM at a time, this structure creates a bottleneck in terms of throughput. In real applications, this bottleneck can be overcome by using different architectures. The architecture to be employed will vary depending on whether the random numbers and responses are used within the SoC or, as in our implementations, transmitted externally, as well as the interface used for external transmission. One possible architectural design involves converting the 4-PLL-TRNG and 64-bit CDC-7-XPUF into intellectual property (IP) cores, enabling communication with relevant units and the hard processor via the Advanced eXtensible

Table 2 Determined Parameters for the PLL-TRNG Implementations

Config.	f_{ref} (MHz)	(M_0, N_0, C_0) (M_1, N_1, C_1)	f_0 (MHz) f_1 (MHz)	K_M	K_D	R (Mbit/s)	S (ps^{-1})	$R \cdot S$	σ_{jit}	$max(\Delta T_{min})$
Max. R	125	(51,4,4) (11,1,3)	398.438 458.333	176	153	2.60417	0.07013	0.18263	76.706	3.56506
Max. S	125	(51,4,4) (32,3,3)	398.438 444.444	512	459	0.86806	0.204	0.177084	100.882	1.22549
Max. $R \cdot S$	125	(37,5,2) (32,3,3)	462.5 444.444	320	333	1.38889	0.148	0.20556	100.882	1.68919

Interface (AXI) bus. Additionally, memory buffers may be incorporated into the design to accommodate the communication speeds of the interacting units.

The aim of the combined design is that in a real-world application, both should be able to work together, but that this interoperability should not adversely affect the performance of the other subsystem. To demonstrate that this aim is achieved, three different test setups are prepared. These three setups share the structure shown in Fig. 7. However, due to the different implementations, only the content of the state machines and hence the wrapper code changes.

The purpose of building the first test setup is to run 4-PLL-TRNG and CDC-7-XPUF sequentially to provide a reference run for the other two test setups. In the first test setup, 4-PLL-TRNG is run first. Then, the 6×64 bit random numbers are taken by CDC-7-XPUF, and CDC-7-XPUF is run.

The purpose of building the second test setup is to show that the continuous operation of the TRNG does not affect the PUF. In the second test setup, the 4-PLL-TRNG is run for one round first. This is because CDC-7-XPUF needs random numbers for the other 6 challenges other than the main challenge. After the random numbers are generated, CDC-7-XPUF starts to run. However, while CDC-7-XPUF runs, 4-PLL-TRNG also keeps running without stopping. In this test run, only the results of the CDC-7-XPUF are recorded.

The purpose of the third test setup is to show that, unlike the second test setup, the continuous operation of CDC-7-XPUF does not cause any deterioration in the performance of 4-PLL-TRNG. In the third test setup, the 4-PLL-TRNG is run for one round first. This is because CDC-7-XPUF needs random numbers for the other 6 challenges other than the main challenge. After the random numbers are generated, CDC-7-XPUF starts to run. However, while 4-PLL-TRNG runs, CDC-7-XPUF also keeps running without stopping. In this test run, only the results of the 4-PLL-TRNG are recorded. In order to be sure that CDC-7-XPUF runs continuously, the result of the fifth run of 4-PLL-TRNG is recorded.

