ON AN EFFICIENT IMPLEMENTATION OF COMBINED TRUE RANDOM NUMBER GENERATOR AND PHYSICALLY UNCLONABLE FUNCTION ON A SOC FPGA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YUNUS EMRE YILMAZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
CRYPTOGRAPHY

SEPTEMBER 2024

Approval of the thesis:

ON AN EFFICIENT IMPLEMENTATION OF COMBINED TRUE RANDOM NUMBER GENERATOR AND PHYSICALLY UNCLONABLE FUNCTION ON A SOC FPGA

submitted by **YUNUS EMRE YILMAZ** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ayşe Sevtap SELÇUK KESTEL
Dean, Graduate School of **Applied Mathematics**

Assoc. Prof. Dr. Oğuz YAYLA
Head of Department, **Cryptography**

Assoc. Prof. Dr. Oğuz YAYLA
Supervisor, **Cryptography, METU**

Examining Committee Members:

Prof. Dr. Zülfükar SAYGI
Department of Mathematics, TOBB

Assoc. Prof. Dr. Oğuz YAYLA
Department of Cryptography, METU

Assist. Prof. Dr. Talha ARIKAN
Department of Mathematics, Hacettepe University

Assist. Prof. Dr. Buket ÖZKAYA
Department of Cryptography, METU

Assist. Prof. Dr. Eda TEKİN
Department of Business Administration, Karabük University

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: YUNUS EMRE YILMAZ

Signature :

ABSTRACT

ON AN EFFICIENT IMPLEMENTATION OF COMBINED TRUE RANDOM NUMBER GENERATOR AND PHYSICALLY UNCLONABLE FUNCTION ON A SOC FPGA

YILMAZ, YUNUS EMRE
Ph.D., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Oğuz YAYLA

September 2024, 86 pages

True Random Number Generators (TRNGs) and Physically Unclonable Functions (PUFs) are two basic and useful primitives in designing cryptographic systems. TRNGs must be invariably random, while PUFs must have repetitive results and instance-specific randomness. In this work, these primitives are implemented in a System-on-Chip Field-Programmable Gate Array (SoC FPGA), or simply SoC. Phase-Locked Loops (PLLs) are essential components in both FPGAs and SoCs, widely implemented for various functions. Within these devices, PLLs offer a promising method for generating random numbers. Due to their isolated operation, broad utilization, and strong entropy generation, as validated by prior research, PLLs integrated into FPGAs or SoCs serve as highly effective foundations for PLL-based true random number generators (PLL-TRNGs). This makes PLL-TRNGs a particularly viable solution for generating secure random numbers in such architectures. The parameter selection in PLL-TRNG is a very critical process since it requires yielding both a sufficient entropy rate and an adequate output bit rate. Hence, in the first part of this thesis, a parameter selection algorithm based on the backtracking method in the literature is chosen and adapted to our selected SoC. In addition to these, a novel methodology is proposed to enhance the rate of random data bit generation of PLL-TRNG by using extra PLLs with a specific interconnection while preserving entropy characteristics. Performance metrics are rigorously evaluated against the criteria set by the German Federal Office for Information Security (BSI) AIS-20/31 Tests and compared to the works in the literature. Other than TRNGs, designing a secure PUF is another motivation for this thesis. The Arbiter PUF, recognized as the first silicon PUF, is capable of generating a substantial number of secret keys instantaneously based on the input, all while maintaining a lightweight design. This advantageous characteristic makes it particularly well-suited for device authentication in applications with constrained resources, especially for Internet of Things (IoT) devices. Despite these advantages, arbiter PUFs are vulnerable to machine learning (ML) attacks. Hence, those arbiter PUF designs are improved to achieve increased resistance against such attacks. These improvements aim to increase resilience against ML attacks while maintaining usefulness and efficiency for IoT applications. In the second part of this thesis, a machine-learning-resistant 32-bit and 64-bit component-differentially challenged XOR Arbiter PUF (CDC-XPUF) is implemented based on a design found in the literature. The 32-bit and 64-bit 7-stream CDC-7-XPUFs are evaluated using PUF metrics in the literature, namely steadiness, correctness, diffuseness, uniformity, and uniqueness. Additionally, the utilization ratios for both TRNG and PUF implementations are presented. In the last part of this thesis, PLL-TRNG with four PLLs (4-PLL-TRNG) and 64-bit 7-stream CDC-XPUF (CDC-7-XPUF) is combined so that they can work together. The random numbers generated by 4-PLL-TRNG are utilized by CDC-7-XPUF to generate other challenges from the main challenge. All the tests applied to TRNG and PUF are also applied to this combined design, and it is shown that that combined design is a suitable candidate to use in an IoT system. Consequently, a total of three different configurations, two of which are discrete implementations of PLL-TRNG and CDC-XPUF and one of which is a combined implementation of these PLL-TRNG and CDC-XPUF, are implemented. All of the tests are implemented using the ZC702 Rev1.1 Evaluation Board, which features the Xilinx Zynq 7020 SoC, and utilizes a configuration involving three boards for experimental validation.

Keywords: True Random Number Generator (TRNG), Physically Unclonable Function (PUF), Field-Programmable Gate Array (FPGA), System-on-Chip (SoC), Phase-Locked Loop (PLL), Component-Differentially Challenged XOR Arbiter PUF (CDC-XPUF)

ÖZ

BİR SOC FPGA ÜZERİNDE KOMBİNE GERÇEK RASTGELE SAYI ÜRETECİ VE FİZİKSEL OLARAK KLONLANAMAYAN FONKSİYONUN VERİMLİ BİR UYGULAMASI ÜZERİNE

YILMAZ, YUNUS EMRE
Doktora, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Oğuz YAYLA

Eylül 2024, 86 sayfa

Gerçek Rastgele Sayı Üreteçleri (TRNG'ler) ve Fiziksel Olarak Klonlanamayan Fonksiyonlar (PUF'lar) kriptografik sistemlerin tasarlanmasında kullanılan iki temel ve kullanışlı ilkel araçtır. TRNG'ler değişmez şekilde rastgele olmalıdır, PUF'lar ise tekrarlayan sonuçlara ve örneğe özgü rastgeleliğe sahip olmalıdır. Bu çalışmada, bu ilkeller bir Çip Üzerinde Sistem Alan Programlanabilir Kapı Dizisinde (SoC FPGA) veya kısaca SoC'de uygulanmıştır. Faz Kilitli Döngüler (PLL'ler) hem FPGA'lerde hem de SoC'lerde çeşitli işlevler için yaygın olarak uygulanan temel bileşenlerdir. Bu cihazlarda PLL'ler rastgele sayılar üretmek için umut verici bir yöntem sunar. Önceki araştırmalarla doğrulandığı üzere, izole çalışmaları, geniş kullanımları ve güçlü entropi üretimleri nedeniyle, FPGA'lara veya SoC'lere entegre edilen PLL'ler, PLL tabanlı gerçek rastgele sayı üreteçleri (PLL-TRNG'ler) için oldukça etkili temeller olarak hizmet eder. Bu da PLL-TRNG'leri bu tür mimarilerde güvenli rastgele sayılar üretmek için özellikle uygun bir çözüm hâline getirmektedir. PLL-TRNG'de parametre seçimi, hem yeterli bir entropi oranı hem de yeterli bir çıkış bit oranı sağlamayı gerektirdiğinden çok kritik bir süreçtir. Bu nedenle, bu çalışmada literatürdeki geri izleme (backtracking) yöntemine dayalı bir parametre seçim algoritması seçilmiş ve seçilen SoC'ye uyarlanmıştır. Bunlara ek olarak, entropi özelliklerini korurken belirli bir ara bağlantıya sahip ekstra PLL'ler kullanarak PLL-TRNG'nin rastgele veri biti üretme oranını artırmak için yeni bir metodoloji önerilmiştir. Performans ölçütleri, Alman Federal Bilgi Güvenliği Ofisi (BSI) AIS-20/31 Testleri tarafından belirlenen kriterlere göre titizlikle değerlendirilmiş ve literatürdeki çalışmalarla karşılaştırılmıştır. İlk silikon PUF olarak kabul edilen hakem (Arbiter) PUF, hafif tasarımını korurken, girdiye bağlı olarak anında önemli sayıda gizli anahtar üretebilmektedir. Bu avantajlı özellik, özellikle Nesnelerin İnterneti (IoT) cihazları gibi kısıtlı kaynaklara sahip uygulamalarda cihaz kimlik doğrulaması için çok uygundur. Bu avantajlara rağmen, hakem PUF'lar makine öğrenimi saldırılarına karşı savunmasızdır. Bu nedenle, bu tür saldırılara karşı daha fazla direnç elde etmek için bu hakem PUF tasarımları geliştirilmiştir. Bu iyileştirmeler, IoT uygulamaları için kullanışlılığı ve verimliliği korurken makine öğrenmesi saldırılarına karşı dayanıklılığı artırmayı amaçlamaktadır. Bu çalışmada, literatürde bulunan bir tasarıma dayalı olarak makine öğrenmesine dirençli 32 bit ve 64 bit bileşen farklılaştırmalı XOR Hakem PUF (CDC-XPUF) gerçeklenmiştir. 32-bit ve 64-bit 7 akışlı CDC-7-XPUF'ler, literatürdeki PUF ölçütleri olan kararlılık, doğruluk, dağınıklık, tekdüzelik ve benzersizlik kullanılarak değerlendirilmiştir. Ek olarak, hem TRNG hem de PUF uygulamaları için kullanım oranları sunulmuştur. Bu çalışmanın son bölümünde, dört PLL'li PLL-TRNG (4-PLL-TRNG) ve 64-bit 7 akışlı CDC-XPUF (CDC-7-XPUF) birlikte çalışabilecek şekilde birleştirilmiştir. 4-PLL-TRNG tarafından üretilen rastgele sayılar CDC-7-XPUF tarafından ana PUF girdisinden (challenge) diğer PUF girdilerini üretmek için kullanılır. TRNG ve PUF'a uygulanan tüm testler bu birleşik tasarıma da uygulanmış ve bu birleşik tasarımın bir IoT sisteminde kullanılmak için uygun bir aday olduğu gösterilmiştir. Sonuç olarak, ikisi PLL-TRNG ve CDC-XPUF'un ayrık uygulamaları ve biri de bu PLL-TRNG ve CDC-XPUF'un birleşik uygulaması olmak üzere toplam üç farklı konfigürasyon gerçeklenmiştir. Tüm testler, Xilinx Zynq 7020 SoC içeren ZC702 Rev1.1 Değerlendirme Kartı kullanılarak uygulanmış ve deneysel doğrulama için üç kart içeren bir yapılandırma kullanılmıştır.

Anahtar Kelimeler: Gerçek Rastgele Sayı Üreteci (TRNG), Fiziksel Klonlanamayan Fonksiyonlar (PUF), Sahada Programlanabilir Kapı Dizileri (FPGA), Çip Üzerinde Sistem (SoC), Faz Kilitli Döngü (PLL), Bileşen-Farklı Zorlanmış XOR Hakem Fiziksel Klonlanamayan Fonksiyonu (CDC-XPUF)

To the love of my life
To my mother
To my brother

ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisor Assoc. Prof. Dr. Oğuz Yayla, for his patient guidance, enthusiastic support, and invaluable advice throughout the development and preparation of this thesis. His generous commitment of time and willingness to share his expertise have significantly shaped my doctoral journey. I am deeply appreciative of the experience, insights, and thorough reviews he provided during the preparation of both this thesis and the related articles.

I would like to acknowledge Aselsan Inc. for its support during the preparation of this thesis, as well as for providing three Xilinx ZC702 Evaluation Boards, which were instrumental in the implementation of the algorithms presented in this thesis. I would like to thank my department manager, Cüneyt Seven, and my team leader, Salih Alper Engin, for their assistance in securing this support and for their contributions to the preparation of this thesis.

I would like to thank Prof. Dr. Zülfikar Saygı, Assist. Prof. Buket Özkaya, Assist. Prof. Dr. Eda Tekin, and Assoc. Prof. Talha Arıkan for being a committee member of my defense.

I would like to thank all the academic and administrative staff of the Institute of Applied Mathematics for their valuable assistance throughout my doctoral study. I am particularly grateful to Serkan Demiröz, Nejla Erdoğdu, and Ebru Gündoğdu for their administrative guidance.

I would like to thank Dr. İzzet Kağan Erünsal, who has consistently supported me throughout my doctoral studies, both as a valued friend and a dedicated academic, despite the distance. My gratitude also extends to Ahmet Ayaşlı for making the doctoral journey more bearable with his humor and for standing by me during difficult times. I am sincerely grateful to Mete Eray for generously sharing his insights on both life and technical matters and for his continuous encouragement to complete my doctorate. I am also thankful to Görkem Uyar for his unwavering support throughout my doctoral journey, both during our time at Aselsan and after his departure. Likewise, I would like to thank my colleague Canberk Tatlı for his friendship and support during this process. Finally, I would like to acknowledge Hakan Erünsal, Sencer Ergin, MD, and Aziz Çelebi for their steadfast support and companionship throughout this journey.

I am profoundly grateful to my mother, Kıvanç Yılmaz, and my brother, Berat Yılmaz. Their invaluable support has made this doctoral journey significantly easier, and I deeply appreciate their help and patience throughout this entire process.

Last but not least, my deepest gratitude goes to the love of my life, Fatma Demirci, whose support and patience throughout this challenging doctoral journey have been invaluable. Her presence provided me with peace whenever I felt overwhelmed by the stress of the process. I am forever grateful for her unwavering patience and encouragement during this time. I feel incredibly fortunate to have her in my life, and I look forward to the future we will share with both excitement and curiosity.

TABLE OF CONTENTS

CHAPTERS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

APUF	Arbiter Physically Unclonable Function
ASIC	Application Specific Integrated Circuit
AXI Bus	Advanced eXtensible Interface Bus
BSI	Bundesamt für Sicherheit in der Informationstechnik (Federal Office for Security in Information Technology)
CDC-XPUF	Component-Differentially Challenged XOR Arbiter Physically (or Physical) Unclonable Function
COSO-TRNG	Coherent Sampling-based True Random Number Generator
CP	Charge Pump
CRP	Challenge-Response Pair
DFF	Data or Delay Flip-Flop
DRNG	Deterministic Random Number Generator
DSP	Digital Signal Processor
ERO-TRNG	Elementary Ring Oscillator-based True Random Number Generator
FIT	Failure In Time
FPGA	Field-Programmable Gate Array
IC	Integrated Circuit
IoT	Internet of Things
IP	Intellectual Property
ML	Machine Learning
MTBF	Mean Time Between Failures
MUX	Multiplexer
NIST	the National Institute of Standards and Technology
NPTRNG	Non-Physical True Random Number Generator
PC	Personal Computer
PFD	Phase Frequency Detector
PL	Programmable Logic
PLL	Phase-Locked Loop

PLL-TRNG	PLL-based True Random Number Generator
P_{VCOd}	Post-Voltage-Controlled Oscillator Divider
PS	Processing System
PUF	Physically (or Physical) Unclonable Function
PRNG	Pseudo-Random Number Generator
PTRNG	Physical True Random Number Generator
RFID	Radio-Frequency Identifier
RNG	Random Number Generator
RoT	Root of Trust
SR-Latch	Set/Reset Latch
SoC	System-on-Chip
TERO	Transient Effect Ring Oscillator
TRNG	True Random Number Generator
VCO	Voltage-Controlled Oscillator
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
XOR-PUF	XOR Arbiter Physically (or Physical) Unclonable Function

CHAPTER 1

INTRODUCTION

Random numbers are essential components in cryptography, used for generating confidential keys, padding data, initialization vectors, and nonces in challenge-response protocols. Additionally, random numbers are employed to generate random masks, which are critical in preventing side-channel attacks. Random number generators (RNGs) serve as cryptographic primitives designed to produce sequences of bits or symbols (e.g., bit groups or vectors) that exhibit no discernible patterns. For these generators, independence and uniform distribution are crucial properties. RNGs are typically classified into two main types: true random number generators (TRNGs) and pseudo-random number generators (PRNGs), also referred to as deterministic random number generators (DRNGs). Each type offers distinct characteristics, benefits, and limitations. Beyond RNGs, a new class of hardware primitives has emerged, sharing some properties with TRNGs, known as physically (or physical) unclonable functions (PUFs). PUFs are utilized for hardware authentication in challenge-response protocols and for generating device-specific confidential keys.

The circuitry of both PUFs and TRNGs necessitates a minimal systematic mismatch to ensure the absence of bias in their respective outputs. Although the responses of both PUFs and TRNGs are inherently unpredictable, a key distinction lies in the behavior of their outputs: for a given challenge, the response of a PUF remains consistent across multiple executions, whereas the output of a TRNG exhibits random variation with each execution, as indicated in [49].

According to Kerckhoff's Principle, the security of any cryptographic system fundamentally depends on safeguarding the keys used in the implemented algorithm. In high-end information security systems, particularly when operating in uncontrolled environments, cryptographic keys and random masks must be generated within the system and should never be exposed or transmitted in an unencrypted form. As a result, when the security system is integrated into a single chip (cryptographic system-on-chip), key generation must occur within the same chip or logic device. However, logic devices are designed to execute deterministic logic operations, not to generate randomness based on analog physical phenomena. Consequently, implementing RNGs and PUFs on logic devices such as field-programmable gate arrays (FPGAs) and digital application-specific integrated circuits (ASICs) presents a significant challenge, as noted in [20].

Recently popular System-on-Chip (SoC) Field-Programmable Gate Arrays (FPGAs) or SoCs are semiconductor devices that integrate programmable logic with hard processor cores. They offer higher integration, lower power, smaller board sizes, and higher bandwidth communication between the processor and FPGA. In this thesis work, SoC is preferred instead of FPGA since SoC has hard processor power, and this power can be used for further applications on a single chip without the need for any external connection.

Both in FPGA and SoCs, Phase-Locked Loops (PLLs) are required and placed. Briefly, those are feedback control systems that automatically adjust the phase of a locally generated signal to match the phase of an input signal. These PLL structures can be used to design TRNGs, as shown in [24].

Phase-locked loops (PLLs), operating in the analog domain, are well-suited for cryptographic TRNG designs due to their inherent analog source of unpredictable randomness, as stated in [24]. PLLs are commonly employed to enhance clock distribution performance and to enable on-chip clock-frequency synthesis. Additionally, these units benefit from dedicated power sources within FPGAs, which helps to isolate them from the rest of the device, minimizing interference. However, the number of available PLLs in FPGAs is limited, which constrains the extent to which TRNG implementations can leverage multiple PLLs.

The primary challenge in PLL-TRNG design is the selection of optimal PLL settings from a vast configuration space. The chosen parameters must yield both a sufficient entropy rate and an adequate output bit rate. This study adopts the parameter determination process outlined in [16].

While it is advantageous to implement PLL-TRNG considering its high entropy and isolated locations of PLL, one of the main drawbacks of the PLL-TRNG is its relatively low random data output speed. A new PLL-TRNG method using four PLLs is proposed to overcome this disadvantage. This research culminates in the implementation of a four-PLL True Random Number Generator (4-PLL TRNG) on a SoC. To elucidate the design progression, a referenced configuration utilizing two PLLs and two intermediate configurations utilizing three PLLs are developed. Subsequently, four distinct PLL-TRNG configurations are implemented.

NIST SP 800-90B [57] and AIS-20/31 [13] are cryptographic standards focused on RNGs, but with different regional origins and scopes. NIST SP 800-90B emphasizes the evaluation and testing of entropy sources for RNGs, focusing on entropy estimation, health tests, and ensuring the unpredictability of random outputs. AIS-20/31 provides a more comprehensive approach by defining standards for both deterministic (DRNG) and true (TRNG) random number generators, classifying them into different security levels. In this work, by considering the properties and the comprehensive approach of the AIS-20/31, the generated random numbers are evaluated using the AIS-20/31 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.

