KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

True Random Number Generators for FPGAs

Bohan Yang

Supervisor:
Prof. dr. ir. Ingrid Verbauwhede
Co-supervisor:
Prof. dr. ir. Nele Mentens

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

September 2018

True Random Number Generators for FPGAs

Bohan YANG

Examination committee:
Prof. dr. ir. Herman Neuckermans, chair
Prof. dr. ir. Ingrid Verbauwhede, supervisor
Prof. dr. ir. Nele Mentens, co-supervisor
Prof. dr. ir. Frederik Vercauteren
Prof. dr. ir. Marian Verhelst
Prof. dr. ir. Dirk Stroobandt
(Gent University, Belgium)
Prof. dr. Leibo Liu
(Tsinghua University, China)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

September 2018

Life is full of randomness.

Uitgegeven in eigen beheer, Bohan Yang, Kasteelpark Arenberg 10 box 2452, B-3001 Leuven (Belgium)

Preface

I like prefaces. I read them. Sometimes I do not read any further.
-Malcolm Lowry

If I had known how difficult to write this page, I would have thought twice before starting this long journey at the very beginning. After spending 21.875% of my current life in Leuven, I know it is impossible to name everyone that I have to express gratitude to.

In the first place, I would like to express my thanks to my supervisors, Prof. Ingrid Verbauwhede and Prof. Nele Mentens who have introduced me to the field of embedded security, and provided me with an unparalleled environment for doing research. I appreciate their guidance, support and numerous valuable discussions.

I would like to extend my gratitude to my assessors, Prof. Frederik Vercauteren and Prof. Marian Verhelst, other members of my jury, Prof. Dirk Stroobandt and Prof. Leibo Liu, for sacrificing their valuable time to sit on my examination committee and providing their remarks to help me to improve the quality of this thesis. I would like to thank Prof. Herman Neuckermans for kindly accepting to be the chairman of my examination committee.

My special thanks go to Vladimir Rožić for being not only an excellent co-author/officemate but also a friend indeed, throughout this "random" and fantastic journey. It is my luck and honor to work together with excellent researchers and to learn from them: Milos Grujić, Stjepan Picek, Josep Balasch, Danilo Šijačić, Eduard Marin, Dave Singelée, Pieter Maene, Bart Preneel, Begül Bilgin, Vincent Rijmen, Tomer Ashur, Ruan De Clercq, Jeroen Delvaux, Jo Vliegen. I wish to thank Benedikt Gierlichs for his much advice and many discussions. I would like to thank Lennert Wouters, Jan-Pieter D'Anvers, Alan Szepieniec, Arthur Beckers for Dutch translations.

COSIC is an ideal place to work. I would like to thank all my former and

current colleagues, who made this journey so remarkable: Anshuman Karmakar, Anthony Van Herrewege, Antoon Bosselaers, Aysajan Abidin, Carl Bootland, Claudia Diaz, Danny De Cock, Daniel Cozzo, Dragos Rotaru, Dušan Božilov, Eleftheria Makri, Enrique Argones Rúa, Fatemeh Shirazi, Filipe Beato, Furkan Turan, Gunes Acar, Iraklis Symeonidis, Jens Hermans, John Kelsey, Kerem Varici, Lauren De Meyer, Marc Juarez, Mustafa Mustafa, Nicky Mouha, Oscar Repáraz, Rafa Galvez, Roel Peeters, Sara Cleemput, Sujoy Sinha Roy, Svetla Petkova-Nikova, Thomas De Cnudde, Victor Arribas Abril, Wouter Castryck, Yulong Chen.

Big thanks go to Péla Noë, for being a versatile colleague, a warmhearted friend and the best neighbor that I could have in COSIC. I also thank Wim Devroye, Saartje Verheyen, Elsy Vermoesen and Dana Brouckmans for handling practicalities.

I thank the China Scholarship Council (CSC, No.201206210295), the Horizon 2020 HECTOR (644052) and Cathedral ERC Advanced Grant (695305), for funding my PhD research at COSIC.

可能是怕过于陷入回忆，大家才往往选择最后来写致谢这一章节。回想起在异国他乡求学的这段人生旅程，各种片段宛如昨日。衷心感谢以下的这些朋友们，让我在这座小城中的生活美好而丰富：黄晓霖夫妇、刘翼鹏夫妇、祝鹏夫妇、张韧夫妇、李超云夫妇、孙兵夫妇、贝荷西夫妇、莊愷莘、刘韵雯、王庆菊、程季秋、许瀚涛、孔令辉、贾旭、陈轩黎、郑学智、马忠坤和王晖。感谢樊俊峰师兄在我博士期间的建议。感谢张文涛老师一直以来的谆谆教导。

感谢我亲爱的父母和岳父母一直以来对我的理解与宽容。由于过去数年博士阶段的学习研究，没能常陪伴于我的妻子袁冰玉身边，这是我内心最为愧疚的事情。感谢她给予我的所有爱与支持，毕竟这是一切的因，也终将是一切的果。愿今后我们真能如同梁上燕一般，岁岁长相见。行文至此，忽然理解了陈之藩先生《谢天》中的一句话："即是无论什么事，得之于人者太多，出之于己者太少。因为需要感谢的人太多了，就感谢天吧。"

If time machine existed, I would have travelled back to the starting point of this long journey and told the young man: "Don't be scared, it will be full of randomness and fun."

Bohan Yang 杨博翰
Leuven, September 2018

Abstract

True randomness is all about unpredictability, which can neither be qualified nor quantified by examining statistics of a sequence of digits. Unpredictability is a property of random phenomena, which is measured in bits of information entropy. Application of randomness span from art to numerical computing and system security. Random numbers enable various cryptographic algorithms, protocols and secured implementations by providing secret keys, initialization vectors, random challenges and masks. As embedded electronics continue to be integrated into our daily lives, security becomes an indispensable requirement for an embedded system. Therefore, it is essential that unpredictable random numbers are available in secure embedded systems. However, designing hardware to harvest random numbers is a challenge, since most digital circuits are primarily developed to behave in a deterministic digital manner. Producing unpredictable output is usually undesired for an integrated circuit, and is sometimes regarded as a design failure. A True Random Number Generator (TRNG) circuit is designed to be sensitive to a particular physical phenomenon when it is in use, and to be resistant to process variations and other unwanted random physical phenomena.

The subject of this thesis is the study of TRNGs, which can be implemented on FPGA hardware. Different types of noise sources are reviewed, and an overview of selective TRNGs is given. Our contributions to TRNG designs include two novel digital noise sources, a method to measure timing jitter, two design methodologies for online tests and the exploration of implementation tradeoffs of one post-processing algorithm and statistical black-box tests.

Beknopte samenvatting

Werkelijke willekeur draait volledig rond onvoorspelbaarheid die niet kan worden gekwalificeerd noch gekwantificeerd door het bepalen van statistische eigenschappen van een reeks getallen. Onvoorspelbaarheid is een eigenschap van willekeurige verschijnselen, deze wordt gemeten in aantal bits informatie-entropie. Toepassingen van willekeurigheid reiken van kunst tot numerieke informatica, en systeembeveiliging. Willekeurige getallen maken verscheidene cryptografische algoritmen, protocollen en beveiligde implementaties mogelijk door het genereren van geheime sleutels, initialisatie vectoren, willekeurige uitdagingen en maskers. Geïntegreerde elektronische systemen worden steeds meer geïntroduceerd in ons dagelijkse leven. Hierdoor wordt beveiliging een onmisbare vereiste voor geïntegreerde elektronische systemen. Daarom is het essentieel dat beveiligde geïntegreerde elektronische systemen toegang hebben tot onvoorspelbare willekeurige getallen. Het ontwerp van hardware voor het genereren van willekeurige getallen is echter een uitdaging, vermits de meeste digitale schakelingen voornamelijk ontworpen zijn om zich deterministisch te gedragen. Het produceren van onvoorspelbare output is meestal ongewenst voor een geïntegreerd circuit en wordt dan aanzien als een mislukt ontwerp. Een werkelijk willekeurige getallengenerator (True Random Number Generator - TRNG) is ontworpen om gevoelig te zijn voor een fysiek fenomeen wanneer deze in gebruik is en om resistent te zijn tegen variaties in het productieproces en andere ongewenste willekeurige fysieke fenomenen.

Dit proefschrift handelt over TRNGs die kunnen worden geïmplementeerd in FPGA hardware. Verschillend types ruisbronnen worden beoordeeld en een overzicht van TRNGs wordt gegeven. Onze bijdragen aan het ontwerp van TRNGs omvatten twee nieuwe digitale ruisbronnen, een methode om timing jitter te meten, twee ontwerpmethoden voor online tests en de exploratie van implementatie afwegingen van een nabewerkingsalgoritme en statistische zwarte doos testen.

List of Abbreviations

ADC Analog-to-Digital Converter.

ALM adaptive logic module.

ASIC Application-Specific Integrated Circuit.

BRAM Block Memory.

BSI Federal Office for Information Security.

CASR Cellular Automata Shift Register.

CLB Configurable Logic Block.

CPU Central Processing Unit.

DC-TRNG Delay Chain based True Random Number Generator.

DFF D-type Flip-Flop.

DSP Digital Signal Processor.

EDA Electronic Design Automation.

EMFI Electro Magnetic Fault Injection.

ES-TRNG Edge Sampling based True Random Number Generator.

FPGA Field-Programmable Gate Array.

GE Gate Equivalent.

GPU Graphics Processing Unit.

HRNG Hybrid Random Number Generator.

HTRNG Hardware True Random Number Generator.

IID Independent and Identically Distributed.

IoE Internet of Everything.

IoT Internet of Things.

IVN iterated Von Neumann.

LFSR Linear Feedback Shift Register.

LSB Least Significant Bit.

LUT Look-Up Tables.

MAC Message Authentication Code.

NIST National Institute for Standards and Technology.

NPTRNG Non-Physical True Random Number Generator.

NVM non-volatile memory.

PLD Programmable Logic Device.

PLL Phase-locked loop.

PRNG Pseudo Random Number Generator.

PROM Programmable Read Only Memory.

PTRNG Physical Random Number Generator.

PUF Physical Unclonable Function.

QRNG Quantum Random Number Generator.

RAM Random Access Memory.

RNG Random Number Generator.

RO Ring Oscillator.

TDC Time-to-Digital Converter.

TERO Transition Effect Ring Oscillator.

TOTAL TRNG On-the-fly Tests for Attack detection using Lightweight hardware.

TRNG True Random Number Generator.

VCO Voltage-controlled oscillators.

Verilog HDL Verilog Hardware Description Language.

WDDL Wave Dynamic Differential Logic.

List of Symbols

α	Lower bound.
β	Upper bound.
ϵ	The bias from an ideal probability.
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ.
$\sigma(X)$	Standard deviation of a random variable X.
$\sigma^2(X)$	Variance of a random variable X.
$\varepsilon(t)$	Phase of an oscillator at time t.
$E(X)$	The expected value of the variable X.
$H_1(X)$	Shannon Entropy of a random variable X.
$H_\alpha(X)$	Renyi Entropy of a random variable X.
$H_\infty(X)$	Min-Entropy of a random variable X.
i_n	Noise current spectral density.
k_b	Boltzmann's constant.
$LSB(x)$	Least significant bit of digit x.
n_f	The order of the parity filter.
$Pr(A)$	Probability of an event A.
R	Resistor value.
T	Temperature.
T_0	Average period of the ring oscillator.

t_A Jitter accumulation time.

t_m Measurement time.

$t_{r/f,1/2}$ Rising delay and falling delay of stage 1 and 2.

t_{step} Step of the time-to-digital conversion

V_S Supply voltage.

Contents

Chapter 1

Introduction

We can only see a short distance ahead,
but we can see plenty there that needs to be done.

-Alan Turing

True randomness is all about unpredictability. Since prehistory, the concept of randomness was intertwined with fate and religion. This mystery was and still is utilized by human beings as a tool to hide secret from adversaries. The ancient representations of random numbers in hiding techniques were often an inadvertent selection or an arbitrary number, such as the diameter of a Scytale, the side length of a Polybius table or the number three for Caesar cipher. The security of the Caesar cipher relies on the unrealistic assumption that the rotating number three is unguessable by the enemies.

Back in the early 20th century, random numbers were generated using various methods for scientific works and for picking the winning numbers in lottery. The first table with random digits was published in [96], which consists of 40000 random digits taken at random from census reports. The first machine producing random numbers was built by Kendall and Babington Smith in 1938 [50], and 100000 digits were generated. The world's first commercially available general-purpose electronic computer, the Ferranti Mark 1 [113], has a miscellaneous special instruction denoted as symbol $/W$ for generating random numbers. Executing this single instruction requires 5.8 milliseconds and yields 20-digit random numbers. This instruction uses a resistance noise generator and this feature was recommended by A. M. Turing [100]. The RAND Corporation published a famous book [22] called *A million random digits with 100,000 normal deviates* in 1955, which was reissued in 2001. The trend of publishing random

digits in a book started in 1947 [12]. An electronic simulation of a roulette wheel was utilized and the results were carefully filtered and tested before being used to generate random digits. Random tables became popular again during the 1990s because of the advances in the storage and the input method. In 1995, George Marsaglia published a CD-ROM of random numbers [64], which also included the DIEHARD test suit.

The new era of computing and consumer electronics focuses on mobile computing, the Internet of Things (IoT) and even the Internet of Everything (IoE). People surround themselves with various electronics such as computers, smartphones, sensor nodes and actuators. Electronic devices are commonly connected via either a local net or the internet. The security of the communication of these devices becomes increasingly important. Considering the indispensable role of true random numbers in any of these systems, new challenges of designing generators with manifold design constraints emerge.

1.1 Securing the security system

According to the renowned Kerckhoffs' principle [51], a cryptographic system should be secure even if the attacker knows everything about the system, except the key. The key may refer to long-term keys, session keys for block ciphers or signature keys, ephemeral keys for public key algorithms. In modern computers and embedded systems, this key is usually generated by executing a True Random Number Generator (TRNG) in a trusted computing zone and is then stored in an embedded, non-volatile memory (NVM), which cannot be directly programmed or accessed by external instructions. This procedure is followed to mitigate potential attacks on the key storage.

Generating the key is not the only cryptographic application of random numbers. They are used for generating initialization vectors, one-time pads, challenges in authentication schemes and masks for countermeasures against side-channel attacks. In authentication schemes, the security relies on the assumption that the random numbers are unbiased, unpredictable and have no statistical flaws. TRNGs are also used by strong Physical Unclonable Function (PUF) obfuscation protocols [127]. Many countermeasures against side-channel attacks rely on the uniformity of the utilized random numbers and the assumption that the RNG implementation itself doesn't have side-channel leakage.

Implementing secure algorithms does not necessarily guarantee a secure implementation. Randomness sources are often the single point of failure in a security system. According to the work of Vanhoef *et al.* in 2016 [103], the group keys of WPA2/802.11 are compromised because its random number

generator is flawed by design and provides an undesirably low amount of entropy. In 2013, the study of Bernstein *et al.* [9] found compromised RSA keys from Taiwan's "Citizen Digital Certificate" database. This problem is caused by randomness-generation failures of the smart cards issued by the local government, although these smart cards have built-in hardware random number generators claimed to be FIPS 140-2 Level 2 compliant. Two independent studies were conducted in 2012 by Lenstra *et al.* [56] and Heninger *et al.*[44]. In [56], the computational properties of millions of public keys collected from the web were checked. Around 0.003% of these public keys were compromised in various ways. An implied reason is that random number generators used for generating public keys may not be properly seeded. In [44], 5.8 million unique TLS certificates from 12.8 million hosts and 6.1 million unique SSH host keys from 10.2 million hosts were collected for analysis. Based on their study of the collected dataset, 0.75% of TLS certificates share keys; the RSA private keys for 0.5% of TLS hosts and 0.03% of SSH hosts were obtained, as well as DSA private keys for 1.03% of SSH hosts. These keys are compromised due to insufficient entropy during key generation. In 2008, a vulnerability of the OpenSSL package from a Debian Linux Distribution was reported [90, 26]. The reason of this issue is that a line of code has been commented out to avoid warning signals. Removing that code cripples the seeding procedure of the OpenSSL Pseudo Random Number Generator (PRNG) and results in a scanty output space with only $32,768$ possible process IDs.

Unfortunately, designing a TRNG is not trivial and different from conventional digital circuit design. Instead of pursuing a stable and predictable behavior of the circuit, the TRNG design aims for a stable and robust unpredictability. Having mistakes or being careless at any step of the TRNG design and fabrication procedure may lead to insufficient entropy or/and a malfunctioned TRNG, such as an overestimated noise strength, an under-estimated process variation, an inappropriate assumption of attackers or even an upgrade of the photolithography process. TRNGs may utilize different physical random phenomena and have different design parameters on different platforms. Standardizing a universal TRNG that can be used for various applications on all platforms is infeasible. However, it is possible to regulate the design procedures, the evaluation methods and the constructions of TRNGs. A German standard AIS-31 [52] was released in 2011 with strict criteria for certifying true random number generators. The final version of the American standard NIST SP 800-90B [99] was published in 2018. It requires that the documentation of an TRNG should justify its entropy claim.

The work of this thesis focuses on the design and evaluation of secure true random number generators.

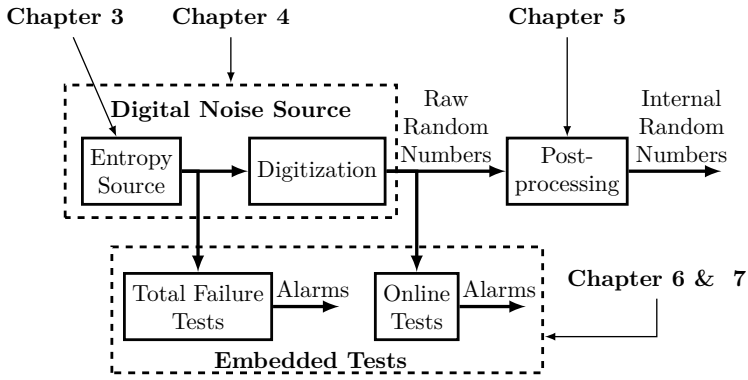


Figure 1.1: Thesis organization and link to TRNG architecture.

1.2 About this Thesis

This thesis deals with different design aspects of the TRNG. Most chapters can be mapped into particular components of the generic TRNG architecture as illustrated in Figure 1.1.

The remainder of this thesis consists of seven chapters. Their contributions are as follows:

- **Chapter 2: Background and Preliminaries.** In addition to introducing the terminology, this chapter also provides sufficient mathematical background to make this thesis self-contained. We also present the generic architecture of TRNGs, briefs of each component composing TRNGs and the requirements of TRNGs from two popular industrial standards. Each component of the TRNG, together with our contributions, is discussed in detail in the following chapters.

- **Chapter 3: Entropy Source.** At beginning of this chapter, we review existing entropy sources. Then, a detailed analysis of the timing jitter based entropy source is provided. As our main contribution to this chapter, we propose a novel method to measure the jitter strength of free-running ring oscillators implemented on Field-Programmable Gate Array (FPGA) for TRNGs. The proposed methodology is designed to measure the strength of the white noise while filtering out contributions from other noise sources such as flicker noise and power supply variations.

  Related publications:

[120], **Yang, B.**, Rožić, V., Grujić, M., Mentens, N., and Verbauwhede, I., *On-chip jitter measurement for true random number generators, In Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2017,* ***Best paper nomination****.*

- **Chapter 4: Digital Noise Source.** This chapter introduces the position of the digital noise source in the architecture of the TRNG. Then an overview of recent digital noise sources is presented with a special focus on timing jitter based TRNGs. At the end of this chapter, a novel TRNG benefiting from high sampling precision is presented and analyzed.

  Related publications:

  [83], Rožić, V., **Yang, B.**, Dehaene, W., and Verbauwhede, I., *Highly Efficient Entropy Extraction for True Random Number Generators on FPGAs, In 52nd Design Automation Conference (DAC), 2015.*

  [39], Grujić, M., Rožić, V., **Yang, B.**, and Verbauwhede, I., *A Closer Look at the Delay-Chain based TRNG, In IEEE International Symposium on Circuits and Systems (ISCAS), 2018.*

  [40], Grujić, M., **Yang, B.**, Rožić, V., and Verbauwhede, I., *Towards Inter-Vendor Compatibility of True Random Number Generators for FPGAs, In Design, Automation and Test in Europe (DATE), 2018.*

- **Chapter 5: Post-processing.** To start this chapter, an ancient oriental divination is analyzed as a post-processing algorithm. We present the difference between algorithmic post-processing and cryptographic post-processing. Pitfalls and common mistakes in designing and implementing post-processing modules are discussed. At the end of this chapter, an implementation tradeoff on iterated Von Neumann is presented.

  Related publications:

  [84], Rožić, V., **Yang, B.**, Dehaene, W., and Verbauwhede, I., *Iterating Von Neumann's Post-Processing under Hardware Constraints, In 9th IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2016.*

- **Chapter 6 and 7: Embedded tests.** These two chapters summarize our works on embedded tests. In Chapter 6, We first discuss various types of embedded tests. Total failure tests are introduced for the TRNG proposed in Chapter 4. Two conventional on-line test solutions based on the black box testing approach are discussed in Chapter 6. We would like to note that the black testing method is not recommended to be used for newly designed TRNGs, however they can be used for existing TRNGs without a stochastic model, or they can be used to build the statistical features pool for the embedded tests introduced in Chapter 7. Two novel methodologies to design on-line tests for TRNGs are introduced in Chapter 7. The first method is an empirical design procedure while the second one explores the testability of the entropy source and entropy extractor.

Related publications:

[123], **Yang, B.**, Rožić, V., Dehaene, W., Mentens, N., and Verbauwhede, I., *TOTAL: TRNG On-the-fly Testing for Attack detection using Lightweight hardware, In Design, Automation and Test in Europe (DATE), 2016,* ***Best paper nomination****.*

[85], Rožić, V., **Yang, B.**, Mentens, N., and Verbauwhede, I., *Canary Numbers: Design for Light-weight Online Testability of True Random Number Generators,In NIST RBG Workshop, 2016.*

[122], **Yang, B.**, Rožić, V., Mentens, N., Dehaene, W., and Verbauwhede, I., *Embedded HW/SW Platform for On-the-Fly Testing of True Random Number Generators, In Design, Automation and Test in Europe (DATE), 2015.*

[124], **Yang, B.**, Rožić, V., Mentens, N., and Verbauwhede, I., *On-the-Fly Tests for Non-Ideal True Random Number Generators, In IEEE International Symposium on Circuits and Systems (ISCAS), 2015.*

- **Chapter 8: A Design Example: ES-TRNG.** This chapter introduces and discusses our work on ES-TRNG: A high-throughput, low-area TRNG based on edge sampling. The proposed design is a culmination of our studies on TRNGs, where the high-resolution sampling, the non-linearity study of the delay chain and the stochastic model of ring oscillators are deployed to design the ES-TRNG.

  Related publications:

  [121], **Yang, B.**, Rožić, V., Grujić, M., Mentens, N., and Verbauwhede, I., *ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling, Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2018.*

1.3 Other Contributions

This thesis only includes a selection of our published works related to True Random Number Generator. In this section, we summarize our remaining contributions in groups according to their main topic.

CRYPTOGRAPHIC IMPLEMENTATIONS

RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms, presented by Zhang, Bao, Lin, Rijmen, Yang and Verbauwhede [128] in Science China Information Sciences. The work proposes a novel lightweight block cipher which is suitable for both hardware and software implementations. RECTANGLE uses an SP-network, in which the substitution layer consists of 16 4x4 S-boxes in parallel. RECTANGLE is extremely hardware-friendly. To meet the needs of different application scenarios, three types of hardware implementations are proposed, namely, round-based implementation,

serialized implementation and parallelized implementation. RECTANGLE also achieves a very competitive software speed due to its bit-slice style. New design criteria are introduced to select the S-box; this results in a good security-performance tradeoff. RECTANGLE remains unbroken.

PHYSICAL UNCLONABLE FUNCTIONS (PUF)

The Monte Carlo PUF, presented by Rozic, Yang, Vliegen, Mentens and Verbauwhede [86] in the International Conference on Field Programmable Logic and Applications. This work introduces a new design approach for implementing weak PUFs on FPGAs. On-chip Monte Carlo simulations are used to measure physical parameters of FPGA primitives. PUF bits can be generated by comparing the physical parameters of two identical primitives. Delays of the carry primitives are used in this work as a case study, due to its dedicated structure and no special requirement over the placement and routing.

SECURITY IN MEDICAL DEVICES

On the Feasibility of Cryptography for a Wireless Insulin Pump System, presented by Marin, Singelée, Yang, Verbauwhede and Preneel [61] in the ACM Conference on Data and Application Security and Privacy (CODASPY 2016). This work analyses the security and privacy properties of a widely used commercial insulin pump. The wireless communication between the insulin pump and its remote is eavesdropped to enable a reverse-engineer on the utilized protocol. Results shows that no standard cryptographic mechanisms are applied and the system is vulnerable to replay and message injection attacks. An AES-based solution is proposed to mitigate the security issue. The proposed solution is implemented on a 16-bit micro-controller and its security properties and energy requirements are evaluated.

Securing wireless neurostimulators, presented by Marin, Singelée, Yang, Volskiy, Vandenbosch, Nuttin and Preneel [62] in the ACM Conference on Data and Application Security and Privacy (CODASPY 2018). In this work, the protocol between a widely used commercial implanted neurostimulator and its programmer is reverse-engineered and analyzed. Results show that the communication over the air is neither encrypted nor authenticated, thus the safety of the patients is compromised. A new scheme is proposed for secure data exchange between the device programmer and the neurostimulator. We propose an experimental solution of using a patient's physiological signal as the entropy source for generating a symmetric key in the neurostimulator.

EVOLUTIONARY COMPUTATIONS FOR CRYPTOGRAPHIC IMPLEMENTATIONS

Evolutionary computation algorithms represent a group of problem-solving techniques inspired by principles of biological evolution. Such algorithms can be used to solve various difficult problems, among which are those from cryptographic applications. We applied different types of evolutionary computation algorithms to several selective problems of designing and implementing cryptographic primitives.

In the work of [76], genetic algorithms were utilized for heuristic searches, which look for cryptographic and implementation properties of affine invariant S-boxes of three sizes, namely, 4×4, 5×5 and 8×8. An updated work of searching for hardware-friendly 4×4 and 5×5 S-boxes was presented in [77]. Heuristics algorithms were compared to other search strategies, such as random search and affine transformations. In [74], a special kind of heuristics, genetic programming, was used to evolve a special type of S-boxes defined with cellular automata rules.

Evolutionary computations can be used to design PRNGs. In works [75] and [78], we presented design methods based on two forms of evolutionary algorithms, namely, Cartesian Genetic Programming and Genetic Algorithms. As a proof-of-concept, three hardware architectures of PRNGs were proposed to meet different scenarios.

Chapter 2

Background and Preliminaries

Random numbers should not be generated
with a method chosen at random.

-Donald Knuth

This chapter is organized as follows. Section 2.1 reviews the usage of random numbers in cryptographic applications. Section 2.2 introduces the classification and the terminology for random number generators. Section 2.3 provides the general architecture of a TRNG. Section 2.4 discusses the mathematical background for TRNGs and introduces the notation that will be used throughout this thesis. Section 2.5 provides a number of definitions applicable to TRNGs. Section 2.6 reviews the requirements and evaluation of TRNGs. Section 2.7 discusses post-processing techniques of TRNGs. Section 2.8 discusses online tests for TRNGs. Section 2.9 briefs an introduction of FPGAs and discussed special primitives used in this thesis. Section 2.10 concludes this chapter.

2.1 Random Numbers in Cryptography

TRNGs are essential building blocks of modern embedded security systems. They enable various cryptographic algorithms, protocols and secured implementations by providing secret keys, initialization vectors, random challenges and masks. The security of these applications relies on the uniformity and unpredictability of the utilized random numbers. A cause of failure in today's security systems is often traced back to a design flaw or an active attack on the used

TRNG [27, 9] rather than to a broken cryptographic algorithm or an unprotected implementation.