The implementation setup employed in this study is illustrated in Fig. 7. It utilizes the ZC702 Rev1.1 Evaluation Board [39], which incorporates the Zynq 7020 XC7Z020-1CLG484C SoC to facilitate the three different test implementations of the combined design 4-PLL-TRNG and CDC-7-XPUF as detailed in this section. In the PL section, three different test setups are developed using Vivado 2019.1 [40] in VHDL [41]. Two dual-access BRAM blocks are employed to enable real-time transmission of generated random numbers and CRPs to the PC. One port of each of these BRAMs is connected to the PL, while the other ports are connected to the PS section. The requisite code for the PS section is written in the C programming language [42]. The PL section generates random numbers and CRPs and writes a predefined value to specific BRAM addresses to indicate that the random bits or CRPs are ready. Once this indication is given, the software in the PS section outputs the random bits or CRPs to the UART serial port, which are then converted to USB and transmitted to the PC. The received random bits or CRPs on the PC are saved in their ASCII-coded hexadecimal form. Random bits are later converted to binary form offline to serve as input for AIS-20/31 Tests [43]. The codes of AIS-20/31 Tests were compiled using [44] as it was. Both Procedure A and Procedure B Tests of AIS-20/31 are conducted for each result. Given that these tests require approximately 7 Mb of random bits, each output file is generated with a size of approximately 7.2 Mb. For the PUF part, as it is indicated in Section 3.2, for the statistical characteristics CRPs, we generated up to 16,000 (challenges) $\times$ 32 (iterations) $\times$ 128 (response length) $\times$ 3 (Zynq 7020 SoCs) CRPs out of each design. The repetition of the CRPs is needed to study the statistical characteristics and investigate related metrics such as correctness and steadiness. In order to calculate scores of metrics for the PUF

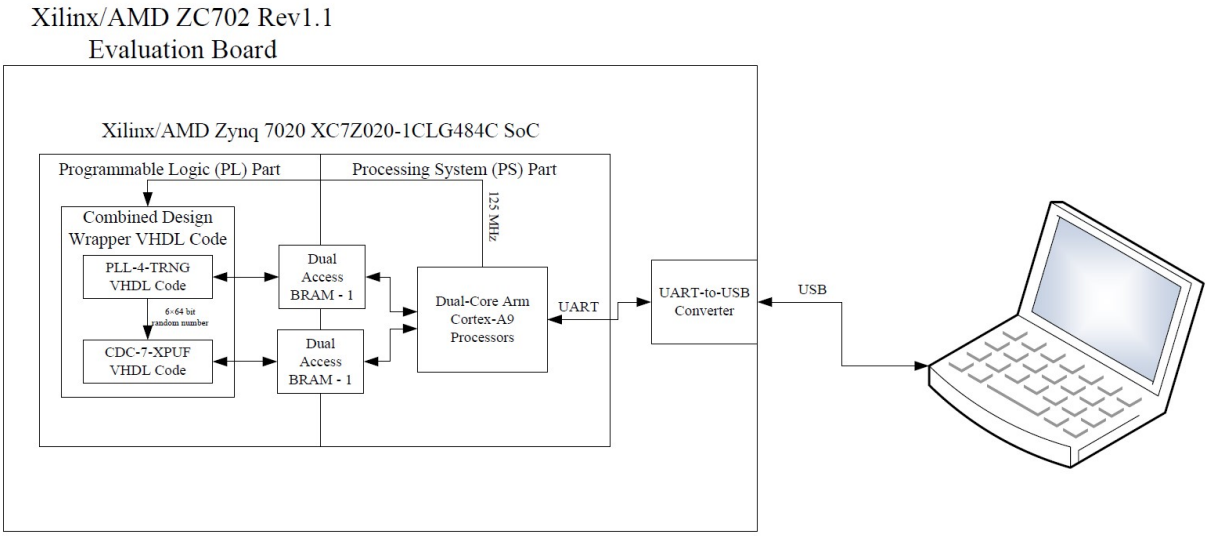

Fig. 7 Block Diagram of Implementation Setup of the Combined Design of 4-PLL-TRNG and 64-bit CDC-7-XPUF

part, test codes are prepared in Python [45] and compiled using Microsoft Studio 2022 [46]. Additionally, a 125 MHz clock frequency was selected for the system's main clock (clk_{in}) due to timing constraints inherent in the SoC. From the PS part via UART, the random bits and the CRPs were sent to the PC with a baud rate of 230,400 bits/second between the PuTTY [47] terminal and the SoCs.