The Arbiter PUF, the first silicon-based PUF, generates numerous secret keys efficiently from input data while maintaining a lightweight design. This makes it well-suited for device authentication in environments with limited resources, such as IoT applications. However, its vulnerability to machine learning attacks highlights the need for enhanced design solutions to improve security.

Consequently, to improve resistance to machine learning (ML) attacks, arbiter PUF designs have been enhanced. In this study, an ML attack-resistant component-differentially challenged XOR arbiter PUF (CDC-XPUF) is implemented, following the reference designs from [43] and [37]. Research in [37] demonstrates that designs with 64-bit or longer challenges and at least 7-stream PUFs are resistant to the most advanced ML attack techniques. Consequently, this work implements a referenced 32-bit CDC-7-XPUF, followed by an improved 64-bit version for enhanced ML attack resilience. The performance results for both the 32-bit and 64-bit CDC-7-XPUFs are presented and compared to the reference design.

In addition to the tests applied to TRNG and PUF designs, the utilization rates of both TRNG and PUF designs are evaluated, showing that they are well-suited for IoT systems by providing sufficient space for other software or firmware.

Considering the intended use of TRNGs and PUFs, these hardware primitives have become a crucial component of modern IoT systems. A root-of-trust for an embedded device can be implemented by combining TRNG and PUF together. Therefore, a design that combines these two primitives would not only be beneficial but also efficient in terms of resource consumption on SoC or FPGA platforms. In this study, after separately implementing both hardware primitives on an SoC, we propose a combined structure where both can operate together. In this structure, the random numbers generated by the TRNG are transferred to the PUF section to generate six of the seven challenges for the CDC-7-XPUF, excluding the main challenge. All tests applied to the TRNG and PUF were also applied to this combined structure, and its resource usage was subsequently calculated. As a result, the combined design of the 4-PLL-TRNG and CDC-7-XPUF successfully passes all tests and demonstrates low resource usage. This promising result indicates that the design can be implemented on the same SoC alongside other firmware and software intended for IoT applications, offering a secure solution as evidenced by the performance metrics.

All of the designs were implemented and tested in a test setup utilizing the ZC702 Rev1.1 Evaluation Board [7], equipped with the Xilinx Zynq 7020 SoC and a configuration of three such boards for experimental validation.

1.1 Motivation

During this study, we are primarily driven by two motivations:

- All network-based embedded systems, such as those used in the Internet of Things (IoT), require hardware primitives like TRNGs and PUFs to serve as the root of trust (RoT). Therefore, the efficient and reliable implementation of these primitives, both individually and in combination, will address the need for a robust RoT in such systems. The goal is to meet the RoT requirements of IoT systems while minimizing the hardware load these primitives impose, thus leaving adequate design space for other firmware and software.
- SoCs have become increasingly common in IoT and similar systems. A key motivation for this work has been the implementation of these hardware primitives, both individually and in combination, on SoC platforms, ensuring their adaptability across different SoCs and facilitating their use in various systems.

1.2 Contributions

Our work presents three primary contributions: We design a new, fast, and adaptive structure of PLL-TRNG, which contains 4 PLLs with a specific interconnection and named 4-PLL-TRNG, implemented on the Xilinx Zynq 7020 SoC, which is compatible with new FPGAs or SoCs and can increase the bit rate without compromising cryptographic properties, and we evaluate its performance with respect to AIS-20/31 tests, comparing the results with previous works. We implement the 64-bit version of the CDC-XPUF (64-bit CDC-7-XPUF), based on the Arbiter PUF, with seven streams. It is reported in [37] that this version is resistant to machine learning (ML) attacks. Additionally, the implemented PUF is evaluated based on the metrics of steadiness, correctness, diffuseness, uniformity, and uniqueness, as well as its resource utilization on the SoC. It is demonstrated that the implemented PUF is appropriate regarding all these metrics and resource utilization.

By combining the 4-PLL-TRNG (with the max. R configuration) 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 combined structure, which results in the new structure having good properties.

1.3 Thesis Organization

The thesis is organized as follows:

- In Chapter 2, preliminary information about PLL-TRNG, CDC-7-XPUF, the combined design of both, applied tests to TRNGs and PUFs, and Zynq-7020 SoCs are presented.
- In Chapter 3, the details of PLL-TRNG implementations are presented. After that, the results and comparison with previous works are demonstrated.
- In Chapter 4, we describe the implementations of both 32-bit and 64-bit CDC-7-XPUF in detail and subsequently present the results and comparisons with the referenced study.
- In Chapter 5, we explain the details of the implementation of the combined design of 4-PLL-TRNG and 64-bit CDC-7-XPUF. The results and the comparisons of them with respect to separate implementations are explained in this chapter.
- In Chapter 6, the conclusion part is presented, and the future works of the thesis are explained.

CHAPTER 2

PRELIMINARIES

In this chapter, preliminary information about the combined PUF-TRNG implementation is presented. Hence, the following sections about PLL-TRNG, PUF, PUF-TRNG, evaluation metrics of TRNGs and PUFs, and Xilinx Zynq SoC FPGA contain required explanations in order to understand the combined PUF-TRNG implementation in this thesis.

2.1 Phase-Locked Loop-based True Random Number Generator (PLL-TRNG)

2.1.1 True Random Number Generators (TRNGs)

Cryptography is a fundamental component of modern information systems, and within cryptographic frameworks, random number generators (RNGs) are essential. RNGs are utilized not only for generating cryptographic keys but also for producing nonces, initialization vectors, and random masks, which are critical in defending against side-channel attacks.

Despite the wide range of applications for random numbers in cryptographic systems, they must meet two primary criteria. First, they must demonstrate strong statistical properties, particularly a uniform probability distribution, to ensure that all possible values have an equal likelihood, thus mitigating vulnerabilities such as frequency attacks. Second, unpredictability is crucial, especially for secret parameters like keys, to prevent adversaries from predicting future or past values based on captured data.

Due to the broad range of RNG applications in cryptography, various RNG principles exist to meet different needs. Two fundamental RNG types are deterministic/pseudo-random number generators (DRNG/PRNG) and true random number generators (TRNGs).

DRNGs generate sequences that appear random in the short term but are periodic in the long term. They use mathematical algorithms and initialization values called seeds to produce less predictable output. On the other hand, TRNGs are not algorithmic; they extract randomness from non-algorithmic phenomena such as temperature fluctuations or radioactive decay, producing real random data.

TRNGs can be physical (PTRNG), utilizing physical noise on the electron level, or non-physical (NPTRNG), relying on non-physical randomness sources like user interactions. While deterministic RNGs ensure unpredictability computationally, true RNGs guarantee unpredictability through random physical phenomena characterized by the entropy rate.

Cryptographic systems often utilize hybrid RNGs, combining the strengths of both TRNGs and DRNGs. Hybrid true RNGs merge a TRNG with cryptographic post-processing to ensure forward and backward secrecy and perfect statistical properties. Hybrid deterministic RNGs use a TRNG to periodically generate seeds for a DRNG, reducing predictability.

To maintain the security of confidential keys, it's essential to generate them within the cryptographic system. As contemporary cryptographic systems are predominantly implemented in logic devices like Field Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs), the focus of research is often on implementing RNGs in these hardware-supported digital logic synthesis devices. Thus, in the following subsection, the randomness sources for these digital logic structures are presented. The details of this section and the following sections can be found in [46].

Those who wish to gain more in-depth knowledge about random numbers can refer to [34] and [35] for further study.

2.1.1.1 Randomness Sources in Logic Devices

TRNGs can rely on either physical or non-physical noise sources; however, in logic devices, the availability of physical noise sources is constrained due to the design focus on maintaining a consistent and well-defined state. To generate random numbers, an inherently uncontrollable random phenomenon is necessary. The primary physical phenomena employed for random number generation in logic devices include **clock jitter** (the deviation of the clock edge from its ideal timing), **metastability** (a circuit's ability to remain in an indeterminate state for an unpredictable duration), **chaos** (the unpredictable behavior of deterministic systems that are highly sensitive to initial conditions), and **analog signals** (such as diode shot noise and thermal noise).

This chapter will focus on clock jitter and metastability. The generation of random numbers using analog signals is beyond the scope of this thesis, as they are difficult to utilize in logic devices. Incorporating an analog signal into a digital system requires an analog-to-digital converter, and most digital logic devices under consideration, such as FPGAs and ASICs, lack such an analog interface.

Chaotic behavior characterizes seemingly deterministic systems, displaying extreme sensitivity to initial conditions, resulting in vastly different outcomes with even the slightest initial state change. This behavior has been investigated for TRNG implementation, as the divergence in results from different initial states disrupts dependencies in the output sequence. Systems exhibiting chaotic behavior typically require analog components like A/D converters or switched capacitors. However, this thesis focuses on randomness sources that do not necessitate such components, as they are generally unavailable in logic devices.

In this thesis, the clock jitter is chosen as the source of the randomness since it is available in every digital logic device and has an unstable nature in some instances, which exhibits a reliable source for randomness.

Clock Jitter

A desired clock signal in digital logic devices should ideally be a square wave with a 50% duty cycle and a consistent period. However, due to the influence of various electronic interferences, the clock signal is never perfectly stable, and its edges deviate from their intended positions. This phenomenon, known as clock jitter, manifests as fluctuations in the phase of the clock signal. In the time domain, these fluctuations are observable as jitter, while in the frequency domain, they manifest as phase noise [19], [46].

Clock jitter is generally undesirable in logic devices but is often inevitable. Extensive research has been conducted to understand and characterize jitter, particularly its negative impact on high-frequency communications and high-speed systems. In analog systems, the jitter is best analyzed in the frequency domain to study its phase and amplitude components separately. Conversely, in digital systems, temporal properties of jitter take precedence, and thus it is characterized in the time domain.

In a digital system, clock jitter refers to the deviation of the actual clock edge from the ideal clock edge, as described by Equation 2.1, where $t(n)$ denotes the time of the n-th period of a clock signal, and T represents the clock signal's period. Due to jitter, real clock signals do not always arrive at precise integer multiples of their period, resulting in variations from the ideal timing.

$$t(n) = n \cdot T \tag{2.1}$$

Various physical phenomena, such as thermal noise, power supply noise, and ambient electromagnetic noise, contribute to the occurrence of jitter, as illustrated in Figure 2.1.

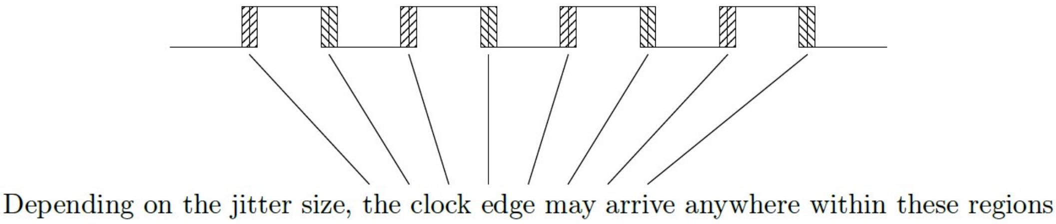

Figure 2.1: Clock jitter [46]

Figure 2.2 demonstrates a primary cause of jitter in digital circuits. These circuits use a reference level, typically located in the middle of the operating voltage range, to detect clock edges. While the reference level should ideally remain stable, it fluctuates in reality due to different noises. When the reference level shifts, it leads to the earlier or later detection of the clock edge than intended, resulting in temporal shifts observed as clock jitter.

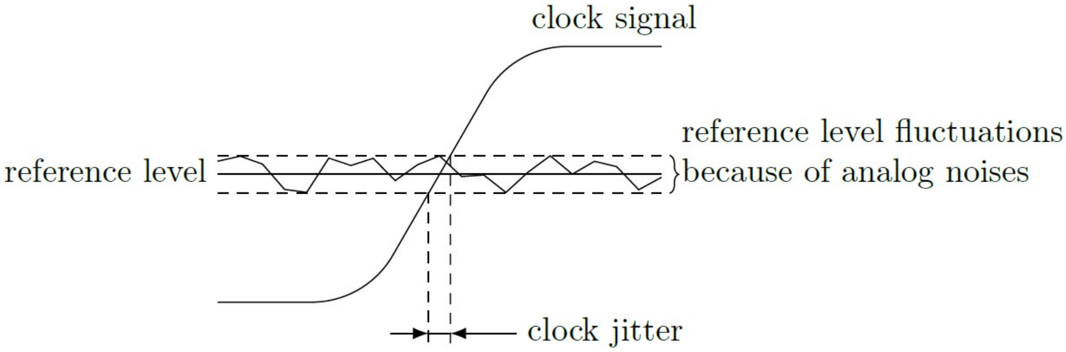

Figure 2.2: Reference level fluctuations originating from analog noises causing clock jitter in digital circuits [46]

Subsequent sections will elaborate on various jitter measurements observed in digital circuits and their interrelations.

Phase Jitter

Phase jitter is defined as the difference between the time of the n-th actual clock edge, denoted as $t_r(n)$, and the time (or phase) of the n-th ideal clock edge. This relationship is expressed in Equation 2.2.

$$\delta_\varphi(n) = t_r(n) - n \cdot T_{ref} \tag{2.2}$$

Figure 2.3 depicts the jitter phenomenon for the case where $n = 3$. To enhance clarity, we specifically showcase the phase jitter pertaining to the rising edges. However, it is important to acknowledge that the phase jitter impacts each edge of the clock, not just the rising edges.

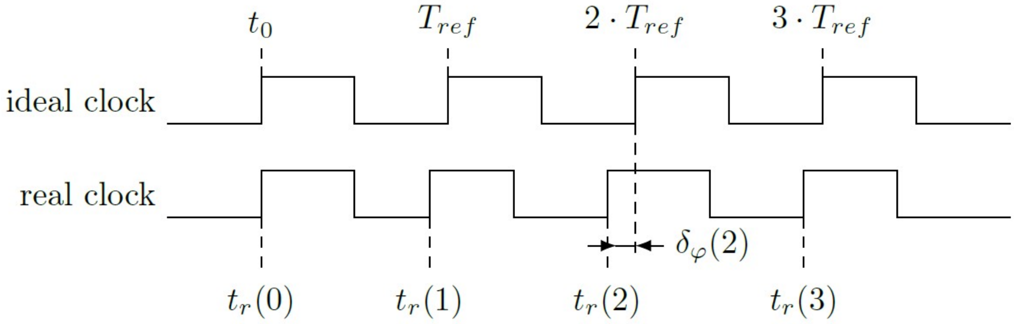

Figure 2.3: Illustration of the phase jitter of the second rising edge of the clock signal [46]

In Figure 2.3, it is evident that the presented phase jitter $\delta_\varphi(2)$ is influenced not solely by the phase discrepancy of $t_r(2)$ but also incorporates contributions from the variation in $t_r(1)$. This phenomenon is termed jitter accumulation, leading to an increase in the observed phase jitter with larger values of n.

Period Jitter

Period jitter is defined as the difference between the actual clock period and the ideal clock period. Additionally, as outlined in Equation 2.3, it corresponds to the first-order difference of the phase jitter.

$$\begin{aligned} \delta_T(n) &= [t_r(n) - t_r(n-1)] - T_{ref} \\ \delta_T(n) &= \delta_\varphi(n) - \delta_\varphi(n-1) \end{aligned} \tag{2.3}$$

Figure 2.4 illustrates period jitter. It is observable that actual periods exhibit variations over time, in contrast to ideal periods that remain constant.

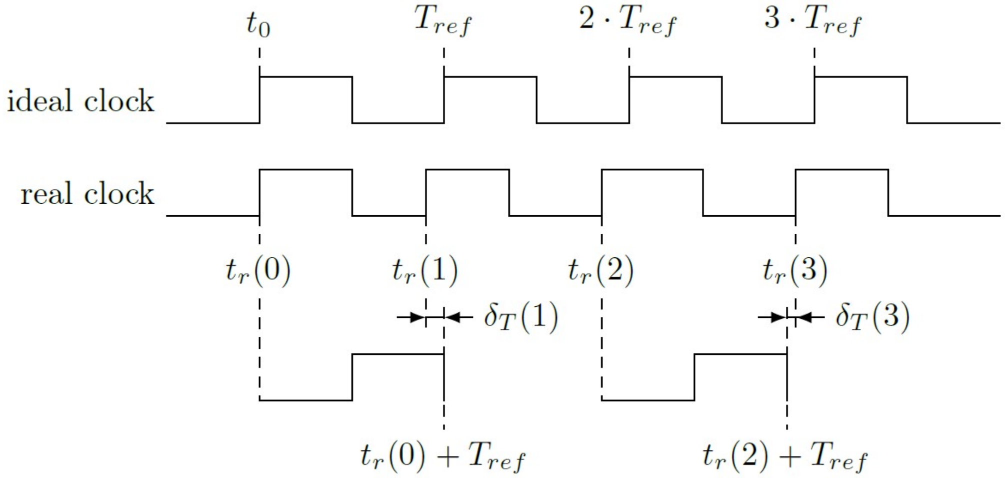

Figure 2.4: Illustration of the period jitter of a real clock signal compared to the ideal clock [46]

Cycle to Cycle Jitter

Cycle-to-cycle jitter is defined as the difference between two consecutive actual clock periods, as expressed in Equation 2.4

$$\begin{aligned} \delta_c &= T_r(n) - T_r(n-1) \\ &= [t_r(n) - t_r(n-1)] - [t_r(n-1) - t_r(n-2)] \\ \delta_c &= \delta_T(n) - \delta_T(n-1) \end{aligned} \tag{2.4}$$

Figure 2.5 portrays this jitter phenomenon.

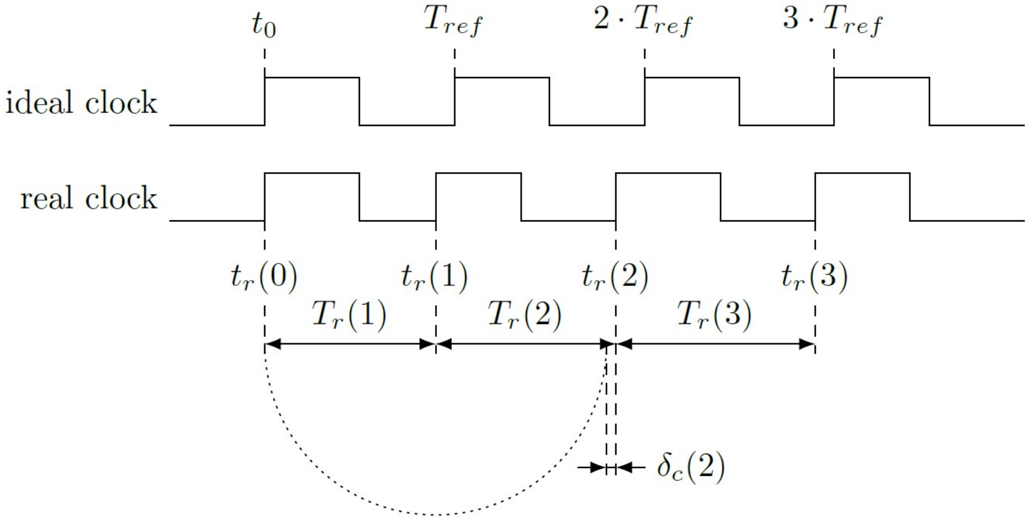

Figure 2.5: Illustration of the cycle to cycle jitter [46]

These various jitter measurements are interconnected: period jitter is essentially the first-order difference of phase jitter, and cycle-to-cycle jitter is the first-order difference of period jitter. Consequently, it is generally adequate to measure just one of these aspects and subsequently calculate any other as necessary.