True randomness cannot be obtained via computational methods. Instead, physical phenomena such as noise in electronic devices should be responsible for the unpredictable nature of TRNGs. Due to their importance for security, TRNGs are subjected to strict evaluations in the process of industrial certification. The SP 800-90B [99], a special publication of the National Institute for Standards and Technology (NIST), contains requirements for the design and evaluation of TRNGs. According to this document, a theoretical rationale for the unpredictable behavior of the entropy source is required. A min-entropy estimation of the generated output and effective online tests are also mandatory. The German Federal Office for Information Security (BSI) standard has published a document for formal security evaluation of random number generators called AIS-31 [52]. The AIS-31 puts forward a stricter requirement for the design and the evaluation of TRNGs. Both the NIST and BSI standards will be discussed in Section 2.6.

2.2 Classification and Terminology of Random Number Generators

A lack of a unified terminology usually results in confusion. As indicated in Table 2.1, different standards favor various terminologies. Nevertheless, all standards categorize RNGs into two main types. To be consistent with our previous work, in this thesis, we name them TRNG and PRNG.

- *TRNGs* are also known as *Non-deterministic RNGs*. TRNGs produce true randomness based on unpredictable effects. The German BSI standard called AIS31 [52] classifies TRNGs into two groups based on the source of randomness. TRNGs are called Physical Random Number Generators (PTRNGs) or Hardware True Random Number Generator (HTRNG), if the randomness sources are implemented on hardware. The non-determinism of these TRNGs is rooted in unpredictable physical processes such as the timing jitter, the thermal noise or the final state of a metastable element. In contrast to HTRNGs, software TRNGs are denoted as Non-Physical True Random Number Generators (NPTRNGs). These generators are used in systems where dedicated hardware noise sources are not available. User behavior such as mouse movement and key strokes, and system data such as disk I/O and interrupts are used as entropy sources with random numbers at the output.

Table 2.1: Taxonomy of Random Number Generator (RNG) types in international standards and this thesis

NIST sp800-22 [87]	n/a	Random Number Generator (RNG)	Pseudorandom Number Generator (PRNG)
NIST sp800-90 [99]	Random Bit Generator (RBG)	Non-Deterministic RBG (NRBG)	Deterministic RBG (DRBG)
AIS31 [52]	Random Number Generator (RNG)	True Random Number Generator (TRNG)	Deterministic RNG (DRNG)
This thesis	**Random Number Generator (RNG)**	**True Random Number Generator (TRNG)**	**Pseudo Random Number Generator (PRNG)**

- *PRNGs* are deterministic algorithms that expand a short input sequence and produce a longer random-like sequence. PRNGs are also denoted as *Deterministic RNGs*. The required short input sequence is called the *seed*. The internal memory of PRNGs is called the *state*. The state is initialized by the seed and updated via a state transition function Φ. The output sequence is generated by applying an output function Ψ to the state. Since PRNGs are purely deterministic, they don't produce any new true randomness.

Academic literature and standards also denote the combined use of TRNGs and PRNGs as Hybrid Random Number Generators (HRNGs). In an HRNG, the state of the PRNG is initialized and constantly updated using the true random output of the TRNG.

In this thesis, we focus on physical true random number generators.

2.3 Generic Architecture of a TRNG

The generic architecture of a TRNG is shown in Figure 2.1. It consists of an entropy source, a digitization module, a post-processing module, a total failure

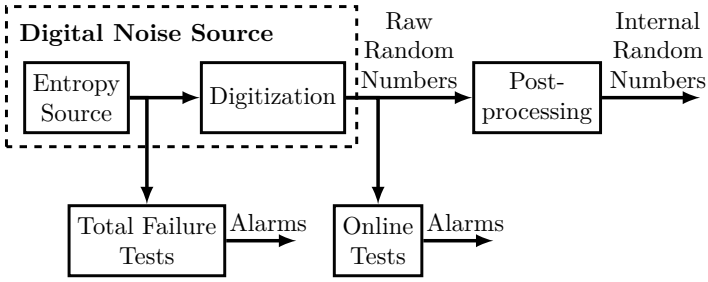


Figure 2.1: The generic architecture of a TRNG.

tests module and a online tests module.

The *entropy source* is the only component in this architecture with non-deterministic behavior. All true randomness is generated by the entropy source, in some cases in the form of analog signals. The *digitization* module is needed to convert these analog signals into a digital form. The combination of the entropy source and the digitization is referred to as the *digital noise source*. The produced outputs of the digital noise source are called the *raw random numbers*. The raw random numbers should be available for inner testability. Usually, the raw random numbers are subject to statistical defects, such as the bias from an ideal probability of ones and the auto-correlation between output bits. The *post-processing (conditioning)* module is applied to these non-ideal raw random numbers to improve their statistical and security characteristics. This module extracts and densifies the randomness from the raw random numbers. Post-processed numbers are called *internal random numbers*.

The digitization module and the post-processing module are completely deterministic and normally implemented as digital circuits. There is no new randomness generated by the digitization module and post-processing module. Some entropy is always lost in the conversion and extraction. Once the internal random numbers are outputted for various applications, they become the *external random numbers*.

The *online tests*, also known as *embedded tests* or *continuous tests*, are used to detect failures in generating raw random numbers. The *total failure tests* are implemented for the fast detection of the total breakdown of the entropy source.

2.4 Mathematical Background

In this section, we present the mathematical background of TRNGs. The mathematical background is the starting point to formally understand and theoretically analyze the TRNG.

2.4.1 Random Variables and Bias

At the core of a modern TRNG design procedure is the stochastic model. And the stochastic model relies on the formal descriptions of the entropy source based on probability theory. Normally, samples of the entropy source correspond to realization of a series of random variables representing the physical phenomena at the entropy source.

This section provides the definition of a random variable. The terminology used in Chapter 1 of [54] is followed here. The definition of a random variable requires the notations of a σ-algebra, a measurable function and a probability space.

Definition 1. A collection S of subsets of a set Ω is called a σ-algebra on the set Ω, if S is an algebra which is closed under countable unions. A σ-algebra has the following properties:

1. $\Omega \in S$.
2. $\forall C \in S \ : \ \Omega \backslash C \in S$.
3. $C_i \in S$ implies that $\cup_{i=1}^{n} C_i \in S$.

Informally, the definition of the σ-algebra is a *set-algebra* that is closed under countable unions. The definition of $set - algebra$ only requires it to be closed under finite unions.

Definition 2. A measurable space is a pair (Ω, S), where Ω is a space of elementary outcomes and S is a σ-algebra of subsets of Ω.

Definition 3. Let (Ω, S) be a measurable space. A function $\xi : \Omega \to \mathbb{R}$ is said to be S-measurable (or simply measureable) if $\omega : a \leq \xi(\omega) < b \in S$ for each $a, b \in \mathbb{R}$.

Definition 4. A measure Pr on a measurable space (Ω, S) is called a probability measure or a probability distribution, if $Pr(\Omega) = 1$.

Definition 5. A probability space is a triplet (Ω, S, Pr), where (Ω, S) is a measurable space and Pr is a probability measure. If $C \in S$, then the number $Pr(C)$ is called the probability of C.

Definition 6. A measurable function defined on a probability space is called a random variable. When the range of the random variable is finite or countable infinite, the random variable is called a discrete random variable.

An informal but intuitive way to comprehend the definition of a random variable is: a random variable, usually denoted as X, is a variable whose possible values are numerical outcomes of a random phenomenon.

There are two types of random variables, discrete and continuous. A discrete random variable has a specified finite or countable number of distinct values as its possible outcomes. A continuous random variable is not defined at a specific value but over an interval of values. A binary discrete random variable is a special type of discrete random variable which only has two possible outcomes.

Bias Bias is a measure of unbalance between the possible outcomes of the random variable. If the random variable is binary, the bias is the difference in probability between the outcome and 0.5.

Definition 7. Given a binary discrete random variable with outcomes $0, 1$, the definition of bias is :

$$\epsilon = |Pr(X = 0) - 0.5| = |Pr(X = 1) - 0.5|. \tag{2.1}$$

If the binary random variable is ideal, $\epsilon = 0$ or the random variable is unbiased.

2.4.2 Entropy

Entropy is a measure of the unpredictability. Before a realization of a random variable, the amount of entropy corresponds to the uncertainty of an attacker in predicting the outcomes. Even though there are various possible measures of entropy, two of them are widely used to evaluate TRNGs: Shannon entropy and min-entropy.

Using coherent mathematic theory to study communication, Shannon [92] initialized the research area called information theory. Later, Rényi [80] extended Shannon's measure of the amount of information to a general and continuous family of entropy measures.

Definition 8. Given a discrete random variable X with possible outcomes $\omega_1, \dots \omega_n$, the Rényi entropy is given by:

$$H_\alpha(X) = \frac{1}{1-\alpha} \log_2 \sum_{i=1}^{n} (Pr(X = \omega_i))^\alpha, \quad \alpha \in [0, 1) \cup (1, +\infty). \tag{2.2}$$

The well-known Shannon entropy can be interpreted as a special case of the Rényi entropy when $\alpha \to 1$.

Definition 9. Given a discrete random variable X with possible outcomes $\omega_1, \dots \omega_n$ and their probabilities of observation as $p_i = Pr(X = \omega_i)$, the Shannon entropy is given by:

$$H_1(X) = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i, \quad p_i > 0. \tag{2.3}$$

The Shannon entropy of a discrete random variable X is the amount of information obtained by observing X. When X is a binary discrete random variable and it has outcomes $0, 1$ with probabilities $p_0 \geq 0$ and $p_1 \geq 0$, the Shannon entropy $H_1(X)$ can be computed as:

$$H_1(X) = -p_0 \log_2 p_0 - p_1 \log_2 p_1. \tag{2.4}$$

Shannon entropy provides a theoretical limitation for compressing data. In other words, when performing a lossless compression, the size of the compressed data can not be smaller than the Shannon entropy of the original data.

The minimal amount of information provided by a single observation of the discrete random variable is called min-entropy. Min-entropy is another special case of the Rényi entropy when $\alpha \to \infty$.

Definition 10. Given a discrete random variable X with possible outcomes $\omega_1, \dots \omega_n$ and their probabilities of observation as $p_i = Pr(X = \omega_i)$, the min-entropy is given by:

$$H_\infty(X) = -\log_2 \max_{i \in 1,n} \{p_i\}, \quad p_i > 0. \tag{2.5}$$

If the discrete random variable X is binary, Equation (2.5) can be simplified as:

$$H_\infty(X) = -\log_2 \max\{p_0, p_1\}. \tag{2.6}$$

The min-entropy is the most conservative measure of the uncertainty, $H_\infty(X)$ bits of information are guaranteed for each observation of the random variable.

According to the NIST standard [99], min-entropy should be used as the metric of randomness for evaluating TRNGs.

If the discrete random variable X is uniformly distributed, i.e., $p_1 = p_2 \cdots = p_n = 1/n$, the Shannon entropy and the min-entropy of X are equal and reach their maximal value:

$$H_\infty(X) = H_1(X) = -\log_2 \frac{1}{n}. \tag{2.7}$$

If X is a non-uniformly distributed random variable, the following relation holds:

$$H_\infty(X) < H_1(X) < -\log_2 \frac{1}{n}. \tag{2.8}$$

2.4.3 Stochastic Model

A stochastic model is a mathematical description of an entropy source using random variables. The goal of having a stochastic model is to support the analysis of the corresponding entropy source. The evolution of the physical events in the noise source can be described as a stochastic process.

Definition 11. A stochastic process is a collection of random variables that are indexed by a totally ordered set T, $X_t : t \in T$. For example, a free-running ring-oscillator is commonly used as an entropy source. The jitter accumulation of the ring-oscillator can be modeled using a simple stochastic process with two parameters: the frequency of the ring-oscillator and the jitter strength.

Normally, a physical model, such as a transistor-level model, is a precise description of the physical process in the entropy source. However, it is not always feasible to construct a physical model of the entropy source. And when it is feasible, the physical model is often too complicated to be employed. A stochastic model is a simplification of a physical model with assumptions, which is better in terms of feasibility and usability. With a suitable stochastic model, a designer should be able to estimate the entropy of the raw random numbers.

2.5 Definition of the TRNG

In this section, we provide definitions for both *ideal TRNGs* and *non-ideal TRNGs*.

Definition 12 (Ideal TRNG). An ideal TRNG is a manufactured building block that realizes a discrete random variable X with a set of output samples.

Each query to the TRNG block is an observation on the realized random variable X and returns a response with a uniform probability on the sample set. By any means, an adversary cannot distinguish the response of an ideal TRNG from the queried result of a uniformly distributed random variable X_{ideal}. All relevant environmental parameters and aging are bounded, e.g., temperature $T \in [\alpha_T, \beta_T]$, supply voltage $V_S \in [\alpha_V, \beta_V]$, and an upper bound of the lifespan β_L.

Definition 12 relies on the indistinguishability between an ideal TRNG and a uniformly distributed random variable. Therefore, practical realizations of an ideal TRNG under Definition 12 are non-existent. However, this definition points out one modern principle for designing TRNGs: the abstract model of a TRNG should be linked with a random variable to enable the analysis of the unpredictability. We also give the definition of a non-ideal TRNG, which hopefully is more realistic to be applied.

Definition 13 (**Non-ideal TRNG**)**.** A non-ideal TRNG is a manufactured building block that realizes a discrete random variable X with a set of output samples. The unpredictability or the true randomness are quantified using the metric of H_∞. An estimation of the lower bound H_∞ for each observation on X can be derived. All relevant environmental parameters and aging are bounded, e.g., temperature $T \in [\alpha_T, \beta_T]$, supply voltage $V_S \in [\alpha_V, \beta_V]$, and the maximum lifespan β_L. For an upper-bounded evaluation time β_E, any query to an instance of the non-ideal TRNG can be regarded as an observation on X with a minimum true randomness of H_∞.

2.6 TRNG Design Procedures and Evaluations

Various design criteria need to be taken into account when designing a TRNG. Conventional design goals include resource consumption, throughput, latency, implementation feasibility on the target platform and design effort. An ongoing trend started more than a decade ago towards following security criteria during the TRNG design procedure. Some essential requirements are resistance against attacks, a stochastic model to prove unpredictability, and inner testability of the entropy source.

Conventional design goals are applicable to the TRNG and other components in the embedded system. A unique and indispensable design criterion for TRNGs is the security assessment. This security assessment is based on two requirements of the generated random numbers: uniformity and unpredictability.

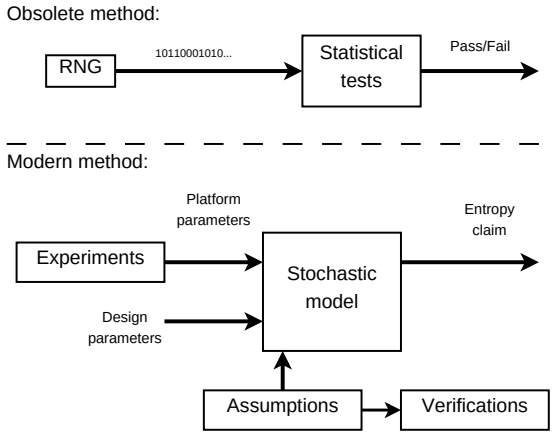


Figure 2.2: TRNG Design methods.

Figure 2.2 shows both the old and the modern TRNG design procedures. They are discussed separately in the following subsections.

2.6.1 The Early TRNG Design Procedure

The old and obsolete design approach relies on statistical evaluations on the output of the proposed TRNG. The proposed TRNGs are supposed to output random numbers with some unpredictable behavior. Then, the designer needs to collect different amounts of random numbers ranging from several megabits to one gigabit according to different standards. The most commonly used statistical test suites are NIST SP 800-22 [87], FIPS 140-1 [33] and DIEHARD [65]. These statistical tests result in a simple pass/fail flag. When the original random numbers don't pass these statistical test suites, additional post-processing is utilized to improve the passing rate. It implies that it is always possible to pass statistical tests by applying more post-processing. However, this design procedure suffers from two severe problems.

First, statistical tests can only evaluate the uniformity of the generated sequences. Pseudo random numbers and manipulated sequences cannot be distinguished from true random numbers. Passing these statistical tests doesn't guarantee the unpredictable requirement is fulfilled. For example, in Figure 2.3, the irrational number π and an output sequence from a TRNG are shown in the form of a bitmap where '1' and '0' are denoted as black and white respectively. There is no visible and straightforward statistical difference between these two bitmaps.

(a) π (b) Output of a TRNG

Figure 2.3: Bitmap representations of π and the output of a TRNG.

Both sequences are able to pass the tests suites mentioned above, although π is completely deterministic.

The second problem is that the data for testing are only collected under a specific set of parameters. TRNGs may behave differently under another set of parameters and fail the tests. For example, the platform parameters of a TRNG may vary due to process variation or even an upgrade of the Electronic Design Automation (EDA) tools. In addition, the working environments may change because of a different application scenario. In a reconfigurable platform, design parameters might be reprogrammed to meet specific non-security performances, such as throughput, latency and power consumption. All changes of the platform parameters, design parameters and working environment may lead to compromised statistical properties.

To conclude, test suites are only checking statistical parameters of the generated bits. This design procedure cannot guarantee the unpredictability of the TRNG. A completely deterministic pseudo random sequence could pass those test suites, despite having no randomness.

2.6.2 The Modern TRNG Design Procedure

In the modern design approach, designers are not only pursuing good statistical properties of the generated random numbers, but are also required to provide reliable and verifiable evidence of the unpredictability of the TRNGs. Normally, a stochastic model of the entropy source is used to prove, verify and evaluate

the true randomness of the proposed TRNG design. Statistical test suites only function as a sanity check or a prototype evaluation.

The non-determinism of a TRNG should be rooted in an unpredictable physical process that evolves over time. A notable example is the timing phase jitter in a free-running ring oscillator.

Before we introduce the modern TRNG design procedure, let us discuss the requirements from industrial standards.

Requirements of Standards

Modern TRNG standards indicate their inclinations towards a stochastic model based security assessment, though they have different specific requirements. We discuss the security requirements of TRNG designs from the two most influential industrial standards.

AIS-31 Requirements The German standard AIS-31 [52], which is also applied as the *de facto* standard in Europe, consists of a formal security evaluation of random number generators. The standard defines three classes of PTRNGs: **PTG.1**, **PTG.2** and **PTG.3**. The security requirements of these three classes are incremental from PTG.1 to PTG.3.

Generators from the PTG.1 class are required to have good statistical properties at the internal random numbers. Total failure tests are required to detect whether the entropy source is totally broken-down. According to the standard, TRNGs from PTG.1 might be used to generate random numbers for applications for which the random numbers need not meet any unpredictability properties.

The PTG.2 class requires a stochastic model of the entropy source to justify the unpredictability of the proposed TRNG. This model is used to estimate the lower bound of the Shannon entropy of the generated data. A minimal Shannon entropy level of 0.997 per bit is required at either the raw random numbers or the output of the algorithmic post-processing. Instead of embedded tests on the internal random numbers, statistical tests are applied to the raw random numbers for a better detection rate and a faster response.

In addition to the requirements of the PTG.2, the PTG.3 class requires cryptographic post-processing to provide extra security anchor. This security anchor relies on the computational security of the cryptographic post-processing algorithm. More details about cryptographic post-processing can be found in Section 2.7.2.

NIST 800-90B Requirements The NIST has published the final version of the special publication SP800-90B [99]. This document provides design and evaluation requirements for TRNGs. The standard does not regard the stochastic model of the entropy source as a compulsory design criterion. However, TRNG developers are required to justify and estimate the bias and the entropy of the generated numbers. This standard also states its requirements of health tests, Independent and Identically Distributed (IID) assumption verification and min-entropy estimation.

The lower part of Figure 2.2 shows the modern TRNG design method. The stochastic model is central in the modern methodology. A stochastic model is a simplified abstraction of the unpredictable physical process based on clearly stated assumptions. There are two types of assumptions. The first type are verifiable assumptions about the underlying physical processes from which the true randomness source originates. These assumptions should be verified experimentally. Wrong assumptions of the first type may ruin the correctness of the developed stochastic model which in turn corrupts the security assessment derived from the model. For example, the assumption about the existence of a specific type of noise in the silicon chip might be incorrect after an upgrade of semiconductor processing or even the software for the lithography process.

The second type of assumptions are about the ignored or unknown physical processes in the stochastic model. Compared to a physical model, such as a transistor-level model, a stochastic model is a simplified version with assumptions of the second type. Even though these ignored or unknown physical processes are contributing to the generated entropy, simplifications are still necessary for several reasons. Firstly, without simplification, a stochastic model can be too complicated to be used. The second reason is that some process parameters cannot be determined precisely or cannot be measured during the design time. It is crucial to always assume the worst case scenario, even it can be a challenging task to identify the worst-case for a particular model.

Platform and design parameters serve as inputs to a stochastic model to enable the entropy estimation of a TRNG. Platform parameters are the intrinsic properties of a specific platform, such as the delays of logic gates and the jitter strength. Platform parameters should be evaluated through experiments on the target platform. In contrast to the platform parameters, design parameters are determined at design time. They are carefully chosen by designers to meet particular security and performance requirements. Typical design parameters include the sampling frequency, the number of delay elements in a ring oscillator, the compression ratio of the post-processing algorithm etc.

2.7 Post-processing

The post-processing module is a deterministic function which is applied to non-ideal raw random numbers to improve their statistical and security characteristics. There are two types of post-processing: algorithmic post-processing and cryptographic post-processing.

2.7.1 Algorithmic Post-processing

Algorithmic post-processing is a term used for all simple methods for processing raw random numbers without using cryptographic primitives. Algorithmic post-processing can be regarded as a compression function with compact hardware implementations. Here we discuss the mathematical principles of two popular methods.

Parity Filter

Parity filter, also known as XOR post-processing, is a simple algorithmic post-processing technique which combines n_f of consecutive input bits to generate one output bit using an XOR function. The number n_f of the consecutive input bits is called the order of the parity filter.

If the n_f consecutive input bits are independent, the bias of the output sequence can be computed as:

$$\epsilon_{internal} = 2^{n_f-1}\epsilon_{raw}^{n_f}, \tag{2.9}$$

where the ϵ_{raw} is the bias of the raw random numbers and $\epsilon_{internal}$ stands for the bias of the internal random numbers.

The parity filter suffers from two problems. The first problem is that the parity filter can only mitigate but not eliminate the bias of the input sequence. The bias of the output bits converges to 0 when $n_f \to \infty$. The second problem is the amount of wasted entropy. Given the bias of the raw random numbers, to achieve a specific bias of the internal random numbers, Equation (2.9) can be transformed and used to compute the required n_f:

$$n_f = \frac{1 + \log_2 \epsilon_{internal}}{1 + \log_2 \epsilon_{raw}}, \tag{2.10}$$

If we use min-entropy as the metric, for a binary discrete random variable, the wasted entropy H_{wasted} can be computed as:

$$H_{wasted} = (-1) \cdot n_f \cdot \log_2(0.5 + \epsilon_{raw}) - (-1) \cdot \log_2(0.5 + \epsilon_{internal}). \tag{2.11}$$

The $(-1)\cdot\log_2(0.5+\epsilon_{internal})$ part of the equation converges to 1 when $n_f \to \infty$. However, if the required n_f increases, the $(-1)\cdot n_f \cdot \log_2(0.5+\epsilon_{raw})$ part of the equation increases linearly with a slope determined by ϵ_{raw}. In other words, when there is a large gap between the target bias $\epsilon_{internal}$ and ϵ_{raw}, a considerable amount of entropy will be wasted.

Von Neumann Post-processing

This debiasing method was introduced by von Neumann in [109]. The generated raw random sequence is partitioned into 2-bit blocks. Blocks equal to 00 and 11 are discarded. For other blocks, the first bit of each block is used as the post-processed bit. The advantage of this method is that it rigorously eliminates the bias of the output bits, if the input bits are mutually independent. However this method suffers from a major drawback: the output bits have a variable rate. The average output rate is $T_{in} \cdot p_1(1-p_1)$, where T_{in} is the throughput of the input and p_1 is the probability of generating bit value 1. Von Neumann doesn't work for sequences with dependencies between bits and it also wastes a considerable amount of entropy. More details on Von Neumann post-processing and one solution to the wasted entropy problem are discussed in Chapter 5.

2.7.2 Cryptographic Post-processing

This type of post-processing takes its name from the fact that it uses cryptographic primitives to either extract randomness from raw bits or to provide extra security anchors.

In the NIST standards [99], a list of vetted cryptographic primitives are recommended to be used for post-processing, such as Message Authentication Code (MAC) algorithms, hash functions and block ciphers. Normally, hardware implementations of these primitives are resource-consuming. However, TRNGs are usually utilized in embedded security systems in which these cryptographic primitives play a vital role. Thus, these primitives can be reused as a post-processing module with a minimal additional area cost. According to [99], with the knowledge of the length and the entropy of the input, the amount of the expected entropy at the output of a particular vetted post-processing module can be calculated. And furthermore, due to the avalanche effect of cryptographic block ciphers and hash functions, the entropy of the output is evenly distributed.

In the AIS31 standard [52], cryptographic post-processing is used to increase the computational complexity and to provide additional security anchors, such

as forward and backward secrecy. Forward secrecy is to assure that future values of internal numbers cannot be determined from the current or previous output values. Backward secrecy is the assurance that previous output cannot be derived from current or future output values.

2.8 Embedded Tests

Embedded tests for TRNGs are different from embedded tests for general-purpose digital circuits. In addition to the testing of the correct functionalities, embedded tests for TRNGs are also expected to be able to detect potential compromises in the entropy source. In this section, we will discuss the requirements of embedded tests from different standards. Then, a description of continuous tests and their positions in a TRNG system is introduced.

NIST 800-90B explicitly requires embedded tests under the name of health *tests*. Three types of embedded tests are proposed for monitoring the quality of the noise source. All three tests are applied on raw random numbers before any possible post-processing.

The first type of embedded tests is called *start-up health tests*. Start-up health tests are performed after every powering up, rebooting, and before the first use of the entropy source. It helps to check whether the entropy source is working properly, before it outputs any random numbers. Samples used for start-up health tests can be discarded. Reusing these samples is only allowed after the completion of the tests and if there are no errors.

Continuous health tests are also referred to as on-the-fly tests, online tests and continuous tests. When the TRNGs are operating, continuous health tests are running continuously on the outputs of the entropy source. Continuous tests are designed to detect many kinds of failures in the underlying entropy source. Since the continuous health tests are running indefinitely, the false alarm rate needs to be carefully selected for particular applications. In designing the continuous health tests, there is the tradeoff between the hardware resource consumption and the ability to detect entropy source failures.

The third type of embedded tests required by SP800-90B are called *On-demand health tests*. On-demand health tests are not required to run continuously. If rebooting and powering up are immediately followed by start-up health tests, they can be regarded as acceptable methods to initialize on-demand health tests.

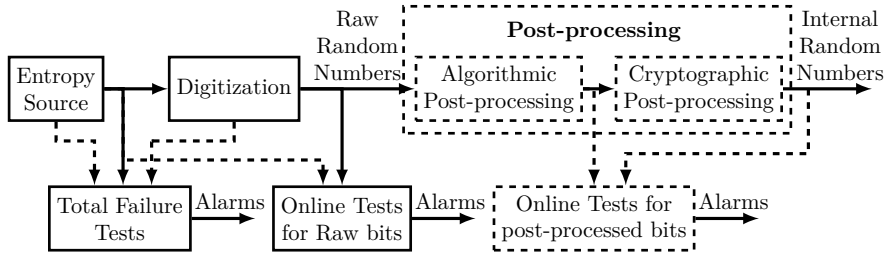


Figure 2.4: TRNG with embedded tests.