In the setup in Fig. 7, the random bits and CRPs were captured at an ambient temperature of approximately 27°C, and the core voltage was set to 1.0V. The ambient temperature does not reflect the temperature of the chip, which has changed as long as the experiments continue. Through a dual-access BRAM, random bits and CRPs were sent to the PS part.

In the following section, the implementation results for those three different test setups are presented.

4 Implementation Results of the Combined Design 4-PLL-TRNG and CDC-7-XPUF

A total of three different test configurations are examined in this section:

1. **Combined Design (a)**: In this first test setup, 4-PLL-TRNG, and CDC-7-XPUF are run sequentially to provide a reference run for the other two test setups. Both generated random numbers and responses are recorded in this configuration.
2. **Combined Design (b)**: In the second test setup, TRNG works continuously, and after the first run, only generated responses by PUF are recorded.
3. **Combined Design (c)**: In the third test setup, PUF works continuously, and after the first run, only generated random numbers by TRNG are recorded.

4.1 Implementation Results of the Random Numbers in 4-PLL-TRNG of Combined Designs

AIS-20/31 test results of the combined designs and for the reference discrete implementation of 4-PLL-TRNG with max. R configuration are presented in Table 3.

The Shannon entropy values of the combined designs and, for reference, the discrete implementation of 4-PLL-TRNG with max. R configuration with respect to AIS-20/31 test are presented in

Table 3 AIS-20/31 Test Results of the Combined Designs and the Reference Design of 4-PLL-TRNG

Implementation Type / Tests	Procedure A - T0 Result	Procedure A - T1-T5 Result	Procedure B - T6-T8 Result
Discrete Implementation of 4-PLL-TRNG with max. R Configuration	PASSED	PASSED	PASSED
Combined Design (a)	PASSED	PASSED	PASSED
Combined Design (c)	PASSED	PASSED	PASSED

Table 4 The Shannon Entropy Results with Respect to AIS-20/31 of the Combined Designs and the Reference Design of 4-PLL-TRNG

Implementation Type	Entropy (Shannon)
Discrete Implementation of 4-PLL-TRNG with max. R Configuration	0.999999972332402
Combined Design (a)	0.999999992061094
Combined Design (c)	0.999999971489810

Table 5 The Steadiness Results of the Combined Designs and the Reference Design of 64-bit CDC-7-XPUF

Implementation Type	Steadiness Score
Discrete Implementation of 64-bit CDC-7-XPUF	96.70%
Combined Design (a)	96.95%
Combined Design (b)	96.75%

Table 4. As expected, the Shannon entropy values of the referenced and the combined designs are very close to each other.

4.2 Implementation Results of the Responses in CDC-7-XPUF of Combined Designs

The results of various metrics for the combined and discrete implementations of the 64-bit CDC-7-XPUF are presented in Tables 5, 6, 7 and 8. *Steadiness*, *correctness*, *diffuseness*, and *uniformity* scores for the combined designs are nearly identical to those of the discrete implementation, as expected. However, a notable improvement in *uniqueness* is observed in the combined designs, primarily due to the use of random numbers generated by the 4-PLL-TRNG for producing additional challenges, as detailed in Table 9.

Table 6 The Correctness Results of the Combined Designs and the Reference Design of 64-bit CDC-7-XPUF

Implementation Type	Correctness Score
Discrete Implementation of 64-bit CDC-7-XPUF	96.19%
Combined Design (a)	96.46%
Combined Design (b)	96.25%

Table 7 The Diffuseness Results of the Combined Designs and the Reference Design of 64-bit CDC-7-XPUF

Implementation Type	Diffuseness Score
Discrete Implementation of 64-bit CDC-7-XPUF	99.99%
Combined Design (a)	99.99%
Combined Design (b)	99.99%

Table 8 The Uniformity Results of the Combined Designs and the Reference Design of 64-bit CDC-7-XPUF

Implementation Type	Uniformity Score
Discrete Implementation of 64-bit CDC-7-XPUF	49.89%
Combined Design (a)	50.05%
Combined Design (b)	49.81%