Jitter Components

Jitter comprises various components stemming from diverse phenomena, categorized as either random or deterministic. Random components, like those arising from thermal or $1/f$ noise, are unpredictable and adhere to some probabilistic law. Deterministic components, contingent on the implementation, rely on specific factors such as processed data and power supplies. Deterministic components lack a probabilistic nature, making their characterization generally impractical. Figure 2.6 visually represents both deterministic and random jitter components and their cumulative contribution to the overall jitter observed in logic devices.

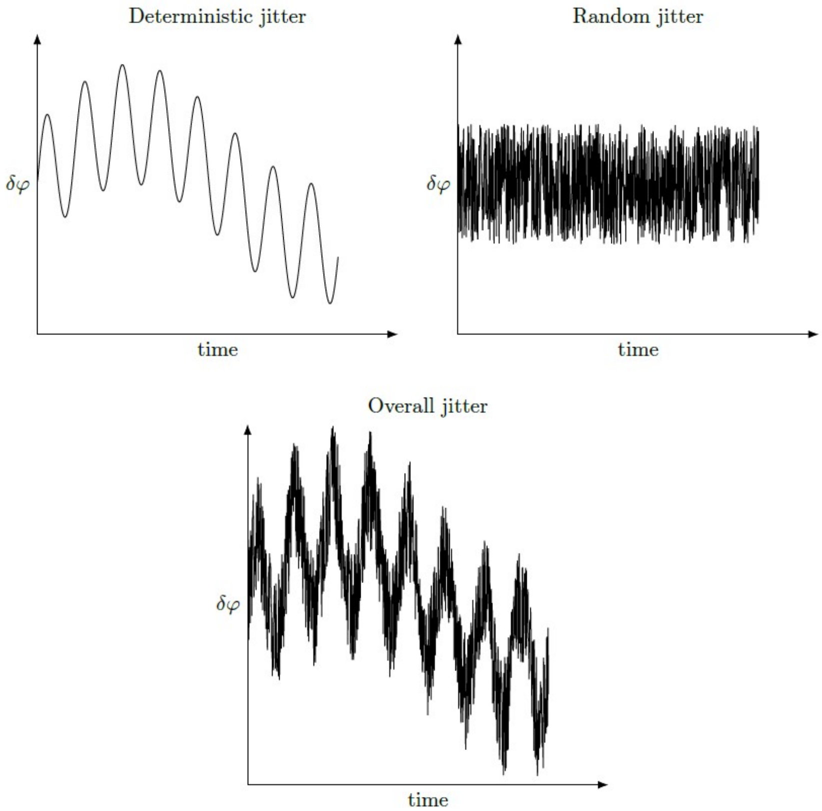

Figure 2.6: Overview of deterministic and random jitter components [46]

Both random and deterministic jitter can originate from either local or global sources. Local sources primarily affect specific regions within the electronic system, often near high-frequency or high-power components such as oscillators and amplifiers. In contrast, global sources include ambient noise and disturbances from power supplies, which impact the entire system.

In the context of TRNGs, deterministic jitter components are undesirable, as they do not produce true randomness. Genuine randomness arises exclusively from jitter caused by random noise. However, before random numbers can be generated from jitter, it is crucial to analyze its statistical properties.

Statistically, noise can be classified as independent or dependent. Independent noise is typically non-manipulable and relatively straightforward to characterize. Consequently, many TRNG designs utilize the sum of independent noise sources, commonly referred to as Gaussian noise, as a randomness source. A key challenge in designing a TRNG based on Gaussian noise is estimating the contribution of only non-correlated (Gaussian) noise to the random numbers while excluding the influence of dependent noise.

Dependent noises include deterministic ones, which are unsuitable for generating random numbers. Additionally, there are non-deterministic autocorrelated noises, such as $1/f$ noise, also known as flicker noise. Flicker noise, a well-known semiconductor phenomenon since the 1950s and 1960s, has gained interest in TRNG applications. Despite being studied extensively in [29], [32], [39], and [41] its physical cause and characterization remain challenging. In TRNG design, efforts are made to exclude the contribution of flicker noise to entropy rate estimation and rely solely on uncorrelated thermal noise.

From a statistical standpoint, considering only uncorrelated random noises, the time of arrival of the nth clock edge is a random variable X_{t_n}. Each variable's probability distribution function has its mean value at $n * T_{ref}$. The variance of these functions provides insight into the extent of clock edge fluctuations. Observing a clock signal reveals a specific realization of each variable X_{t_n}, as demonstrated in previous sections.

Metastability

Metastability refers to a system's ability to endure an unlawful state for an indefinite duration. To illustrate, consider a coin flip, as depicted in Figure 2.7. Ideally, when we flip a coin, we expect it to land on one of its two faces, constituting the legal states. However, if the coin lands on its side, the outcome becomes uncertain, leading to an indecisive and, hence, illegal state known as metastability. When a coin lands on either face, it attains a stable state. To remain in a metastable state, a coin must maintain perfect equilibrium, as even the slightest force applied to it will prompt it to fall onto one of its faces.

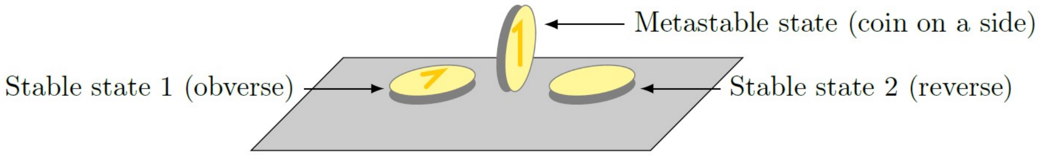

Figure 2.7: Metastability of a coin flip [46]

Metastability in FPGAs

In digital devices such as FPGAs, all registers must meet specific signal timing requirements to correctly capture input data and produce output signals. For proper functionality, a register's input must remain stable for a minimum period before the clock edge (referred to as setup time, t_{SU}) and for a minimum period after the clock edge (referred to as hold time, t_H). The register's output becomes available after a defined clock-to-output delay (t_{CO}). If a transition in the data signal occurs in violation of the register's t_{SU} or t_H requirements, the register may enter a metastable state. In this state, the output oscillates between high and low values for a period, delaying the transition to a stable state beyond the expected t_{CO}.

In synchronous systems, adherence to register timing requirements prevents metastability. However, metastability issues arise when transferring a signal between unrelated or asynchronous clock domains, as the designer cannot guarantee meeting t_{SU} and t_H requirements due to variable arrival times relative to the destination clock. Not every violation of a register's t_{SU} or t_H results in metastable output; the likelihood and the time needed to return to a stable state depend on the device's manufacturing process and operating conditions.

A useful analogy for visualizing a register sampling a data signal at a clock edge is that of a ball dropped onto a hill. The stable states are represented by the sides of the hill, corresponding to the signal's old and new data values after a transition, while the peak of the hill represents a metastable state. If the ball is dropped precisely at the top, it may remain balanced indefinitely; however, in practice, it will eventually shift slightly and roll to one side. The further the ball is from the peak, the more quickly it reaches a stable state. This concept is illustrated in Figure 2.8.

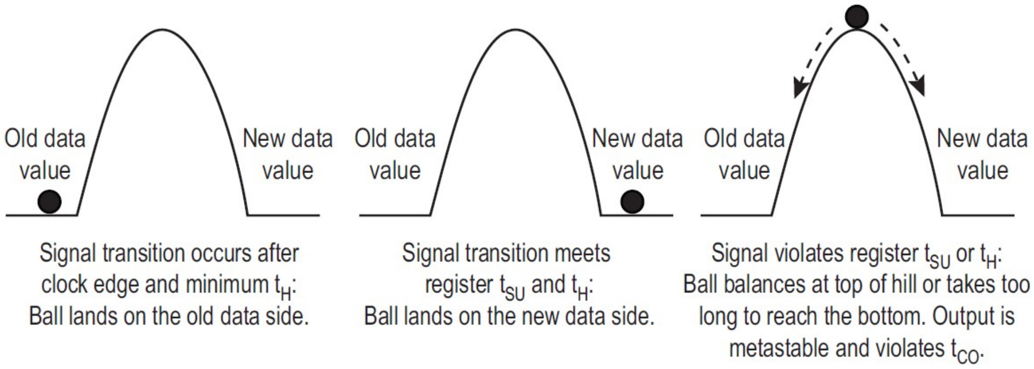

Figure 2.8: Metastability Illustrated as a Ball Dropped on a Hill [3]

When a data signal transitions after the clock edge and the minimum hold time (t_H), it is comparable to dropping the ball on the *old data value* side, preserving the original output value for that clock cycle. Conversely, if the data input transitions before the clock edge and satisfies the minimum t_H, it is akin to dropping the ball on the *new data value* side, allowing the output to promptly stabilize at the new value within the specified clock-to-output delay (t_{CO}). However, if the data input violates either the setup time (t_{SU}) or the hold time (t_H), it resembles dropping the ball near the top of the hill. In such cases, the ball takes longer to reach a stable state, causing the output delay to exceed the expected t_{CO}. Figure 2.9 demonstrates this metastability, where a violation of t_{SU} causes the output to oscillate between high and low states, delaying the transition to a stable value [3].

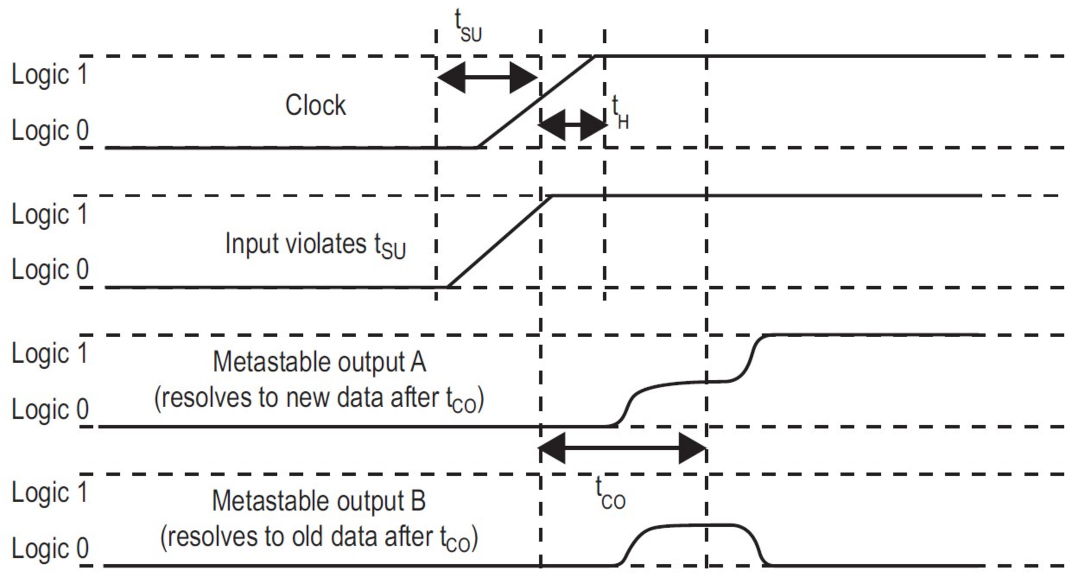

Figure 2.9: Examples of Metastable Output Signals [3]

The outcome of whether the register settles in a new or previous state is determined by random factors, resulting in a random transition. However, generating random numbers in this manner faces a significant challenge—the precise synchronization of two signals arriving at the register simultaneously. This difficulty arises due to substantial efforts by device manufacturers to minimize setup and hold times, preventing registers from entering metastable states.

Device manufacturers perform comprehensive lifetime studies to assess the Failure In Time (FIT) rate of a device, where one FIT represents a failure occurring once every 10^9 hours, as explained in [46]. This FIT rating is then used to calculate the Mean Time Between Failures (MTBF) for a particular device and design. MTBF provides an estimate of the average time interval between two system failures caused by metastability, typically measured on the order of several decades.

Considering the typical MTBF, it would take a long time, possibly years, to generate a single random bit using only the metastability of a circuit as a source of randomness. Therefore, it is deemed impractical to rely solely on metastability for generating large quantities of random data.

Oscillatory Metastability

Due to the high MTBF values, instead of waiting for metastability to occur spontaneously, one of the types of metastability intentionally created for random number generation is known as oscillatory metastability. Unlike the metastable behavior seen in registers, this form of metastability does not lead a system into an undefined state but causes it to oscillate between low and high states for an undetermined period. In a study referenced as [52], it is demonstrated that oscillatory metastability can be induced by introducing an additional delay to a set/reset latch (SR-latch) circuit. This modified circuit is then initialized to an illegal state to exhibit its oscillatory metastable behavior.

In [58], the transient effect ring oscillator (TERO) is introduced as a mechanism that leverages oscillatory metastability for randomness generation. The TERO consists of a modified SR-latch that is periodically forced into an illegal state through simultaneous set and reset operations, thereby violating the latch's setup and hold times. The internal architecture of the TERO is depicted in Figure 2.10.

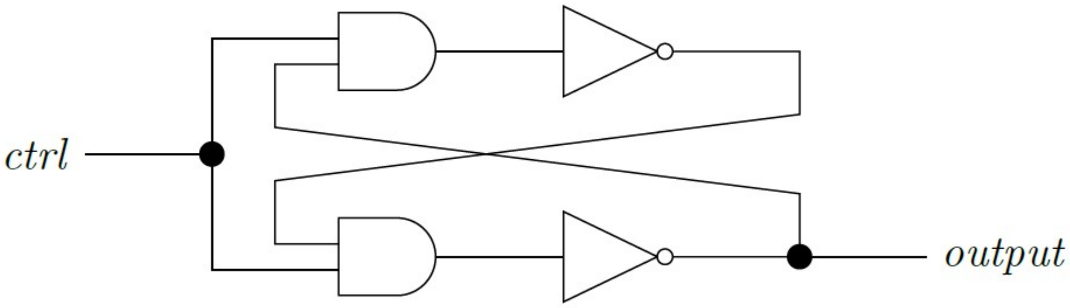

Figure 2.10: Internal structure of a TERO [46]

When the control signal is activated, the TERO enters an oscillatory state, remaining in this state for a random duration. Following the oscillatory phase, the cell stabilizes at one of the two logic levels (high or low), with the final state being random as well. Figure 2.11 illustrates that the number of oscillations at the TERO's output, as well as its final state, are variable. Unlike the analog metastability shown in Figure 2.7, in this case, the output oscillates between two discrete states, as depicted in Figure 2.8.

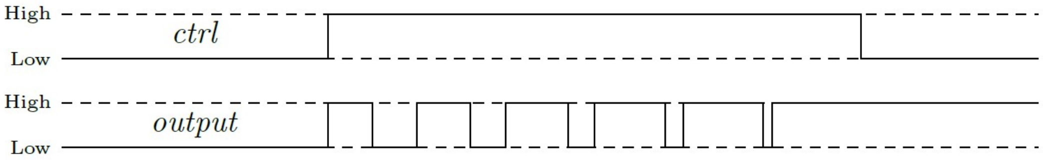

Figure 2.11: Example waveforms of a TERO [46]

2.1.1.2 Extraction of Randomness from the Clock Jitter

Clock jitter is regarded as a promising source of randomness in digital devices due to its constant presence and intrinsic random elements. To generate random numbers from the jitter, a digitization process is required, and the most commonly employed method involves sampling the jittered clock edge, as depicted in Figure 2.12.

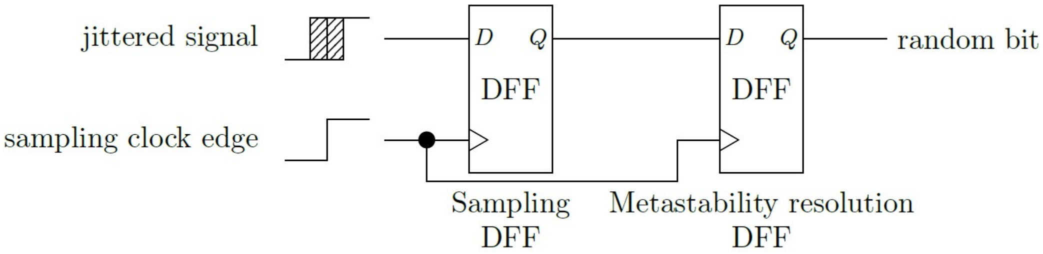

Figure 2.12: Randomness extraction from the jittered clock signal by its sampling on the rising edge of the reference clock signal [46]

Generating random bits requires the clock signal to align with the jitter-affected edge of the jittered signal. This process demands highly precise clock timing, as jitter is typically very small, often on the order of picoseconds or approximately $\frac{1}{1000}$ of the clock period. Although jitter is inevitable, the sampling clock signal is also susceptible to jitter, complicating the timing precision further. Additionally, random bits generated through this sampling method may exhibit bias, which is significantly influenced by the duty cycle of the sampled clock. With a 50% duty cycle, there is an equal probability (50%) of the output bit being 1. However, if the duty cycle is imbalanced, the probabilities of generating a 1 or 0 are no longer equal but rather proportional to the duty cycle. Despite these limitations, this sampling method remains the most commonly employed approach for extracting randomness from clock jitter as described in works of [11], [15], [53], [56].

One approach to converting random jitter into random bits involves accumulating the jitter until its size surpasses the sampled signal period [11]. In this scenario, each sampling of such a signal would yield a completely unpredictable result [46].

A TRNG employing jitter accumulation is the Elementary Ring Oscillator-based TRNG (ERO-TRNG), which was suggested and modeled in [11]. The internal configuration of this TRNG is illustrated in Figure 2.13 and includes two ring oscillators, a frequency divider, and a D flip-flop.

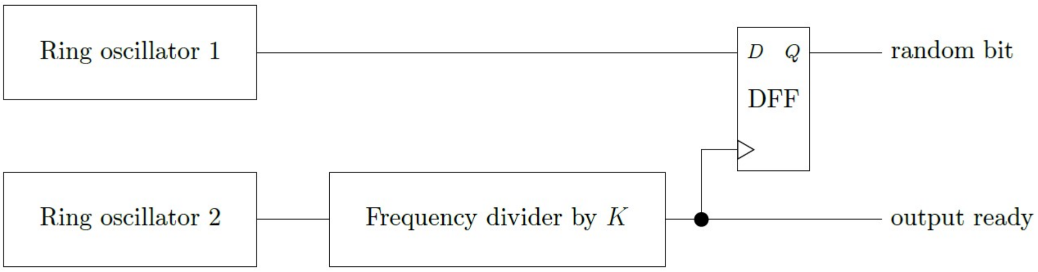

Figure 2.13: Elementary ring oscillator TRNG [46]

The inclusion of a frequency divider enables the extension of the time interval between two samplings of Oscillator 1 in the ERO-TRNG. This elongation facilitates the accumulation of phase jitter, and when the K value is sufficiently large, each output bit becomes entirely unpredictable due to the accumulated jitter.

Alternatively, the jitter accumulation time can be controlled using a frequency divider, as depicted in Figure 2.13, or by designing the periods T_1 and T_2 in such a way that their difference is smaller than the jitter standard deviation, as defined in Equation 2.5. This latter method is employed in the Coherent Sampling-based TRNG (COSO-TRNG) in [36]. In this method, instead of sampling, a counter is utilized to extract randomness from the clock jitter.

$$\sigma > |T_1| - |T_2| \tag{2.5}$$

Coherent sampling, also referred to as subsampling, is a technique that improves sampling accuracy without increasing the frequency of the sampling clock. In traditional sampling, an image of each period of the sampled signal is captured, whereas subsampling reconstructs the signal by taking samples over multiple periods. For this method to be effective, two conditions must be satisfied: the sampled signal must be periodic, and the ratio between the sampling frequency and the signal frequency must be known. Although subsampling typically operates at a slower rate than conventional sampling, it allows lower sampling frequencies to be used while maintaining high sampling precision.