AIS-31 has different requirements of embedded tests for PTG.1, PTG.2 and PTG.3. *Continuous tests* are denoted as online tests in AIS-31. The name of start-up tests remains the same as in NIST 800-90B. AIS-31 has an additional type of tests called *tot tests* or *total failure tests*. Total failure tests detect the total breakdown of the entropy source. If the entropy source is totally broken-down, the entropy of the future raw random numbers is equal to zero. If a total breakdown of the entropy source occurs, the total failure tests must respond to the failure fast and stop the TRNG from further outputting random numbers. PTG.1, PTG.2 and PTG.3 all require total failure tests of the noise source. For PTG.1-compliant TRNGs, continuous tests are used to detect the intolerable statistical defects of the internal random numbers. According to the requirements of PTG.2 and PTG.3, TRNGs from these two classes require continuous tests to detect statistical failures in the raw random numbers.

According to both NIST and AIS standards, continuous tests and total failure tests are essential for monitoring the health of TRNGs. Figure 2.4 shows a typical TRNG architecture with all possible testing building blocks. Post-processing is optional, if the raw random numbers after digitization fulfill the requirement. Dashed lines in the figure indicate the possible inputs for various tests.

Total failure tests are supposed to quickly respond to the total breakdown of the TRNG. Therefore, they are measuring the entropy source and digitization directly. For example, for a ring-oscillator-based TRNG, the total failure tests should block the TRNG's output and trigger an alarm, if the ring oscillator stop oscillating.

According to standards, online statistical tests can be applied to the raw random numbers and/or the post-processed random numbers. However, in our opinion, online tests should work with intermediate random numbers before any post-processing. One reason is that the post-processing algorithms are whitening the potential flaws in the raw random numbers. In addition, an appropriate

post-processing algorithm can be regarded as a compression function. If the online tests are working with a fixed length sequence, it takes a longer execution time before the online tests can draw a conclusion. Logically, it is meaningless to apply the online tests to the output of the cryptographic post-processing. However, they might serve as an additional security anchor to detect fault attacks on the post-processing algorithms.

A more detailed discussion of online tests can be found in Chapter 6 and Chapter 7.

2.9 Field-Programmable Gate Array

An FPGA is an integrated circuit providing configurability to customers or designers after manufacturing. The FPGA industry originated from both Programmable Logic Devices (PLDs) and Programmable Read Only Memory (PROM), which can be programmed by the user (field programmable). FPGAs were invented to explore tradeoffs between general-purpose Central Processing Unit (CPU) and Application-Specific Integrated Circuit (ASIC), which implies tradeoffs between computational flexibility and computational efficiency. At the very beginning, the FPGA was mainly used as the prototype for ASIC designs to avoid extra design spins. However, the performance gap between ASICs and FPGAs becomes smaller because of the efforts of FPGA manufacturing companies. There is also a trend of integrating FPGAs and other dedicated resources such as Digital Signal Processor (DSP), Random Access Memory (RAM) blocks and microcontrollers into a heterogeneous system. Today's FPGAs are widely used as end products to meet requirements of faster time-to market, reduced development time and cost.

There are both advantages and challenges in using FPGAs as the platforms for security applications. First, security applications on FPGA are upgradable. Design bugs and implementation flaws can be seamlessly patched using updated configurations. It's also easy to add support for newer security standards and algorithms to existing systems. However, using FPGAs as platforms can be a great challenge for security applications that require balanced differential routing such as the side-channel resistance technique called Wave Dynamic Differential Logic (WDDL), metastability-based PUFs.

Designing a high throughput TRNG on FPGAs for cryptographic applications is non-trivial, since the conventional goals of digital designs on FPGA consists of low noise and stable digital behavior of logic functions. Primitive components on FPGA can however be utilized for efficient TRNG implementations, as will be illustrated in this thesis.

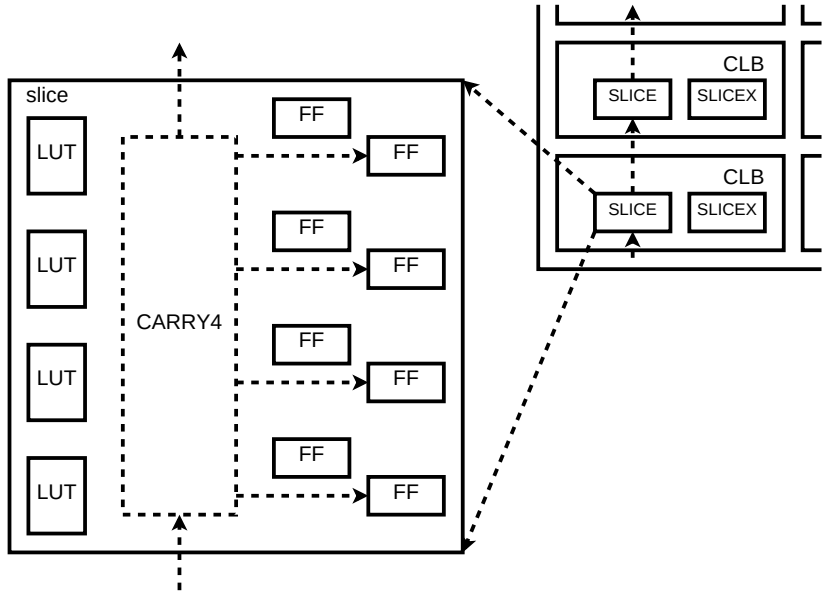


Figure 2.5: A conceptual diagram of Spartan-6.

To extend the discussion, a conceptual diagram of Xilinx Spartan-6 FPGA is depicted in Figure 2.5. The latest Xilinx FPGA families are similar in the level of primitive building blocks, although some variance may exist. In this discussion, we focus on the architecture of the Spartan-6 family. The main logic resources for implementing combinatorial circuits are the Configurable Logic Blocks (CLBs). Each CLB block is connected to a switch matrix which can be programmed to meet different requirements of routings. A CLB element contains a pair of slices. According to the user guide of Spartan-6 TRNGs [118], one slice of this pair in a CLB is marked as SLICEX, and another slice is marked as either SLICEL or SLICEM depending on the position of the CLB. Approximately, 50% available slices are SLICEX and 25% each are SLICEL and SLICEM. All three types of slices consists of four Look-Up Tabless (LUTs) with six-input and eight storage elements. LUTs can be programmed to be any arbitrarily defined six-input Boolean function. Four of these eight storage elements can be configured as either edge-triggered D-type Flip-Flops (DFFs) or level sensitive latches, while other four elements can only be used as DFFs. A special carry structure called *CARRY4* is included in SLICEL and SLICEM, in other words it is included in 50% of available slices on Spartan-6 and every slice on the latest families such as Virtex-7 [117]. The dedicated *CARRY4* primitive is designed to perform fast arithmetic addition and subtraction. Several *CARRY4* modules

from the same slice column can be concatenated vertically up to form a wider carry chain.

Throughout this thesis, most of the proposed techniques and designs were implemented and validated on FPGA platforms, although most of them are also suitable for ASIC implementations. There are several notable implementations which utilize special FPGA primitives. LUTs are used to implement the free-running ring oscillators for noise source. CARRY4 primitives are used to implement time-to-digital Time-to-Digital Converter (TDC) for jitter measurements and digitizer.

2.10 Summary

This chapter serves as the theoretical foundation of this thesis. It starts with the introduction of TRNGs, which is followed by a discussion of the mathematical backgrounds. After the definition of the TRNG, the modern design methodology of a TRNG is compared with the obsolete one. Other components of TRNGs, such as embedded tests and post-processing, are also discussed.

Chapter 3

Entropy Source

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.

-John von Neumann

Content Source

Yang, B., Rožić, V., Grujić, M., Mentens, N., and Verbauwhede, I. On-chip jitter measurement for true random number generators, In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2017.
Contribution: Principle author, contribute to the methodology and responsible for the implementation and the experiment.

3.1 Introduction

As depicted in Figure 3.1, the digital noise source consists of an entropy source and a digitization. For some TRNGs, there is no absolute boundary between the entropy source and the digitization module. Generally speaking, the entropy source is referred to as the component where the unpredictable physical processes are taking place. And the digitization module samples the entropy source and converts the obtained value into digital numbers to be compliant with other digital components.

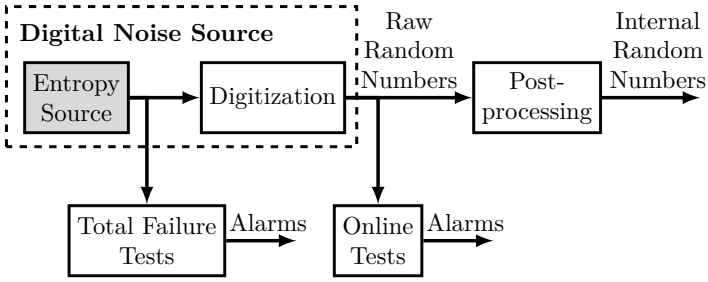


Figure 3.1: The generic architecture of a TRNG. The gray part is discussed in this chapter

In this chapter, the entropy source module of the TRNG is discussed. In Section 3.2, we give an overview of existing entropy sources. In Section 3.3, an on-chip jitter measurement method for TRNG applications is introduced and discussed. The content of Section 3.3 was partially published at [120]. Section 3.4 summarizes this chapter.

3.2 Entropy Sources for TRNGs

A wide variety of physical phenomena has been utilized as the noise source. These physical phenomena have in common that they can be described as unpredictable random physical processes. In this section, several types of entropy sources are discussed.

3.2.1 Thermal Noise

Thermal noise, also known as Johnson–Nyquist noise [68], is intrinsic electronic noise which occurs regardless of any applied voltage. This electronic noise is generated by the thermal agitation of the charge carriers inside an electrical conductor at equilibrium. Figure 3.2 provides a common equivalent model of the thermal noise over a resistor. A noisy resistor under a specific temperature can be regarded as a noiseless resistor in parallel with a noise-creating current source. According to [68], the current spectral density can be computed as:

$$i_n = \sqrt{(4k_bT\Delta f)/R}, \tag{3.1}$$

where k_b is Boltzmann's constant in joules per kelvin, T is the absolute temperature in K and R is the resistor value in ohms (Ω). The amplified thermal

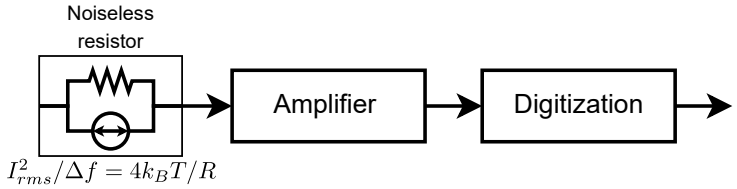


Figure 3.2: The conceptual architecture of thermal noise based entropy source.

noise across a resistor is usually utilized as the entropy source of a TRNG. This type of entropy source is normally suitable for ASIC implementations.

A well-known TRNG from this family was introduced by Intel [49]. In this design, the Johnson thermal noise over a resistor is amplified and then used as the input of Voltage-controlled oscillatorss (VCOs). The output of this VCO is used to sample a high-speed oscillator. A von Neumann post-processing module is used to improve the statistical properties of the generated random numbers. More recent TRNGs from this family can be found in [71, 15].

3.2.2 Metastability

Metastability is the most commonly used entropy source for both FPGA and ASIC TRNGs. Various forms of metastability are explored. In early designs, the metastable operation of latches was utilized as a noise source [6]. TRNGs of this type rely on the circuit symmetry to achieve unbiased outputs. The TRNG proposed in [6] is implemented in 2 μm technology and required a careful dimensioning and layout. However, due to the process variation of the latest technologies, it is probably not likely to achieve a sufficient level of symmetry and robustness. Recent designs from this family try to evaluate or compensate the asymmetry with post-silicon methods. Walker et al. evaluated the metastability of a DFF circuit for random number generation in [112]. The last passage time of ring oscillators is utilized as the entropy source in [81]. Tokunaga et al. present a metastability based TRNG with a quality control in [98]. The metastable resolution time, measured by a TDC, is used to grade the quality of the output bitstream and as feedback to tune the system. Random bits marked with low quality are discarded. Utilizing the write collisions in Block Memory (BRAM)s of TRNGs as entropy sources was presented in [41]. The proposed TRNG provides a throughput of more than 100 $Mbps$ with good statistical quality. The drawback of this TRNG is that, due to the lack of the low-level understanding of BRAM, as it is a company secret, it is almost impossible to characterize the randomness-generating process and to evaluate its security. Programmable delay lines are used in [60] to accurately equalize

the signal arrival times to flip-flops. The proposed design was implemented on a Xilinx Virtex5 FPGA, and achieved a throughput of 2 $Mbps$.

3.2.3 Timing Jitter

In electronic systems, timing jitter is defined as the deviation from a periodic signal, such as a reference clock signal. It is popular to use the timing jitter of free-running oscillators or Phase-locked loops (PLLs) as randomness sources for TRNGs. PLL-based TRNGs were introduced in [36, 30]. The optimizations of PLL-based TRNGs were presented in [72, 2]. Roover and Steyaert introduced an entropy source with jitter amplification in [25]. A TRNG based on multiple ring oscillators was introduced in [94], and was improved by Wold et al. in [116]. Utilizing multiple edges of a ring oscillator as the randomness source was presented by Yang et al. in [125]. In [79], Rahman et al. introduced a technology-independent TRNG with bias detection. A timing jitter based TRNG which utilizes a self-timed ring oscillator was introduced in [16], its stochastic model was given by [17], and its threat model and countermeasures against attacks were presented in [38].

3.2.4 Chaos Circuits

Chaotic circuits are used to denote electronic circuits that exhibit classic chaotic behavior. They amplify small changes at the initial states to produce significantly different future outputs. A TRNG using a double-scroll attractor was proposed and analyzed in [119]. In [82], a simple chaotic discrete-time systems based TRNG was introduced. The proposed TRNG relies on a simple mathematical model called discrete maps to generate chaotic signals. Another TRNG based on discrete-time chaotic maps was presented by Pareschi et al. in [69]. It was implemented in 0.35 μm technology, achieved a throughput of 40 $Mbps$ with an area of 0.52 mm^2 and a power consumption less than 30 mW. It requires post-processing to pass the statistical tests. Analog-to-Digital Converters (ADCs) are also used for chaos-based TRNG. An ADC-based TRNG exploiting nonlinear signal processing and chaos was presented by Callegari et al. in [14]. In this work, a 1.5-bit ADC module is used. According to their theoretical analysis and extensive simulation, a throughput of 10 $Mbits$ can be achieved using a simple parity filter. The post-processed data could pass the statistical tests [87].

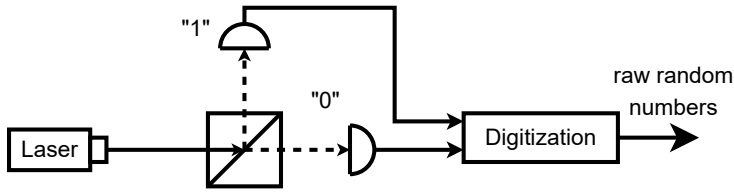


Figure 3.3: The conceptual architecture of an optical quantum based entropy source.

3.2.5 Quantum Effect

Various quantum effects can be utilized as a randomness source. A nuclear decay radiation source measured by a Geiger counter can be regarded as an entropy source [111]. A popular type of Quantum Random Number Generators (QRNGs) is based on the detection of a single photon between the two outputs of a beam splitter. A conceptual diagram is illustrated in Figure 3.3. The typical sources of photons are lasers, light emitting diodes and single-photon emitters. The photons pass through a balanced beam splitter with equal transmissivity and reflectivity, and reach one of these two detectors. The results are encoded to the raw random numbers using digitization.

There are various TRNG designs in this family, ranging from commercial products [45] to academic literature [93]. In [48], Jennewein et al. present a physical QRNG with a throughput of $1\,Mbps$. A single-electron based QRNG was introduced in [101]. The proposed QRNG consists of single-electron transistor and a single-electron trap. A QRNG, which utilizes a 16x16 array of detector pairs, was recently proposed by Massari et al. [66]. Arbiter modules were implemented to determine which detector in the pair of two received a photon first. The proposed design was fabricated in a CMOS 150nm technology and achieved a throughput of $128\,Mbps$.

3.2.6 Mixture Entropy Sources

There are several TRNGs of which the entropy sources rely on the mixture of true randomness and pseudo randomness. An early TRNG [1] design combines true randomness from the metastability of latches with pseudo randomness from a Linear Feedback Shift Register (LFSR). The Motorola TRNG was presented in [97]. Two free-running ring oscillators are used to clock a 43-bit LFSR and a 37-bit Cellular Automata Shift Register (CASR), respectively. The true randomness of this TRNG comes from the timing jitter of these two

free-running ring oscillators, while the LFSR and the CASR contribute to the pseudo randomness part. A digital true random number generator based on asynchronous logic circuits with feedback is introduced by Golić in [47]. Two ring oscillators called Galois and Fibonacci serve as the entropy sources in which the true randomness fuses the pseudo randomness.

It is worth to note that if a chaotic circuit is implemented with a digital synchronous circuit only, it functions like a PRNG expanding the true randomness from the initial state. If other forms of true randomness have an influence on the operation of the chaotic circuit, the chaos source can be regarded as a mixture entropy source. The chaotic behavior of the chaos circuit may improve the computational security of the TRNG. However, it does not provide any additional true randomness, since the output of a chaos circuit is only determined by its initial state.

Paralleling Kerckhoffs's principle, the unpredictability of a TRNG should not rely on the obscurity introduced by the pseudo random processes. A practical attack [27] on the Motorola TRNG was presented by Dichtl. When the TRNG is sampled at high bit rates, the generator produces little entropy. In principle, due to the small amount of generated entropy and the linearity of the LFSR and the CASR, it is possible to predict the output bits from the TRNG.

We would like to note that the true random process and the pseudo random process may exist in the entropy source at the same time. However, for the TRNG evaluation and the entropy estimation, the unpredictability of the entropy source should only rely on the true randomness, while always considering the worst-case value of other pseudo random processes.

3.3 Jitter Measurement for TRNGs

Timing jitter is a critical platform parameter for ring oscillator based TRNGs and affects the entropy of the TRNG. The description of the measurement methodology is an obligatory part of the TRNG certification procedure.

In this section, we present our novel methodology for measuring the timing jitter of free-running ring oscillators on FPGA platforms. This methodology is designed to measure the strength of the white noise while filtering out contributions from other noise sources such as flicker noise and power supply variations.

3.3.1 Jitter in Ring Oscillators

Free-running ring oscillators are implemented by connecting any number of delay elements (including an odd number of inverting elements) into a loop configuration. The delay of each logic element in a ring oscillator consists of an average component and a variable component. Several types of noise could affect the variable component:

- Global noise from the supply voltage variations. This noise affects all components on the platform.
- Environment noise such as slow changes of the operating conditions (e.g. temperature changes).
- Correlated noise such as flicker noise and telegraph noise.
- White noise which is also called Gaussian noise.

Mathematical models of TRNG designs usually rely only on the Gaussian noise because other noise sources are either difficult to model and analyze or can be manipulated by an attacker. These noise sources are usually treated as deterministic.

The period of the ring oscillator is given by:

$$T_i = T_0 + T_{Global} + T_E + T_{Corr} + T_{Gauss}\,, \tag{3.2}$$

where T_{Gauss} is a random variable with standard deviation σ_G, T_0 is the average period and T_{Global}, T_E and T_{Corr} are the contributions of the global noise, environment noise and the correlated noise sources, respectively.

For a Ring Oscillator (RO) that is running for time t_m, the following relation holds:

$$t_m = n \cdot T_0 + t_D + t_G\,, \tag{3.3}$$

where n is the number of full oscillations made during the $(0, t_m)$ time interval, t_D is the deterministic noise accumulated during this interval and t_G is the accumulated Gaussian noise. Let σ_m^2 denote the variance of t_G. Then the following holds:

$$\sigma_m^2 = \sigma^2(t_G) = \frac{\sigma^2(T_{Gauss})}{T_0} \cdot t_m\,. \tag{3.4}$$

The parameter σ_m^2/t_m characterizes the rate of the Gaussian jitter accumulation. This quantity is the most important platform parameter for RO-based TRNGs because it is the only parameter that characterizes the randomness generation process.

3.3.2 Methodology

A procedure for accurate and precise characterization of the white noise should satisfy the following requirements:

- All measurements should be performed on-chip – only the digital data should be sent to the output for post-processing.
- Digitization should be performed using a small quantization step to reduce the quantization noise.
- All measurements should be differential in order to reduce the influence of the global noise sources.
- The jitter accumulation time should be as short as possible in order to minimize the effect of the low frequency noise sources.

The proposed jitter measurement method is based on time-to-digital conversion, which is implemented using the carry chain primitives. Carry chain primitives are available on most commercial FPGAs and they are designed to constitute high-speed adders and multipliers. They can also be configured to operate as high-resolution TDCs.

The proposed measurement methodology consists of three steps:

1. T_0 measurement.
2. Delay line characterization.
3. Differential TDC measurement.

In continuation of this section we describe these three steps in more detail.

T_0 measurement

For measuring the average period T_0 of a ring oscillator, the architecture shown in Figure 3.4 is proposed. The output signal of the ring oscillator under

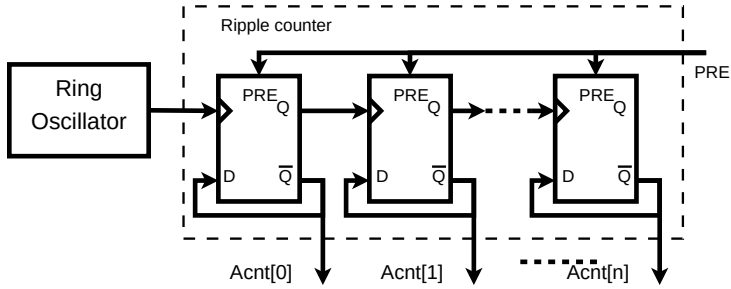


Figure 3.4: Ring oscillator period measurement setup.

evaluation connects to an n-bit ripple counter to count the number of $0 \rightarrow 1$ transitions. The ripple counter consists of n positive-edge-triggered DFFs.

To perform an experiment, all flip-flops in the ripple counter are preset to 1 before enabling the ring oscillator. Then, the ring oscillator is enabled for N_{acc} system clock cycles. The system frequency is denoted as f_{system}. After N_{acc} cycles, the ripple counter result $Acnt$ is stored.

This experiment is repeated N_{exp} times. At the beginning of every experiment, the ripple counter is preset, and the value of it is stored as $Acnt_i$. In order to cancel out the influence of low frequency noise over the ring oscillator period, the average of all stored $Acnt_i$ is computed. In the last step, T_0, the average period of the ring oscillator, can be computed as:

$$T_0 = \frac{N_{acc} \cdot N_{exp}}{f_{system} \cdot \sum_{i=1}^{N_{exp}} Acnt_i}. \tag{3.5}$$

Delay line characterization

In our proposed jitter measurement method, two delay line based TDCs are used. For a precise measurement, the steps of elements in the delay line need to be small and accurate. Therefore, the delay line should be characterized first.

We propose a method based on Monte Carlo experiments to evaluate the delay line. As shown in Figure 3.5, a signal osc generated by a ring oscillator propagates through an n elements delay chain under evaluation. The average period of the signal osc is T_0. The length of the delay line should be able to cover more than one and a half full periods of the signal osc. An independent system clock Clk is used to sample this delay line N_{exp} times. The captured states (bit vectors) of the delay line are denoted as:

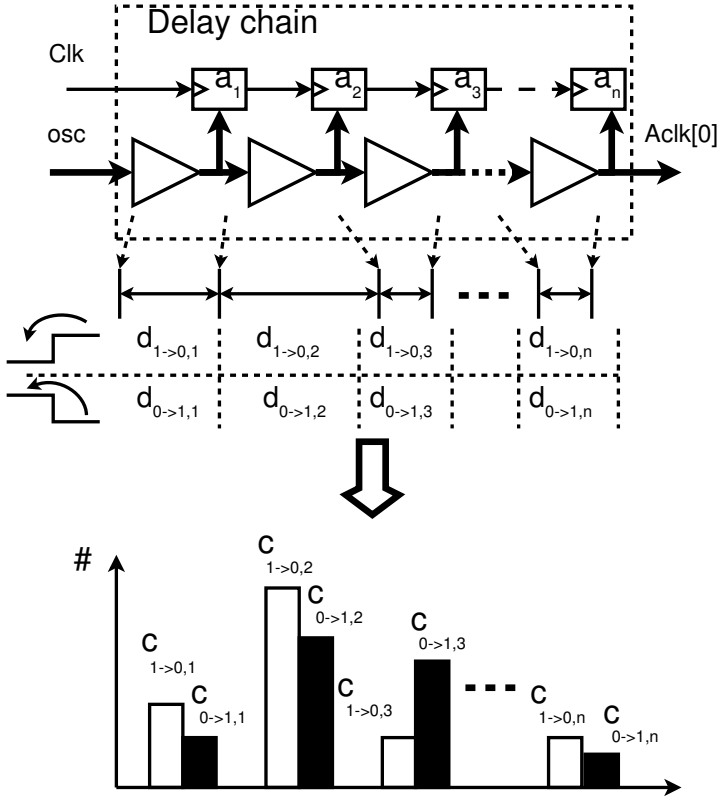


Figure 3.5: Monte Carlo method for delay measurement.

$$m_i = (a_{i,0}, a_{i,1} \dots a_{i,n}),\ a_{i,j} \in \{0, 1\}, \tag{3.6}$$

where $a_{i,0}$ is the most recent bit and $a_{i,n}$ is the earliest bit in the i_{th} sampled sequence.

The bit vector m_i consists of consecutive $'1's$ and $'0's$, representing the signal *osc*. If the delays of all n elements are the same, the probabilities of having a transition are equal at every position. However, due to the nonlinearity of the delay line, the probability of having a transition is higher for a delay element with larger delay.

We denote the delay of a $1 \to 0$ and $0 \to 1$ transition for the j_{th} element with $d_{1\to0,j}$ and $d_{0\to1,j}$, respectively. The goal of characterizing the delay line is to estimate the value of $d_{1\to0,j}$ and $d_{0\to1,j}$.

$c_{1\to0,j}$ and $c_{0\to1,j}$ are defined as:

$$\begin{aligned} c_{1\to0,j} &= \#(m_i),\ for\ (a_{i,j}, a_{i,j+1}) = (0,1), \\ c_{0\to1,j} &= \#(m_i),\ for\ (a_{i,j}, a_{i,j+1}) = (1,0). \end{aligned} \tag{3.7}$$

$c_{1\to0,j}$ is the number of samples that have the subvector $(0,1)$ at position $(j, j+1)$.

The proportion of any two elements from a union set of $c_{0\to1,j}$ and $c_{1\to0,j}$ is equal to the proportion of the corresponding delays of a union set of $d_{0\to1,j}$ and $d_{1\to0,j}$.

We define the following functions for the summation of consecutive $c_{0\to1,j}$ and $c_{1\to0,j}$:

$$\begin{aligned} S_{1\to0}(i,j) &= \sum_{x=i}^{j} c_{1\to0,x} \\ S_{0\to1}(i,j) &= \sum_{x=i}^{j} c_{0\to1,x}. \end{aligned} \tag{3.8}$$

We denote $e_{1\to0,j}$ and $e_{0\to1,j}$ as the edge positions of the j_{th} latest $1 \to 0$ and $0 \to 1$ transitions in the snapshot, respectively. Therefore, $(a_{i,e_{1\to0,1}}, \ldots, a_{i,e_{0\to1,1}}, \ldots, a_{i,e_{1\to0,2}-1})$ represents the first full period of *osc* captured in this snapshot.

In the next step, the function $W(m_i)$ is defined and used to represent the period of the first captured full period of the signal *osc*:

$$W(m_i) = S_{0\to1}(e_{1\to0,1}, e_{0\to1,1} - 1) + S_{1\to0}(e_{0\to1,1}, e_{1\to0,2} - 1). \tag{3.9}$$

And, W_{cnt} is the average $W(m_i)$ value of N_{exp} experiments:

$$W_{cnt} = \frac{\sum_{i=1}^{N_{exp}} W(m_i)}{N_{exp}}. \tag{3.10}$$

The delays $d_{1\to0,j}$ and $d_{0\to1,j}$ of every element of the delay line can be estimated by:

$$\begin{aligned} d_{1\to0,j} &= \frac{c_{1\to0,j}}{W_{cnt}} \cdot T_0 \\ d_{0\to1,j} &= \frac{c_{0\to1,j}}{W_{cnt}} \cdot T_0. \end{aligned} \tag{3.11}$$

Differential TDC measurement

Figure 3.6 depicts the architecture proposed for jitter measurement. This architecture consists of two TDCs, two ring oscillators and two ripple counters. Signals $osc1$ and $osc2$ are the outputs of the corresponding free running oscillators. Then $osc1$ and $osc2$ propagate through two n-stage tapped delay lines. The rising transitions in $osc1$ and $osc2$ are counted by two ripple counters.