4.3 Utilizations of Combined Designs of Zynq-7020 SoCs

The utilization results of the combined designs and, for reference, the discrete implementation of 4-PLL-TRNG with max. R configuration and 64-bit CDC-7-XPUF are presented in Table 10. As expected, the utilization rates of the combined designs are very close to each other. Although these usage rates are higher than those of discrete implementations, they still consume fewer resources than a design where discrete components are used separately, each consuming resources individually. In other words, the combined design offers a lower utilization rate than a design that

Table 9 The Uniqueness Results of the Combined Designs and the Reference Design of 64-bit CDC-7-XPUF

Implementation Type	Uniqueness Score
Discrete Implementation of 64-bit CDC-7-XPUF	18.96%
Combined Design (a)	50.30%
Combined Design (b)	50.07%

includes the discrete 4-PLL-TRNG and 64-bit CDC-7-XPUF implementations.

5 Discussion About Implementation Result of the Combined Designs

The aim of combining the 4-PLL-TRNG and the 64-bit CDC-7-XPUF in a unified design is to maintain the cryptographic properties of these two subsystems while achieving a more compact solution rather than utilizing them separately and embedding them within the SoC. In this integration process, the random numbers generated by the TRNG are used to create additional challenges within the PUF aside from the main challenge. This approach not only allows for a mutualistic integration where the output of one subsystem is utilized by the other but also simplifies the operations required for generating additional challenges in the CDC-7-XPUF by replacing the multiplication and addition processes with a simple XOR operation.

In order to test this combined structure, three different test configurations are created, one of which serves as the reference. In the first test configuration, which is also the reference configuration, random numbers are generated in the initial stage via the 4-PLL-TRNG. Once this subsystem completes its operation, a 6×64-bit random number required for the operation of the 64-bit CDC-7-XPUF is transferred to the PUF subsystem, and the additional challenges are generated by XORing these random numbers with the main challenge. The PUF then begins the process of generating the response. It should be noted at this point that the two subsystems are not operated simultaneously. However, in a real application, such as in an IoT system, the concurrent operation of these two subsystems would be naturally desirable. For this purpose, two additional test configurations are designed. In one of these configurations, the PUF results are tested while the TRNG continuously operates, and in the other, the opposite scenario is tested.

In the second test configuration, which is created to examine these scenarios, random numbers are generated first, similar to the first configuration. The 6×64-bit random number is then transferred to the PUF for challenge generation. However, unlike the first configuration, the TRNG is not stopped and continues to operate. Meanwhile, the PUF generates the response, and the results are recorded on the PC. In this configuration, the effect of continuous TRNG operation on the PUF is examined.

In the third test configuration, as in the previous ones, the TRNG is initially activated, and the random number transfer process to the PUF subsystem is repeated. Once the PUF starts generating challenges with these random numbers, the TRNG continues to run to observe its potential interference with the PUF. During this period, the TRNG is reactivated three more times while the PUF continues to operate in a loop without interruption. The results of these four TRNG activations, including the first one, are not recorded. On the fifth and final activation, the TRNG is run again, and the random number outputs are recorded on the PC. Consequently, in this last test configuration, the effect of continuous PUF operation on the TRNG is analyzed.

These three configurations are named *Combined Design (a)*, *Combined Design (b)*, and *Combined Design (c)*. All of these test configurations are implemented in the test setup whose block diagram is presented in Fig. 7.