2.1.2 PLL-TRNG

2.1.2.1 Basics of PLL

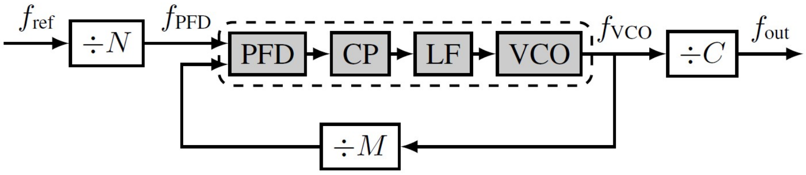

Figure 2.14: Block diagram of a PLL (PFD: phase frequency detector, CP: charge pump, LF: loop filter, VCO: voltage-controlled oscillator) [16]

A phase-locked loop (PLL) is a circuit, illustrated in Figure 2.14, that synchronizes a signal from an internal oscillator to an external input signal. The grey blocks represent analog components, which are fixed and cannot be parameterized, whereas the integer division coefficients M, N, and C, shown in white blocks, must be configured. These coefficients are crucial for determining the output frequency of the PLL (f_{out}) based on the reference frequency (f_{ref}), as defined in Equation (2.6).

$$f_{out} = f_{ref} \times \frac{M}{N \times C} \tag{2.6}$$

2.1.3 Random Bit Generation Principle of the PLL-TRNG

The working principle of the PLL-TRNG with one PLL is presented in Figure 2.15, and also PLL-TRNG with two PLLS is presented in Figure 2.16.

The jittered clock signal clk_1 from the PLL is sampled by a data or delay flip-flop (DFF) using the reference clock signal clk_0. A 1-bit counter is used to track the number of samples that equal one. Due to the frequency relationship established by the PLL, a periodic pattern with a period of $T_Q = K_D \times T_0 = K_M \times T_1$ emerges at the flip-flop output. As a result, certain samples are consistently one (shown as blue in Figure 2.15 and these are 4^{th} and 7^{th} dots), some are always zero (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 described in [23] and [24].

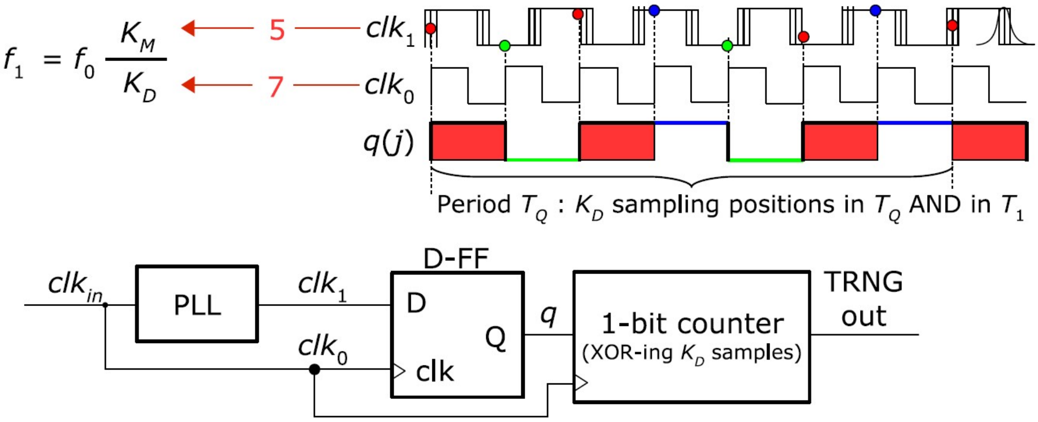

Figure 2.15: Principle of the PLL-TRNG with one PLL [23]

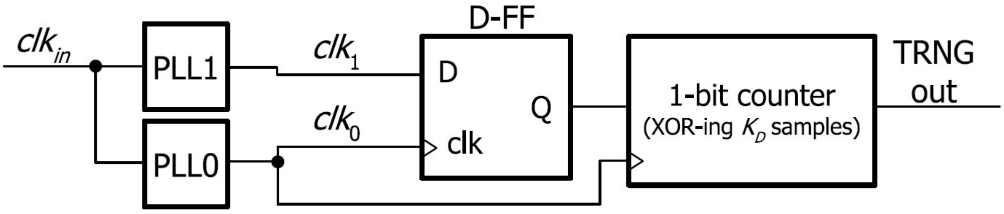

Figure 2.16: PLL-TRNG with two PLLs configuration [23]

This work adopts a PLL-TRNG architecture containing two PLLs 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 [47].

In this two PLLs case, firstly, as it is stated in Figure 2.16:

$$\frac{f_1}{f_0} = \frac{K_M}{K_D} \tag{2.7}$$

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:

$$K_M = K_{M_1} \cdot K_{D_0} \tag{2.8}$$

$$K_D = K_{M_0} \cdot K_{D_1} \tag{2.9}$$

The output (Q) of DFF in Figure 2.15 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:

$$R = \frac{f_0}{K_D} = \frac{f_1}{K_M} \tag{2.10}$$

The entropy rate per bit at the generator's output is influenced by both the jitter parameters and the generator's characteristics, specifically its sensitivity to the jitter:

$$S = \Delta^{-1} = f_0 \cdot K_M = f_1 \cdot K_D \tag{2.11}$$

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 3.1.1 for Zynq 7020 SoC values listed in Table 3.2.

2.2 Physically (or Physical) Unclonable Function (PUF)

2.2.1 Basics of PUFs

A Physically Random Function or Physical Unclonable Function (PUF) is a function that establishes a mapping from a set of challenges to a set of responses based on the inherently intricate nature of a physical system. Consequently, this static mapping results in a random assignment, as stated in [55]. The evaluation of the function can only occur with the specific physical system, and it is distinct for each individual physical instance. While PUFs can be implemented using various physical systems, this thesis focuses on silicon PUFs, which rely on the concealed timing and delay information inherent in integrated circuits. Even when sharing identical layout masks, variations in the manufacturing process lead to significant delay discrepancies among different integrated circuits.

As highlighted in the introduction, PUFs offer notably heightened physical security by deriving secrets from intricate physical systems rather than storing them in non-volatile memory. Another advantageous aspect of PUFs is that they do not necessitate any specialized manufacturing process or specific programming and testing steps, as noted in [55].

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, as shown in Figure 2.17. In contrast to TRNGs, 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 described in [14].

A PUF essentially produces a sequence (response) that serves as a unique signature in response to an input state (challenge), forming what are known as challenge-response pairs (CRPs). Each PUF can be modeled as a black box, $R = f(C)$, as shown in Figure 2.18, where the function $f()$ remains secret, as highlighted in [14].

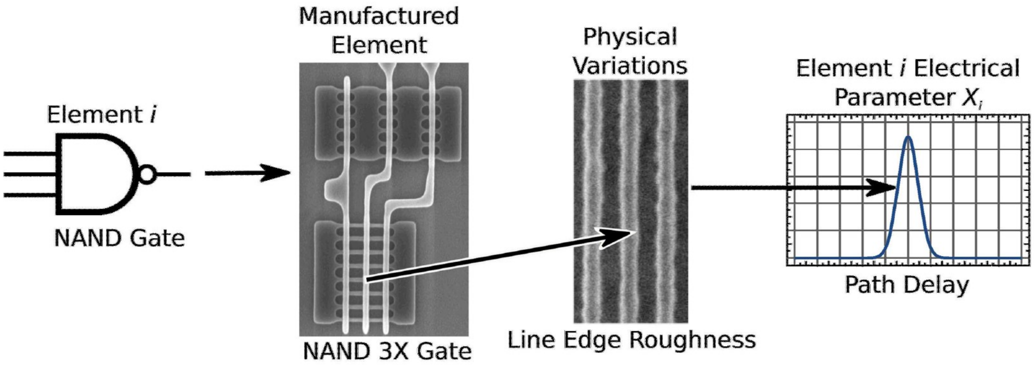

Figure 2.17: Extracting manufacturing process variations in an IC for PUF [27]

Figure 2.18: PUF model [14]

PUF circuits are often characterized by their resilience and compact size, making them particularly suitable for applications in radio-frequency identifiers (RFIDs), smart cards, and other small, cost-effective Internet of Things (IoT) devices as stated in [21]. More detailed information about PUFs can be found in [28].

2.2.2 A Basic Form of PUF-Based Authentication

As outlined in [14], the most fundamental PUF-based authentication protocol involves two phases: registration and authentication, as illustrated in Figure 2.19. During the registration phase, conducted in a secure environment, a trusted entity with access to the authentic PUF device (referred to as A) selects a random subset of possible challenges and applies them to the PUF, generating a corresponding set of responses. The CRPs for each device are securely stored by the server in a database for future authentication. Due to the large CRP space in strong PUFs and the confidentiality of the chosen subset, only a small number of CRPs need to be stored per device, making it difficult for an adversary to replicate the token for impersonation. In the verification phase, the server selects a challenge that was previously recorded but not yet used in authentication and requests a response from PUF device A. If the response closely matches a previously stored response, the device is verified as authentic.

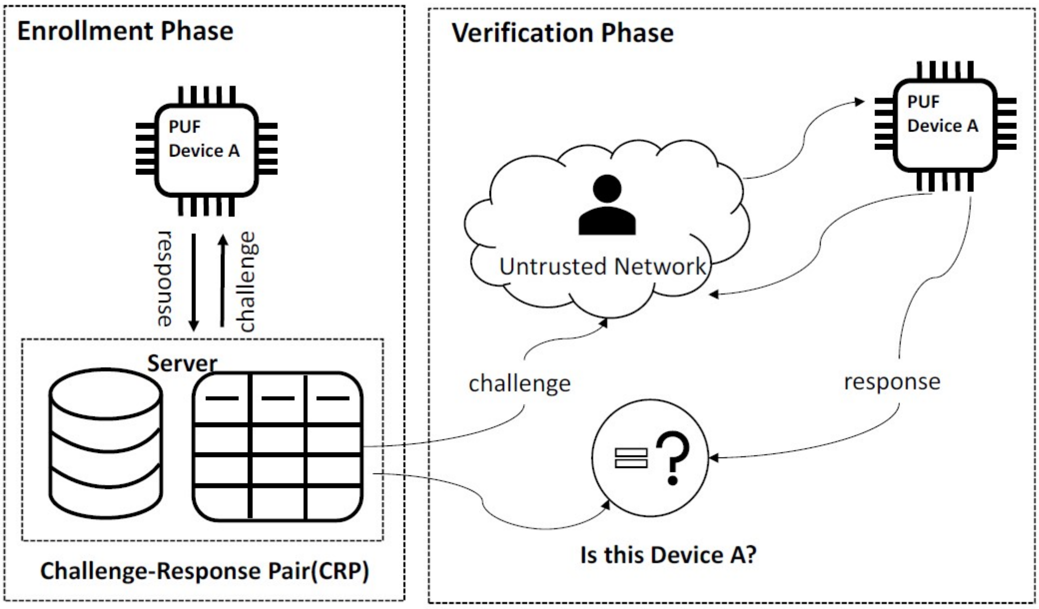

Figure 2.19: PUF-based authentication [14]

2.2.3 Types of PUFs

In the literature, there are various types of PUFs, as shown in Figure 2.20, and they can be classified with respect to their entropy sources and their CRPs [54]. 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. In order to understand the importance of this, we first need to explain the definitions of PUFs according to CRP types. With respect to CRPs, PUF types can be classified as given in [54]:

1. **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. **Weak PUF:** A Weak PUF has a limited number of CRPs, often accommodating only a small number or none at all. They are typically used in applications where limited challenge-response space is sufficient and high complexity is not required.

3. **Controlled PUF:** Controlled PUFs incorporate an additional layer of control logic that manages the interaction with the PUF. This control logic can filter, modify, or restrict access to the CRPs, enhancing security and functionality by preventing direct access to the raw responses of the PUF.

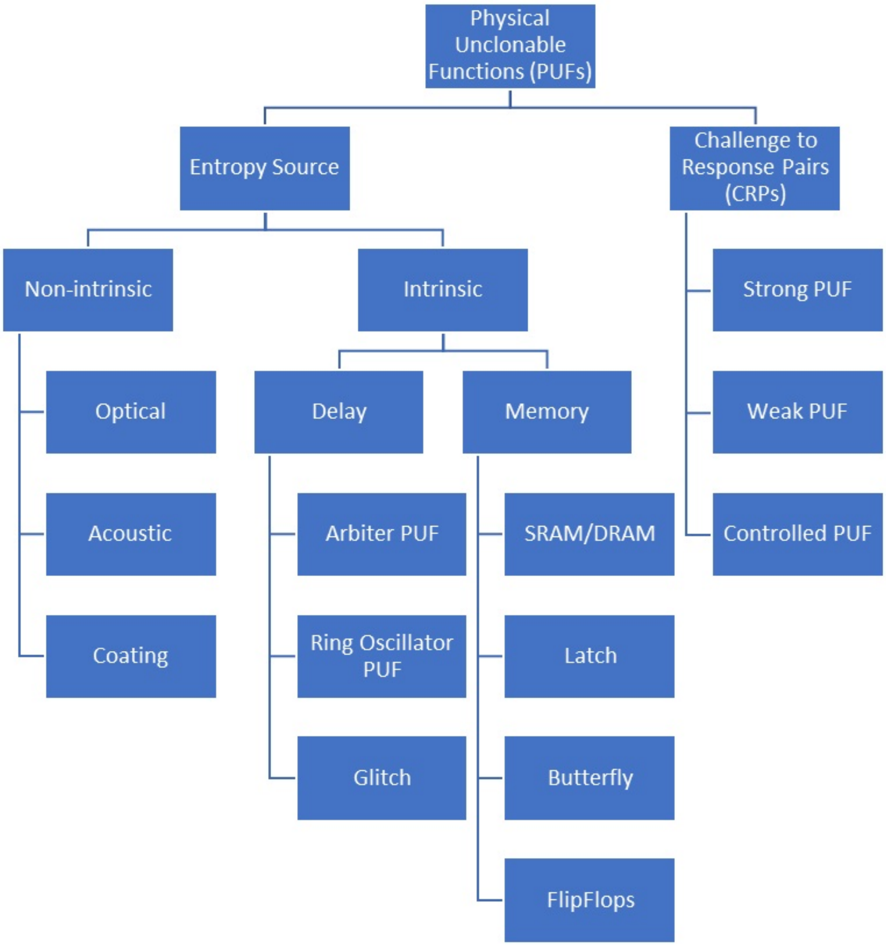

Figure 2.20: Classification of PUFs [54]

2.2.4 Types of Arbiter PUFs

2.2.4.1 Basic Arbiter PUF

An Arbiter Physical Unclonable Function (APUF) is a robust PUF relying on delay, featuring a race condition between two symmetrical digital paths. In each delay stage, two multiplexers are incorporated, and their operation is governed by challenges $(C_0, \ldots, C_{n-1})$, as illustrated in Figure 2.21.

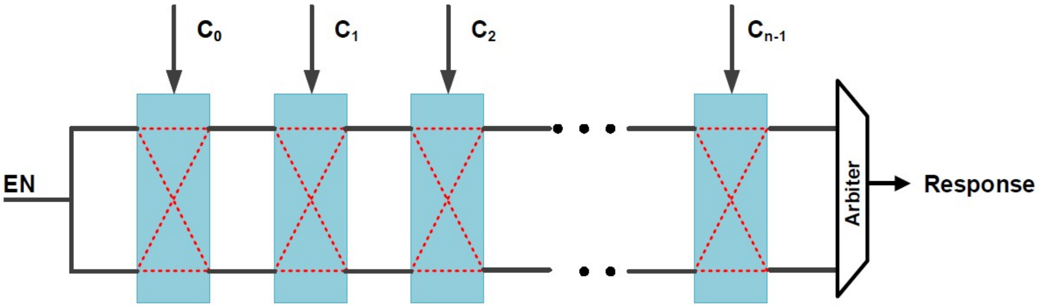

Figure 2.21: The basic APUF [14]

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 ($R = f(C)$), where C is the challenge and R is the response.

2.2.4.2 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 [55]. This new design incorporates a non-linear XOR gate into multiple arbiter PUFs to generate the final response, which is referred to as the XOR arbiter PUF. Figure 2.22 depicts a simple example of an n-bit 2-XOR-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 [37].

2.2.4.3 Component-differentially challenged XOR-PUF (CDC-XPUF)

Component-differentially challenged XOR-PUF (CDC-XPUF) 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,

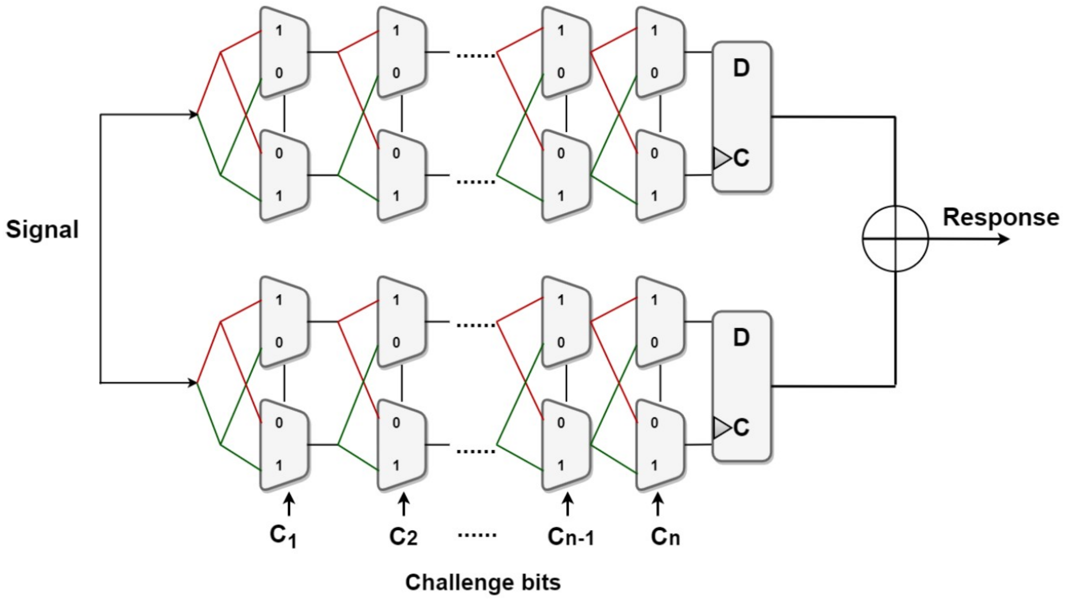

Figure 2.22: An XOR-PUF with 2 sub-streams and n bits of each stream [37]

whereas all component arbiter PUFs in an XOR-PUF receive the same challenges [37]. Figure 2.23 illustrates the structure of CDC-XPUF.