To estimate the jitter strength, we first measure the timing phase difference of the corresponding edges in Snapshot 1 and 2 after an accumulation time t_m. This measurement is repeated N_{exp} times. The number of $1 \to 0$ transitions in Snapshot 1 and 2 are denoted as l_{S1} and l_{S2} respectively. We denote $g_{1\to0,i}$ as the position of the i_{th} latest $1 \to 0$ transition in the bit vector. Vectors m_{S1} and m_{S2} are captured sequences in Snapshot1 and Snapshot2. The timing of the i_{th} $1 \to 0$ transition in sequence m can be computed by:

$$T(m, i_{th}) = \sum_{i=g_{1\to0,i_{th}}}^{n} d_i + T_0 \cdot Acnt. \tag{3.12}$$

The phase difference of the corresponding edges for the j_{th} experiment is computed by:

$$t_{diff,j} = \begin{cases} T(m_{S1}, l_{S1}) - T(m_{S2}, l_{S2}), & for\ Acnt1 = Acnt2 \\ T(m_{S1}, l_{S1}) - T(m_{S2}, l_{S2} - 1), & for\ Acnt1 = Acnt2 + 1 \\ T(m_{S1}, l_{S1} - 1) - T(m_{S2}, l_{S2}), & for\ Acnt1 = Acnt2 - 1 \end{cases} \tag{3.13}$$

Then, the average value of $t_{diff,j}$ is:

$$\bar{t}_{diff} = \frac{\sum_{j=1}^{N_{exp}} t_{diff,j}}{N_{exp}}. \tag{3.14}$$

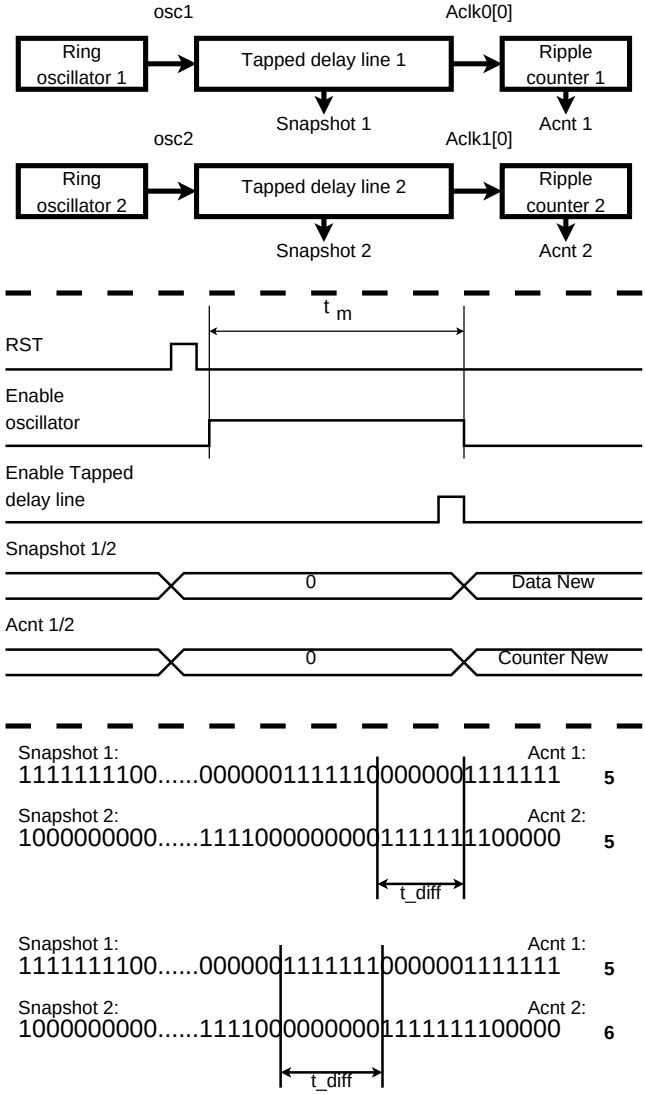


Figure 3.6: Setup of the differential jitter measurement.

The expected value of $\bar{t}_{diff}$ is around 0. However, it may be different due to the effect of manufacture process variations.

In the last step, jitter accumulated during time t_m can be computed by:

$$\sigma_m(t_m) = \sqrt{\frac{\sum_{i=1}^{N_{exp}} (t_{diff,i} - \bar{t}_{diff})^2}{N_{exp} - 1}} \tag{3.15}$$

The parameter σ_m^2/t_m characterizes the rate of the Gaussian jitter accumulation.

3.3.3 Experiments

The proposed method for jitter measurement is applied to two case studies on two different FPGA platforms.

Xilinx FPGA

Figure 3.7 shows the implementation of the proposed jitter measurement on a Xilinx Spartan-6 FPGA. Ring oscillators are implemented using a single LUT. CARRY4 primitives are used to implement tapped delay lines. In total, 64 CARRY4 primitives are cascaded together to form a 256-stage delay line.

The frequency of the system clock Clk is set to 100 MHz. We select parameters $N_{acc} = 2^{13}$ and $N_{exp} = 100000$ for the T_0 measurement. The collected $\sum_{x=1}^{N_{exp}} Acnt_x$ are 6482648529 for ring oscillator 1 and 6500063667 for ring oscillator 2. Therefore, the computed T_0 of ring oscillator 1 and 2 are $1.2637ns$ and $1.2603ns$ respectively.

While characterizing the delay line, bubbles appear in the captured snapshots. In [32], this problem is handled by counting the number of $'1's$ and $'0's$. The origin of the bubble might be the Carry Look Ahead Logic structure and the process variation during fabrication. In total, we found 8 types of bubbles. All of them can be easily corrected by reordering. After reordering, the $1 \rightarrow 0$ and $0 \rightarrow 1$ delays of all 256 stages are characterized and shown in Figure 3.8. The average delay of those stages is $16.8ps$.

In each measurement, the oscillators were enabled for t_m ns before the snapshots were taken. The snapshots and the counter values were transferred to the PC for post-processing. Based on the 100000 measurements for each, the computed variance of t_{diff} for $t_m = 20$ ns, 40 ns, 60 ns and 80 ns, are shown in Figure 3.9. The lower bound noise strength is obtained at $t_m = 80$ ns. The computed

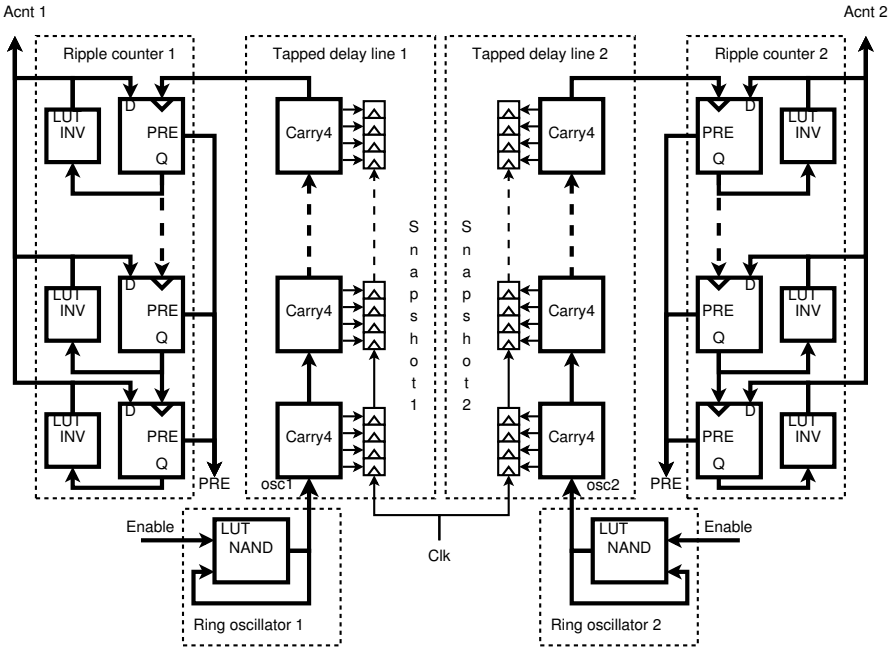


Figure 3.7: Implementation of the jitter measurement on a Xilinx Spartan-6 platform.

standard deviation $\sigma(t_{diff}) = 41.9413\ ps$. Then, the computed noise strength is:

$$\frac{\sigma_m^2}{t_m} = 21.98\ fs\,. \tag{3.16}$$

Intel FPGA

The jitter measurement circuit for the Intel (Cyclone IV) FPGA is implemented in an analogous way as on the Xilinx FPGA. The stages of the ring oscillator are instantiated in the LUTs of neighboring LEs (Cyclone IV Logic Elements) by using low-level primitives. Fast (tapped) delay lines are implemented using small 3-input LUTs, which are specially dedicated for fast carry generation [46]. The delay lines are placed in LEs right below the ring oscillator by using placement constraints. Ripple counters are then placed below these delay lines in order to minimize interconnection delays. Bubbles in the delay line appear only near the signal edges, meaning that they originate from uneven propagation

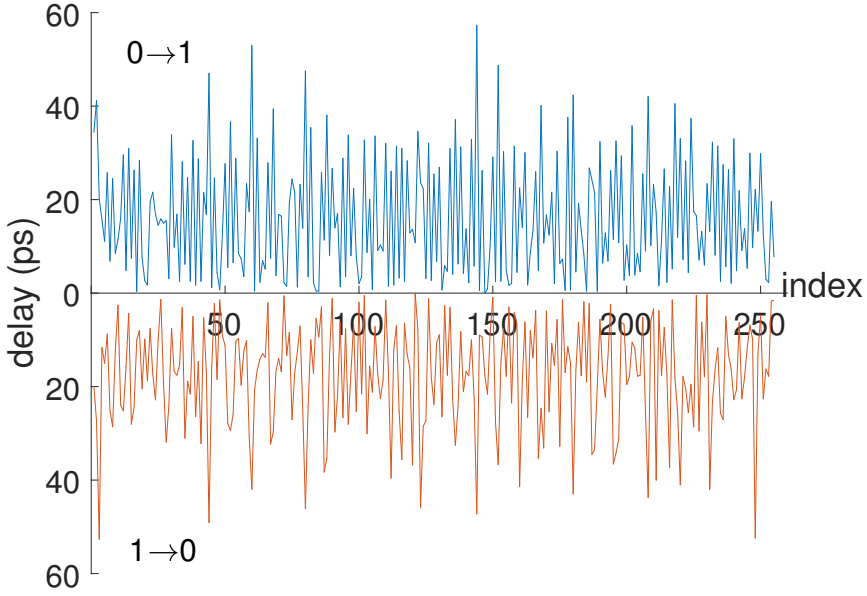


Figure 3.8: Delays of each carry bit after characterization.

delays in the FPGA structure and timing violations on delay line capturing registers.

The average periods of ring oscillator 1 and 2 are 2394.2712 ps and 2529.605 ps. In each measurement, the oscillators were enabled for $t_m = 40$ ns before the snapshots were taken. The snapshots and the counter values were transferred to the PC for post-processing. Based on the 100000 measurements, the computed standard deviation of t_{diff} is $\sigma(t_{diff}) = 5.9739$ ps. The computed noise strength is:

$$\frac{\sigma_m^2}{t_m} = 0.89 \ fs\,. \tag{3.17}$$

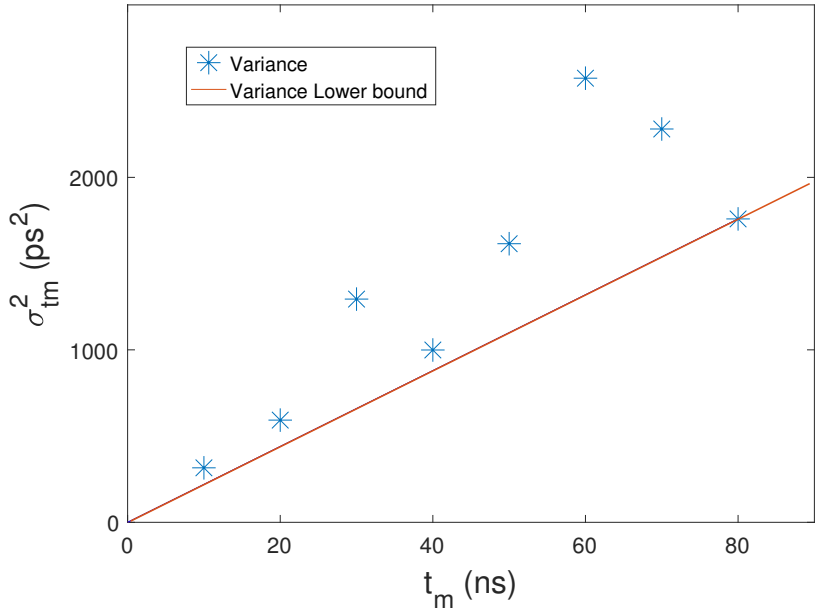


Figure 3.9: Variance result of t_{diff}.

3.3.4 Conclusion and Discussion

While the oscillator period T_0 can be estimated accurately using very simple counter-based methods, estimating the strength of the jitter is a more challenging task. Jitter in ring oscillators is often studied from the reliability perspective, where the goal is to provide the upper boundaries of jitter strength. However, for TRNG applications, the goal is to derive the lower bound on jitter strength in order to provide a conservative estimate of the output entropy. For this reason, different measurement procedures that isolate the contribution of the white noise need to be developed. In this section we compare the proposed jitter measurement methodology with other, more commonly used methods, and discuss the consequences of choosing a wrong jitter estimation.

The most straightforward solution for jitter measurement is to connect the oscillator output to the output pin and to analyze the signal using an oscilloscope. This approach was taken in [73]. The problem with this strategy is that the oscillator signal is corrupted by the pads, pins and measurement probe. In addition, commonly used oscilloscopes have a bandwidth of up to 4 GHz,

Table 3.1: Jitter measurement results using three different methodologies.

Method	Counter	Single Delay Line	Differential Delay Line
σ_G^2/T_0	11 ps	154 fs	22 fs
σ_G	118 ps	13.9 ps	5.26 ps

which is not sufficient to analyze the Gaussian jitter which is in the order of picoseconds. The results presented in [73] show that oscillators with a period ranging from 2 ns to 10 ns, all have a comparable Gaussian jitter. This result is not in line with the theory on the white noise which is a strong indicator that the measurement setup introduces some systematic error.

The simplest on-chip method for jitter evaluation consists of counting the toggles of the ring oscillator output over a fixed amount of time and computing the jitter based on the variance of the obtained results. This methodology is used in [102] and [59]. We will refer to this technique as *the counter methodology*. The errors of this methodology are caused by the quantization noise and the low frequency noise. If the measurement time is too short, the quantization noise of the counter is dominant and it is impossible to measure the Gaussian jitter precisely. For longer measurement times the quantization noise is reduced but the contribution of the low-frequency noise becomes dominant. In either case, the Gaussian noise is significantly overestimated.

To illustrate the importance of differential measurements, a single delay line based measurement method was used for comparison. We refer to this method as *Single Delay Line* methodology. Jitter is computed using only the data from $Snapshot1$. The jitter is computed from the variance of the timing of the last rising edge $T(m_{S1}, l_{S1})$. This method reduces the quantization noise and low frequency noise but it doesn't filter out the global noise from the power supply.

Table 3.1 summarizes the jitter results measured using the three different methodologies on a Xilinx Spartan-6 FPGA. The proposed *Differential Delay Line* methodology filters out the quantization noise, low frequency noises and the global noise and thus results in the lowest jitter estimate. The single delay line method doesn't filter out the global noise and thus overestimates the jitter variance by a factor of 5. The counter methodology grossly overestimates the jitter strength.

Another related work is the coherent sampling method for jitter measurement presented in [58]. We don't see any problems with this methodology and the reported results are similar to those obtained by the *Differential Delay Line* method.

Finally we present an example to illustrate how improper jitter measurement procedure can affect the design decisions. We consider a Spartan-6 implementation platform and we assume that the parameter $T_0 = 1,26\ ns$ can be estimated accurately. Using the proposed methodology we obtain the jitter accumulation rate of $\sigma_m^2/t_m = 22\ fs$. To illustrate the importance of differential measurements, a single delay chain based measurement method was used for comparison. This method consists of computing the standard deviation of the timing of the last rising edge $Snapshoot1$.

3.4 Summary

The noise source is the critical building block for FPGAs. It is the only component generating true randomness. Six types of entropy sources in the literature are discussed. Blindly mixing true randomness with pseudo randomness should be avoided, since it may result in an overestimated entropy and provides over-confidence to the designer. The worst-case values of the pseudo randomness process should be taken into account for entropy estimation.

One popular type of entropy source, timing jitter, is discussed in detail. Accurate jitter characterization is an essential step in designing RO-based TRNGs for FPGAs. The measurement setup and the methodology for jitter characterization on FPGA platforms have been described. The proposed setup utilizes fast carry chain elements that are available on most FPGAs. Differential measurements are used to reduce the influence of the global noise sources. The methodology has been applied to measure jitter on Xilinx Spartan-6 and Intel Cyclone IV FPGAs.

Chapter 4

Digital Noise Source

The generation of random numbers is too important to be left to chance.

-Robert R. Coveyou

Content Source

Rožić, V., Yang, B., Dehaene, W., and Verbauwhede, I. Highly Efficient Entropy Extraction for True Random Number Generators on FPGAs, In *52nd Design Automation Conference (DAC)*, 2015.
Contribution: Responsible for the implementation.

4.1 Introduction

This chapter covers the topic of the Digital Noise Source module, indicated as the gray part of Figure 4.1. Digital noise sources consists of the Entropy Source module and the Digitization module. The Entropy Source module is introduced and discussed in Chapter 3. A digitization module is required to convert these output samples to bits, if the sample of the noise source is something other than binary data. However, if the sample of the noise source is in a satisfactory form, the digitization module is not mandatory. In other words, the digitization can

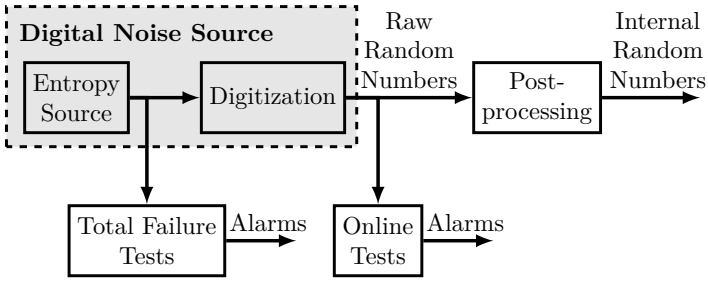


Figure 4.1: The generic architecture of a TRNG. The digital noise source is discussed in this chapter.

be the identity mapping. Normally, the digital noise source defines the type of TRNG, since the same entropy source can be connected to different samplers. We would like to note that it is not always possible to locate a clear boundary between the entropy source and the digitization.

This chapter is organized as follows. Section 4.2 reviews the existing digital noise sources with a special focus on timing jitter based TRNGs. In Section 4.3, we introduce a novel timing jitter based TRNG, which follows an approach that is different from elementary TRNGs and coherent sampling TRNG to improve the performance of ring oscillator based TRNGs. The content of Section 4.3 was partially published at [83]. Finally, Section 4.4 concludes this chapter.

4.2 Digitization: An Overview

Various methods are used for digitization, such as ADC, TDC etc. However, the digitization module is fused with the entropy source in TRNG designs. For example, the boundary between the entropy source and the digitization is not clear for most of the metastability based TRNGs. These TRNGs produce output digits by resolving their metastable states. In this section, several digital noise sources are discussed.

4.2.1 Motorola TRNG

As mentioned in Section 3.2.6, the Motorola TRNG was published in [97], and was used within Motorola for a number of years before the publication. The architecture of the Motorola TRNG is shown in Figure 4.2. The Motorola

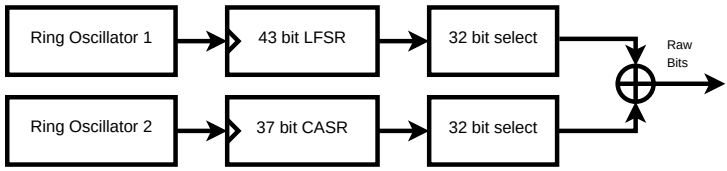


Figure 4.2: The Motorola TRNG.

TRNG is based on a 43-bit LFSR and a 37-bit CASR. The LFSR and the CASR are clocked by two independent free-running ring oscillators. When a new number is requested, 32 bits from each register are selected, permuted and XORed to generated the raw random bits.

It is not easy to locate the digitization module of the Motorola TRNG. Since these two ring oscillators resonate independently, LFSR and CASR are updated individually. The output random number depends on both the timing jitter from the ring oscillators and the metastability while sampling. The states of the LFSR and the CASR depend on the update function and the initial states. The pseudo randomness from the registers and the true randomness from the free running oscillators are mixed together, which makes it difficult to design a stochastic model for the TRNG. Another disadvantage of blending the pseudo randomness with true randomness is that the generated random sequence is able to pass any statistical test easily even when there is no true randomness.

4.2.2 TERO TRNG

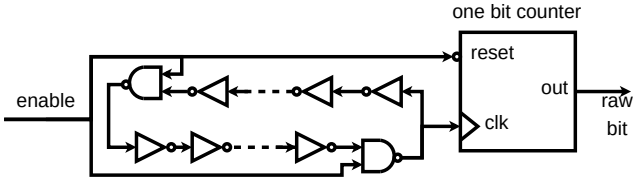


Figure 4.3: The TERO TRNG.

The Transition Effect Ring Oscillator (TERO) TRNG was first proposed by Varchola and Drutarovsky [105] in 2010. The robustness of the TERO TRNG against voltage variations was discussed in [104]. Its stochastic model was published by Haddad *et al.* [43] in 2015.

The block diagram of the TERO TRNG is shown in Figure 4.3. The entropy source of the TERO TRNG is the TERO loop, which consists of an even number

of inverters and two NAND gates that restart the oscillation. Triggered by the rising edge of the enable signal, the TERO resonates temporarily. Due to the electronic noise in the transistors in the TERO structure, the number of oscillations it takes to resolve the metastable state is not constant. A one-bit counter is utilized to keep track of the least significant bit of the number of oscillations, and the final count is used as the raw random bit. For TERO TRNG, the digitization module and the entropy source are one unit.

Implementing the TERO TRNG on FPGA requires careful placement and routing, since the delays between the forward path and the backward path should be balanced to reach a metastable state.

4.2.3 Open-loop TRNG

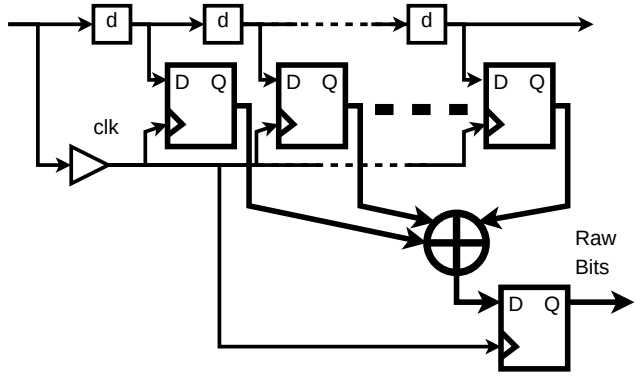


Figure 4.4: The Open-loop TRNG.

The Open-loop TRNG depicted in Figure 4.4 was first proposed by Danger *et al.* [23, 24], and its stochastic model was introduced by Ben-Romdhane *et al.* [7]. In 2013, Lozach *et al.* [57] present a design methodology aiming to adapt a generic architecture for the Open-loop TRNG.

The randomness of the Open-loop TRNG comes from the metastable behavior of a DFF. The input signal is fed into a chain of delay elements, of which each output is sampled by its own DFF. All DFFs are sampled with the same input signal. The small delays of the elements in the chain are carefully selected so that at least one of these elements reaches the metastable state. Using a relatively long chain provides robustness against changes of the operating conditions, such as the temperature and the voltage fluctuation in the power supply introduced by the adversary. All digital samples are XORed together to yield the raw binary bit.

The main drawback of the Open-loop TRNG is the requirement of manual placement and routing since the delay chains are very sensitive to routing differences. The digitization and entropy source are also integrated in one module. In the original design of the Open-loop TRNG, for the reason of debiasing, the raw random sequence is XORed with the output of a LFSR to be able to pass the standard statistical tests. This is not a good practice, since XORing with the output of a LFSR does not increase the entropy in the output. Masking the statistical defects in the raw sequence with the LFSR output also reduces the testability of the TRNG.

4.2.4 The Elementary Ring Oscillator based TRNG

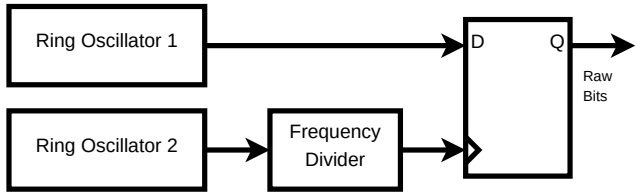


Figure 4.5: The elementary TRNG.

As shown in Figure 4.5, the Elementary ring oscillator based TRNG (ERO-TRNG) has a simple architecture, which consists of only two free-running ring oscillators, one frequency divider and a DFF. The ERO-TRNG was widely applied [11] and has been the subject to rigorous mathematical analysis [59, 5, 21] before it was named by Fischer *et al.* in 2014 [37].

The true randomness of the ERO-TRNG origins from the timing jitter of the free-running oscillator indicated as the *Ring Oscillator 1* in the figure. The second oscillator is followed by a frequency divider to generate a slow sampling frequency. According to [35], implementing identical ring oscillators as the entropy source and the sampler helps to significantly reduce the impact of the global deterministic jitter that can easily be manipulated by an attacker. Because of its simple structure, the ERO-TRNG can be used as a basic building block for almost any TRNG relying on the timing jitter in oscillators. In addition to the simple structure, the ERO-TRNG also has advantages of compact implementation, a detailed and applicable stochastic model, and online testability. However, the main drawback is the low throughput. This is because the preferable Gaussian jitter accumulates proportionally to the square root of time and much more slowly than the accumulation speed of the deterministic jitter [102]. For the ERO-TRNG, ring oscillator 1 is used as the entropy source,

while the digitization is done by ring oscillator 2, the frequency diver and the sampling DFF.

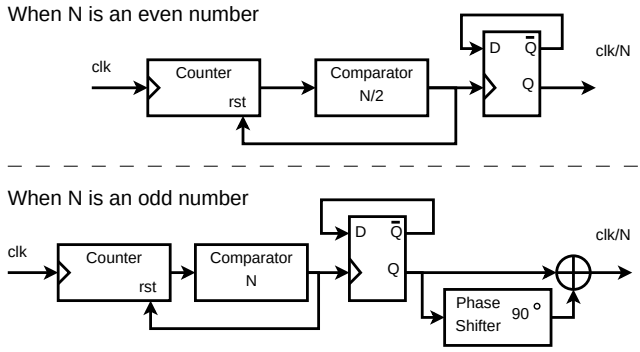


Figure 4.6: Conceptual architectures of frequency dividers.

Frequency dividers can be implemented for both analog and digital applications. For FPGA implementations, frequency dividers are normally implemented using dedicated onchip PLLs. It is also possible to implement digital frequency dividers with 50% duty cycle for on FPGAs from scratch. If the original clock is divided by a power of 2, the straightforward solution is to cascade a series of DFFs. However, sometimes, it is desirable to divide a master frequency by an arbitrary odd or even number N. The conceptual architecture of a frequency divider with 50% duty cycle for an arbitrary N is illustrated in Figure 4.6. When N is an even number, a frequency divider can be implemented using a counter that counts up to $N/2$. When N is an odd number, the simplest way is to generate two clocks at half of the target frequency with a 90° phase difference. More details about frequency dividers can be found in [3].