Starting with the evaluation of the random number generation results in this combined structure, the random numbers are assessed using AIS-20/31 tests, and their Shannon entropies are calculated using the formula in Equation (8). For these evaluations, the reference 4-PLL-TRNG in [9] (with the max. R configuration) is used alongside Combined Design (a) and Combined Design (c). Tables 3 and 4 are prepared for this assessment. As expected, all three configurations pass the AIS-20/31 tests, and very similar Shannon entropy values are measured in each case. Consequently, these tests demonstrate that there is no

Table 10 The Utilization Rates of the Combined Designs and the Reference Design of 4-PLL-TRNG and 64-bit CDC-7-XPUF

Resource Type	Avaliable Resource Quantity	Utilization % of 4-PLL-TRNG with max. R Configuration (Utilization)	Utilization % of 64-bit CDC-7-XPUF (Utilization)	Utilization % of Combined Design (a) (Utilization)	Utilization % of Combined Design (b) (Utilization)	Utilization % of Combined Design (c) (Utilization)
LUT	53200	2.89% (1539)	2.82% (1500)	4.64% (2469)	4.65% (2474)	4.64% (2466)
LUTRAM	17400	0.41% (72)	0.41% (72)	0.47% (82)	0.47% (82)	0.47% (82)
FF	106400	1.77% (1884)	1.67% (1781)	2.79% (2965)	2.79% (2966)	2.76% (2936)
BRAM	140	2.86% (2)	2.86% (2)	2.86% (4)	2.86% (4)	2.86% (4)
DSP	220	0% (0)	5.45% (12)	4.55% (10)	4.55% (10)	4.55% (10)
IO	200	4.00% (8)	4.00% (8)	4.00% (8)	4.00% (8)	4.00% (8)
PLL	4	100% (4)	0% (0)	100% (4)	100% (4)	100% (4)

issue in utilizing the combined design in terms of random number generation.

The evaluation metrics for PUFs are thoroughly examined in Section 2.5.2. These metrics include resilience against ML attacks, steadiness, correctness, diffuseness, uniformity, and uniqueness. For these evaluations, the reference 64-bit CDC-7-XPUF in [11] is used in conjunction with Combined Design (a) and Combined Design (b).

The evaluation of the combined PUF design focuses on six key metrics: *resilience against ML attacks*, *steadiness*, *correctness*, *diffuseness*, *uniformity*, and *uniqueness*. *Resilience against ML attacks* is enhanced through XORing challenges with random numbers from the 4-PLL-TRNG, as opposed to a fixed transformation, improving defense against ML attacks as it is explained in Section 3.3 and mentioned in [38]. *Steadiness*, *correctness*, *diffuseness*, and *uniformity* results are nearly identical to those of the reference design, confirming the design's suitability. Notably, *uniqueness* shows significant improvement due to better challenge generation, confirming the combined design's overall effectiveness.

Lastly, we compared the utilization rate of the separate designs with the combined designs in Table 10. Through simple mathematics, it is shown that resources expected to be more heavily utilized when used separately are used more efficiently in the combined design. As can be seen in Table 10, the utilization of all resources except the PLL remained below 5%. This indicates that when this combined design is used, there is still room in the SoC for other designs to be implemented for additional applications.

6 Conclusion and Future Works

In conclusion, we have introduced and implemented a combined 4-PLL-TRNG and 64-bit CDC-7-XPUF in a unified design. While these components have been implemented independently in previous works [9] and [11], their integration follows the current trend of combining TRNG and PUF hardware primitives for enhanced efficiency and security. This combined design is evaluated in terms of the TRNG and PUF metrics, and we concluded that the combined design is highly suitable for use in IoT systems. The analysis and tests conducted in this work have demonstrated that the combined design retains all the features of the separately designed 4-PLL-TRNG and 64-bit CDC-7-XPUF.

As a future work, in addition to AIS-20/31, NIST's test suite [7] would also be applied. On the other hand, the results of the combined design would be tested in various environmental conditions such as varying temperature and varying voltage. Our combined design is also applicable to other FPGA and SoC platforms, and so this design would be tested on other platforms.

Acknowledgements. The authors acknowledge Aselsan Inc. for its support during the preparation of this paper and for providing three Xilinx ZC702 Evaluation Boards that were utilized during the implementation of the combined 4-PLL-TRNG and 64-bit CDC-7-XPUF algorithms described in the paper.