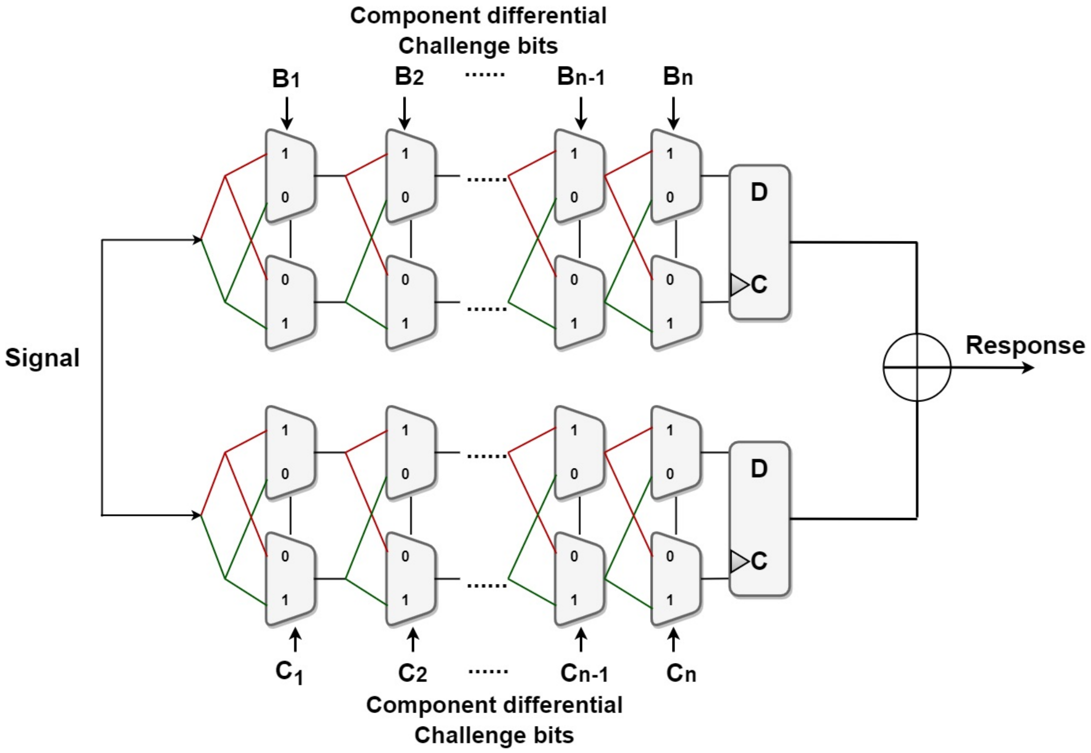

Figure 2.23: A CDC-XPUF with 2 sub-streams and n bits of each stream [37]

In order to generate different challenge bits in [37], a PRNG structure is proposed as follows:

$$C_{n+1} = (a * C_n + g) \;\; mod \; m \tag{2.12}$$

where C is the sequence of the generated random number, a is a multiplier, g is a given constant, and m is $2\dot{K}$, where K is the number of stages.

2.3 Combined PUF-TRNG Design

Security assurance traditionally depends on well-established cryptographic protocols that serve various functions, such as key generation, identification, and authentication. These protocols are founded on cryptographic algorithms, which are, in turn, based on physical implementations of hardware primitives. For robust security, hardware primitives must be resilient to physical attacks, providing a foundational layer of physical security assurance. The hierarchical security structure in IoT systems is illustrated in Figure 2.24.

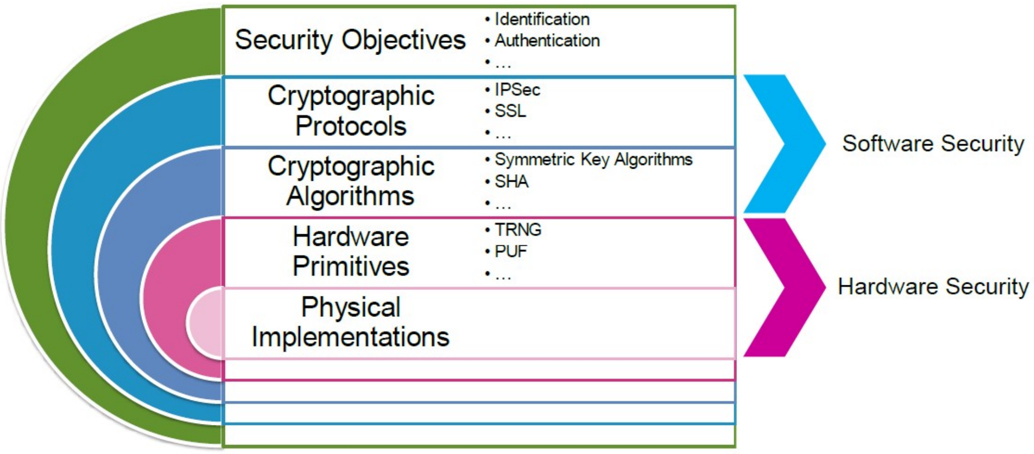

Figure 2.24: Hierarchy of security in IoT [63]

Historically, security architects have primarily focused on software security, particularly on cryptographic algorithms and protocols. However, with the increasing effectiveness of physical attacks—often more efficient and impactful than traditional attacks, even against advanced cryptographic algorithms that are mathematically secure—the focus has shifted to the hardware domain. In IoT devices, constrained resources such as memory and power present challenges to implementing conventional cryptographic security solutions. Consequently, hardware-based primitives, such as TRNGs and PUFs, which are the main focus of this thesis, are preferable [63].

As the entropy sources for conventional TRNGs and PUFs differ, these components are typically designed as separate modules or chips. Nevertheless, there is an anticipation to integrate both hardware security primitives, namely TRNG and PUF, into a single chip to enhance the security of applications relying on security measures [14]. Hence, an integrated PUF-TRNG design is implemented in this thesis. TRNG part uses clock jitter as a randomness source, while PUF part uses process variation of an SoC FPGA. Details of this combined design are presented in Chapter 5.

2.4 Evaluation Metrics of TRNGs and PUFs

2.4.1 Evaluation Criteria of TRNGs

The Bundesamt für Sicherheit in der Informationstechnik (BSI), Germany's national cybersecurity agency, plays a crucial role in developing and promoting standards, guidelines, and best practices in the fields of cybersecurity and cryptography, including random number generation.The BSI collaborates with government bodies, industry stakeholders, and international organizations to strengthen cybersecurity and safeguard critical infrastructure. Through the establishment of rigorous standards and guidelines, the BSI ensures the security and reliability of cryptographic systems. Additionally, various other organizations and research groups contribute tools and resources for evaluating RNGs, which complement the BSI's test suites, as discussed in [18].

The BSI test suites [13] 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 [13] is chosen.

The summary of the tests found in AIS-20/31 under Procedure A and Procedure B in [13], 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:

$$H_S = -2^{-H_{min}} \cdot \log_2\left(2^{-H_{min}}\right) - \left(1 - 2^{-H_{min}}\right) \cdot \log_2\left(1 - 2^{-H_{min}}\right) \tag{2.13}$$

The BSI standard also specifies a minimum requirement for Shannon entropy that must be met to validate a module. The formula for Shannon entropy is applied to min-entropy normalized to a bit unit. For example, if the H_{min} value from the BSI suite corresponds to entropy samples with a bit width greater than 1 bit, it must be normalized by dividing the H_{min} value by the bit width. For example, values of 7.999 and 7.888 were obtained, corresponding to normalized min-entropy values of 0.999 and 0.986, respectively. Applying the Shannon entropy formula, the resulting H_S values were both approximately 1.000. In addition to these, the BSI specifies that the confidence level of the results is 99.87% [13].

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.

2.4.2 Evaluation Criteria 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 chapters following the initial chapter, the evaluation criteria studied and constructed by either Hori et al. [30] or Maiti et al. [40] 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. Additionally, in the list below, it is indicated that each metric is defined by whom.

1. Reliability of responses from the same PUFs
    - Steadiness in Hori et al. [30]
    - Correctness in Hori et al. [30]
2. Entropy of responses from the same PUFs
    - Diffuseness in Hori et al. [30]
    - Uniformity in Maiti et al. [40]
3. Fingerprint property
    - Uniqueness in Maiti et al. [40]

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. [30] defines the randomness metric, Maiti et al. [40] 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. [30] indicates that randomness is evaluated, only some kind of uniformity is evaluated as in Maiti et al. [40]. This choice can be understood better by the explanation in Section 2.4.2.4. In addition to these, as indicated in [9], 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.

2.4.2.1 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, as explained in [38]. Additionally, if adversaries gain access to a sufficient number of silicon CRPs, PUFs may become susceptible to machine learning attacks, as explained in [1], [2], [10], [44]. 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 [37], 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 [37] 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 [37] 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.

2.4.2.2 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 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. [30]. When generating identical responses multiple times on the same device, it is expected that all responses remain consistent. Steadiness measures the stability of a PUF in producing the same responses to identical challenge sets. A steadiness value of 1 indicates that no variations occurred in the responses recorded during the experiment. Steadiness is calculated as follows:

$$S = 1 + \frac{1}{N_c} \sum_{k=1}^{N_c} \log_2 max\{\frac{\sum_{j=1}^{N_a} b_{k,j}}{N_a}, 1 - \frac{\sum_{j=1}^{N_a} b_{k,j}}{N_a}\} \tag{2.14}$$

where N_c represents the number of distinct challenges used, N_a refers to the number of times each challenge is applied, and $b_{k,j}$ denotes the j-th response out of all N_a responses to the k-th challenge in the set of N_c challenges. The challenge-response pairs (CRPs) that pass the steadiness test are referred to as *Correct ID*, as noted in [43].

Correctness

This metric is defined by Hori et al. [30] and is almost the same metric as reliability, which is defined by Maiti et al. [40]. The primary distinction between their equations lies in the normalization factor. Correctness is normalized by the maximum value of the Fractional Hamming Distance of the responses, whereas reliability is normalized by the average. Therefore, only the correctness value was calculated, and reliability was not considered. The ideal correctness value is 1, which is computed as follows:

$$C = 1 - \frac{2}{N_c \times N_a} \sum_{k=1}^{N_c} \sum_{j=1}^{N_a} (b_k \oplus b_{k,j}) \tag{2.15}$$

where b_k is the *Correct ID*. The *Correct ID* is determined by majority voting among all responses provided for a given input challenge. In this case, N_c denotes the number of challenges in the dataset, and $b_{k,j}$ represents the j-th response within the set of N_a responses corresponding to the k-th challenge.

2.4.2.3 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. [30] introduced the diffuseness metric, while

Maiti et al. [40] proposed the uniformity metric. Given the close resemblance between Hori's [30] randomness metric and Maiti's [40] uniformity metric, only the uniformity metric is assessed in this context.

Diffuseness

The diffuseness metric, introduced by Hori et al. [30], 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:

$$D = \frac{4}{K^2 \times L} \sum_{l=1}^{L} \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} (b_{i,l} \oplus b_{j,l}) \tag{2.16}$$

where L represents the length of the responses, measured in bits, while K denotes the number of multi-bit responses utilized in the experimental study.

Uniformity

The uniformity, as introduced by Maiti et al. [40], evaluates the balance between zeros and ones in the responses generated by a PUF. The ideal value for uniformity is 0.5. It can be computed as follows:

$$U = \frac{1}{N_r} \sum_{i=1}^{N_r} b_i \tag{2.17}$$

where N_r represents the length of the response in the set, and b_i refers to the i-th bit of the response.

The randomness metric, defined by Hori et al. [30], 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:

$$H = -\log_2 max(p, 1-p), \tag{2.18}$$

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

$$p = \frac{1}{N_r} \sum_{i=1}^{N_r} b_i \tag{2.19}$$

where N_r is the response length in a set, and b_i is the i-th response bit.

It is obvious that the Equations (2.17) and (2.19) are nearly the same. These two equations define the same thing actually, and it is the uniformity of the responses. Hori et al. [30] claim that taking this uniformity and using them in (2.18) calculates the randomness. The approach presented by Hori et al. [30] is not suitable for accurately calculating randomness. Equation (2.18) 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. [40] in Equation (2.17). Thus, using this equation does not contribute to a deeper understanding of randomness beyond what uniformity already indicates. As stated in [9], 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. [40].

2.4.2.4 Fingerprint Property

Uniqueness

The uniqueness, as introduced by Maiti et al. [40], is determined by calculating the Hamming Distance between the responses of two devices. It can be computed as follows:

$$U_k = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{HD(ID_i, ID_j)}{L} \tag{2.20}$$

where ID_i and ID_j represent two L-bit responses generated by a PUF implemented on two different chips (the i-th and j-th chip) in response to the k-th challenge, which is applied repeatedly L times. The ideal value of Maiti's uniqueness metric [40] is 0.5.

As indicated in the last part of the previous section, the resource utilization rate is a metric for both TRNGs and PUFs.

2.5 Xilinx Zynq SoC FPGA

The algorithm in this thesis work is implemented in SoC FPGA or simply SoC. Using SoC instead of FPGA has some advantages. They offer higher integration, lower power, smaller board sizes, and higher bandwidth communication between the processor and FPGA. Additionally, since SoCs have hard processors, these processors can also be used for future implementations.

Xilinx Zynq-7000 SoC ZC702 HW-Z7-ZC702 Rev1.1 Evaluation Board is chosen for the PUF-TRNG implementation in this thesis. In Figure 2.25, this board is shown. Using this board, the algorithm is executed, and the results from both TRNG and PUF are saved in an external memory. After that, these results are sent to a PC to be examined using compliance tests.

Figure 2.25: Xilinx Zynq-7000 SoC ZC702 Evaluation Kit [7]

The ZC702 Evaluation Board includes an SoC FPGA from Xilinx Zynq-7000 Series, and the exact part number is XC7Z020-CLG484-1. The internal architecture of these SoCs is depicted in Figure 2.26. These SoCs are composed of two primary sections: the processing system (PS) and the programmable logic (PL). In this work, the algorithms are implemented within the PL section, where the PLLs are also utilized.

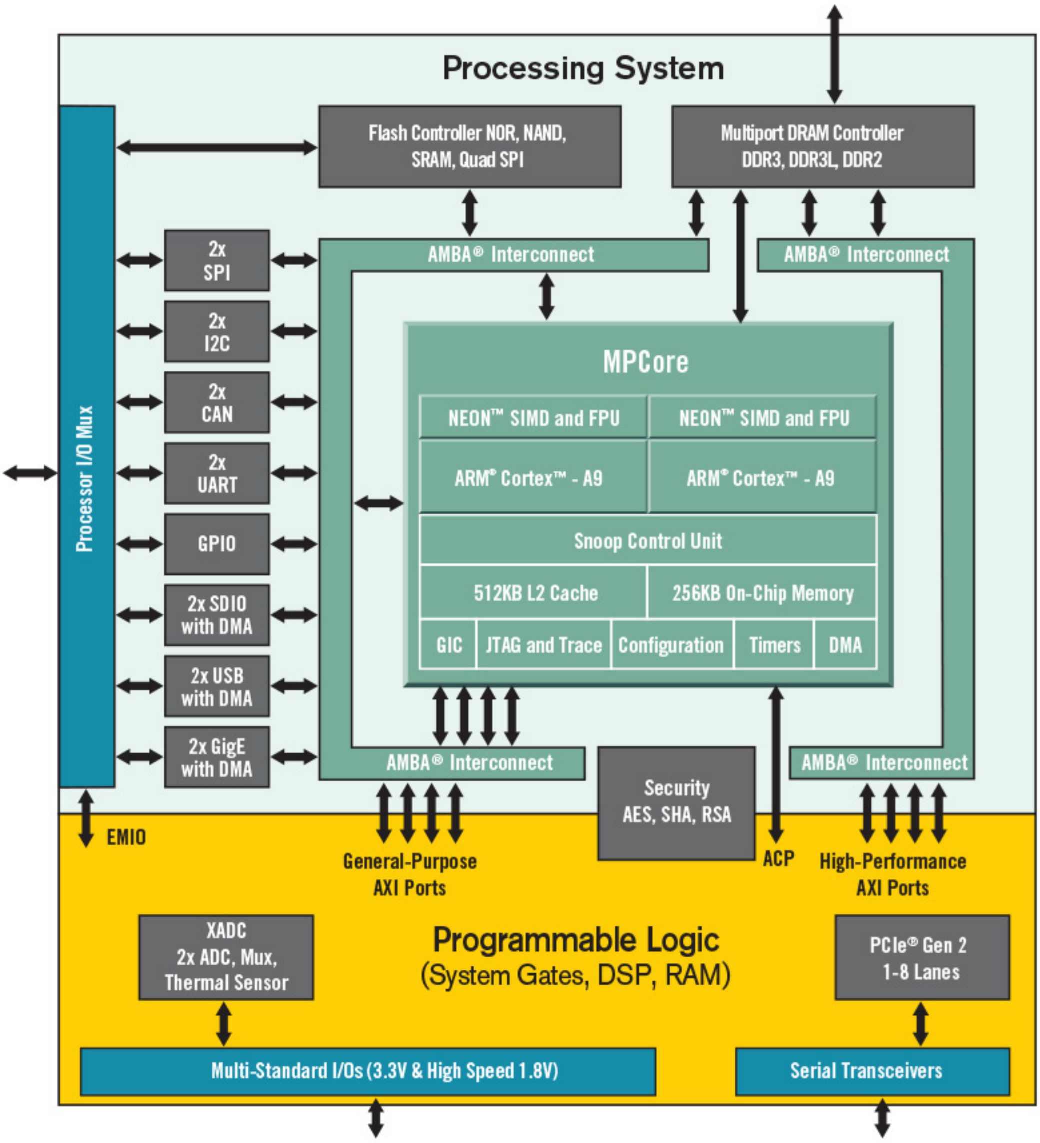

Figure 2.26: Block Scheme of Internal Structure of Xilinx Zynq-7000 SoC [6]

CHAPTER 3

AN AIS-20/31 COMPLIANT PLL-TRNG IMPLEMENTATION ON A ZYNQ-7020 SOC

In this chapter, the implementation details of the PLL-TRNG are presented. Also, the results of the implementation and comparisons with the previous work are demonstrated. The outcome of this chapter was accepted at 17^{th} International Conference on Computational Intelligence in Security for Information Systems (CISIS 2024), see [62].

3.1 PLL-TRNG Implementation

The PLL-TRNG is particularly well-suited for integration in FPGAs or SoC FPGAs. Its implementation in ASICs is relatively expensive due to the substantial silicon area required by PLLs. However, in many FPGAs, multiple PLLs are available, significantly reducing the implementation cost. Additionally, PLLs in FPGAs are physically and electrically isolated as they are implemented as hardwired blocks. The repeatability of the PLL-TRNG is high because the implementation relies solely on the configuration of digital components within the PLL, such as multiplication and division factors. This chapter provides a comprehensive review of PLL-TRNG design, discusses its advantages and potential challenges, and proposes automated methods to optimize PLL-TRNG design. The reference design of the PLL-TRNG with two PLLs, described in detail in citepeturathesis, is improved by adding other PLLs.