4.2.5 Multi-Ring Oscillator based TRNG

The digital noise source of the Multi-Ring Oscillator based TRNG (MURO-TRNG), which employs ring oscillators as a source of timing jitter, is depicted in Figure 4.7.

The MURO-TRNG and its original stochastic model were introduced by Sunar *et al.* in 2007 [94]. A FPGA implementation of MURO-TRNG was proposed by Schellekens *et al.* in 2006 [89]. This implementation consists of 110 ring oscillators of 3 inverters, and achieves a throughput of more than 2 Mbps. The sampling DFFs, indicated with dashed lines, are not present in the designs from both [94] and [89]. In other words, all signals from the oscillators are

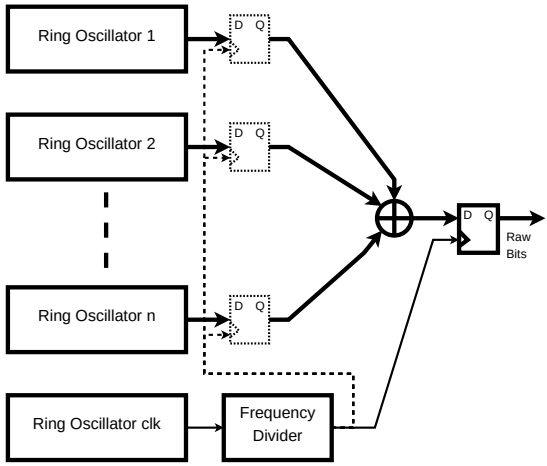


Figure 4.7: The Multi-Ring Oscillator TRNG.

XORed together without being sampled and synchronized to the same frequency. This practice, pointed out by Dichtl *et al.* in [29], is based on an unrealistic assumption on the speed of the utilized XOR gates. And the sampling DFF also cannot handle the great amount of transitions from the XOR gate. The solution to these issues was proposed by Wold *et al.* [116], in which each oscillator is sampled by a DFF before being fed to XOR-tree. A further investigation of the interaction between the oscillators on different FPGA platforms was carried out and discussed in [115].

The simulation results presented in [10] indicate that a MURO-TRNG consisting of 18 jitter-free oscillators of slightly different frequencies passes all statistical tests. The pseudo randomness, which masks the insufficiency of true randomness, is caused by XOR-ing clock signals having different frequencies. Moreover, a considerable number of oscillators are locked and become dependent. According to [10], to ensure a sufficient quality of the generated true randomness, the required number of ring-oscillators should be determined by using the stochastic model instead of considering the used architecture. Yoo *et al.* [126] investigated implementation aspects of the MURO-TRNG on FPGA and discussed the effects of the operating conditions on the performance of the MURO-TRNG. An improved behavioral stochastic model was proposed in [114]. This improved model is based on the number of hits in the transition regions accounting for the true randomness.

4.2.6 Coherent Sampling TRNG

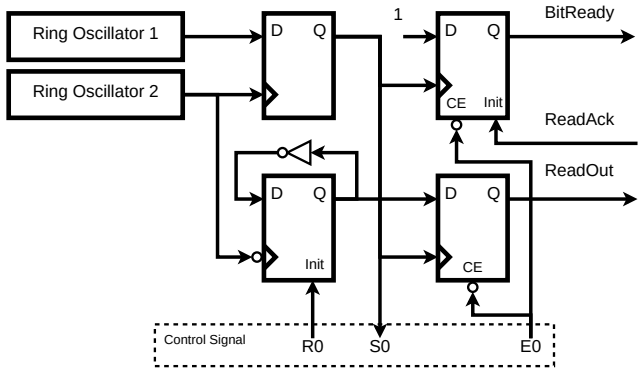


Figure 4.8: The Coherent Sampling TRNG.

The principle of the Coherent Sampling TRNG (COSO-TRNG) was introduced by Kohlbrenner and Gaj in 2004 [53]. The COSO-TRNG was proposed to be used for FPGAs that lack a PLL primitive.

Figure 4.8 shows the architecture of the COSO-TRNG. Two oscillators, which have the same number of delay elements and the same topology of the placement, are used in the COSO-TRNG to achieve two close but not identical frequencies. The signal *S0* is generated using the output of *oscillator 2* to sample the output of *oscillator 1*. A one-bit counter is used to count the transitions of *oscillator 2* within the period of *S0*. The theoretical resolution of the coherent sampling is the difference in periods of these two oscillators and can be computed as $T_\Delta = T_{RO1} - T_{RO2}$, where T_{RO1} and T_{RO2} denote the average period of *oscillator 1* and *oscillator 2* respectively. Improved implementations of the COSO-TRNG on FPGAs of three mainstream vendors were proposed by Petura *et al.* in 2016 [73].

The COSO-TRNG achieves a relatively high entropy throughput with an extremely compact implementation. However, according to [73], the main drawback of the COSO-TRNG is its repeatability. The performance of the COSO-TRNG relies on a sufficiently small value of the parameter T_Δ, which is not repeatable even for identical implementations on different devices.

4.2.7 PLL TRNG

The idea of using the jitter existing in analog PLLs as the noise source for TRNG designs was introduced by Fischer *et al.* in 2002 [36]. The stochastic of

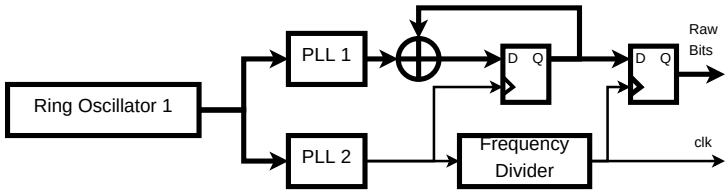


Figure 4.9: The PLL-based TRNG.

the PLL-based TRNG (PLL-TRNG) was proposed in [8].

The coherent sampling technique is also used in PLL-TRNGs. However, the PLL-TRNG has several advantages over the COSO-TRNG. First, PLLs are available as dedicated components on FPGAs of mainstream vendors and are physically isolated from the rest of the device. Implementing a TRNG with dedicated components saves configurable logic for other designs in the system. This physical isolation provides additional security, because the impact of other designs running in the configurable logic area is reduced. Another important advantage of PLL-TRNG is that using PLLs as a noise source ensures a good repeatability on different platforms.

A conceptual architecture of the PLL-TRNG implemented with two PLLs is depicted in Figure 4.9. A single free-running ring oscillator is used as the input signal for both PLL 1 and PLL 2, which leads to the favorable result that the outputs at both PLLs are mutually related. The ratio between these two PLLs only depends on their configurations. The output signals of these two PLLs are used to generated the raw bits by following the coherent sampling principle. Thanks to the capability of tuning the ratio between these two output signals of PLLs, the entropy level and throughput of the PLL-TRNG can be optimized by selecting the best parameters.

However, choosing appropriate PLL configurations from a large set of parameter combinations is the most challenging work in the PLL-TRNG design. A genetic algorithm based parameters selection for PLL-TRNGs was proposed by Petura *et al.* in 2017 [72]. One year later in [2], Allini *et al.* proposed an algorithm enabling the research of all suitable configurations for PLL-TRNGs implemented with either one or two PLLs.

4.3 Delay Chain Based TRNG

In this section, we present a timing jitter based high-throughput TRNG suitable for FPGA implementations. The Delay Chain based True Random Number

Generator (DC-TRNG) was originally introduced in 2015 [83] for Xilinx FPGA platforms. The original design was improved and ported to an Intel FPGA platform in 2018 [40]. An updated stochastic model was published in 2018 [39].

Our goal is to improve the efficiency of the entropy extraction from the accumulated timing jitter in the free-running ring oscillators. Rather than boosting throughput by increasing the number of transition events at the cost of high energy and/or area, we focus on improving the entropy extraction from a single transition event.

The main motivation of our idea is the fact that the TRNG throughput increases proportionally to the square of the timing resolution of the sampled signal. If the sampling precision is increased by a factor of k then, in order to achieve the same level of entropy, the required accumulated jitter is k times smaller. Since Gaussian jitter accumulates proportionally to the square root of time [102], to achieve the same level of jitter strength, the required accumulation time is k^2 times smaller. For this reason, a modest improvement in the timing resolution leads to a high reduction in accumulation time and a proportional throughput gain.

The carry-chain primitive was discussed in Section 3.3 for on-chip jitter measurement. In this section, we utilize the carry chains for entropy extraction. Carry chains from the neighboring slices can be cascaded through dedicated routing paths to form a larger delay line. This configuration can be used for sampling signals with a high timing accuracy and it has been used for implementing high-resolution TDCs in [67]. Benefiting from this high resolution sampling, our TRNG implementation achieved a compact footprint with high throughput.

4.3.1 DC-TRNG Architecture

The DC-TRNG is based on the concept of sampling the unstable signal with high precision, thereby extracting more entropy compared to the standard approach, because the throughput gain scales with the square of the improvement in precision.

The architecture of the digital noise source in the DC-TRNG is shown in Figure 4.10. In this architecture, the entropy source and the digitizer are implemented as separate blocks. A tapped Delay Chain, a Priority Encoder and a DFF comprise the digitizer of the DC-TRNG.

The entropy source is implemented as a free-running oscillator using a single LUT configured as a two-input NAND gate. All randomness in the system

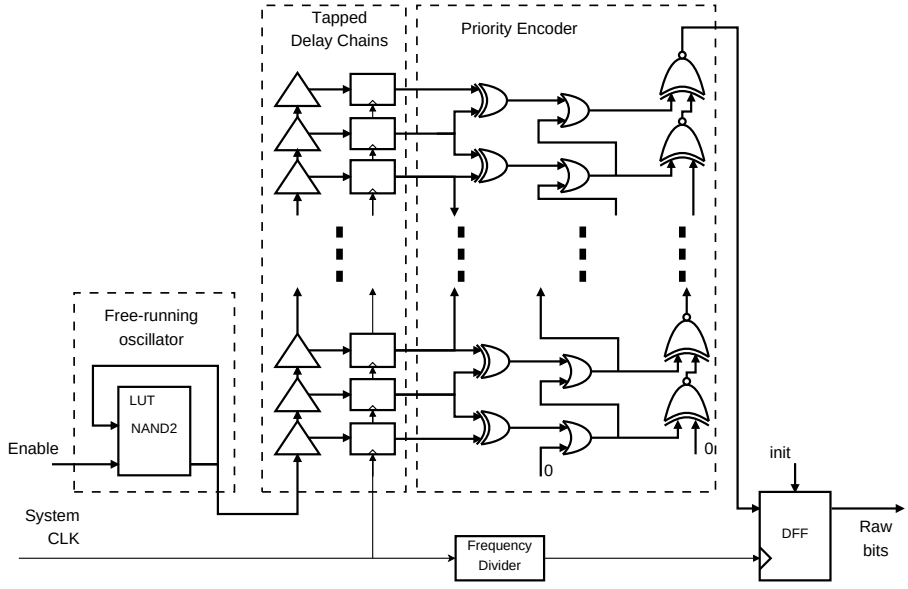


Figure 4.10: The digital noise source of the DC-TRNG.

is generated by the timing uncertainty of the signal in this oscillator. Other components of the DC-TRNG are purely deterministic.

Oscillator signals are sampled using the tapped delay chain. The tapped delay chain is performing the time-to-digital conversion of the noisy signals. The timing resolution of the conversion is equal to the delay of a single stage of the tapped delay line. After a sufficient jitter accumulation period, the timing phase of the signal becomes unpredictable due to the accumulated Gaussian noise. This jittered signal coming from the RO propagates through a delay chain with m fast buffers and is sampled by a set of DFFs, each of which is attached to a delay element. Taking a sample of the delay chain is equal to snapshotting a snippet of the signal in the oscillator. The position of the signal edge is determined by examining the saved snapshot.

The selection of parameter m depends on the period of the used noisy ring oscillator and the timing delays of these fast buffers. m should be chosen such that the delay chain can capture the full state of the oscillator. This selection should be done under the worst-case conditions, because the frequency of the oscillator as well as the delays of carry chain elements may vary due to voltage or temperature variations within the range according to a particular design specification. Otherwise, the captured state will be all '1's or all '0's, in other words, the position information of the signal will be lost. An expected snapshot

of the delay chain consists of a run of consecutive '1's followed by a run of '0's or vice versa.

Due to the timing violations during sampling, some flip-flops may be driven to a metastable state which can produce "bubbles" in the code. "Bubbles" can also be the consequence of a specific architecture of the carry primitives in FPGAs. For example, the delay chain can be implemented as a carry-lookahead adder, where all carry bits may not be generated sequentially. Multiple edges in the captured delay chain can occur if the total delay of the tapped line is larger than the period of an oscillator.

The position of signal edges in the captured snapshot is used to generate the raw bit. A priority encoder is utilized to address the sampling issues with "bubbles" and multiple edges. The priority encoder always encodes the first edge in the snapshot and ignores the rest, which also filters out the "bubbles". The Least Significant Bit (LSB) of the first edge position is used as the raw bit so that the odd positions are encoded as '1' and even positions as '0'. As depicted in Figure 4.10, the architecture of the priority encoder ensures that an output generated from the edge at the bottom overrides those outputs generated from edges above.

4.3.2 Security Analysis of the DC-TRNG

In this section, we present the formal security evaluation of the DC-TRNG. We follow the modern design procedure proposed in section 2.6.2. Assumptions about the FPGA platform are discussed first and then used to derive a simple stochastic model for binary probability calculation and design parameter optimization. The ultimate objective is to provide a lower bound of entropy per bit.

Assumptions We initialize the security analysis by making the following assumptions:

- The period of the free ring oscillator in the entropy source is influenced by the jitter. The period of the oscillator consists of a deterministic component T_D and a Gaussian component T_G. The Gaussian jitter originating from the thermal noise can be modeled by a normal distribution centered around zero, $\mathcal{N}(0, \sigma^2)$ where σ^2 is the jitter variance. Observations of this Gaussian jitter are mutually independent and also independent of other noise components in the system. More discussion on jitter in ring oscillators can be found in section 3.3.1.

- Noise sources, other than those generating white noise, are not exploited for entropy extraction, but are treated as deterministic jitter in this section. We assume that attackers can observe and manipulative deterministic jitter via power supply variations, fluctuations of temperature, and other active attacks such as Electro Magnetic Fault Injection (EMFI). Deterministic jitter is not quantified in our model, therefore we always assume the worst case values when estimating the lower bound of entropy.

- In contrast to deterministic jitter, we assume that attackers cannot predict or manipulate the amount of random jitter. The strength of the white noise jitter should be measured via experiments and should not be overestimated.

- The utilized TDC for high resolution sampling has perfect linearity, in other words, all elements in the carry chain have the same delay t_{step}. This assumption is used for the simplified stochastic model presented here. A more sophisticated model accounting for the nonlinearity of the TDC was introduced in [39].

- A stable system clock is available on the platform and can be used to generate the sampling signal. Jitter of both the system clock and the sampling signal are negligible compared to the jitter in the noise source.

Binary probability and Lower Bound of Entropy The DC-TRNG operates in the following way. The entropy source oscillator is reset before generating a bit and then enabled for a time t_A. After this time t_A, the sampling signal is activated. During the sampling, the signal from the oscillator is propagated through the fast delay lines. Since the oscillator is free-running, the jitter accumulates over time. It is critical that the delay line is long enough ($m \cdot t_{step} > T_D$), which guarantees that at least one noisy signal edge is captured in the delay line.

A phase-oriented model is used to describe the behavior of the free-running ring oscillator. The phase of the oscillator changes with time:

$$\xi(t) = \frac{t}{T_0} + \xi_{noise}(t), \tag{4.1}$$

where T_0 is the average period of the free-running ring oscillator and $\xi_{noise}(t)$ is the component of the phase originating from the noise. This phase component consists of both the deterministic phase jitter $\xi_D(t)$ and the Gaussian phase jitter $\xi_G(t)$:

$$\xi_{noise}(t) = \xi_D(t) + \xi_G(t). \tag{4.2}$$

From the assumption of σ^2 and the central limit theorem, the variance of the Gaussian jitter accumulates proportionally with time:

$$\sigma^2(\xi_g(t)) = \sigma_G^2 \cdot \frac{t}{T_0}, \tag{4.3}$$

where σ_g is the standard deviation of the Gaussian phase jitter accumulated during a single clock period T_0.

The phase of the oscillator after an accumulation time t_A since the latest reset equals:

$$\xi(t_A) = \frac{t_A}{T_0} + \xi_{noise}(t_A). \tag{4.4}$$

Due to the assumption that the Gaussian jitter is independent of all other sources, we have:

$$\sigma^2(\xi(t_A)) = \sigma^2(\xi_G(t_A)) + \sigma^2(\xi_D(t_A)) \geq \sigma^2(\xi_G(t_A)). \tag{4.5}$$

The timing variance of the jitter accumulated during time t_A is denoted as σ_A^2 and can be computed by following the procedure in Section 3.3:

$$\sigma_A^2 = \frac{1}{T_0^2} \cdot \frac{\sigma_m^2}{t_m} \cdot t_A. \tag{4.6}$$

The expected value of the oscillator phase at time t_A is:

$$E(\xi(t_A)) = E\left(\frac{t_A}{T_0} + \xi_{noise}(t_A)\right) = \frac{t_A}{T_0} + E(\xi_D(t_A)) + E(\xi_G(t_A)). \tag{4.7}$$

The average value of the Gaussian phase component is 0, therefore:

$$E(\xi(t_A)) = \frac{t_A}{T_0} + E(\xi_D(t_A)). \tag{4.8}$$

Since we assume that the attacker can control the deterministic jitter, we treat $\xi_D(t_A)$ as a constant:

$$E(\xi(t_A)) = \frac{t_A}{T_0} + \xi_D(t_A). \tag{4.9}$$

We denote the edge position sampled in the delay chain at time t_A as X. The phase $\xi(t_A)$ lies within the following boundaries:

$$X\frac{t_{step}}{T_0} < \xi(t_A) - \lfloor \xi(t_A) \rfloor < (X+1)\frac{t_{step}}{T_0}. \tag{4.10}$$

X sampled at time t_A is a random variable. The most likely value of this variable is:

$$E(X) = \left\lfloor (E(\xi(t_A)) - \lfloor E(\xi(t_A)) \rfloor) \cdot \frac{T_0}{t_{step}} \right\rfloor \tag{4.11}$$

Using the priority encoder, the raw random bit is computed as the LSB of X:

$$RawBit = LSB(X). \tag{4.12}$$

Since $\xi_D(t_A)$ cannot be predicted with reasonable precision at design time. We have to estimate the entropy based on the worst case of the impact of $\xi_D(t_A)$. Assume without loss of generality that $LSB(E(X)) = 1$. We define ξ_{offset} as:

$$\xi_{offset} = (E(\xi(t_A)) - \lfloor E(\xi(t_A)) \rfloor) \cdot \frac{T_0}{t_{step}} - E(X). \tag{4.13}$$

The minimal value of the entropy is obtained as follows:

$$\xi_{offset} = \frac{t_{step}}{2T_0}. \tag{4.14}$$

In this case, the raw bit probability is:

$$p_{raw} = Pr(LSB(X = 1)) \tag{4.15}$$

$$= \sum_{i=-\infty}^{\infty} \left(F_{\xi_{offset},\sigma_A}\left((2i+1) \cdot \frac{t_{step}}{T_0} \right) - F_{\xi_{offset},\sigma_A}\left((2i) \cdot \frac{t_{step}}{T_0} \right) \right)$$

The lower bound on min-entropy per bit and Shannon entropy per bit can be computed as:

$$H_\infty(RawBit) = -log_2(max(p_{raw}, 1 - p_{raw})), \tag{4.16}$$

$$H_1(RawBit) = -p_{raw} \cdot log_2(p_{raw}) - (1 - p_{raw}) \cdot log_2(1 - p_{raw}). \tag{4.17}$$

Once the platform parameters (T_0, t_{step} and σ_m^2/t_m) are measured and the design parameter (t_A) is determined. The lower bounds on the Shannon entropy and the min-entropy can be computed using equations (4.6), (4.14), (4.15), (4.16) and (4.17).

Use of the Model

The presented stochastic model can be used to estimate the lower bound of entropy from the platform parameters and the design parameters. A Xilinx

Spartan-6 FPGA was used for implementation. Stages of the ring oscillator are implemented using LUTs, and fast delay lines are implemented using carry-chain primitives. The platform parameter $T_0 = 2.2\ ns$ was measured by counting the number of transitions. The TDC step was determined by capturing an oscillator output in a long carry chain, and counting the number of stages of a clock period. We would like to note that the DC-TRNG is an early work of us, the nonlinearity of the TDC was not taken into account for the stochastic model. An updated model addressing the nonlinearity of the TDC was published at [39]. For the stochastic model presented in this section, an average TDC step $t_{step} = 17\ ps$ is used. We use the thermal jitter strength measured in Section 3.3, $\sigma_m^2/t_m = 22\ fs$.

The number of stages in the fast-delay line m should be able to cover at least half period of T_0:

$$m \geq \frac{T_0}{2 \cdot t_{step}} = 64.7. \tag{4.18}$$

In order to provide enough safety margin, we choose $m = 80$, which leads to a TDC using 20 CARRY4 primitives.

The jitter accumulation time t_A has to be a multiple of 10 ns, since the platform system clock frequency is 100 MHz. We performed an exhaustive search on all reasonable design parameters. For a given bias constraint of 2^{-64}, an optimal throughput of 9.09 $Mbps$ is achieved under $t_A = 10ns$ and the stages of parity filter $n_f = 11$.

4.4 Summary

The digital noise source is the central component of TRNGs, consisting of both the entropy source and the digitization module. Different forms of physical noise signals are sampled and converted into the proper format for the subsequent digital systems. Digital noise source modules of seven different TRNGs in literature are discussed. A novel digital noise source efficiently extracting entropy from free-running ring oscillators is presented to resolve the slow jitter accumulation speed for timing jitter based TRNGs on FPGAs. A stochastic model of the proposed DC-TRNG is introduced and used to analyze the security of the DC-TRNG.

Chapter 5

Post-processing

Basic research is what I am doing when I don't know what I am doing.

-Wernher von Braun

Content Source

Rožić, V., Yang, B., Dehaene, W., Mentens, N., and Verbauwhede, I. Iterating Von Neumann's Post-Processing under Hardware Constraints, In *9th IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2016.
Contribution: Responsible for the HW implementation and HW evaluation and contribute to the searching algorithm.

5.1 Introduction

The position of the post-processing module in the generic architecture of a TRNG is illustrated in Figure 5.1. The output of a digitization module normally deviates from the statistical ideal. Post-processing modules can be used to improve the statistical properties of these outputs. In contrast to a noise source module, a post-processing module is purely deterministic. Therefore, executing post-processing operations cannot add any additional true randomness. However, it is possible to apply an adequate post-processing to compress non-ideal raw random numbers into internal random numbers with better entropy density.

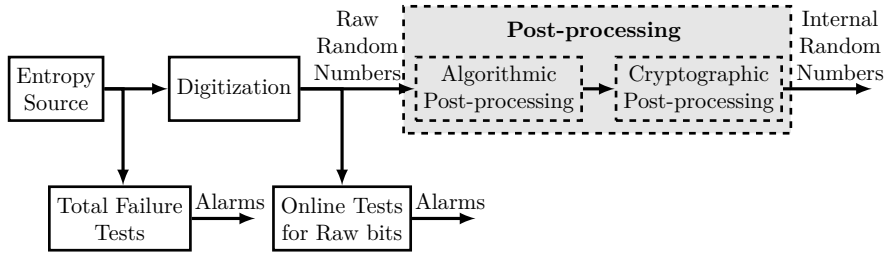


Figure 5.1: Post-processing in the TRNG.

This chapter is organized as follows. Section 5.2 analyzes an ancient divination algorithm from a classical Chinese book called I Ching, based on which people can quickly generate unbiased random numbers. We use this to warm up on the importance of post-processing. In Section 5.3, we discuss two types of post-processing techniques. Section 5.4 introduces a novel implementation of iterating Von Neumann post-processing. The content of Section 5.4 was originally published at [84]. We conclude this chapter in Section 5.5.

5.2 Yarrow Stalks Divination in I Ching

Generating good random numbers is never a gift of human beings. People can be easily biased consciously or unconsciously by culture, religion or experience. Unpredictability brings the mysterious property to divinations. Many tools are utilized in divinations to provide randomness such as: stones, shells, bones, coins and even Coffee residues.

In this section, we analyze an algorithm from an ancient oriental divination book: I Ching.

5.2.1 Algorithm Description and Simplification

Here is the description of the yarrow stalks divination from I Ching translated by Legge in [18]:

> *The numbers of the Great Expansion, (multiplied together), make 50, of which (only) 49 are used (in divination). (The stalks representing these) are divided into two heaps to represent the two (emblematic lines, or heaven and earth). One is then taken (from the heap on*

> *the right), and placed (between the little finger of the left hand and the next), that there may thus be symbolized the three (powers of heaven, earth, and man). (The heaps on both sides) are manipulated by fours to represent the four seasons; and then the remainders are returned, and placed (between) the two middle fingers of the left hand, to represent the intercalary month. In five years there are two intercalations, and therefore there are two operations; and afterwards the whole process is repeated.*

This original description of the yarrow stalks divination is in the language of astrology. To ease our following analysis and discussion, we simplify and formalize the divination into Algorithm 1.

Algorithm 1: Algorithm of the Yarrow Stalks Divination

Result: D

$N_1 \leftarrow 49$;
for $r = 1$ **to** 3 **do**
  $S_L = Rand(N_r)$;
  $S_R = N - S_L$;
  $S_R = S_R - 1$;
  if $S_L \,(mod\, 4) = 0$ **then**
    $S_T = 4$;
  else
    $S_T = S_L \,(mod\, 4)$;
  end
  if $S_R \,(mod\, 4) = 0$ **then**
    $S_T = S_T + 4$;
  else
    $S_T = S_T + (S_R \,(mod\, 4))$;
  end
  $S_T = S_T + 1$;
  $N_{r+1} = N_r - S_T$;
end
$D = N_4/4$.

In Algorithm 1, N_r, with $r \in 1, 2, 3$, denotes the number of remaining stalks before round r. After the third round, N_4 stalks stay in the diviner's hands. S_L and S_R denote the number of stalks in the left hand and the right hand, respectively. S_T represents the number of stalks taken away at the end of each round. We define a function as $Rand(N)$ which generates an integer from the range $[1, N-2]$. It simulates the action of splitting the heap into two parts. The output D has only four possible values $D \in [6, 7, 8, 9]$. Results 6 and 8 are

interpreted as *Yin*, while 7 and 9 are interpreted as *Yang*. To simplify notations, we denote Yin and Yang as binary 0 and 1 respectively.

Historically, because the imprecise description of Yarrow Stalks divination, there were two popular debates over the fifth Step of this algorithm. The first debate was described as: from which hand this extra stalk should be taken away. But people had quickly realized, without the influence of the handedness, the roles of two hands are exchangeable. The second debate was on whether the removal of the extra stalk is performed only in the first round or in all three rounds. This debate remains unsolved today, and we discuss it in this section.

We assume that an ancient divination algorithm, which survives the test of time, could generate almost unbiased outputs. Therefore, to resolve this debate, we have to analyze the probabilities of the binary output. The output probability depends on the splitting procedure of the heap, which is not clearly described in the original text. In other words, the probability distribution of the outputs of the function $Rand(N)$ remains unknown. We analyze Algorithm 1 under two different assumptions of this distribution.

Case Study I: Uniform Distribution

A straightforward assumption is that the random generation function $Rand(N)$ outputs an integer from the interval $[1, N-2]$ with an equal probability. The value of this probability can be computed as:

$$Pr(Rand(N)=1) = \cdots = Pr(Rand(N)=N-2) = \frac{1}{N-2}. \tag{5.1}$$

Tabel 5.1 shows possible S_T values at the end of the first round. By checking the result, there are two possible values of N_2 with two different probabilities: $Pr(N_2 = 40) = \frac{11}{47}$ and $Pr(N_2 = 44) = \frac{36}{47}$.