References

[1] D.E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*

(Addison-Wesley, Boston, 1997)
[2] Ç.K. Koç, *Cryptographic Engineering* (Springer, New York, 2009)
[3] M. Hofer, C. Böhm, *Physical Unclonable Functions in Theory and Practice*, 1st edn. (Springer New York, NY, 2012). https://doi.org/10.1007/978-1-4614-5040-5
[4] K. Pratihar, U. Chatterjee, M. Alam, R.S. Chakraborty, D. Mukhopadhyay, Birds of the Same Feather Flock Together: A Dual-Mode Circuit Candidate for Strong PUF-TRNG Functionalities. IEEE Transactions on Computers **72**(6), 1636–1651 (2023). https://doi.org/10.1109/TC.2022.3218986
[5] V. Fischer, M. Drutarovský, *True Random Number Generator Embedded in Reconfigurable Hardware*, in *Cryptographic Hardware and Embedded Systems - CHES 2002*, ed. by B.S. Kaliski, Ç.K. Koç, C. Paar (Springer Berlin Heidelberg, Berlin, Heidelberg, 2003), pp. 415–430
[6] B. Colombier, N. Bochard, F. Bernard, L. Bossuet, *Backtracking Search for Optimal Parameters of a PLL-based True Random Number Generator*, in *Proceedings of the 23rd Conference on Design, Automation and Test in Europe* (EDA Consortium, San Jose, CA, USA, 2020), DATE '20, p. 1–6
[7] N.I. of Standards, Technology, A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special Publication 800-22, Revision 1a, National Institute of Standards and Technology, Gaithersburg, MD, USA (2010). URL https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final
[8] B. für Sicherheit in der Informationstechnik (BSI). AIS 20/31 - Functionality Classes for Random Number Generators. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.html (2011)
[9] O. Yayla, Y.E. Yılmaz. Design and Implementation of a Fast, Platform-Adaptive, AIS-20/31 Compliant PLL-Based True Random Number Generator on a Zynq 7020 SoC FPGA. Cryptology ePrint Archive, Paper 2024/1442 (2024). https://eprint.iacr.org/2024/1442
[10] G.E. Suh, S. Devadas, *Physical Unclonable Functions for Device Authentication and Secret Key Generation*, in *2007 44th ACM/IEEE Design Automation Conference* (2007), pp. 9–14
[11] O. Yayla, Y.E. Yılmaz. 32-bit and 64-bit CDC-7-XPUF Implementation on a Zynq-7020 SoC. Cryptology ePrint Archive, Paper 2024/1443 (2024). URL https://eprint.iacr.org/2024/1443
[12] K.T. Mursi, From XOR PUF to CDC XOR PUF: Cost-Effectiveness, Statistical Characteristics, and Security Assessment. Ph.D. thesis, Texas Tech University (2021)
[13] G. Li, K.T. Mursi, A.O. Aseeri, M.S. Alkatheiri, Y. Zhuang, A New Security Boundary of Component Differentially Challenged XOR PUFs Against Machine Learning Modeling Attacks. International Journal of Computer Networks & Communications (IJCNC) **14**(03), 1–15 (2022). https://doi.org/10.5121/ijcnc.2022.14301. https://aircconline.com/ijcnc/V14N3/14322cnc01.pdf
[14] V. Fischer, M. Drutarovský, *True Random Number Generator Embedded in Reconfigurable Hardware*, in *Workshop on Cryptographic Hardware and Embedded Systems* (2002). https://api.semanticscholar.org/CorpusID:5441670
[15] V. Fischer, F. Bernard, N. Bochard, Modern Random Number Generator Design – Case Study on a Secured PLL-Based TRNG. it - Information Technology **61**(1), 3–13 (2019). https://doi.org/doi:10.1515/itit-2018-0025
[16] O. Petura, U. Mureddu, N. Bochard, V. Fischer, *Optimization of the PLL based TRNG design using the genetic algorithm*, in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (2017), pp. 1–4. https://doi.org/10.1109/ISCAS.2017.8050839
[17] 7 Series FPGAs Clocking Resources User Guide (UG472) (v1.14). https://docs.amd.com/v/u/en-US/ug472_7Series_Clocking. Accessed: 2024-06-25
[18] Zynq-7000 SoC: DC and AC Switching Characteristics (DS187) (v1.21). https://docs.amd.com/v/u/en-US/ds187-XC7Z010-XC7Z020-Data-Sheet. Accessed: 2024-02-25