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. However, prior to full-scale implementation, intermediate configurations employing three PLLs are investigated to facilitate a systematic design process. This study elucidates the design rationale for the four-PLL system by providing detailed explanations of these intermediate steps. Consequently, four distinct PLL-TRNG configurations are presented in Table 3.1 and visually depicted in Figure 3.1:

Table 3.1: Configurations of PLL-TRNG Implementations

Codes of PLL-TRNG Designs Depicted in Figure 3.1	**Number of PLLs**	**Purpose of Use of PLL**		**PLL-TRNG Design Type**
		As Reference Clock	**As Jittered Clock**	
(a)	2	1	1	Referenced Design
(b)	3	1	2	Intermediate Step for Proposed Approach
(c)	3	2	1	Intermediate Step for Proposed Approach
(d)	4	2	2	Proposed Design

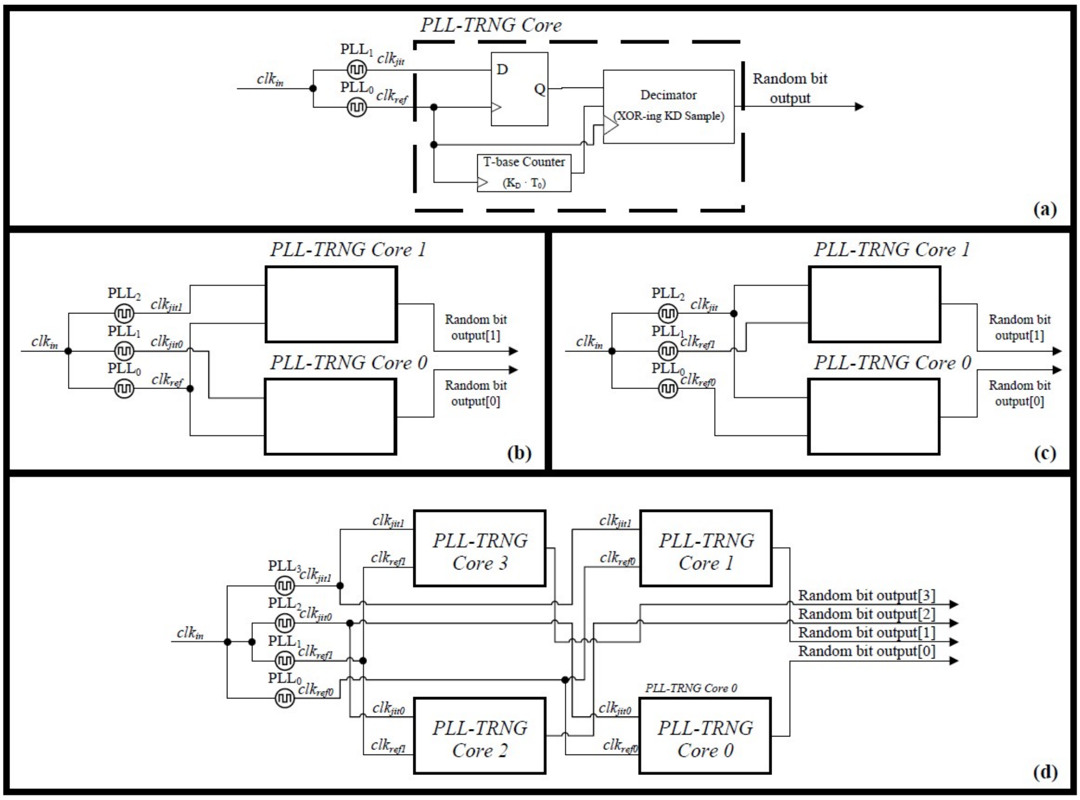

Figure 3.1: Implemented PLL-TRNG Configurations: (a), (b), (c), and (d)

3.1.1 Determining PLL-TRNG Parameters

In this work, as the parameter search algorithm, the backtracking algorithm in [16] is selected. Given a set of variables explained in Section 2.1.2.1 and Section 2.1.3 and constraints listed in Table 3.2, 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 [16] involves determining the PLL-TRNG parameters that comply with both physical constraints and application requirements.

Table 3.2: Table of ranges of possible values for the PLL parameters and frequencies for Zynq-7000 SoC [4], [8]

Parameters	**Xilinx Zynq-7000**	
	Min	**Max**
f_{ref} (MHz)	19	800
P_{VCOd}	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

The code in the backtracking is open-source shared in [17]. Hence, we can modify it for Zynq 7020 SoC parameters provided in Figure 2.14. The open source code of this algorithm is in Python programming language [51] and modified in Visual Studio 2022 [42]. Table 3.2 is generated with PLL properties of the SoC except for f_{out} which is determined concerning the maximum frequency value of BUFG clock buffer in Zynq 7000 Series [8]. BUFG must be used in the SoC design, and hence, it restricts f_{out} value for the search algorithm.

After generating results for our case, 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, those results must be tested with one more criterion. The sampling process of the jittered clock with the reference clock is illustrated in Figure 2.15. To generate random numbers at the output of this PLL-TRNG, it is necessary for at least one sample to be influenced by jitter. This requires that the distance between any edge of clk_0 and the corresponding edge of clk_1 be less than Δ. This condition is satisfied if the following equation holds, as discussed in [25] and [45]:

$$\sigma_{jit} > max(\Delta T_{min}) \tag{3.1}$$

where σ_{jit} represents the standard deviation of the jitter at the PLL output, and $max(\Delta T_{min})$ denotes the maximum distance between the two nearest edges of clk_0 and clk_1. This can be calculated as outlined in [25] and [45]:

$$max(\Delta T_{min}) = \frac{T_{clk_0}}{4K_M} gcd(2K_M, K_D) = \frac{T_{clk_1}}{4K_D} gcd(2K_M, K_D), \tag{3.2}$$

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 (3.1). 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 3.3. As it can be seen, all the selected configurations satisfy Equation (3.1).

Table 3.3: Determined Parameters for the PLL-TRNG Implementations for $f_{ref} = 125$ MHz

Config.	(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	(51,4,4) (11,1,3)	398.438 458.333	176	153	2.60417	0.07013	0.18263	76.706	3.56506
Max. S	(51,4,4) (32,3,3)	398.438 444.444	512	459	0.86806	0.204	0.177084	100.882	1.22549
Max. $R \cdot S$	(37,5,2) (32,3,3)	462.5 444.444	320	333	1.38889	0.148	0.20556	100.882	1.68919

3.1.2 PLL-TRNG Implementation Setup

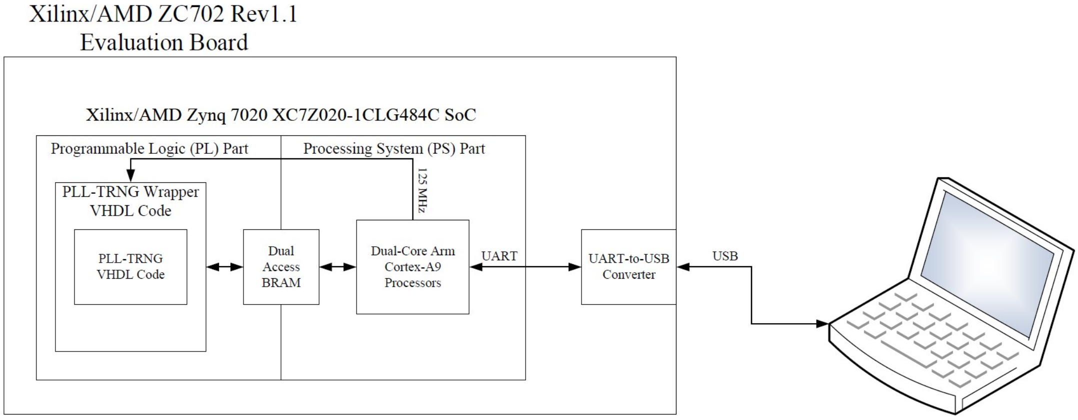

Figure 3.2: Block Diagram of Implementation Setup

The implementation setup employed in this study is illustrated in Figure 3.2. It utilizes the ZC702 Rev1.1 Evaluation Board [7], which incorporates the Zynq 7020 XC7Z020-1CLG484C SoC to facilitate the implementation of four distinct PLL-TRNG configurations as detailed in Section 3.1. In the Programmable Logic (PL) section, four distinct designs, specified in Table 3.1, are developed using Vivado 2019.1 [5] in VHDL[31]. To enable real-time transmission of generated random numbers to a personal computer (PC), a dual-access Block RAM (BRAM) was employed. One port of this BRAM is connected to the PL, while the other is connected to the processing system (PS) section. The requisite code for the PS section is written in the C programming language [33]. The PL section generates random numbers and writes a predefined value to a specific BRAM address to indicate that the random bits are ready. Once this indication is given, the software in the PS section outputs the random bits to the UART serial port, which are then converted to USB and transmitted to the PC. The received bits on the PC are saved in their ASCII-coded hexadecimal form and later converted to binary form offline to serve as input for AIS-20/31 Tests [12]. The codes of AIS-20/31 Tests in [12] were compiled using [22] 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 to have a size of approximately 7.2 Mb. Additionally, a 125 MHz clock frequency was selected for the system's main clock (clk_{in}) due to timing constraints inherent in the SoC.

In conclusion, three ZC702 Rev1.1 Evaluation Boards are employed to demonstrate that the implemented PLL-TRNG configurations are not specific to a particular device. The backtracking algorithm and the elimination criteria outlined in Equation (3.1) are used to determine the PLL-TRNG configuration parameters for maximizing S, R, and the product $R \cdot S$. Subsequently, five random output bit files are generated for each of the four configurations described in Section 3.1 and tested on the three evaluation boards. The results are stored on a PC, and AIS-20/31 Tests are conducted. The outcomes of these tests are detailed in the following section.

3.2 PLL-TRNG Results and Comparisons with Previous Works

The implementation results are presented in Table 3.4. In this table, each row corresponds to a unique configuration defined by the number of PLLs in the PLL-TRNG and the parameter configuration. By considering our four different PLL-TRNG configurations and three different PLL parameter selections, we have twelve distinct rows, in other words, twelve different results. For each row, five different approximately 7.2 Mb random number files are generated for each of the ZC702 Boards. Hence, the arithmetic mean of these fifteen values (3 boards $\times$ 5 repetitions) is used for the Shannon entropy calculation in Table 3.4.

Procedure B of the BSI Test Suite calculates min-entropy H_{min} for a given bitstream. After normalizing these values and using Equation 2.13, the Shannon entropies are calculated and presented in Table 3.4. Additionally, it is not indicated in Table 3.4, but it must be emphasized that all four PLL-TRNG designs passed all the AIS-20/31 Tests for all three different configurations on three distinct boards for all generated files.

Table 3.5 provides a comparative analysis of our work with previously implemented PLL-TRNGs. The results indicate that our 4-PLL implementation significantly enhances the output bit rate of the PLL-TRNG design while maintaining robust cryptographic properties. Specifically, the table shows that a speed of approximately 10.4 Mb/s can be achieved, which is notably higher than any other reported PLL-TRNG implementation. Additionally, our results exhibit superior Shannon Entropy compared to earlier PLL-TRNG designs in [45] and [48].

Table 3.4: PLL-TRNG Implementation Results

PLL Configuration	Parameter Configuration	R (Mbit/s)	Output Bit Rate (Mbit/s)	S (ps^{-1})	$R \cdot S$	Entropy (Shannon)
2-PLL with one reference clock and one jittered clock	Max. R	2.6042	2.60417	0.0701	0.18263	**0.999999986833568**
	Max. S	0.8681	0.86806	0.204	0.17708	0.999999986516413
	Max. $R \cdot S$	1.3889	1.38889	0.148	0.20556	0.999999981069240
3-PLL with one reference clock and two jittered clocks	Max. R	2.6042	5.20834	0.0701	0.18263	0.999999976364641
	Max. S	0.8681	1.73612	0.204	0.17708	0.999999977771549
	Max. $R \cdot S$	1.3889	2.77778	0.148	0.20556	0.999999980834110
3-PLL with two reference clocks and one jittered clock	Max. R	2.6042	5.20834	0.0701	0.18263	0.999999985962200
	Max. S	0.8681	1.73612	0.204	0.17708	0.999999961780714
	Max. $R \cdot S$	1.3889	2.77778	0.148	0.20556	0.999999966076834
4-PLL with two reference clocks and two jittered clocks	Max. R	2.6042	**10.41668**	0.0701	0.18263	0.999999972332402
	Max. S	0.8681	3.47224	0.204	0.17708	0.999999971434251
	Max. $R \cdot S$	1.3889	5.55556	0.148	0.20556	0.999999956486246

Table 3.5: PLL-TRNG Implementation Results Comparison with [16], [45], and [48]

	Results of 4-PLL-TRNG			Results in [16] for Xilinx Spartan-6			Results in [45] for Xilinx Spartan-6		
Parameter Configs.	Output Bit Rate (Mbit/s)	S (ps^{-1})	Entropy (Shannon)	Output Bit Rate (Mbit/s)	S (ps^{-1})	Entropy (Shannon)	Output Bit Rate (Mbit/s)	S (ps^{-1})	Entropy (Shannon)
Max. R	10.41668	0.07013	0.999999972332402	1.042	0.094	1	0.555	0.0913	0.997
Max. S	3.47224	0.204	0.999999971434251	0.521	0.167	0.99999	0.555	0.0913	0.997
Max. $R \cdot S$	5.55556	0.148	0.999999956486246	N/A	N/A	N/A	0.555	0.0913	0.997
	Results of 4-PLL-TRNG			Results in [48] for Xilinx Spartan-6					
Parameter Configs.	Output Bit Rate (Mbit/s)	S (ps^{-1})	Entropy (Shannon)	Output Bit Rate (Mbit/s)	S (ps^{-1})	Entropy (Shannon)			
Max. R	10.41668	0.07013	0.999999972332402	0.44	N/A	0.999931407560694			
Max. S	3.47224	0.204	0.999999971434251	0.44	N/A	0.999931407560694			
Max. $R \cdot S$	5.55556	0.148	0.999999956486246	0.44	N/A	0.999931407560694			

3.3 Utilization Results of 4-PLL-TRNG in Zynq-7020 SoC FPGA

Our final design is a PLL-TRNG consisting of 4 PLLs, named 4-PLL-TRNG, and as a result, we focus on analyzing the utilization rate of its various configurations. The implementation employs state machines in the PL part, enabling the generation of random numbers within the PL part, which are then transmitted to the PS part via Dual Access BRAM. Despite the PL part containing not only the 4-PLL-TRNG but also the necessary state machines for the implementation, the overall utilization rate remains relatively low, with the exception of the PLLs, as demonstrated in Table 3.6 for the three different configurations of the 4-PLL-TRNG design. These low utilization rates highlight the potential of the 4-PLL-TRNG as a promising candidate for applications requiring a TRNG.

Table 3.6: Utilization Table Generated Using Vivado 2019.1 [5] for 4-PLL-TRNG Implementation

Resource Type	**Avaliable Resource Quantity**	**Utilization Quantity (Utilization Rate as %) of Max. R Configuration**	**Utilization Quantity (Utilization Rate as %) of Max. S Configuration**	**Utilization Quantity (Utilization Rate as %) of Max. $R \cdot S$ Configuration**
LUT	53200	1539 (2.89%)	1542 (2.90%)	1536 (2.89%)
LUTRAM	17400	72 (0.41%)	72 (0.41%)	72 (0.41%)
FF	106400	1884 (1.77%)	1884 (1.77%)	1884 (1.77%)
BRAM	140	2 (1.43%)	2 (1.43%)	2 (1.43%)
IO	200	8 (4.00%)	8 (4.00%)	8 (4.00%)
PLL	4	4 (100.00%)	4 (100.00%)	4 (100.00%)

3.3.1 Discussion About PLL-TRNG Implementation Results

In this chapter, a total of four versions of the PLL-TRNG were implemented, including a 2-PLL version as the reference design and a 4-PLL version as the final design. The details of these two versions, along with the other two intermediate versions, are provided in Table 3.1. During these implementations, three different ZC702 Evaluation Boards were used. The AIS-20/31 test was employed to assess the randomness of the generated outputs.

One of the most critical aspects of PLL-TRNG designs is the appropriate selection of PLL parameters, as shown in Figure 2.14. To achieve this, the backtracking-based algorithm in [16] was utilized, and suitable candidates were identified. These candidates were ranked according to the max. R, max. S, and max. $R \cdot S$ criteria, and from the results, three different configurations were created. The tests were performed on all three boards for each of the three different configurations. Additionally, each configuration was tested five times on each board, and the arithmetic mean of the results was calculated for Shannon entropy calculations. The block diagram of the implementation setup is illustrated in Figure 3.2

The generated random bitstreams successfully passed all tests defined in AIS-20/31, and the Shannon entropy values were equal to or better than those reported in the literature. Furthermore, the 4-PLL-TRNG with the max. R configuration demonstrated that this design offers a higher output probability than all other PLL-TRNGs in the literature.

In addition, the resource utilization of the Zynq-7020 SoC used for this design was examined for the three different configurations of the 4-PLL-TRNG, and in all three cases, the resource usage was low enough not to hinder its use in any IoT system. The results are listed in Table 3.6. Naturally, due to the use of all four PLLs in the SoC (which contains only four), the utilization of this resource is 100%. For all other resources, the usage does not exceed 4%. These resource utilization results indicate that the 4-PLL-TRNG with the max. R configuration is suitable for use in IoT systems due to its low resource consumption, high output rate, and high entropy.

CHAPTER 4

32-BIT AND 64-BIT CDC-7-XPUF IMPLEMENTATION ON A ZYNQ-7020 SOC

In this chapter, details of the implementations of the 32-bit and 64-bit CDC-7-XPUF are presented. Also, the results of the implementation and comparison with the referenced work are demonstrated. The results of this PUF implementation are presented in the paper [60].

4.1 CDC-XPUF Implementation Details

In this study, our aim is to implement an ML-resistant PUF with good cryptographic properties explained in Section 2.4.2.2 - 2.4.2.4. As it is stated in [37], 64-bit CDC-7-XPUF is ML-resistant. However, in [37], the cryptographic properties are examined. These are examined in [43], but only for a maximum of 32-bit CDC-7-XPUF. Hence, we decided that firstly, we implemented 32-bit CDC-7-XPUF and showed that the design satisfies good cryptographic properties, as the referenced PUF design does. After that, we implemented the 64-bit or ML-resistant version of CDC-7-XPUF. In this chapter, we present all of our results with respect to the metrics explained in Section 2.4.2.2 - 2.4.2.4 and compare our results to the referenced design [43].

The implementation details of CDC-XPUF are given in both [43] and [37] and also explained in Section2.2.4.3. The results of the implementation and comparisons with the previous works are presented in Section 4.2.

The MUX-based CDC-XPUF arbiter structures are implemented in Vivado 2019.1.

As an illustration of this structure, the schematic of four MUXes of the MUX array is shown in Figure 4.1. Both the 32-bit design and the 64-bit design have the same MUX-based structure illustrated in Figure 4.1.

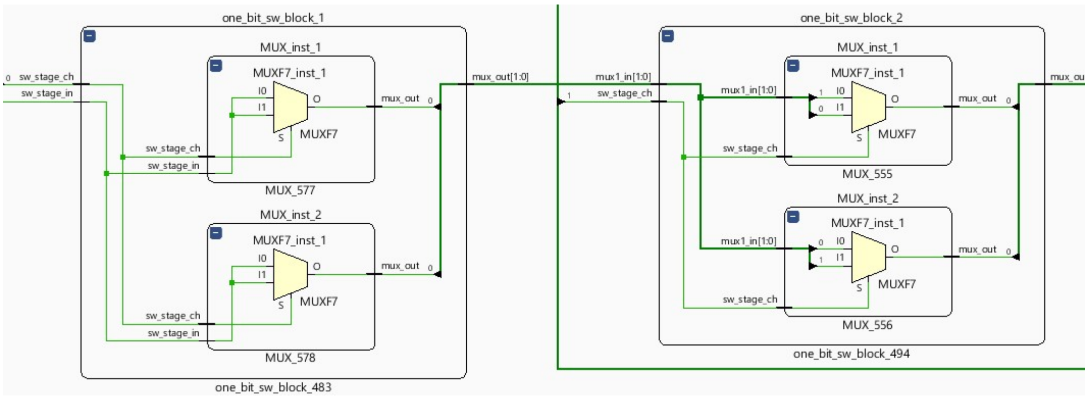

Figure 4.1: Vivado 2019.1 Schematic Design View of CDC-7-XPUFs

Since the CDC-XPUF is a delay-based PUF, relying on the calculation of delays incurred by the internal gates and interconnections, the correct placement of its components is crucial. To ensure equal delay lines, the top and bottom of each stage in the CDC XPUF must be precisely aligned. Figure 4.2 illustrates a bad placement example of the CDC XPUF's MUXes. In this figure, the bottom path is longer than the top path due to the MUXes highlighted in orange, resulting in the bottom path's delay consistently exceeding the top path's delay and causing biased responses. In contrast, Figure 4.3 demonstrates a good placement example and the way how we implement MUXes in the actual design.