Table 5.1: Possible S_T values for different S_L, after the first round.

S_L	$\cdots$	20	21	22	23	24	25	26	27	28	$\cdots$
S_T	$\cdots$	9	5	5	5	9	5	5	5	9	$\cdots$

We compute all possible intermediate results and derive the probabilities of all possible final outputs. The result is shown in Table 5.2. As illustated in the result table, the output bias is 0.0097% when Step 5 is applied for all three rounds, and 7.5855% when Step 5 is only applied to the first round.

Table 5.2: Probabilities of the final output when the output of $Rand(N)$ is uniformly distributed.

Binary output	Output	Probabilities			
		Step 5 in All Rounds		Step 5 Only in Round 1	
0	6	0.051736	0.500097	0.012196	0.424145
	8	0.448361		0.411949	
1	7	0.288737	0.499903	0.123680	0.575855
	9	0.2111662		0.452175	

Case Study II: Normal Distribution

It is also reasonable to assume that people always try to evenly split the stalks into two heaps. We use a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with $\mu = N/2$ and $\sigma = 3$, to model this splitting operation. We calculate the output probabilities under this assumption, and show the result in Table 5.3. There is no bias when Step 5 is applied to all three rounds. However, when Step 5 is only applied to the first round, the output bias is equal to 6.25%. We note that the parameter $\sigma = 3$ is chosen arbitrarily, but similar results can be obtained for other values of σ.

Table 5.3: The output probabilities of the Divination Algorithm when the output of $Rand(N)$ is normally distributed

Output (binary)	Output	Probabilities			
		Step 5 in All Rounds		Step 5 Only in Round 1	
0	6	0.0625	0.5	0.015625	0.4375
	8	0.4375		0.421875	
1	7	0.3125	0.5	0.140625	0.5625
	9	0.1875		0.421875	

5.2.2 Conclusion: Resolved a Historical Debate?

Analyzing a divination using mathematical models may seem to be irrational and self-contradictory. However, it is reasonable to assume that the yarrow stalks divination was developed in countless religious practices. As a result, it is capable to produce fairly good and almost unbiased random outputs. Having Step 5 in all three rounds always produces output with a smaller bias than

having Step 5 only in the first round under two different assumptions. Although it is not possible to retrieve the original explanation of the algorithm, based on our analysis above, we believe that Step 5 is performed in all three rounds of this divination.

5.3 Two Types of Post-processing

In this section, we first review definitions of algorithmic post-processing and cryptographic post-processing. Then our focus lays on algorithmic post-processing algorithms. In addition to the algorithms presented in Chapter 2, two algorithmic post-processing algorithms are reviewed. We discuss a common mistake in designing post-processing.

We briefly introduced the post-processing techniques in Section 2.7. Two popular algorithmic post-processing methods, namely parity filter and von Neumann post-processing, were discussed. The requirements from two industrial standards were explained.

There is no clear definition to categorize a post-processing technique as algorithmic post-processing or cryptographic post-processing. As a rule of thumb, we regard post-processing algorithms, which utilize well studied cryptographic primitives such as block ciphers and hash functions, as cryptographic post-processing and regard other post-processing algorithms as algorithmic post-processing. Both algorithmic post-processing and cryptographic post-processing are optional, if the entropy source is strong enough and the generated raw bits contain sufficient entropy.

A cryptographic post-processing algorithm employs cryptographic primitives to extract randomness and to provide extra security anchors. It also provides safety nets when a total breakdown occurs in the noise source. In that case, the TRNG falls back to a pseudo random number generator if the post-processor is properly implemented. If one-way hash functions are used as post-processors, the analysis of the distribution of random bits becomes very difficult. These cryptographic algorithms are usually available in systems which put high security requirements on TRNGs.

In contrast to cryptographic post-processing, algorithmic post-processing aims to improve the entropy density and the statistical properties of raw random bits with a lightweight implementation. With a better understanding of the statistical defects of the raw bits, a specific algorithmic post-processing technique can be developed.

5.3.1 Algorithmic Post-processing

A resilient corrector A (n, m, t)-resilient function maps n bits to m bits such that if t input bits are fixed, there is no influence on the output.

Definition 14. [19] A (n, m, t)-resilient function is a function f mapping $\mathbf{F}_2^n$ to $\mathbf{F}_2^m$ such that for any coordinates $i_1, \ldots, i_t$ and for any binary constant $c_1, \ldots, c_t$ and for all $y \in \mathbf{F}_2^m$, we have

$$Pr(f(x) = y | x_{i_1} = c_1, \ldots, x_{i_t} = c_t) = 2^{-m},$$

where x_i with $i \notin i_1, \ldots, i_t$ verify $Pr(x_i = 1) = Pr(x_i = 0) = 0.5$.

Informally speaking, if up to any t bits of the input are deterministic or known by the adversary, as long as the rest of the bits of the input are random, the output of a resilient function remains perfectly unpredictable. With the knowledge of any t values of the input, an attacker can not make anything better than a random guess on the output.

The resilient function was first proposed to be used as the post-processing algorithm for ring oscillator based TRNGs in [94]. A lightweight implementation of the resilient corrector was introduced by Schellekens *et al.* in [89].

A linear code corrector A simple technique to construct a resilient function is using a linear code corrector. A linear code corrector was proposed in [28] to post-process independent but biased raw bits from a stationary entropy source. The proposed method is further studied and formalized by Patrick Lacharme in [55].

The first proposed algorithm in [28] has a fixed compression rate and maps a 16-bit input vector to a 8-bit output vector with lightweight hardware. We denote the input vector as $x = (x_0, \ldots, x_{15})$ and output vector as $y = (y_0, \ldots, y_7)$. The corrector H can be written in pseudocode:

$$H(X_1, X_2) = X_1 \oplus ROL(X_1, 1) \oplus X_2,$$

where $X_1 = (x_0, \ldots, x_7)$, $X_2 = (x_8, \ldots, x_{15})$. The operations $\oplus$ and $ROL(X, i)$ correspond to bitwise xor and circular left rotation of i bits, respectively.

Two further improved correctors are proposed in [28]:

$$H_2(X_1, X_2) = X_1 \oplus ROL(X_1, 1) \oplus ROL(X_1, 2) \oplus X_2,$$

$$H_3(X_1, X_2) = X_1 \oplus ROL(X_1, 1) \oplus ROL(X_1, 2) \oplus ROL(X_1, 4) \oplus X_2.$$

According to [28], if the bias of any input bits is ε, the lowest power of ε in the bias of output bits is three for H, four for H_2 and five for H_3.

These correctors have been investigated in [55] and it shows that the transformations of correctors H_1, H_2 and H_3 can be seen as matrix representations $G = (g_{i,j})$ of linear correctors:

$$\begin{pmatrix} g_{0,0} & \cdots & g_{0,15} \\ g_{1,0} & \cdots & g_{1,15} \\ \vdots & & \vdots \\ g_{7,0} & \cdots & g_{7,15} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ x_{15} \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_7 \end{pmatrix},$$

where the matrix G can be seen as a generator matrix of $[16, 8, 3]$, $[16, 8, 4]$ and $[16, 8, 5]$ linear codes. [55] further shows that a matrix G with a generator matrix of a $[n, k, d]$ linear code can be used to construct a post-processing algorithm which reduces the bias of the output bits to $2^{d-1}\varepsilon^d$.

5.3.2 A Common Design Mistake

A common mistake in designing a post-processing technique for TRNGs is using a non-compression scheme. Post-processing should reduce the weakness and not just simply transform one weakness into another. For example no-compression post-processors usually convert a biased random sequence into a sequence with bit dependency. Trivial tricks are used to mask the output of the random number generator in order to pass the statistical tests. For example, LFSRs are widely used to mask statistical defects in the generated raw numbers, such as [15], [23] and [25]. More sophisticated non-compression algorithms were also used to design post-processing, such as the E-transform proposed in [63]. As pointed out by Dichtl in [28], a bijective algorithm such as E-transform cannot extract entropy from its input bits, but applying a bijection to raw random bits also does not do any harm.

5.4 Implementation of Iterating Von Neumann

The content of this section is based on our published work at HOST2016 [84].

In this work we present a design methodology and hardware implementations of lightweight post-processing modules for debiasing random bit sequences. This work is based on the iterated Von Neumann (IVN) procedure. Given the high throughput, low area and low energy demands of good random number

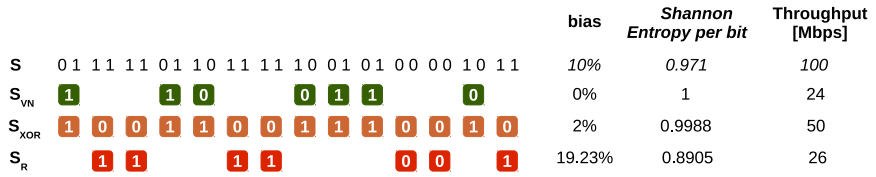


Figure 5.2: Principle of operation of the iterated von Neumann post-processing.

generators on IoT devices, there is a need for lightweight, high-throughput post-processing techniques. We present a method to maximize the efficiency of IVN for applications with area and throughput constraints.

Von Neumann (VN) post-processing [109] is a famous debiasing method. A brief introduction of it can be found in Section 2.7. This method guarantees full entropy output provided that there are no dependencies between the raw bits. However, a lot of entropy is wasted in the process, causing a substantial reduction of throughput. An extension of the VN procedure that achieves improved efficiency of entropy extraction was proposed by Peter Elias in [31]. The theoretical paper of Yuval Peres [70] proposes a more practical solution based on IVN post-processing in order to improve the efficiency. The central idea of IVN is to re-apply the VN post-processing on the outputs that are discarded in the conventional VN. The extracted entropy converges to the theoretical maximum of entropy, by iterating the VN procedure to infinity.

We have three main contributions in this work. First, we are the first to explore the incomplete binary tree topologies of processing elements for throughput or area optimization. Second, we provide an algorithm for finding the optimal circuit structure given the bias value and the hardware area budget. Last, we explore the design space for various bias values.

5.4.1 IVN Post-processing

Here we provide an overview of the IVN algorithm proposed in [70]. We use the following example to explain the principle of operation. A $100Mb/s$ entropy source produces bits with bias equal to $\varepsilon = 10\%$. This source produces $97.1Mb/s$ of Shannon entropy. At the top of Figure 5.2, we see an example of a sequence S generated by this source. We can decompose this sequence into three sequences which are denoted as S_{VN}, S_{XOR} and S_R. The S_{VN} sequence (shown in green)

is constructed by applying the classical Von Neumann procedure on the input bits. The S_{VN} sequence is always unbiased and, in the presented example, its throughput is $24Mb/s$, which can be computed by:

$$Throughput(S_{VN}) = (0.5 - \varepsilon)(0.5 + \varepsilon) \cdot Throughput(S). \tag{5.2}$$

When $\varepsilon = 0$, $Throughput(S_{VN})$ reaches its maximal value.

The S_{XOR} sequence (shown in orange) is generated by xoring two consecutive bits. The bias of the S_{XOR} sequence can be calculated by $2 \cdot \varepsilon^2$. The sequence S_{XOR} has a fixed throughput which is half of the original bit sequence.

The S_R sequence (shown in red) is the residual sequence that contains all information that is not extracted by the other 2 sequences. It is generated by producing a 1 whenever a block 11 is detected and producing a 0 whenever 00 is detected in the original sequence. The bias of this sequence is $2 \cdot \varepsilon/(1 + 4 \cdot \varepsilon^2)$. The average throughput of sequence S_R can be computed by $Throughput(S_R) = (0.25 + \varepsilon^2) \cdot Throughput(S)$.

It is clear that no information entropy is lost during the decomposition because the original sequence S can be reconstructed from the sequences S_{VN}, S_{XOR} and S_R.

For classical Von Neumann post-processing, only S_{VN} is used as the output, while the other two sequences are discarded. These two sequences contain all the discarded information.

According to [70], these three sequences are independent and the Von Neumann procedure can be applied again on the discarded information (S_{XOR} and S_R). In the presented example, S_{XOR} has a throughput of $50Mb/s$ and more than 0.99 bits of Shannon entropy per output bit. Note that this sequence has a lower bias than the original one. The sequence S_R, on the other hand has a higher bias. However, a substantial amount of information can be extracted from this sequence as well. With a throughput of $26Mb/s$ and around 0.89 bits of Shannon entropy per output bit there is a Shannon entropy throughput of more than $23Mb/s$.

The procedure can be iterated. S_{XOR} and S_R can be further decomposed into three components and the same procedure can be repeated. By iterating the procedure to infinity, the theoretical maximum of $97.1Mb/s$ can be achieved, i.e. all available entropy can be extracted.

Table 5.4: Elementary IVN operation.

S		S_{VN}	S_{XOR}	S_R
0	0	-	0	0
0	1	1	1	-
1	0	0	1	-
1	1	-	0	1

5.4.2 IVN Optimization

In this section, we explore the efficiency of IVN, using finite computational resources.

Elementary Operation

VN is based on processing pairs of consecutive bits. The elementary operation of VN consists of reading a pair of bits and producing one or no output bits as a result. The elementary operation of IVN is only slightly more complex. Processing one pair of input bits results in exactly 2 bits at the output (one at S_{XOR} and one at either S_{VN} or S_R) as shown in Table 5.4. A full IVN implementation is obtained by connecting these elementary operations in a tree-like structure.

Our work shows how to maximize the amount of extracted entropy using a fixed amount of elementary operations. The results can be applied to both hardware and software implementations. The elementary operations have a fixed cost in terms of the number of cycles for software operations or the number of processing modules operating in parallel for hardware implementations. In hardware designs, the proposed method can be used to maximize the throughput within the given area constraints. In this section, we use hardware implementations to illustrate the benefits of the proposed method.

Hardware Implementation of IVN Module

Figure 5.3 shows the architecture of the hardware module performing the IVN elementary operation. The design has two data inputs: the signal S, which represents the bit value, and the signal S_{Valid}, which indicates if the value of S is valid. The output consists of 3 data signals (S_{VN}, S_{XOR} and S_R) and 3 signals to indicate when the output is valid ($S_{VN,Valid}$, $S_{XOR,Valid}$ and $S_{R,Valid}$). The hardware implementation of this module requires only 2 flip-flops and several

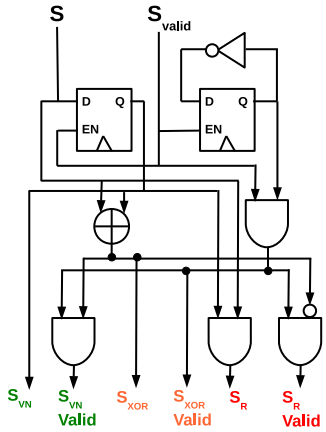


Figure 5.3: Processing element architecture.

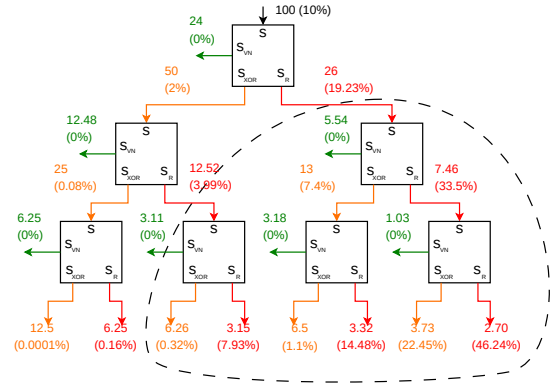
logic gates. On a Xilinx Spartan-6, the implementation consumes only 2 slices. An ASIC implementation using the open-source NanGate 45nm standard cell library consumes only 25 Gate Equivalents (GEs).

An Example of an Optimization Strategy

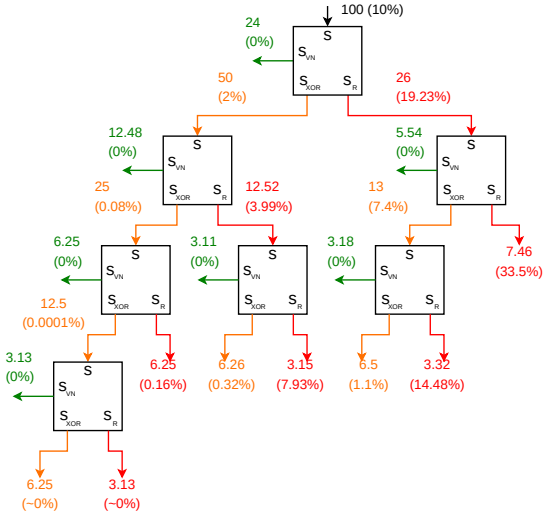
To illustrate the benefits of the IVN optimization, we use the example of the biased generator in Figure 5.2.

A $100Mb/s$ steam is post-processed using the IVN procedure with three iterations. Figure 5.4a shows the post-processing block implemented using seven processing elements. The throughput and bias of each branch is indicated in the figure.

The S_{VN} output of each processing element is directly used as the output, while the other 2 outputs are sent for recycling. It is immediately obvious that the root node (classical 1-stage VN), contributes with $24Mb/s$, while the lower right element contributes with only $1.03Mb/s$. Moreover, the elements enclosed within the dashed line contribute with only $9.75Mb/s$ (out of $55.59Mb/s$) while consuming more than 50% of the area. Throughput optimization can be performed by finding the optimal post-processing structure with the given number of processing elements. This optimal structure may depend on the



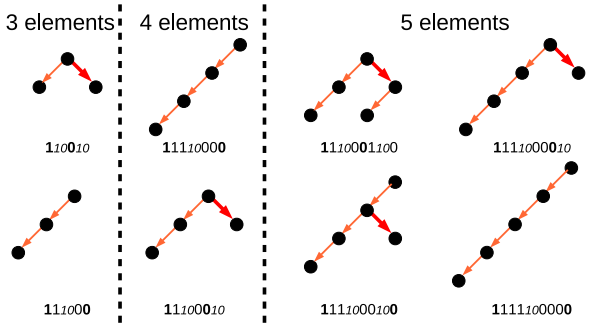
(a) Classical IVN with 3 iterations. The part enclosed within the dashed line contributes very little to the total throughput.



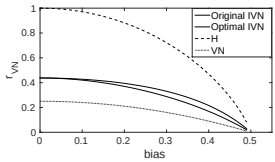
(b) IVN structure with 7 processing elements, optimized for high throughput.

Figure 5.4: Different IVN post-processing structures. The numbers indicate the throughput [Mb/s] and bias of each branch.

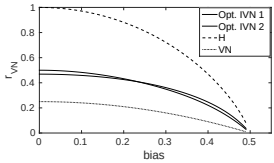
bias. The optimal structure for input data with 10% bias and seven processing elements is shown in Figure 5.4b.



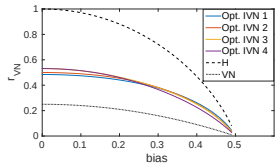
(a) Binary trees



(b) Throughput efficiency of 3-element structures.



(c) Throughput efficiency of 4-element structures.



(d) Throughput efficiency of 5-element structures.

Figure 5.5: Optimal binary trees and the corresponding throughput efficiency.

Binary Tree Representation

Note that the post-processing block using classical IVN has the structure of a complete binary tree of n levels where n is the number of iterations. This block can be implemented using exactly $2^n - 1$ processing elements. The optimized solution shown in Figure 5.4b also has the structure of a binary tree. However, this tree is incomplete.

The main contribution of our work is exploring the incomplete binary tree structures for IVN and providing an algorithm for finding the optimal structure for a specific bias value. Our results show that the classical IVN (complete binary tree) is rarely the optimal solution for any bias value. Using incomplete

binary trees enables us to use any arbitrary number of processing elements, not only $2^n - 1$.

To simplify the notation of different post-processing architecture, we use the representation of binary trees. Each node of the binary tree architectures represents one processing element. The left branch of the node corresponds to the output S_{XOR} and the right branch corresponds to the output S_R. We use a string of 1s and 0s to note the binary tree structure. An empty tree is represented using the empty string '_'. Any other tree is represented using the following recursive definition:

$$< Tree > = \ 1 < LTree > 0 < RTree >, \tag{5.3}$$

where $< LTree >$ and $< RTree >$ stand for the sub-trees connected to the left (S_{XOR}) and right (S_R) branch of the root node respectively. For example, a tree consisting of only one node is represented using a string '10'. More examples of binary trees and their string representations are shown in Figure 5.5a. For better readability, digits corresponding to the root node are shown using bold font and the digits corresponding to the leaf nodes are shown using italic font and a smaller font size.

When the number of elements is small, it is possible to find the optimal binary tree using exhaustive search. The optimal post-processing methods using 3, 4 and 5 processing elements were explored and the results are summarized in Figure 5.5. The throughput efficiency (defined as the ratio of the input throughput to the output throughput) of the 3-element binary trees (shown in Figure 5.5a) are computed for different bias values and the results are summed up in Figure 5.5b. The dashed line shows the Shannon entropy available in the sequence (maximal throughput efficiency) and the dotted line shows the efficiency of the Von Neumann procedure. Both binary trees show significant improvement of the post-processing efficiency over the VN procedure. The complete 3-element binary tree represents the classical IVN with one iteration of recycling, it is therefore denoted as *Original IVN*. The other binary tree is denoted as *Optimal IVN*. It results in the highest efficiency for all bias values.

When using more than 3 elements, the optimal tree depends on the bias value. Two optimal trees can be constructed using 4 elements as shown in Figure 5.5a. One of these trees is optimal for a bias lower than 25% and the other one for higher bias values, as shown in Figure 5.5c. Using 5 processing elements, we can construct 4 binary trees that are optimal for different bias intervals. Other 5-element trees are not optimal for any bias value. Therefore, the designer can choose the optimal post-processing configuration based on the bias of the original sequence. If the bias is not known at design time or it cannot be

precisely estimated, the wrong structure can be chosen. However, this is not a big problem because the structure that is optimal for one bias value has close-to-optimal performance for all bias values. This can be clearly seen in Figures 5.5c and 5.5d where all graphs corresponding to the optimal structures are clustered together. Therefore, the wrong estimation of the bias does not cause high throughput reduction as long as the chosen structure is optimal for some bias value.

Finding the Optimal Trees

For a higher number of elements, it becomes increasingly difficult to find the optimal binary tree by exploring all combinations. We propose an algorithm for finding the optimal tree given the bias and the number of processing elements. The intention is that the designer computes the available number of elements from the area budget and uses the algorithm to find the post-processing structure that extracts the most entropy.

Algorithm 2: Algorithm for finding optimal trees.

Function `FindTree`(b,n)**:**
  if $n = 0$ **then**
    $r = 0$;
    $T =' \_'$;
  else
    $r_{VN} = 0.25 - b^2$;
    $r_{XOR} = 0.5$;
    $r_R = 0.25 + b^2$;
    $b_{XOR} = 2b^2$;
    $b_R = (2b)/(1 + 4b^2)$;
    $r = 0$;
    for $i = 0$ **to** $n - 1$ **do**
      $(new\_r_{XOR}, new\_T_{XOR}) =$ `FindTree`(b_{XOR}, i);
      $(new\_r_R, new\_T_R) =$ `FindTree`$(b_R, n - 1 - i)$;
      $new\_r = r_{VN} + r_{XOR} \cdot new\_r_{XOR} + r_R \cdot new\_r_R$;
      if $new\_r > r$ **then**
        $r = new\_r$;
        $T =' 1' new\_t_{XOR} '0' new\_t_R$;
      end
    end
  end
  return (r,T)
end

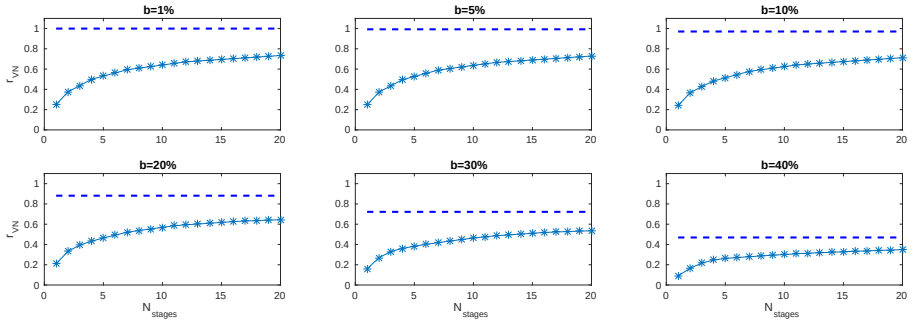


Figure 5.6: Throughput efficiency given the number of elements for different bias values.

The presented recursive function takes bias b and the number of elements n as input and produces the maximal throughput rate r and the corresponding binary tree structure T at the output. The optimal binary tree is constructed by assigning the root node and allocating the remaining $n-1$ nodes to the left and the right sub-trees. All n combinations are explored in a loop and the optimal sub-trees are found using the $FindTree$ function. The recursion stops when the number of nodes is 0. The combination resulting in maximal throughput is returned. The tree structure is reported as a string of ones and zeros according to the definition 5.3.

Figure 5.6 shows the throughput efficiency for different bias values and the number of elements ranging from 1 to 20. The theoretical maximum throughput is shown using dashed lines, which corresponds to the Shannon entropy. The same trend is observed for all bias values: 1-stage post-processing (VN) extracts around one quarter of the available entropy. Optimal configurations using 5 processing elements always extract more than half of the available entropy. Adding more elements improves the throughput at smaller and smaller steps. After 15 elements are used, the additional contribution of each added element is less than 1%.

To summarize our contributions, we have proposed a lightweight hardware module for debiasing binary data based on the iterated Von Neumann procedure, where the incomplete binary tree structures of the VN iteration are used. This approach enables us to explore the tradeoff between area consumption and penalties in throughput reduction.

It was found that the optimal post-processing structure depends on the bias of

the input sequence, therefore the designer needs to estimate the bias of the raw numbers at the design phase. However, bias estimation does not have to be very precise, especially when using a small number of elements. For example, for finding the optimal 4-element structure the designer only needs to determine if the expected bias is below or above 25%. Another observation is that the structures that are optimal for some bias value are close-to-optimal for any bias value. Therefore, a wrong bias estimation at the design phase does not result in a high penalty in throughput reduction.

We note that restrictions of VN and IVN procedures also apply to the proposed methods, i.e. the presented post-processing structures should only be used on sequences without dependencies between the generated bits.

5.5 Summary

Post-processing modules are used to improve statistical and security properties of raw random numbers. In this chapter, the difference between algorithmic post-processing and cryptographic post-processing are discussed. Two additional algorithmic post-processing algorithms are presented. A common mistake in designing the post-processor is discussed. An exploration on implementation dimensions of the IVN is given at the end of this chapter.

Chapter 6

Embedded tests: the black-box tests approach

You can't make what you can't measure,
because you don't know when you've got it made.

-Dr.Irving Gardner

Content Sources

Yang, B., Rožić, V., Mentens, N., Dehaene, W., and Verbauwhede, I. Embedded HW/SW Platform for On-the-Fly Testing of True Random Number Generators, In *Design, Automation and Test in Europe (DATE)*, 2015.
Contribution: Main author, contribute to the test simplification and responsible for the implementation and the experiment.

Yang, B., Rožić, V., Mentens, N., and Verbauwhede, I. On-the-Fly Tests for Non-Ideal True Random Number Generators, In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015.
Contribution: Main author, contribute to the test simplification and responsible for the implementation and the experiment.

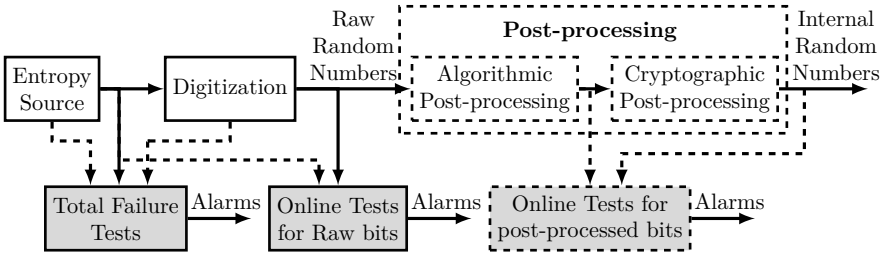


Figure 6.1: Embedded tests in the TRNG.