[19] Y. Cao, W. Liu, L. Qin, B. Liu, S. Chen, J. Ye, X. Xia, C. Wang, Entropy Sources Based on Silicon Chips: True Random Number Generator and Physical Unclonable Function. Entropy **24**(11) (2022). https://doi.org/10.3390/e24111566

[20] M.B.R. Srinivas, K. Elango, Era of sentinel tech: Charting hardware security landscapes through post-silicon innovation, threat mitigation and future trajectories. IEEE Access **12**, 68061–68108 (2024). http://dx.doi.org/10.1109/ACCESS.2024.3400624

[21] B. Gassend, D. Clarke, M. van Dijk, S. Devadas, *Silicon physical random functions*, in *Proceedings of the 9th ACM Conference on Computer and Communications Security* (Association for Computing Machinery, New York, NY, USA, 2002), CCS '02, p. 148–160. https://doi.org/10.1145/586110.586132

[22] M. Majzoobi, F. Koushanfar, M. Potkonjak, *Lightweight secure PUFs*, in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design* (IEEE Press, 2008), ICCAD '08, p. 670–673

[23] M.D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, I. Verbauwhede, A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. IEEE Transactions on Multi-Scale Computing Systems **2**(3), 146–159 (2016). https://doi.org/10.1109/TMSCS.2016.2553027

[24] K.T. Mursi, Y. Zhuang, *Experimental Examination of Component-Differentially-Challenged XOR PUF Circuits*, in *Journal of Physics: Conference Series*, vol. 1729 (IOP Publishing, 2021), p. 012006. https://dx.doi.org/10.1088/1742-6596/1729/1/012006

[25] K.T. Mursi, B. Thapaliya, Y. Zhuang, A.O. Aseeri, M.S. Alkatheiri, A Fast Deep Learning Method for Security Vulnerability Study of XOR PUFs. Electronics **9**(10) (2020). https://www.mdpi.com/2079-9292/9/10/1715

[26] N. Wisiol, G.T. Becker, M. Margraf, T.A.A. Soroceanu, J. Tobisch, B. Zengin. Breaking the lightweight secure PUF: Understanding the relation of input transformations and machine learning resistance. Cryptology ePrint Archive, Paper 2019/799 (2019). https://eprint.iacr.org/2019/799

[27] Y. Hori, T. Yoshida, T. Katashita, A. Satoh, *Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs*, in *2010 International Conference on Reconfigurable Computing and FPGAs* (2010), pp. 298–303. https://doi.org/10.1109/ReConFig.2010.24. https://dl.acm.org/doi/10.1109/ReConFig.2010.24

[28] A. Maiti, V. Gunreddy, P. Schaumont, *A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions* (Springer New York, New York, NY, 2013), pp. 245–267. https://doi.org/10.1007/978-1-4614-1362-2_11. https://doi.org/10.1007/978-1-4614-1362-2_11

[29] N.N. Anandakumar, M. Hashmi, M. Tehranipoor, FPGA-based Physical Unclonable Functions: A comprehensive overview of theory and architectures. Integration **81** (2021). https://doi.org/10.1016/j.vlsi.2021.06.001

[30] D. Lim, Extracting Secret Keys from Integrated Circuits. Master's thesis, Institute of Technology (MIT), Massachusetts, Mass, USA (2004)