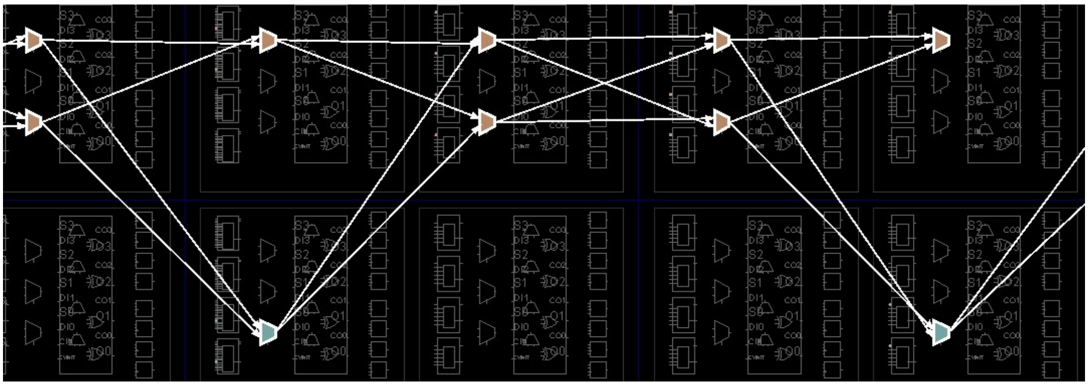

Figure 4.2: An Example of Bad Placement of CDC-7-XPUF MUXes

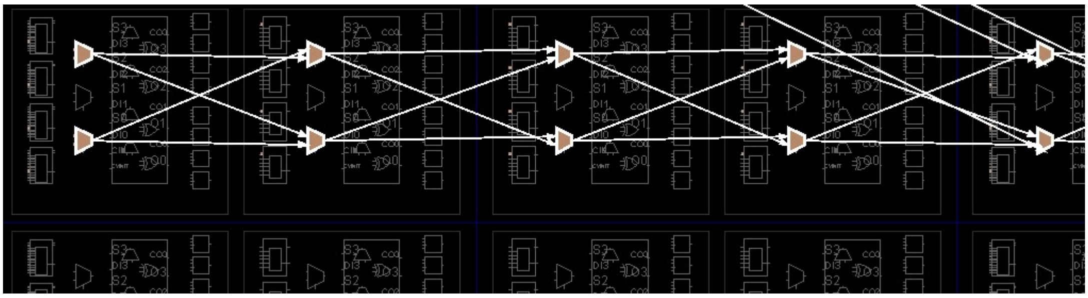

Figure 4.3: An Example of Good Placement of CDC-7-XPUF MUXes

For generating different challenges for different stages, a PRNG is proposed in Equation 2.12. Obviously, two PRNGs with two different parameter sets are used for the 32-bit and 64-bit designs.

The implementation setup illustrated in Figure 4.4 is used. It is very similar to the PLL-TRNG setup in Figure 3.2. The only difference is that PLL-TRNG parts are changed with CDC-7-XPUF codes. Obviously, both the 32-bit and the 64-bit designs have the same setup with different VHDL codes.

The software developed in Python [51] using Visual Studio 2022 [42] is utilized to calculate the scores for the five evaluation metrics, which are detailed in Section 2.4.2 from the generated bitstreams.

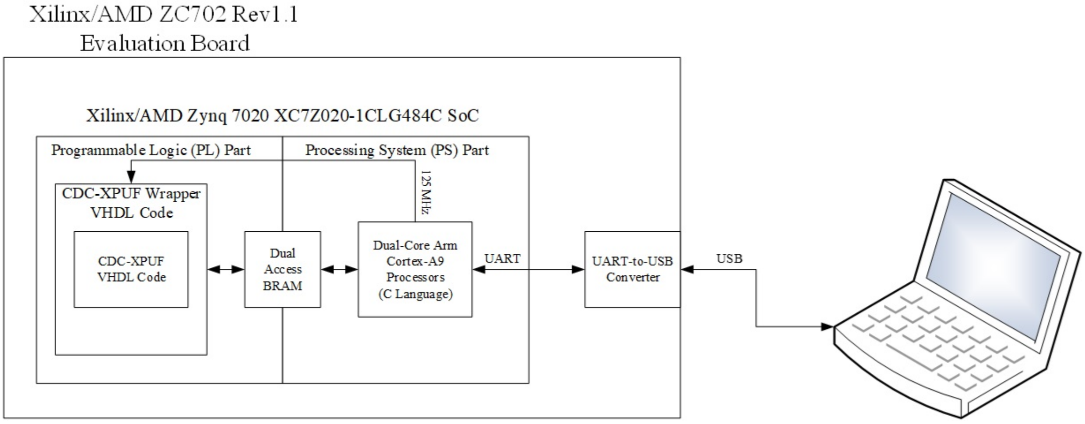

Figure 4.4: Block Diagram of Implementation Setup of CDC-7-XPUFes

Using the setup in Figure 4.4, for the statistical characteristics CRPs, we generated up to 16,000 (challenges) ×32 (iterations) ×128 (response length) ×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. Through a dual-access BRAM, CRPs are sent to the PS part. From the PS part via UART, the CRPs are sent to the PC with a baud rate of 230,400 bits/second between the PuTTY [50] terminal and the SoCs.

4.2 32-bit and 64-bit CDC-7-XPUF Experimental Results and Comparisons

The evaluation metrics of PUFs are explained in Section 2.4.2. As explained in Section 2.4.2, respectively, the implementation results of steadiness, correctness, diffuseness, uniformity, and uniqueness are presented. Furthermore, the calculated metric scores of the reference PUF implemented on [43] are presented in the result tables in the following sections for comparison purposes.

4.2.1 Steadiness

The steadiness scores of the referenced 32-bit CDC-7-XPUF in [43], our implemented 32-bit CDC-7-XPUF, and 64-bit CDC-7-XPUF are presented in Table 4.1. In Equation 2.14, the steadiness score is calculated between 0 and 1. Hence, we normalize it using percentages to calculate the score in the table. As stated before, 32 iterations of 128-bit responses are generated. In the steadiness calculation, these 4096-bit long responses are used.

Our result for the 32-bit is only slightly worse than the reference design in an acceptable range. Also, the 64-bit result is slightly worse than the 32-bit results. As it can be seen in [43], increasing the stage number has a negative effect on the steadiness.

Table 4.1: Steadiness Results

Steadiness Score of Referenced Work 32-bit CDC-7-XPUF [43]	**Steadiness Score of 32-bit CDC-7-XPUF Implementation**	**Steadiness Score of 64-bit CDC-7-XPUF Implementation**
98.18%	97.09%	96.70%

4.2.2 Correctness

The correctness scores of the referenced 32-bit CDC-7-XPUF in [43], our implemented 32-bit CDC-7-XPUF, and 64-bit CDC-7-XPUF are presented in Table 4.2. In Equation 2.15, the correctness score is calculated between 0 and 1. Hence, we normalize it using percentages to calculate the score in the table.

Our result for the 32-bit is only slightly worse than the reference design in an acceptable range. Also, the 64-bit result is slightly worse than the 32-bit results. As it can be seen in [43], increasing the stage number has a negative effect on the correctness.

Table 4.2: Correctness Results

Correctness Score of Referenced Work 32-bit CDC-7-XPUF [43]	**Correctness Score of 32-bit CDC-7-XPUF Implementation**	**Correctness Score of 64-bit CDC-7-XPUF Implementation**
97.63%	96.64%	96.19%

4.2.3 Diffuseness

The diffuseness scores of the referenced 32-bit CDC-7-XPUF in [43], our implemented 32-bit CDC-7-XPUF, and 64-bit CDC-7-XPUF are presented in Table 4.3. In Equation 2.16, the diffuseness score is calculated between 0 and 1. Hence, we normalize it using percentages to calculate the score in the table. For the diffusion calculation, "Correct ID"s are used.

Our result for the 32-bit is only slightly better than the reference design. Also, the 64-bit result is slightly better than the 32-bit results. As it can be seen in [43], increasing the stage number has a positive effect on the diffuseness.

Table 4.3: Diffuseness Results

Diffuseness Score of Referenced Work 32-bit CDC-7-XPUF [43]	**Diffuseness Score of 32-bit CDC-7-XPUF Implementation**	**Diffuseness Score of 64-bit CDC-7-XPUF Implementation**
99.90%	99.96%	99.99%

4.2.4 Uniformity

The uniformity scores of the referenced 32-bit CDC-7-XPUF in [43], our implemented 32-bit CDC-7-XPUF, and 64-bit CDC-7-XPUF are presented in Table 4.4. In Equation 2.17, the uniformity score is calculated between 0 and 1, whose expected score is 0.5. Hence, we normalize it using percentages to calculate the score in the table. For the uniformity calculation, "Correct ID"s are used. In [43], the results of the uniformity are not direct, yet they can be inferred from the results of the randomness, as it can be seen from Equations 2.17, 2.18, and 2.19. However, the result of this inference is ambiguous. Since it can not be known that p or $1-p$ is greater, the result in Table 4.4 for the referenced 32-bit work can be 49.60% also. But it is not important. Because, in terms of uniformity, the proximity of the value to 50% is important, not the percentage value. Hence, in terms of proximity, 50.40% and 49.60% are the same. Consequently, that approach is applied to the comparison in the following paragraph.

Our result for the 32-bit is only slightly worse than the reference design. However, the 64-bit result is slightly better than the result of our 32-bit design and the referenced 32-bit design. Although, as it can be seen in [43], increasing the stage number has a negative effect on the uniformity, in our case, it increased the uniformity.

Table 4.4: Uniformity Results

Uniformity Score of Referenced Work 32-bit CDC-7-XPUF [43]	**Uniformity Score of 32-bit CDC-7-XPUF Implementation**	**Uniformity Score of 64-bit CDC-7-XPUF Implementation**
50.40%	50.94%	49.89%

4.2.5 Uniqueness

The uniqueness scores of the referenced 32-bit CDC-7-XPUF in [43], our implemented 32-bit CDC-7-XPUF, and 64-bit CDC-7-XPUF are presented in Table 4.5. In Equation 2.20, the uniqueness score is calculated between 0 and 1. Hence, we normalize it using percentages to calculate the score in the table. For the uniqueness calculation, "Correct ID"s are used.

Our result for the 32-bit is only slightly better than the reference design. Also, the 64-

bit result is slightly better than the 32-bit results. As it can be seen in [43], increasing the stage number has a positive effect on the uniqueness.

Table 4.5: Uniqueness Results

Uniqueness Score of Referenced Work 32-bit CDC-7-XPUF [43]	**Uniqueness Score of 32-bit CDC-7-XPUF Implementation**	**Uniqueness Score of 64-bit CDC-7-XPUF Implementation**
17.90%	18.06%	18.96%

4.2.6 Utilization Results of CDC-7-XPUFs in Zynq-7020 SoC FPGA

For the implementation, we use state machines in the PL part so that we can take the challenges from the PS part, and we can send responses derived from these challenges through the Dual Access BRAM. Although the PL part consists of not only CDC-7-XPUFs but also state machines which are necessary for the implementation, the utilization rate is relatively low, as it can be seen in Table 4.6 for both 32-bit and 64-bit CDC-7-XPUF implementations.

As expected, the 64-bit design has a higher utilization rate, especially in DSP resources. In order to generate different challenges for each of the streams, we use PRNGs, which multiply 64-bit numbers requiring more DSP than 32-bit design. That relatively low utilization result makes 64-bit CDC-7-XPUF a promising candidate for applications that require a PUF.

Table 4.6: Utilization Table Generated Using Vivado 2019.1 [5] for 32-bit and 64-bit CDC-7-XPUF Implementations

Resource Type	**Avaliable Resource Quantity**	**Utilization Quantity (Utilization Rate as %) of 32-bit CDC-7-XPUF**	**Utilization Quantity (Utilization Rate as %) of 64-bit CDC-7-XPUF**
LUT	53200	1500 (2.82%)	1740 (3.27%)
LUTRAM	17400	72 (0.41%)	72 (0.41%)
FF	106400	1781 (1.67%)	1933 (1.82%)
BRAM	140	2 (1.43%)	2 (1.43%)
DSP	220	12 (5.45%)	68 (30.91%)
IO	200	8 (4.00%)	4 (100.00%)

4.2.7 Discussion About CDC-7-XPUF Implementation Results

We have thoroughly examined the resilience against machine learning attacks in Section 2.4.2.1. As discussed in this section and demonstrated in [43], the 64-bit CDC-XPUF designs with 7 streams are resistant to machine learning attacks.

In the following part, evaluation criteria are examined. The PL part of Zynq 7020 shown in Figure 2.26 has a very similar architecture to Artix-7, which is used to implement the reference design of CDC-XPUF in [43]. Hence, we expected similar results found in [43], and, as expected, we observed similar results as can be seen in the previous sections.

In the last part, the utilization rate of both 32-bit and 64-bit designs are examined, and it is shown that both designs are suitable for an IoT application since they provide a lot of space in the PL part.

CHAPTER 5

A COMBINED DESIGN OF 4-PLL-TRNG AND 64-BIT CDC-7-XPUF ON A ZYNQ-7020 SOC

In this chapter, after a brief introduction section, the combined design of the 4-PLL-TRNG and CDC-7-XPUF is explained, and the implementation results on Xilinx Zynq-7000 SoC ZC702HW-Z7-ZC702 Rev1.1 Evaluation Board are presented. The findings from this part of the research are detailed in [61].

5.1 Introduction

The foundational concepts of PLL-TRNG and CDC-XPUF are introduced in Chapter 2, with detailed implementation discussions provided in Chapter 3 and Chapter 4. Therefore, a comprehensive understanding of these three chapters is essential for grasping the combined design presented in this chapter.

As shown in Figure 2.24 in Section 2.3, TRNGs and PUFs are two basic hardware primitives in the security of IoT systems. They can be implemented separately in an IoT device. However, if they are implemented in a combined way so that the overall security qualifications of the device are improved, it would be efficient, which will ensure effective utilization of the computational resources of the device. Based on this principle of working in combination, we have worked on a design in which these two hardware primitives are both related to each other and also can work separately.

In the following section, details of the implementation of the combined design are explained.

5.2 Implementation Details 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 Figure 3.1 as (d) proposed design in Section 3.1. In the combined design, this 4-PLL-TRNG is used as it is and has its own BRAM block to write random numbers. There are three different configurations of PLL-TRNGs in this work, as explained in Section 3.1.1. 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 3.3 for PLL configuration.

In the PUF part, in order to differentiate the challenges, a new approach is applied instead of PRNGs. The results in [59] 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 [59] 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 [59], we expect resistance to ML attacks.

The block diagram of this combined design is shown in Figure 5.1. 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 Interface (AXI) bus. Additionally, memory buffers may be incorporated into the design to accommodate the communication speeds of the interacting units.

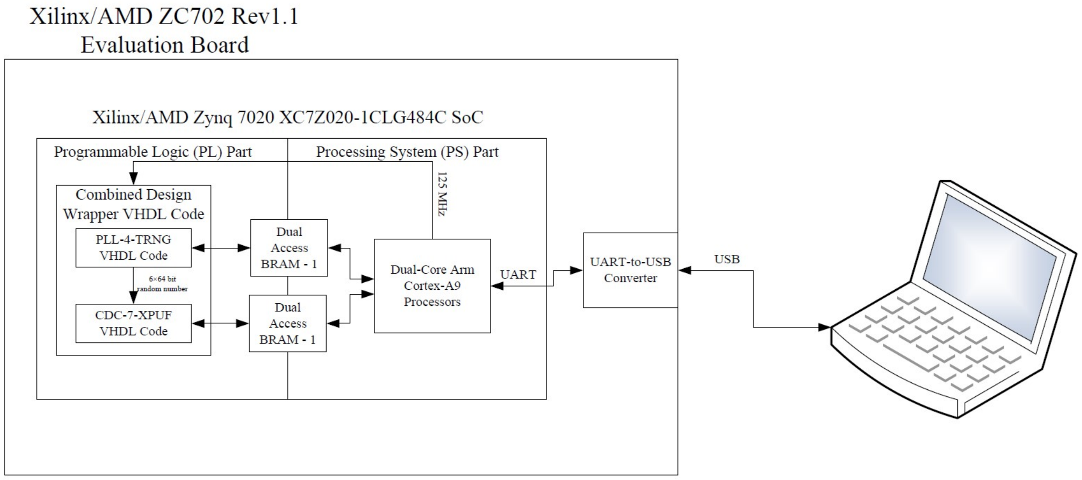

Figure 5.1: Block Diagram of Implementation Setup of the Combined Design of 4-PLL-TRNG and CDC-7-XPUF

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 Figure 5.1. 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 Figure 5.1. It utilizes the ZC702 Rev1.1 Evaluation Board [7], 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 [5] in VHDL [31]. 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 [33]. 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 [12]. The codes of AIS-20/31 Tests in [12] were compiled using [22] 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 4.1, 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 part, test codes are prepared in Python [51] and compiled using Microsoft Studio 2022 [42]. 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 [50] terminal and the SoCs.

In the setup in Figure 5.1, 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.

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

As explained in the previous section, a total of three different test configurations are examined in this section. These are:

1. 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. This configuration is named in the tables of results as **Combined Design (a)**.

2. In the second test setup, TRNG works continuously, and after the first run, only generated responses by PUF are recorded. This configuration is named in the tables of results as **Combined Design (b)**.
3. In the third test setup, PUF works continuously, and after the first run, only generated random numbers by TRNG are recorded. This configuration is named in the tables of results as **Combined Design (c)**.

5.3.1 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 5.1.

Table 5.1: 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

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

Table 5.2: 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

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

5.3.2.1 The Steadiness Results of the Combined Designs

The steadiness scores of the combined designs and, for reference, the discrete implementation of 64-bit CDC-7-XPUF are presented in Table 5.3. As expected, the steadiness scores of the referenced and the combined designs are very close to each other.

Table 5.3: 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%

5.3.2.2 The Correctness Results of the Combined Designs

The correctness scores of the combined designs and, for reference, the discrete implementation of 64-bit CDC-7-XPUF are presented in Table 5.4. As expected, the correctness scores of the referenced and the combined designs are very close to each other.

Table 5.4: 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%

5.3.2.3 The Diffuseness Results of the Combined Designs

The diffuseness scores of the combined designs and, for reference, the discrete implementation of 64-bit CDC-7-XPUF are presented in Table 5.5. It is expected that the diffuseness scores of the referenced and the combined designs are very close, but they have the same score value.

Table 5.5: 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%

5.3.2.4 The Uniformity Results of the Combined Designs

The uniformity scores of the combined designs and, for reference, the discrete implementation of 64-bit CDC-7-XPUF are presented in Table 5.6. As expected, the uniformity scores of the referenced and the combined designs are very close to each other.