6.1 Introduction

Chapter 6 and 7 cover the topic of embedded tests. The embedded tests indicated as the gray part of Figure 6.1, consist of total failure test modules, online test modules for raw random numbers and online test modules for internal random numbers. Section 6.2 discusses different types of embedded tests. Section 6.2.1 presents different methodologies for designing online tests. Two implementations of black-box statistical testing for IID sources and non-IID sources are introduced in Section 6.3 and Section 6.4, respectively. The content of Section 6.3 and Section 6.4 was originally published in [122] and [124]. Section 6.5 summarizes this chapter.

6.2 Types of Embedded Tests

Embedded tests are necessary for on-the-fly monitoring the working status of an entropy source. As discussed in Section 2.8, embedded tests are required by industrial standards. Normally, embedded tests are implemented adjacently to entropy sources to avoid attacking and eavesdropping on generated random numbers. Two different types of embedded tests are commonly used, namely total failure test and online test.

Total Failure Test The total failure test is designed to detect a totally broken down of a digital noise source. If a digital noise source is totally break-down, the entropy of further generated random numbers is equal to zero. Thus a total failure test module must quickly respond to the failure: stopping the TRNG from outputting compromised data and informing the user via setting alarms.

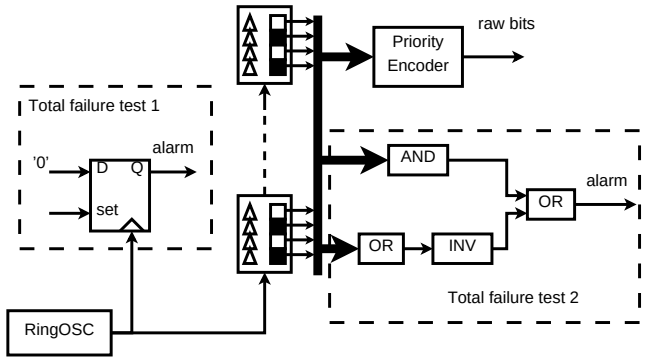


Figure 6.2: Two total failure tests for the DC-TRNG.

As a case study, we implement two total failure test modules for the DC-TRNG introduced in Section 4.3. The architecture of these two total failure tests is depicted in Figure 6.2. The *total failure test 1* is implemented as an edge detector, which examines the continuation of toggling in a ring oscillator. The used DFF is periodically set to '1' and its output will remain to be '1' if there is no transition at the clock input.

The *total failure test 2* works on the signal captured in the delay chain. We implemented the following Boolean function as total failure test 2:

$$alarm = (\wedge delaychain) \ \vee \ \neg(\vee delaychain)\,.$$

The sampled dated in the delay chain is referred to as a vector *delaychain*. Unary operations $\vee$ and $\wedge$ applied on the vector are defined as applying 'AND' and 'OR' on all bits of the vector to generate a single-bit result. Operation $\vee$ used between two Boolean bits is defined as a conjunction of these two bits. An alarm signal is triggered when the bits from the vector *delaychain* are all zeros or all ones. This total failure test can be further simplified based on an assumption of the minimal period of the free-running ring oscillator.

Online Test An online test module is designed to detect many kinds of failures in an entropy source. Typical failures include but are not limited to aging effects, a working environment outside the specification, or active attacks. Together with total failure test modules, online test modules should be running non-stop when entropy sources are enabled. Similar to other hypothesis testing, the false alarm rate is a design parameter of online testing, which should be carefully selected according to particular application scenarios.

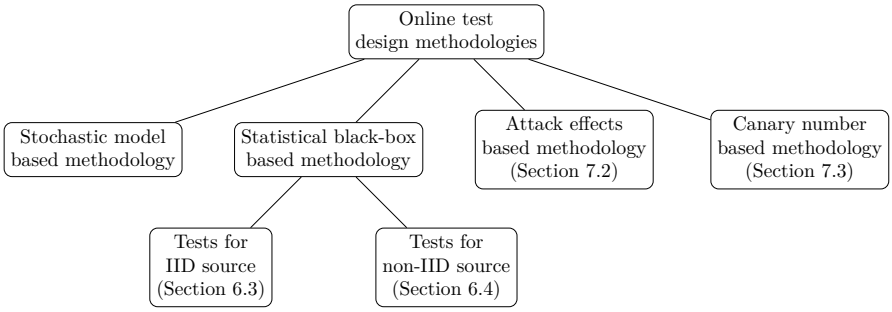


Figure 6.3: Design methodologies for online tests.

The rest of this chapter and Chapter 7 focus on the topic of online tests.

6.2.1 Design Methodologies for Online Tests

Existing online tests can be roughly categorized into four different types based on the applied design methods. The principle of the first type of design methodology is deriving online tests from the stochastic model of the target entropy sources. This method is followed when the applicable stochastic model exists for a particular TRNG and effects of potential attacks are well studied. Design examples of online tests based on stochastic models are [5], [37] and [52].

The design methodology called "TRNG On-the-fly Tests for Attack detection using Lightweight hardware (TOTAL)" was proposed by Yang et al. [123]. As its full name indicates, TOTAL is an empirical design methodology based on analyzing attack effects over particular TRNGs. The details of the TOTAL methodology are given in Section 7.2.

The canary number based methodology was introduced by Rožić et al. in 2016 [85]. The canary entropy source and the canary entropy extractor are used for early-warning attack detection before the generated random numbers are compromised. Section 7.3 presents a detailed discussion on the canary number methodology.

Statistical black-box testing is based on observing the generated bits without accounting for the internal structure of the entropy source. The raw random numbers or internal random numbers are collected and used for searching specific patterns that imply a bias from an ideal probability or bit dependencies.

There are pros and cons of using statistical black-box tests as online tests.

Some existing TRNGs lack an applicable stochastic model, and it is not always easy to derive a useful stochastic model for every TRNG. Statistical black-box testing can be used for the online testing of TRNGs without stochastic models. Black-box testing works directly on the generated data. If no alarm is triggered during operation, the generated random numbers are free of examined statistical defects. However, like other hypothesis testing, conclusions about all generated and to-be-generated random numbers are drawn based on a limited amount of generated random numbers. If we choose the null hypothesis H_0 that the sequence being tested is random, there is a non-zero false rejection rate.

We would like to note that the effectiveness of using statistical black-box testing is relatively limited. The absence of a triggered alarm from the implemented black-box online tests does not guarantee sufficient true randomness, because simple post-processing could hide all statistical flaws in the generated random numbers.

In the remaining two sections of this chapters, we proposed two online test implementations based on statistical black-box testing. The contents of these two sections were published before at [122] and [124] in 2015. In Section 6.3, we implement nine out of all thirteen tests from the NIST-22 [87] test suite as the online test for internal random numbers. A hardware (HW)/software (SW) co-designed architecture was proposed to provide both efficiency and flexibility. In Section 6.4, we implement the tests from the NIST-90b [99] test suite as the online test for raw random numbers. Instead of generating a one-bit alarm signal, we provide a rough estimation of the entropy level based on the test results.

6.3 Black-Box Tests on Internal Random Numbers

In this section we present a HW/SW co-design platform for performing online tests. We provide three different versions that operate on bit sequences of different lengths. The random sequence is read bit by bit. Tests are selected from the NIST battery of tests [87]. This test suite is not designed for on-the-fly testing and in general, it is not suitable for this purpose due to the high latency and high computational requirements. However, a subset of the test suite can be optimized for compact implementation and low latency. We have selected those tests from the suite that can be optimized for compact implementation in order to make them suitable for on-the-fly monitoring.

The splitting of calculations between hardware and software is carefully carried out to enable resource sharing between different tests for minimizing the hardware consumption. Each test is carried out using two types of operations.

The first type consists of operations on the incoming bits such as: counting ones and zeros, finding the maximal longest run of the same value, counting the appearance of a given pattern or keeping track of a random walk. These operations are implemented in hardware using counters, comparators and registers, and they are performed while the TRNG is active.

The second type of operations is used for generating the final security assessment. These operations, which include addition, multiplication, squaring and comparison, are performed on the obtained counter values after the sequence has been generated. These operations are only required once in a while, therefore it makes sense to reduce the datapath area by having a single instance of these operations shared between different tests. These operations are very common and available on most microcontrollers or soft-core processors. For this specific work, we assume that the TRNG and tests operate in a heterogeneous system which consists of both microcontrollers and configurable fabric. This is different from the standard approach where the hardware blocks complete the full test and report the failure by activating the alarm signal. Our approach assumes that the microcontroller reads the counter values from the hardware blocks and performs the remaining operations. This approach makes probing attacks difficult because there is no single alarm signal, but rather a set of numerical values that are being transmitted.

Another advantage of HW/SW co-design is the flexibility. Each test can be carried out with a critical value α, indicating the level of significance, where the recommendations by NIST are that α is chosen in the interval $[0.001, 0.01]$. The presented hardware blocks analyze the generated sequence and provide the results that do not depend on α. The level of significance only figures in the operations performed by the software which can be easily programmed.

6.3.1 Statistical Tests

The idea behind statistical tests is to start with the hypothesis that the RNG is ideal, we will denote this hypothesis as H_0. A statistical test is used to measure a property of the generated sequence, for example the longest run of zeros. Based on the test result, the hypothesis H_0 is either accepted or rejected. If the sequence is indeed random, it is still possible to reject H_0 with some small probability. This is known as a type 1 error. The probability of this error is a design parameter called the level of significance and denoted as α. The other type of error that can occur is accepting H_0 when the sequence is not random. This is known as a type 2 error, and the purpose of the test is to minimize the probability of this error. As summed up in table 6.1, The NIST sp 800-22 suite consists of 15 statistical tests which estimate different properties of a random

Table 6.1: The NIST sp 800-22 test suite. Some tests are suitable for HW implementation.

TEST	HW/SW co-design
1. Frequency (Monobit) Test	Yes
2. Frequency Test within a Block	Yes
3. Runs Test	Yes
4. Test for Longest-Run-of-Ones in a Block	Yes
5. Binary Matrix Rank Test	No
6. Discrete Fourier Transform (Spectral) Test	No
7. Non-overlapping Template Matching Test	Yes
8. Overlapping Template Matching Test	Yes
9. Maurer's "Universal Statistical" Test	No
10. Linear Complexity Test	No
11. Serial Test	Yes
12. Approximate Entropy Test	Yes
13. Cumulative Sums (Cusums) Test	Yes
14. Random Excursions Test	No
15. Random Excursions Variant Test	No

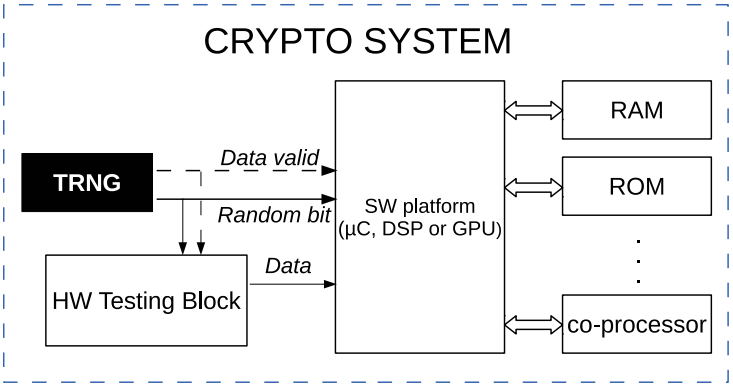


Figure 6.4: Testing environment

variable. These tests are originally designed to evaluate the statistical properties of PRNGs, but they are also used for the evaluation of TRNGs.

6.3.2 Implementation

As depicted in Figure. 6.4, the crypto system consists of a TRNG, the HW Testing Block, at least one component that performs arithmetic (microcontroller, DSP or a Graphics Processing Unit (GPU)), and possibly other components such as embedded RAM and crypto co-processors. We split the testing implementation into two parts: hardware and software. To reduce the area and power consumption of the HW testing block, it is implemented in a compact manner using only basic components such as counters, comparators and registers, while all power-hungry arithmetic operations are moved to the software part. Squaring, multiplication, logarithm and comparison with the precomputed constants are performed in software.

As pointed out in [34], on-the-fly tests should be active while the TRNG is working in order to ensure that they are always operating in the same conditions as when they are being tested. The proposed approach allows us to run the hardware block all the time and check the test results only when needed.

HW/SW Partitioning

All calculations needed for the statistical tests are divided between a HW testing block and software executed on the SW platform. The boundary between hardware and software is chosen to minimize the hardware block. Each test consists of operations that have to be executed while the bit stream is generated (the HW part) and basic arithmetic operations (SW part). It is also important to minimize the amount of data that need to be transferred from HW to a co-processor in order to simplify the interface between the two modules.

We have selected nine tests from the test suite which are suitable for this type of implementation, as indicated in Table 6.1. The remaining six tests either require too much data storage in the HW module which would result in large area, too complex operations in the software part which would result in high latency, or too much data to be transferred between the two modules which would result in an overly complicated interface.

Table 6.2 presents how the required operations were divided between HW and SW. The middle column (Hardware), lists all the values that are computed by the HW module and transferred to the co-processor. We used the following notation:

- N_{ones} - total number of ones.

Table 6.2: Calculation partitioning between hardware and software.

Test	**Hardware**	**Software**
Test 1	N_{ones}	Comparison operations
Test 2	$\varepsilon_1, \varepsilon_2 ... \varepsilon_N$	$\sum_{i=0}^{N} (\varepsilon_i - \frac{M}{2})^2$
Test 3	N_{ones}, N_{runs}	Comparison operations
Test 4	$\nu_{runs,1}, \nu_{runs,2} ... \nu_{runs,N}$	$\sum_{i=0}^{N} \nu_{runs,i}^2 (\frac{1}{\pi_i})$
Test 7	$W_1, W_2 ... W_N$	$\sum_{i=1}^{N} (2^m W_i - \mu 2^m)^2$
Test 8	$\nu_{temp0} ... \nu_{tempN}$	$\sum_{i=0}^{N} \nu_{temp,i}^2 (\frac{1}{\pi_i})$
Test 11	$\nu_{0000}, \nu_{0001}, ... \nu_{1111}$, $\nu_{000}, \nu_{001}, \nu_{010} ... \nu_{111}$, $\nu_{00}, \nu_{01}, \nu_{10}, \nu_{11}$	$\Psi_m^2 = \frac{2^m}{n} \sum_{i_1 ... i_m} \nu_{i_1 ... i_m} - n$ $\Psi_{m-i}^2 = \frac{2^{m-1}}{n} \sum_{i_1 ... i_{m-1}} \nu_{i_1 ... i_{m-1}} - n$ $\Psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1 ... i_{m-2}} \nu_{i_1 ... i_{m-2}} - n$ $\nabla \Psi_m^2 = \Psi_m^2 - \Psi_{m-1}^2$ $\nabla^2 \Psi_m^2 = \Psi_m^2 - 2\Psi_{m-1}^2 + \Psi_2^2\}$
Test 12	$\nu_{0000}, \nu_{0001}, ... \nu_{1111}$, $\nu_{000}, \nu_{001}, \nu_{010} ... \nu_{111}$	$\sum_{i(m=3)} \frac{\nu_i}{n} log \frac{\nu_i}{n} - \sum_{i(m=4)} \frac{\nu_i}{n} log \frac{\nu_i}{n}$
Test 13	$S_{max}, S_{min}, S_{final}$	$max(S_{final} - S_{min}, S_{max} - S_{final})$ Comparison operations

- N_{runs} - total number of runs in a sequence. A single run is a consecutive appearance of a single value (0 or 1).
- ε_i - number of ones within a block of data.
- M - length of a single block of data.
- N - number of data blocks.
- ν_i - number of runs within a block of data.
- W_i - number of non-overlapping appearances of a given template within a block of data.
- $\nu_{temp,i}$ - number of data blocks in each category depending on the number of overlapping appearances of a given template.
- m - length of a template.

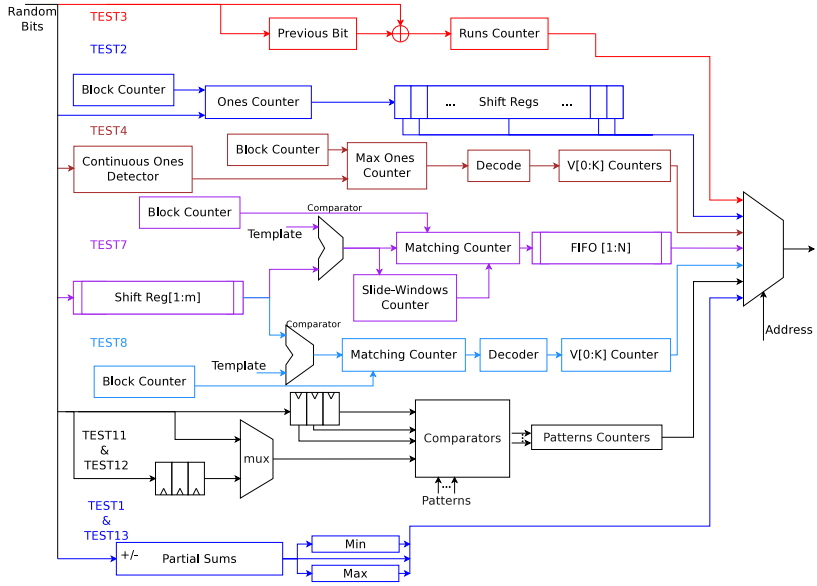


Figure 6.5: Hardware module containing all tests

- $S_{max}, S_{min}, S_{final}$ - partial sums obtained by the up/down counter. The maximal, the minimal (negative) and the final value are recorded.

All other values are test-specific precomputed constants.

Software routines operate on these obtained data values. As can be seen from the last column of Table 6.2, the required operations for software are basic multiplication, addition and comparison operations. The only difficulty is implementing the $x \cdot log(x)$ function needed for the approximate entropy test, which will be described in more detail in Section 6.3.2.

HW Implementations

We have implemented several versions of the HW testing block. Figure 6.5 shows the implementation of the largest version that contains all 9 tests and operates on a sequence of 2^{20} bits. Clock and enable signals are omitted for better clarity. The global bit counter is also not shown. This counter is used to count the total number of bits in order to detect the end of the sequence. A memory-mapped interface is implemented using a large multiplexer, where

the 7-bit address is used as a select signal. Since this interface contributes significantly to the overall area, we can save resources by reducing the number of transmitted values. After receiving each random bit from the generator, all update calculations finish within one clock cycle.

This type of unified implementation enables us to share more resources between different tests to obtain a higher area reduction. Four tricks are used to reduce the area of this module:

- **Omitting a redundant counter:** Tests 1 and 3 use the total number of 1's in a sequence to compute the test result. However, it is possible to obtain this result without this counter. For the implementation of test 13, an up/down counter is used to keep track of the random walk. The total number of ones can be calculated from the final value of this counter. For this reason, the counter of ones can be omitted.
- **Block detection:** For the implementations of tests 2, 4, 7 and 8 it is necessary to divide the sequence into sub-blocks and look for certain properties in each block (the total number of ones, longest run of the same bit value, and occurrences of different patterns). We have selected test parameters such that block lengths are equal to powers of 2, which enables us to detect the beginning and the end of each block by simply observing specific bits of the global bit counter.
- **Unified implementation:** The approximate entropy test (test 12) uses the number of all 4-bit and 3-bit patterns in a sequence. These values are already provided by the serial test (test 11) implementation, therefore there is no need for the separate implementation of test 12.
- **Shared shift register:** The non-overlapping and the overlapping template match tests compare the generated numbers with the pre-defined 9-bit patterns. The same shift register can be used for both tests.

SW Implementations

Typical software implementations of statistical tests operate by computing the P-value, and comparing it with the required level of significance. The P-value is the probability that an ideal random number generator produces a sequence which is worse than the measured sequence with respect to the metric used by the test (for example bias, or the longest run of the same bit value). This is a computationally intensive task because calculating P-values requires complicated functions such as *erfc* and the *gamma* function. We use a simple approach for computing the inverse functions of the critical value and

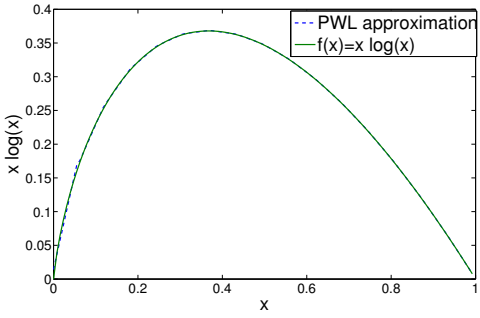


Figure 6.6: PWL approximation of the function $x \cdot log(x)$

storing the precomputed constants, thereby skipping the most computationally intensive step. This approach is also used in [95, 106, 108].

As shown in Table 6.2, implementations of tests 1 and 3 only require comparison operations. Test 1 only compares N_{ones} with the critical value. For test 3, the critical values for N_{runs} are stored in the program memory as constants and they depend on N_{ones}. The SW procedure first checks the interval where N_{ones} belong and based on the result, compares N_{runs} with the appropriate constant. A similar approach was used for the FPGA implementation in [108].

Other tests require simple calculations on the obtained values before the comparison. The required operations are comparison, addition (subtraction), multiplication and squaring. A typical processor has dedicated instructions for these operations. The main difficulty is related to the implementation of the Approximate Entropy Test, which requires the implementation of the function $x \cdot log(x)$. In order to avoid computationally intensive logarithm calculation, we implement this function using piece-wise linear approximations with 32 segments. As can be seen on Figure 6.6, the approximation (dash line) is almost indistinguishable from the function (full line) resulting in less than 3% error.

6.3.3 Implementation Results

We provide 8 different designs which cover nine tests from the NIST test suite. We implement our hardware designs in the Verilog Hardware Description Language (Verilog HDL) and use Mentor Graphics Modelsim SE PLUS 6.6d for functional simulation. All proposed hardware designs are synthesized using Xilinx ISE14.7 on Spartan-6 XC6SLX45 FPGA and Synopsys Design Compiler

Table 6.3: Implementation results

	n=128		n=65536			n=1048576		
	light	medium	light	medium	high	light	medium	high
test1	•	•	•	•	•	•	•	•
test2	•	•	•	•	•	•	•	•
test3	•	•	•	•	•	•	•	•
test4	•	•	•	•	•	•	•	•
test7				•	•		•	•
test8							•	•
test11		•			•			•
test12		•			•			•
test13	•	•	•	•	•	•	•	•
FPGA:								
Slice	52 0.8%	149 2.2%	144 2.1%	168 2.5%	377 5.5%	173 2.5%	291 4.3%	552 8.1%
FF	110	329	307	375	836	379	585	1156
LUTs	158	471	420	454	1103	546	828	1699
MaxFreq (MHz)	156	147	143	136	133	125	122	121
ASIC:								
GE	1210	3632	3243	3850	8983	4013	5993	12416
SW:	16-bit instructions							
ADD	9	153	108	122	266	130	358	890
SUB	8	14	16	24	30	24	40	50
MUL	4	28	24	24	48	15	47	91
SQR	8	36	14	22	50	23	45	101
SHIFT	0	3	0	8	11	0	8	11
COMP	22	28	42	44	50	34	42	48
LUT	0	24	0	0	24	0	0	24
READ	10	24	18	22	50	21	35	91

D-2010.03-SP4 with UMC's 0.13μm.1P8M Low Leakage Standard cell Library with typical values (voltage of 1.2V and temperature of 25 °C).

As with most practical implementations, there is no golden way to the perfect system in a generic way, and different applications demand different design trade-offs. As shown in Table 6.4, we propose 8 different implementations which support three different input lengths: 128/65536/1048576 bits. For hardware design, with the merit of a compact hardware footprint, the 128-bit version

can be utilized for lightweight designs for up to seven different tests. On the other hand, the 1048576-bit version has the capability to support long term evaluation and up to nine different tests. The 65536-bit version provides a balanced trade-off between the hardware area and the input length of the random sequence. All our implementations on FPGA have a maximum working frequency larger than $100MHz$. In other words, they can handle an input bit rate of $100Mbit/s$, which is enough for most of the TRNGs on FPGA.

Our designs are also suitable for ASIC, either as an individual unit or as a building block for processors. In Table 6.4 we provide the area results in GE.

Our software designs can be implemented on different hardware platforms, such as microcontrollers, GPUs and DSPs. These embedded systems might utilize different dedicated peripherals, such as a HW multiplier and squarer. The number of required clock cycles is greatly dependent on the SW platform. In Table 6.4, we present the instruction count of software implementations for a 16-bit architecture. We can see that the largest version requires more than 900 ADD/SUB and almost 200 MUL/SQR. The reason is that instructions operating on data larger than 16-bit have to be decomposed into several 16-bit operations. We can expect that, on 32-bit or 64-bit platforms, considerably lower latency could be achieved.

In this work, we present a unified implementation of different NIST tests based on splitting the operation between hardware and software. By keeping only the necessary operations in hardware, we have achieved our goal of a low area cost of the hardware part and a sufficient flexibility for the software part.

6.4 Black-Box Tests on Raw Random Numbers

All presented online tests based on black-box statistics are tailored for IID number generators. TRNGs rarely produce statistically perfect numbers and they require arithmetic post-processing to produce a full-entropy output. In order to detect TRNG failures quickly and reliably, we need online tests that are suitable for non-IID data.

In this work, we present implementations of four statistical tests that report the level of entropy, rather than an alarm signal. These test algorithms are originally developed for estimating the min-entropy of non-IID sources during the prototype evaluation [4].

6.4.1 Test Algorithms: Description and Simplification

A set of five algorithms for min-entropy evaluation is presented in [4]. We find that one of those, namely the compression test, is not suitable for hardware implementation due to large memory requirements. The remaining four algorithms can be implemented after some simplifications. The algorithms are very general in the sense that they can be applied to any number of possible output values. We have implemented tests for a generator that produces one bit at a time, which simplifies the algorithms to some extent.

Our implemented online tests report eight different levels of entropy on a scale from 0 to 7. The levels are corresponding to the 3 most significant bits of the binary representation of min-entropy per bit. This approach enables us to provide compact implementations that require storing of a small number of precomputed constants. All statistical tests provide more reliable results when the length of the tested sequence is longer. However, testing very long sequences means that the failures are not detected fast enough. As a compromise solution we have chosen a sequence length of 2^{13} bits.

Since each algorithm tests a sequence for a different type of failure, some statistical weaknesses will remain undetected by some tests (the test overestimates the entropy). For this reason, it is necessary to implement several tests and to observe the minimal result at the output.

The Collision Test

We say that a sub-sequence of a dataset contains a collision if it contains two equal data values. Since we are testing a TRNG that produces only 2 distinct data values, a collision can happen after 2 bits (if they are equal) or after 3 bits (if the first 2 bits are different). The whole sequence can be divided into segments that contain exactly one collision. These segments consist of either 2 or 3 bits, as illustrated in Figure 6.7.

The collision test measures the mean time until the first collision. Since a collision always happens after either 2 or 3 generated bits, the min-entropy of a sequence of fixed length can be determined using only one parameter. This parameter can be the total number of collisions, the number of 2-bit segments, or the number of 3-bit segments containing a collision. We chose to track the number of 3-bit segments since the maximal value of this parameter is the smallest, which results in the most compact counter and comparators. The estimated min-entropy is a monotonically decreasing function of this parameter. Therefore, since the test reports only 8 levels of min-entropy, the hardware implementation requires only one counter and 7 cut-off values.