[31] M.A. Alamro, K.T. Mursi, Y. Zhuang, A.O. Aseeri, M.S. Alkatheiri, Robustness and Unpredictability for Double Arbiter PUFs on Silicon Data: Performance Evaluation and Modeling Accuracy. Electronics **9**(5) (2020). https://doi.org/10.3390/electronics9050870

[32] M.S. Alkatheiri, Y. Zhuang, *Towards Fast and Accurate Machine Learning Attacks of Feed-Forward Arbiter PUFs*, in *2017 IEEE Conference on Dependable and Secure Computing* (2017), pp. 181–187. http://dx.doi.org/10.1109/DESEC.2017.8073845

[33] A.O. Aseeri, Y. Zhuang, M.S. Alkatheiri, *A Machine Learning-Based Security Vulnerability Study on XOR PUFs for Resource-Constraint Internet of Things*, in *2018 IEEE International Congress on Internet of Things (ICIOT)* (2018), pp. 49–56. http://dx.doi.org/10.1109/ICIOT.2018.00014

[34] K.T. Mursi, Y. Zhuang, M.S. Alkatheiri, A.O. Aseeri, *Extensive Examination of XOR Arbiter PUFs as Security Primitives for Resource-Constrained IoT Devices*, in *2019 17th International Conference on Privacy,*

Security and Trust (PST) (2019), pp. 1–9. http://dx.doi.org/10.1109/PST47121.2019.8949070

[35] The source code of the backtracking algorithm in [6]. https://gitlab.univ-st-etienne.fr/sesam/pll-trng-constraint-programming/tree/master. Accessed: 2024-02-25

[36] V. Fischer, M. Drutarovský, M. Simka, N. Bochard, *High Performance True Random Number Generator in Altera Stratix FPLDs* (2004), pp. 555–564. https://doi.org/10.1007/978-3-540-30117-2_57

[37] E. Noumon Allini, Characterisation, Evaluation and Use of Clock Jitter as a Source of Randomness in Data Security. Ph.D. thesis, Université de Lyon (2020). https://ujm.hal.science/tel-02952931

[38] N. Wisiol, G.T. Becker, M. Margraf, T.A.A. Soroceanu, J. Tobisch, B. Zengin, *Breaking the Lightweight Secure PUF: Understanding the Relation of Input Transformations and Machine Learning Resistance*, in *Smart Card Research and Advanced Applications*, ed. by S. Belaïd, T. Güneysu (Springer International Publishing, Cham, 2020), pp. 40–54

[39] Xilinx Zynq-7000 SoC ZC702 Evaluation Kit. https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html. Accessed: 2023-12-25

[40] Xilinx (AMD) Vivado 2019.1 Design Software for Xilinx (AMD) Adaptive SoCs and FPGAs. https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html. Accessed: 2023-11-25

[41] IEEE Computer Society. IEEE Standard VHDL Language Reference Manual. IEEE Std 1076-2008 (2008)

[42] B.W. Kernighan, D.M. Ritchie, *The C Programming Language*, 2nd edn. (Prentice Hall, Englewood Cliffs, NJ, 1988)

[43] BSI. Implementation of Test Procedure A and Test Procedure B for Application Notes and Interpretation of the Scheme (AIS) 20/31 Standard. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_testsuit_zip.zip. Accessed: 2023-11-25

[44] Eclipse Foundation, *Eclipse IDE for Java Developers - 2023-12*. URL https://www.eclipse.org/downloads/packages/. Version 2023-12 https://www.eclipse.org/downloads/packages/

[45] Python Software Foundation, *Python Language Reference, version 3.x*. {https://www.python.org/}Accessed:2023-01-12

[46] Microsoft Corporation. Visual Studio 2022 (2022). https://visualstudio.microsoft.com/ Accessed: 2023-01-12

[47] PuTTY - a free and open-source terminal emulator, serial console, and network file transfer application. https://www.putty.org/. Accessed: 2023-12-25