Table 5.6: 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%

5.3.2.5 The Uniqueness Results of the Combined Designs

The uniqueness scores of the combined designs and, for reference, the discrete implementation of 64-bit CDC-7-XPUF are presented in Table 5.7. Significant improvements in the ’Uniqueness’ parameter have been observed due to the random numbers generated by the 4-PLL-TRNG for the production of challenges other than the main challenge.

Table 5.7: 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%

5.3.3 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 5.8. 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 includes the discrete 4-PLL-TRNG and 64-bit CDC-7-XPUF implementations.

Table 5.8: 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)

5.4 Discussion About Combined Designs Implementation Results

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 Figure 5.1.

Starting with the evaluation of the random number generation results in this combined structure, as in Chapter 3, the random numbers were assessed using AIS-20/31 tests, and their Shannon entropies are calculated using the formula in Equation 2.13. For these evaluations, the reference 4-PLL-TRNG (with the max. R configuration) from Chapter 3 is used alongside Combined Design (a) and Combined Design (c). Tables 5.1 and 5.2 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 issue in utilizing the combined design in terms of random number generation.

The evaluation metrics for PUFs are thoroughly examined in Section 2.4.2. These metrics include resilience against ML attacks, steadiness, correctness, diffuseness, uniformity, and uniqueness. For these evaluations, the reference 64-bit CDC-7-XPUF from Chapter 4 is used in conjunction with Combined Design (a) and Combined Design (b).

The first metric to evaluate is the resilience of the PUF against ML attacks. As discussed in Section 5.2 and mentioned in [59], it is expected that the process of generating new challenges by XORing the main challenge with random numbers produced by the proposed 4-PLL-TRNG, instead of using the fix-point-free transformation, provides resistance to ML attacks.

The second metric in the PUF evaluation is steadiness. Our expectation for this metric is to obtain results similar to those of the independently implemented reference design. As shown in Table 5.3, very similar results are observed. Therefore, the combined design is appropriate in terms of steadiness.

The third metric in the PUF evaluation is correctness. Similar to steadiness, we expect to achieve results comparable to those of the independently implemented reference design. As seen in Table 5.4, very similar results were observed, indicating that the combined design is also suitable in terms of correctness.

The fourth metric in the PUF evaluation is diffuseness. Again, our expectation for this metric is to achieve results similar to those of the independently implemented reference design. The reference value is already 99.99%. As shown in Table 5.5, the same results are observed. Therefore, the combined design is appropriate in terms of diffuseness.

The fifth metric in the PUF evaluation is uniformity. Our expectation is to achieve results comparable to those of the independently implemented reference design. As shown in Table 5.6, very similar results are observed. Therefore, the combined design is appropriate in terms of uniformity.

The final metric in the PUF evaluation is uniqueness. In this metric, our expectation is to achieve slightly better results than the independently implemented reference designs, as we anticipate that XORing would produce better challenges compared to a PRNG. As shown in Table 5.7, significantly better results are observed. Therefore, the combined design is appropriate in terms of uniqueness.

Lastly, we compared the utilization rate of the separate designs with the combined designs in Table 5.8. 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 5.8, 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.

In conclusion, when the combined design is evaluated in terms of the TRNG and PUF metrics, it is clear that the combined design is highly suitable for use in IoT systems. The analysis and tests conducted in this chapter have demonstrated that the combined design retains all the features of the separately designed 4-PLL-TRNG and 64-bit CDC-7-XPUF.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

In this chapter, the conclusion of the research and the future works of this thesis are presented.

6.1 Conclusion

In this research, as the first part, we delineated the design and implementation procedures of an innovative and fast PLL-TRNG utilizing the coherent sampling method of jittered PLL clocks. For parameter selection, we employ the backtracking algorithm [16]. Unlike conventional designs, which typically incorporate two PLLs, our approach leverages four PLLs to enhance the output bit rate. The choice of four PLLs is constrained by the Xilinx Zynq 7020 SoC, which accommodates exactly four PLLs. Nevertheless, the methodology illustrated in Figure 3.1 is adaptable to any FPGA or SoC platform. This flexibility ensures the broad applicability of our approach across various hardware configurations. We show that our proposed methods can generate random numbers with AIS-20/31 compliance. With their excellent results compared to the previous works, it can be concluded that our proposed method is promising. Also, the resource utilization is very low, except for PLLs, since we use all of them to increase the output bit rate, as shown. The outcome of this TRNG part is detailed in [62].

In the second part, we implemented 32-bit and 64-bit CDC-7-XPUFs for PUF applications, building on the improved Arbiter PUF design [26]. This implementation thoroughly examines resilience against ML attacks, as outlined in Section 2.4.2.1, and supported by [43], showing that 64-bit CDC-XPUF with seven streams is resistant to such attacks. Metrics such as steadiness, correctness, diffuseness, uniformity, and uniqueness were used to evaluate the PUF results. To ensure a meaningful comparison, we first implemented the 32-bit design, followed by the 64-bit design for enhanced ML resilience. Similar to the TRNG section, the utilization rates of both designs were assessed, demonstrating their suitability for IoT applications due to their efficient use of space in the PL part. The results of this PUF implementation are presented in [60].

In the last part of the thesis, 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 [62] and [60], 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. The output of this PUF part is presented in [61].

6.2 Future Works

As a future work, the following items can be listed:

- In addition to AIS-20/31 tests, NIST SP 800-90B [57] test suite would also be applied.
- 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. Hence, this design would be tested for these other platforms.

REFERENCES

[1] M. A. Alamro, K. T. Mursi, Y. Zhuang, A. O. Aseeri, and M. S. Alkatheiri, Robustness and Unpredictability for Double Arbiter PUFs on Silicon Data: Performance Evaluation and Modeling Accuracy, Electronics, 9(5), 2020, ISSN 2079-9292, `https://www.mdpi.com/2079-9292/9/5/870`.

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

[3] Altera, Understanding Metastability in FPGAs, White Paper WP-01082-1.2, July 2009, `https://www.intel.com/content/www/us/en/content-details/650346/understanding-metastability-in-fpgas.html`, accessed: 2023-08-01.

[4] AMD, 7 Series FPGAs Clocking Resources User Guide (UG472) (v1.14), `https://docs.amd.com/v/u/en-US/ug472_7Series_Clocking`, accessed: 2023-10-01.

[5] AMD, 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: 2023-09-01.

[6] AMD, Xilinx Zynq-7000 SoC Block Scheme, `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`, accessed: 2023-09-01.

[7] AMD, Xilinx Zynq-7000 SoC ZC702 Evaluation Kit, `https://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html`, accessed: 2023-09-01.

[8] AMD, 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: 2023-10-01.

[9] N. N. Anandakumar, M. Hashmi, and M. Tehranipoor, FPGA-based Physical Unclonable Functions: A comprehensive overview of theory and architectures,

Integration, 81, 07 2021, `https://doi.org/10.1016/j.vlsi.2021.06.001`.

[10] A. O. Aseeri, Y. Zhuang, and 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)*, pp. 49–56, 2018, `http://dx.doi.org/10.1109/ICIOT.2018.00014`.

[11] M. Baudet, D. Lubicz, J. Micolod, and A. Tassiaux, On the Security of Oscillator-Based Random Number Generators, Journal of Cryptology, 24(2), pp. 398–425, 2011, `https://doi.org/10.1007/s00145-010-9089-3`.

[12] Bundesamt für Sicherheit in der Informationstechnik (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: 2023-09-01.

[13] Bundesamt 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.

[14] Y. Cao, W. Liu, L. Qin, B. Liu, S. Chen, J. Ye, X. Xia, and C. Wang, Entropy Sources Based on Silicon Chips: True Random Number Generator and Physical Unclonable Function, Entropy, 24(11), 2022, ISSN 1099-4300, `https://www.mdpi.com/1099-4300/24/11/1566`.

[15] A. Cherkaoui, V. Fischer, A. Aubert, and L. Fesquet, A Self-Timed Ring Based True Random Number Generator, in *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems*, pp. 99–106, 2013, `https://doi.org/10.1109/ASYNC.2013.15`.

[16] B. Colombier, N. Bochard, F. Bernard, and 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*, DATE '20, p. 1–6, EDA Consortium, San Jose, CA, USA, 2020, ISBN 9783981926347, `https://doi.org/10.23919/DATE48585.2020.9116307`.

[17] B. Colombier, N. Bochard, F. Bernard, and L. Bossuet, The source code of the backtracking algorithm in [16], `https://gitlab.univ-st-etienne.fr/sesam/pll-trng-constraint-programming/tree/master`, 2020, accessed: 2023-11-01.

[18] L. Crocetti, P. Nannipieri, S. Di Matteo, L. Fanucci, and S. Saponara, Review of Methodologies and Metrics for Assessing the Quality of Random Number Generators, Electronics, 12(3), 2023, ISSN 2079-9292, `https://www.mdpi.com/2079-9292/12/3/723`.

[19] N. D. Dalt and A. Sheikholeslami, *Understanding Jitter and Phase Noise: A Circuits and Systems Perspective*, Cambridge University Press, USA, 1st edition, 2018, ISBN 1107188571.

[20] M. Deutschmann, S. Lattecher, J. Delvaux, V. Rozic, B. Yang, D. Singelee, L. Bossuet, V. Fischer, U. Mureddu, O. Petura, A. A. Yamajako, B. Kasser, and G. Battum, Hardware Enabled Crypto and Randomness (HECTOR) d2.1 - Report on Selected TRNG and PUF Principles, February 2016, `https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5a6bd305c&appId=PPGMS` accessed: 2023-07-01.

[21] M. Ebrahimabadi, M. Younis, W. Lalouani, and N. Karimi, A Novel Modeling-Attack Resilient Arbiter-PUF Design, in *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*, pp. 123–128, 2021, `https://doi.org/10.1109/VLSID51830.2021.00026`.

[22] Eclipse Foundation, *Eclipse IDE for Java Developers - 2023-12*, version 2023-12 `https://www.eclipse.org/downloads/packages/` accessed: 2024-01-15.

[23] V. Fischer, F. Bernard, and N. Bochard, Modern random number generator design – Case study on a secured PLL-based TRNG, it - Information Technology, 61(1), pp. 3–13, 2019, `https://doi.org/10.1515/itit-2018-0025`.

[24] V. Fischer and M. Drutarovský, True Random Number Generator Embedded in Reconfigurable Hardware, in B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pp. 415–430, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ISBN 978-3-540-36400-9, `https://doi.org/10.1007/3-540-36400-5_30`.

[25] V. Fischer, M. Drutarovský, M. Simka, and N. Bochard, High Performance True Random Number Generator in Altera Stratix FPLDs, volume 3203, pp. 555–564, 08 2004, ISBN 978-3-540-22989-6, `https://doi.org/10.1007/978-3-540-30117-2_57`.

[26] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, Silicon Physical Random Functions, in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS ’02, p. 148–160, Association for Computing

Machinery, New York, NY, USA, 2002, ISBN 1581136129, `https://doi.org/10.1145/586110.586132`.

[27] R. Helinski, Evaluating Physical Unclonable Functions, February 2020, `https://www.osti.gov/biblio/1766751` accessed: 2023-11-01.

[28] M. Hofer and C. Böhm, *Physical Unclonable Functions in Theory and Practice*, Springer New York, NY, 1st edition, 2012, `https://doi.org/10.1007/978-1-4614-5040-5`.

[29] F. Hooge, 1/*f* Noise Is No Surface Effect, Physics Letters A, 29(3), pp. 139–140, 1969, ISSN 0375-9601, `https://doi.org/10.1016/0375-9601(69)90076-0`.

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

[31] IEEE Computer Society, IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008, 2008, `https://standards.ieee.org/ieee/1076/3666/` accessed: 2023-07-01.

[32] N. Kasdin, Discrete Simulation of Colored Noise and Stochastic Processes and $1/f\alpha$ Power Law Noise Generation, Proceedings of the IEEE, 83(5), pp. 802–827, 1995, `https://doi.org/10.1109/5.381848`.

[33] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1988, ISBN 978-0131103627.

[34] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Addison-Wesley, Boston, 1997, ISBN 0201896842.

[35] Ç. K. Koç, *Cryptographic Engineering*, Springer, New York, 2009, `https://doi.org/10.1007/978-0-387-71817-0`.

[36] P. Kohlbrenner and K. Gaj, An Embedded True Random Number Generator for FPGAs, Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, 2004, `https://doi.org/10.1145/968280.968292`.

[37] G. Li, K. T. Mursi, A. O. Aseeri, M. S. Alkatheiri, and 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), pp. 1–15, 2022, `https://doi.org/10.5121/ijcnc.2022.14301`.

[38] D. Lim, J. Lee, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, Extracting Secret Keys From Integrated Circuits, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 13(10), pp. 1200–1205, 2005, `https://doi.org/10.1109/TVLSI.2005.859470`.

[39] C. Liu, *Jitter in Oscillators with 1/f Noise Sources and Application to True RNG for Cryptography*, Ph.D. Thesis, Worcester Polytechnic Institute, Jan 2006, `https://digital.wpi.edu/show/sj1392036`.

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

[41] A. L. McWhorter, 1/f Noise and Germanium Surface Properties, in *Semiconductor Surface Physics*, p. 207–228, Philadelphia, PA: Univ. Pennsylvania Press, 1957.

[42] Microsoft Corporation, Visual Studio 2022, 2022, `https://visualstudio.microsoft.com/` Accessed: 2023-11-01.

[43] K. T. Mursi, *From XOR PUF to CDC XOR PUF: Cost-Effectiveness, Statistical Characteristics, and Security Assessment*, Ph.D. Thesis, Texas Tech University, 2021, `https://ttu-ir.tdl.org/bitstreams/d26a326c-9494-47e4-a9e2-9af57d949e88/download`.

[44] K. T. Mursi, Y. Zhuang, M. S. Alkatheiri, and 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)*, pp. 1–9, 2019, `http://dx.doi.org/10.1109/PST47121.2019.8949070`.

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

[46] O. Petura, *True Random Number Generators for Cryptography : Design, Securing and Evaluation*, Ph.D. Thesis, Université de Lyon, October 2019, `https://theses.hal.science/tel-02895861`.

[47] O. Petura, U. Mureddu, N. Bochard, and V. Fischer, Optimization of the PLL Based TRNG Design Using the Genetic Algorithm, in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017, `https://doi.org/10.1109/ISCAS.2017.8050839`.

[48] O. Petura, U. Mureddu, N. Bochard, V. Fischer, and L. Bossuet, A Survey of AIS-20/31 Compliant TRNG Cores Suitable for FPGA Devices, in *2016*

26th International Conference on Field Programmable Logic and Applications (FPL), pp. 1–10, 2016, `https://doi.org/10.1109/FPL.2016.7577379`.

[49] K. Pratihar, U. Chatterjee, M. Alam, R. S. Chakraborty, and 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), pp. 1636–1651, 2023, `https://doi.org/10.1109/TC.2022.3218986`.

[50] PuTTY, PuTTY - a free and open-source terminal emulator, serial console, and network file transfer application, `https://www.putty.org/`, accessed: 2023-09-01.

[51] Python Software Foundation, *Python Language Reference, version 3.x*, `https://www.python.org/` accessed: 2023-11-01.

[52] L. Reyneri, D. Del Corso, and B. Sacco, Oscillatory Metastability in Homogeneous and Inhomogeneous Flip-Flops, IEEE Journal of Solid-State Circuits, 25(1), pp. 254–264, 1990, `https://doi.org/10.1109/4.50312`.

[53] V. Rozic, B. Yang, W. Dehaene, and I. Verbauwhede, Highly Efficient Entropy Extraction for True Random Number Generators on FPGAs, in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015, `https://doi.org/10.1145/2744769.2744852`.

[54] M. B. R. Srinivas and K. Elango, Era of Sentinel Tech: Charting Hardware Security Landscapes Through Post-Silicon Innovation, Threat Mitigation and Future Trajectories, IEEE Access, 12, pp. 68061–68108, 2024, `http://dx.doi.org/10.1109/ACCESS.2024.3400624`.

[55] G. E. Suh and S. Devadas, Physical Unclonable Functions for Device Authentication and Secret Key Generation, in *2007 44th ACM/IEEE Design Automation Conference*, pp. 9–14, 2007, `https://doi.org/10.1145/1278480.1278484`.

[56] B. Sunar, W. J. Martin, and D. R. Stinson, A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks, IEEE Transactions on Computers, 56(1), pp. 109–119, 2007, `https://doi.org/10.1109/TC.2007.250627`.

[57] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. L. Baish, and M. Boyle, Recommendation for the Entropy Sources Used for Random Bit Generation: NIST SP 800-90B, January 2018, `https://doi.org/10.6028/NIST.SP.800-90b`.

[58] M. Varchola and M. Drutarovsky, New High Entropy Element for FPGA Based True Random Number Generators, in S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pp. 351–365, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, ISBN 978-3-642-15031-9, `https://doi.org/10.1007/978-3-642-15031-9_24`.

[59] N. Wisiol, G. T. Becker, M. Margraf, T. A. A. Soroceanu, J. Tobisch, and B. Zengin, Breaking the Lightweight Secure PUF: Understanding the Relation of Input Transformations and Machine Learning Resistance, in S. Belaïd and T. Güneysu, editors, *Smart Card Research and Advanced Applications*, pp. 40–54, Springer International Publishing, Cham, 2020, ISBN 978-3-030-42068-0, `https://doi.org/10.1007/978-3-030-42068-0_3`.

[60] O. Yayla and Y. E. Yılmaz, 32-bit and 64-bit CDC-7-XPUF Implementation on a Zynq-7020 SoC, Cryptology ePrint Archive, Paper 2024/1443, 2024, `https://eprint.iacr.org/2024/1443`.

[61] O. Yayla and Y. E. Yılmaz, A Combined Design of 4-PLL-TRNG and 64-bit CDC-7-XPUF on a Zynq-7020 SoC, Cryptology ePrint Archive, Paper 2024/1457, 2024, `https://eprint.iacr.org/2024/1457`.

[62] O. Yayla and 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`.

[63] Y. Yu, *Design and Security Analysis of TRNGs and PUFs*, Ph.D. Thesis, KTH, Electronic and Embedded Systems, 2022, `https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-307501`.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Yılmaz, Yunus Emre

EDUCATION

Degree	Institution	Year of Graduation
M.S.	METU - Electrical & Electronics Engineering	2016
B.S.	METU - Electrical & Electronics Engineering	2011

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
Jan. 2015 - present	Aselsan Inc.	Digital Hardware Design Engineer
Dec. 2011 - Jan. 2015	Mikes Inc.	Digital Design Engineer

PUBLICATIONS

International Conference Publications

- B. Aksoy, Y. A. Bilgin, M. Cenk, M. B. İlter, N. Koçak, and Y. E. Yılmaz, Analyzing NIST 2nd-round Lattice-based Postquantum KEM Algorithms, *Information Security and Cryptology Conference*, Ankara, Turkey, 2019, `https://bilgiguvenligi.org.tr/BGD/ISCTurkey%202019%20Bildiriler%20Kitab%C4%B1.pdf`.

- O. Yayla and 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, *17th International Conference on Computational Intelligence in Security for Information Systems* Salamanca, Spain, 2024, `https://eprint.iacr.org/2024/1442`.
- O. Yayla and Y. E. Yılmaz, 32-bit and 64-bit CDC-7-XPUF Implementation on a Zynq-7020 SoC, Cryptology ePrint Archive, Paper 2024/1443, 2024, `https://eprint.iacr.org/2024/1443`.
- O. Yayla and Y. E. Yılmaz, A Combined Design of 4-PLL-TRNG and 64-bit CDC-7-XPUF on a Zynq-7020 SoC, Cryptology ePrint Archive, Paper 2024/1457, 2024, `https://eprint.iacr.org/2024/1457`.