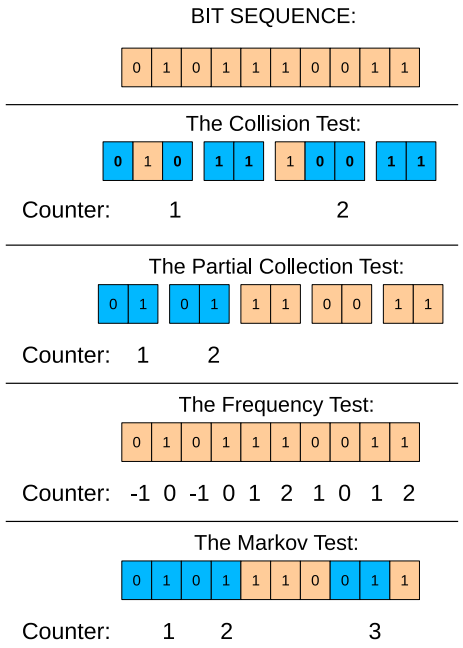


Figure 6.7: An example illustrating the principles of the test algorithms.

The Partial Collection Test

This test measures the number of distinct data values in each block of data. The length of a block is equal to the size of the output space (in this case 2). The concept is shown in Figure 6.7. The test statistics can be described using only one parameter. In this case it is the number of non-overlapping blocks containing 2 different bits. This test can be implemented using one counter and comparing the final value with the precomputed cut-off values.

The Frequency Test

The purpose of this test is to find the frequency of the most common value in a data set (in this case either 0 or 1). This test can be implemented by using an up/down counter to track $|C_1 - C_0|$ and by comparing the final result with the precomputed cut-off values.

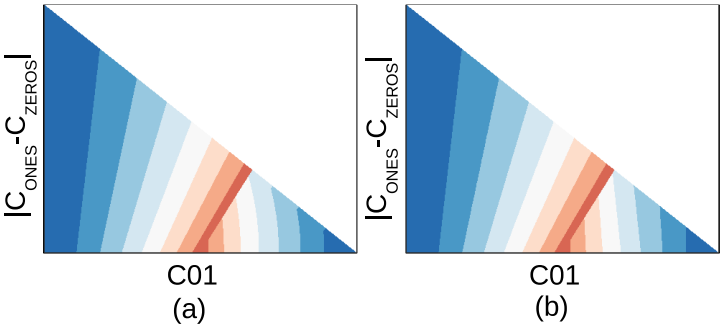


Figure 6.8: Heatmap diagrams showing entropy levels for the Markov test: (a) Before approximations. (b) After approximations.

The Markov Test

This test relies on approximating the TRNG with a Markov process. This is a process where the output depends not only on the current state but also on the previous states. This test requires to estimate the probabilities of all states and all state transitions, and then to estimate the probability of the most likely bit sequence. In order to calculate the min-entropy, six parameters have to be obtained from the bit sequence, namely a count of occurrences of different states (C_0 and C_1) and counts of all state transitions (C_{00},C_{01},C_{10},C_{11}).

In order to reduce the number of parameters we make the following observations:

- $C_{01} \approx C_{10}$ These two values can differ by at most one which is negligible compared to the length of the sequence.
- $C_0 + C_1 = N$
- $C_{01} + C_{00} = C_0$
- $C_{11} + C_{10} = C_1$

Since there are six parameters with four constraints, we only need to keep track of two parameters. We have chosen to track $|C_1 - C_0|$ which is already available from the frequency test, and C_{01}. The entropy level was calculated for all possible parameter values and the results are shown in the heatmap diagram in Figure 6.8a. Different colors are used to denote entropy levels (blue for low entropy, red for high entropy). Values above the diagonal never appear because $C_{01} \leqslant min(C_0, C_1)$.

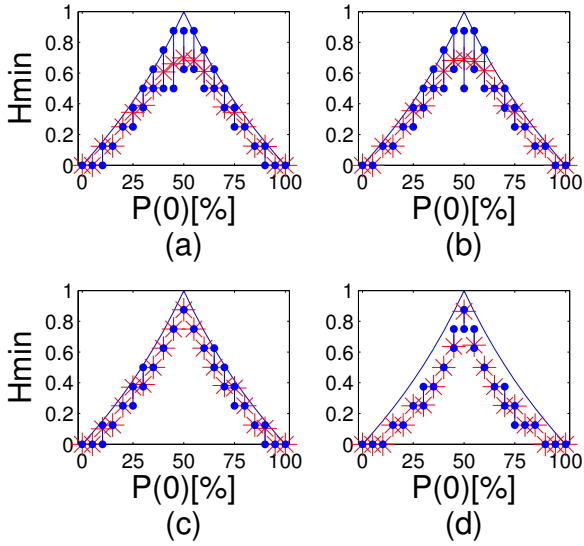


Figure 6.9: Test results of the sequences generated by the biased generators: (a) The collision test. (b) The partial collection test. (c) The frequency test. (d) The Markov test.

Borders between different entropy level regions can be approximated with straight lines, as shown in Figure 6.8b. There are 15 lines in total, and each line is determined by two parameters: the 15-bit slope coefficient and the 19-bit value at zero. In total only 15 constants of 34 bits have to be precomputed. The entropy level can be determined using at most 15 multiplications, additions and comparison operations:

$$|C_1 - C_0| < C_{01} \cdot k_i + n_i, \tag{6.1}$$

where k_i and n_i are the precomputed constants corresponding to the linear coefficients of the 15 borders between entropy regions.

6.4.2 Test Validation

In order to verify the functionality of the simplified tests, 100 sequences of 2^{13} bits were generated using simulated non-IID sources and the test results were

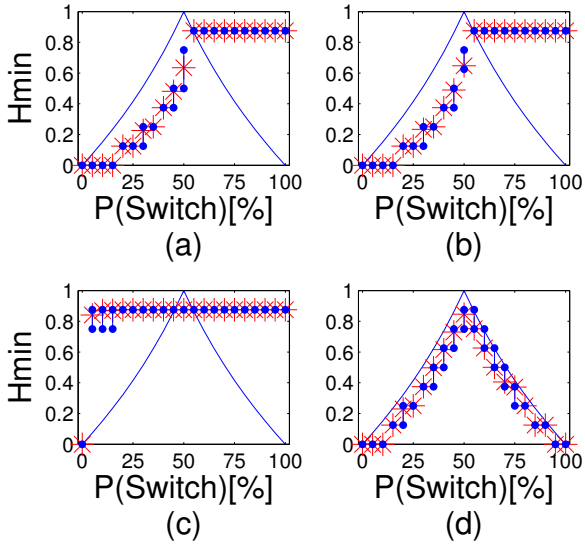


Figure 6.10: Test results of the sequences generated by the generators with bit-dependencies: (a) The collision test. (b) The partial collection test. (c) The frequency test. (d) The Markov test.

compared with the theoretical min-entropy values. Two types of non-IID sources have been used for this purpose: a biased generator without bit dependencies and an unbiased generator with correlation between the consecutive bits.

Figure 6.9 shows the test results obtained by testing the biased generator without bit dependencies. We used 21 different biased generators with the probability of 0 varying from 0% to 100% in steps of 5%. For each bias value, 100 sequences were generated and the test results were recorded. Minimal and maximal values are shown as dots, and the average value is represented using the star symbol. The theoretical value of entropy is shown as a continuous line. As expected, the entropy estimations of all four tests follow the theoretical value of min-entropy.

Figure 6.10 shows the test results obtained using an unbiased generator with dependencies. For this experiment, we varied the probability of switching the bit value. Again, this probability was varied in steps of 5% and 100 sequences were generated for each value. It can be seen that the frequency test almost always reports the full entropy, which is expected because this test is not designed

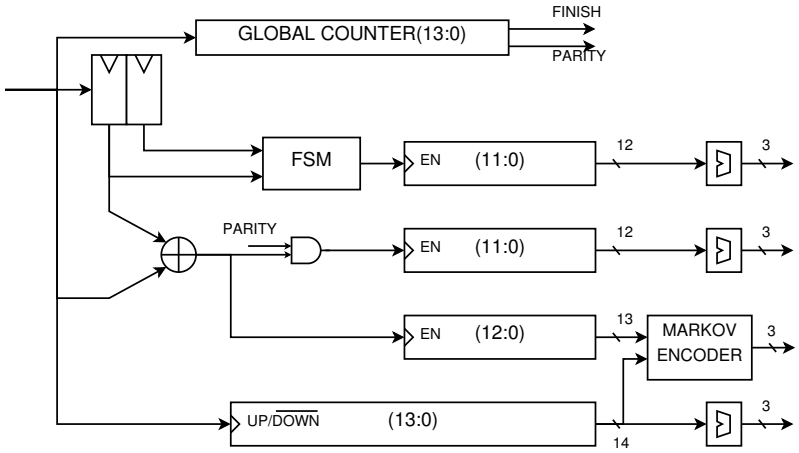


Figure 6.11: Architecture overview of the hardware module containing all 4 tests running in parallel.

to detect this type of failure. The collision test and the partial collection test detect weaknesses only when the probability of switching is low. The only test that reliably estimates the min-entropy in the presence of bit dependencies is the Markov test.

We note that, even though the Markov test can successfully estimate entropy for both types of failures that we tested, there are failures that are only detectable by other tests. For example, a sequence of alternating patterns of 2 zeros and 2 ones (00110011...) which can appear when oversampling the ring oscillator based TRNG, will not cause failing the Markov test which reports the highest entropy level. However, the collision test and the partial collection test correctly estimate the entropy level as 0. Therefore, it is important to have a variety of tests because no single test can detect all possible weaknesses.

6.4.3 Implementation and Result

Figure 6.11 shows the general architecture of the implemented hardware module. The clock signal and some enable signals are omitted for better clarity. Input signals coming from the TRNG are the data bit and the data valid signal. The tests operate on bit sequences of length 2^{13}. The proposed design takes 15 clock cycles to calculate the min-entropy level.

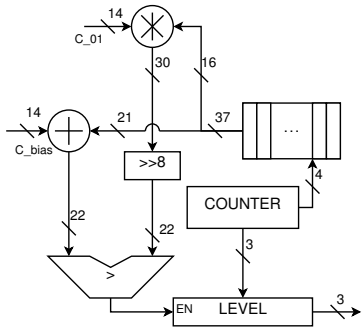


Figure 6.12: Architecture of the encoder used in the Markov test.

The implementation boundaries of each individual test were removed to obtain a better unified hardware implementation. A global counter is used to indicate the end of the sequence and to provide the parity bit used by the partial collection test. The up/down counter result is used by both the frequency test and the Markov test. Collisions are counted using a 2-bit finite state machine.

Each test result, with the exception of the Markov test, is computed by comparing the available counter values with the precomputed boundaries corresponding to each level. The Markov test uses a 2-parameter function which requires a more complex encoder. The result obtained from the counters corresponds to a single location in the heat map diagram and the min-entropy level is determined by finding the region on the diagram that contains this point. This is achieved by solving 15 linear equations of which each one requires one multiplication and one addition. The datapath used by the Markov encoder is shown in Figure 6.12. The coefficients of the linear equations are stored as 15 words of 37 bits. The comparison operation from Equation 6.1 was implemented as:

$$|C_1 - C_0| - n_i < C_{01} \cdot k_i \tag{6.2}$$

This implementation enables us to ignore the fractional part of the product $C_{01} \cdot k_i$ (the last 8 bits) which results in a more compact multiplier. The test goes through the precomputed constants, and reports the entropy level when it finds a match.

Table 6.4: Implementation Results

FPGA:				
Slices	FFs	LUTs	DSP48A1s	MaxFreq(MHz)
60	81	174	1	130
ASIC:				
Area(GE)			Energy/bit(pJ) @1MHz	
2394			1.4	

Implementation Result

The design environments for experiments, such as the FPGA platform and the design tools, are the same as in Section 6.3.3. The implementation result is given in Table 6.4. With the utilization of 60 slices and 1 DSP48A1 slice, our design has a maximum working frequency of 130 MHz on FPGA. In other words, it can handle an input bit rate up to 130 $Mbit/s$, which is sufficient for most of the TRNGs on FPGA. Our design is also suitable for ASIC. The area consumption is 2394 GE. The most area consuming part is the multiplier in the Markov encoder. The power consumption is estimated at the gate level by PowerCompiler, based on the switching activities generated by a real testbench. The power strongly depends on the clock frequency and technology. In order to draw a fair comparison, we use energy per bit to represent the energy efficiency. The result shows that our design has a rather low energy consumption.

In this work, we presented a compact hardware implementation of statistical tests suitable for monitoring non-IID entropy sources. We demonstrated that the presented tests are suitable for estimating the min-entropy of biased generators and generators with bit dependencies. ASIC implementations are very compact consuming around 2.4 kGE, and FPGA implementations on Xilinx Spartan-6 consume less than 1% of the available resources.

6.5 Summary

At beginning of this chapter, we discuss two types of embedded tests, namely total failure test and online test. A total failure test designed for DC-TRNG is introduced as a case study. Four different design methodologies for online tests are discussed. Two implementations of statistical black-box testing are proposed in Section 6.3 and Section 6.4 for IID sources and non-IID sources, respectively.

Chapter 7

Embedded tests: the modern approach

Beware of false knowledge; it is more dangerous than ignorance.

-George Bernard Shaw

Content Sources

Yang, B., Rožić, V., Dehaene, W., Mentens, N., and Verbauwhede, I. TOTAL: TRNG On-the-fly Testing for Attack detection using Lightweight hardware, In *Design, Automation and Test in Europe (DATE)*, 2016, pp. 127-132.
Contribution: Main author.

Rožić, V., Yang, B., Mentens, N., and Verbauwhede, I. Canary Numbers: Design for Light-weight Online Testability of True Random Number Generators, In *NIST RBG Workshop*, 2016.
Contribution: Responsible for the implementations and experiments, contribute to the idea of replica based architecture.

7.1 Introduction

This chapter presents two novel methodologies for designing online test modules for TRNGs. The content of this chapter was originally published in [123] and [85]. In Section 7.2, we propose the TOTAL methodology. A different approach to design online tests based on canary numbers is introduced in Section 7.3.

7.2 TOTAL Design Methodology

This section presents a design methodology called TOTAL (TRNG On-the-fly Tests for Attack detection using Lightweight hardware).

The principle of TOTAL is using an empirical design procedure consisting of a data collection and a search for the useful statistical features. We recommend to test the entropy source under normal operating conditions and under different attacks such as underpowering, oversampling, glitching and temperature attacks. The sequences of the produced bits should be recorded. The next phase consists of applying different statistical features on the collected data and finding a useful feature for distinguishing between the normal operation and the operation under attack. In this context, the usefulness of the feature is evaluated according to the attack detection probability. We focus on the features that have compact hardware implementations. We created an extensive list of these features including: the number of ones in a sequence, the auto-correlation coefficients, frequencies of different patterns and the number of runs of a predefined length. Once useful features are found, the tests are implemented by setting the appropriate threshold values.

This methodology enables the designer to specify the length of a sequence under test and the false alarm trigger level, as well as to estimate the usefulness of the test. The effectiveness of the proposed methodology is higher when more features and more attacks are tested. The attacks that don't change any of the features are not likely to create any harm.

7.2.1 Design Steps

The proposed design methodology for on-the-fly tests relies on extensive testing of the entropy source under various operating conditions to simulate all practical attack scenarios. The goal is to find one or more statistical features that are suitable for attack detection and that are also implementable using lightweight

hardware. Another design goal is a low length of the test sequence. This requirement is needed because of the requirement to stop the TRNG output when an attack or a weakness is detected. If a statistical test is used on a sequence of length n, then by the time a weakness is detected, n compromised bits are sent to the output. The only way to prevent compromising the output is to buffer the generated data and send them to the output only if the data passed the test. This solution is practical only for short sequences which don't require much storage. This methodology aims for compact hardware implementations, short bit sequences, low computational latency and reliable attack detection.

The proposed design methodology consists of the following steps:

1. Data Collection
2. Preliminary detection of useful features
3. Feature verification
4. Attack effort analysis
5. HW implementation
6. HW Validation

Here we explain each step in more detail:

Data Collection Firstly, data produced by the entropy source (the raw bits) should be collected under the normal operating conditions. We call these data the golden reference. We suggest collecting data sets of 512 consecutive bits. In our experience, this length is enough to design useful, robust tests that are significantly faster than generic statistical tests. For the preliminary analysis, we recommend collecting around 8000 sequences.

Afterwards, the entropy source should be exposed to different attacks and new sets of data should be collected. Again, we recommend collecting around 8000 sequences of 512 bits for each attack. At this step it is important to include all low-cost attacks such as changes in temperature, voltage and clock frequency. The security of the method depends on the extensiveness of the testing.

Each attack can be performed with different effort, for example the underpower attack can be performed using different voltage levels. In this phase, we are only collecting data using the highest attack effort such as the lowest voltage level for the underpower attack or the highest frequency for the oversampling attack. The collected data will be used to detect the useful statistical features that will be used in the test design.

Preliminary detection of useful features Once the first set of data is collected, the goal is to compute the statistical features. We provide a recommended list of features that can be implemented in a compact manner:

1. Auto-correlation coefficients

$$C_i = \sum_{k=1}^{n-i} a_k \oplus a_{k+i} \tag{7.1}$$

   where $a_1..a_n$ are the bits of the test sequence. Auto-correlation coefficients are computed for $i = 1...16$.

2. Number of ones in a sequence - N_1.

3. Number of overlapping 2-, 3- and 4-bit templates in a sequence - $N_{00}...N_{11}, N_{000}...N_{111}, N_{0000}, N_{0001}...N_{1111}$.

4. Number of non-overlapping 2-, 3- and 4-bit templates in a sequence $n_{00}...n_{11}, n_{000}...n_{111}, n_{0000}, n_{0001}...n_{1111}$.

5. Maximal Random Walk Divergence from Zero - S_{max}

$$S_i = \sum_{k=1}^{i} (2 \cdot a_k - 1) \tag{7.2}$$

$$S_{max} = max_{k=1..n}|S_k| \tag{7.3}$$

6. Number of runs of fixed length - $r_1...r_8$

7. Length of the longest run of the same bit value - r_{max}.

The features were selected based on previous implementations of online tests. We looked into the implementations presented in Chapter 6 and selected the features that can be implemented in a simple and compact manner. This list is not comprehensive and other features may be included. Each of these features can be implemented using one counter and a small amount of logic gates.

Each feature is computed for each of the collected sequences and the probabilities of the distributions are studied. Useful features are selected based on two criteria: the distribution of the golden reference, and the distinguishability of the distributions during attack. Only the features that show the normal distribution for the golden reference are considered. If no such features are found, the data collection should be repeated using sequences longer than 512

bits. For each of the selected features the mean μ and the standard deviation σ are computed.

Further selection is done by evaluating the usefulness of features for attack detection with respect to type II error probability. We define usefulness as the probability of detecting an attack: $1 - P_2$ where P_2 is the probability of a type II error (false negative). We explore usefulness with respect to two types of tests: a sensitive test and a robust test. We define a sensitive test as a test with 1% type I error probability. This means that one in 100 sequences will trigger the alarm when the entropy source is not under attack. This test is implemented by checking if the measured feature is within the $(\mu - 2.58\sigma, \mu + 2.58\sigma)$ interval. The robust test is defined as a test with 10^{-9} type I error probability. It is implemented by checking if the measured feature is within the $(\mu - 6.11\sigma, \mu + 6.11\sigma)$ interval.

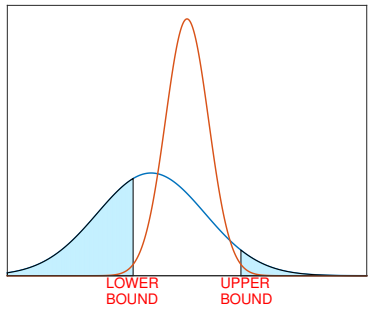
Other tests may be defined depending on the design specifications. The usefulness of each test is estimated as the ratio of sequences under attack that are detectable by the test.

$$usefulness = \frac{\#\text{Sequences outside of the range}}{\#\text{All sequences}} \tag{7.4}$$

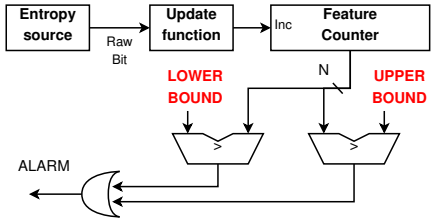
The features are ranked according to the computed usefulness for both types of test and for all attacks.

Feature verification At this phase, the top ranked features for the sensitive test and the robust test are selected. More data are collected to verify the choice of the selected features. We suggest collecting 128000 sequences of 512 bits. These data are used to verify that the golden reference remains normally distributed and that detected features remain useful. If problems with the distribution of the golden reference are detected, all previous steps should be repeated using longer bit sequences or using different design parameters of the entropy source. If the usefulness is too low, then the features with lower rank are explored. If no useful feature is found then previous steps should be repeated using longer sequences or different parameters of the entropy source. In the end, two features are selected for each attack, one for the sensitive and one for the robust test. There may be some overlap between the features, for example when one feature is useful for detecting 2 attacks.

Attack Impact Analysis In previous phases, we have used data under the highest attack impact. At this phase, we evaluate how different levels of attack impact affect the usefulness of the test. Additional data should be collected using



(a) Determining bounds for on-the-fly tests



(b) The generic architecture of the on-the-fly test

Figure 7.1: The principle for hardware implementation

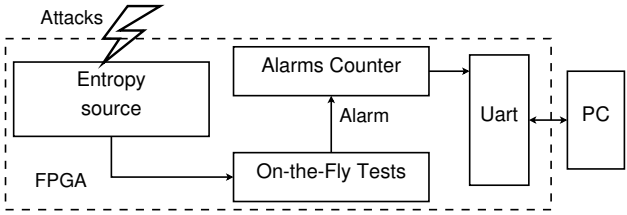


Figure 7.2: Platform for hardware validation

different levels of attack impact such as different voltage levels (for underpower attacks) or different frequencies (for oversampling attacks). Test usefulness should be computed for each level of the attack impact.

HW Implementation As a general rule, tests with a high false positive (Type I) error rate are more efficient for attack detection. Type I error rate is a design parameter which should be set based on the application requirements. If the

maximal allowed error rate is once in a lifetime of the product (for example when a manual reset is required whenever an alarm signal is detected), as required by [99], then a robust test with very low error rate should be implemented. The usefulness of this test for attack detection may be very low. For some applications, sensitive tests may be more appropriate. If the application doesn't require manual reset after attack detection, then a test with higher Type I error rate (for example 1%) may be appropriate. Such a test can detect attacks with higher success rate.

The generic architecture of the online test is shown in Figure 7.1b. Raw bits from the entropy source are sent to the update function which generates the Inc signal, for example in the template matching tests, the signal will be activated whenever a specific pattern is detected. This signal is used to increase the counter. At the end of the sequence, the counter value is compared to the predefined lower bound and the upper bound. These bounds are computed based on the required error rates and the parameters of the golden reference Gaussian distribution. If the counter value is outside of the range, the alarm signal is activated. The example given in Figure 7.1a shows the counter distributions under normal conditions (orange line) and under attack (blue line). The bounds are shown for 1% type I error rate. The shaded area under the curve is equal to the usefulness of the test.

HW Validation As the last step of the methodology, the hardware implementation should be evaluated. The platform shown in Figure 7.2 is used for this purpose. The entropy source under normal operating conditions is used to generate 1024000 sequences. The embedded test evaluates a sequence and produces an alarm signal if the test fails. A counter keeps track of the alarm signals. After all sequences are tested, the counter value is sent to the output and the error rate is computed. The purpose of this validation is to verify if the error rate is within the predicted limits. The same procedure is repeated for the entropy source under attack and the attack usefulness is computed. Validation should be done for all attacks and all implemented tests.

7.2.2 Case study I: ERO-TRNG

In this section we provide an example to illustrate the proposed design methodology. We use the ERO-TRNG discussed in Section 4.2.4 as a case study and apply our methodology to design a test for detection of the oversampling attack. In a commercial product, more tests to detect attacks should be included (such as interlock attacks, glitching attacks and temperature attacks).

Table 7.1: Features with 100% usefulness for the elementary TRNG

Sensitive tests	Robust tests
C_1,	
N_{00}, N_{01}, N_{10},	
N_{000},N_{010},N_{101},N_{111},	C_1,
N_{0000}, N_{0010}, N_{0100},	N_{01}, N_{10},
N_{0101}, N_{1010}, N_{1011},	N_{010}, N_{101},
N_{1101},	n_{01}, n_{10},
n_{01}, n_{10},	n_{010}, n_{101},
n_{010}, n_{101},	n_{1011}, n_{1101}
n_{0010}, n_{0100},	
n_{1011}, n_{1101}	

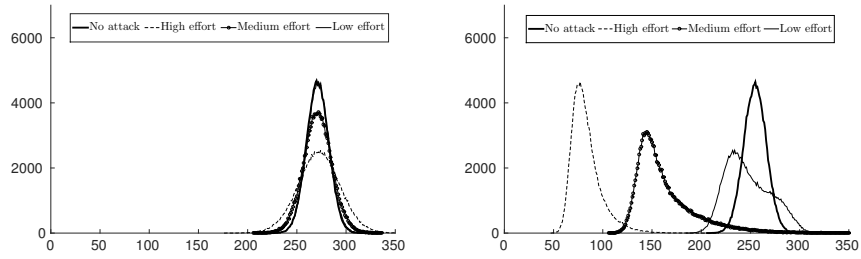
In this case study, the ERO-TRNG consists of a fast oscillator implemented using 3 LUTs, which is sampled by a slow oscillator implemented using a 5-stage LUT-based ring oscillator and a frequency divider. This TRNG relies on the timing jitter for generating entropy. For longer jitter accumulation time, the output approaches the full-entropy. However, the timing jitter accumulates very slowly. In order to achieve more than 0.99 bits of entropy per output bit, the output needs to be down-sampled by a factor of 32. The entropy per bit can be effectively reduced by decreasing the down-sampling rate.

Design Online Test

The over-sampling attack is simulated by reducing the down-sampling rate. We select the down-sampling factor by choosing the parameter of the frequency divider in Figure 4.5. The design was evaluated using three down-sampling factors (2, 4 and 8) corresponding to a high, medium and low attack effort respectively. For the initial analysis only the highest attack effort was used.

The usefulness of the features was evaluated with respect to the sensitive test (corresponding to 1% type I error rate) and the robust test (10^{-9} error rate). The features are ranked based on the measured usefulness for both tests and the features with maximal usefulness are shown in Table 7.1. The overlapping 4-bit template N_{0101} and the auto-correlation coefficient C_1 were chosen for the implementations. Any feature listed in the table could be used for test design.

Additional data were collected to justify the choice of the features and to analyze the test performance under different attack impacts. Figure 7.3 shows examples of a useful and a useless feature. An additional 128000 sequences were collected under normal operating conditions and under high, medium and low attack



(a) N_1 (b) C_1

Figure 7.3: Feature distributions

effort. The features N_1 and C_1 were computed for each sequence and the corresponding distributions are shown. Figure 7.3a shows the distribution of the number of ones within the sequence. As can be seen, this feature is not useful because the distributions under attack are centered around the same mean value as under normal operating conditions, which means that the attack doesn't significantly affect the bias. Standard deviations seem to increase with higher attack effort but not enough to be useful for test design. On the other hand, the auto-correlation coefficient C_1 is very useful for attack detection. The mean of the distribution shifts to the left with higher attack effort as shown in Figure 7.3b. Under highest attack effort the distribution is completely distinguishable from the original one resulting in almost perfect usefulness of the test. This usefulness is slightly lower for lower attack effort as expected.

We tried to perform the underpower attack using the same methodology. Our implementation platform uses the core power supply voltage of $1.2V$. For voltages below $0.86V$, the communication interface stopped working. We have collected data for different voltage values between $0.86V$ and $1.2V$ but couldn't find any useful features. We conclude that the ERO-TRNG is not affected by the underpower attack in a significant way.

Implementation and Validation

We have implemented both the sensitive (1% error rate) and the robust test (10^{-9} error rate). The sensitive test was implemented using an overlapping 0101 template. The test verifies if the number of templates in the sequence is within the $[17, 47]$ interval. The robust test is implemented using one XOR gate and a counter for computing C_1. Comparators are used to check if C_1 is within

Table 7.2: Hardware utilization results for the elementary TRNG

	FPGA			ASIC
	Slices	LUTs	FFs	(GE)
Sensitive test	9	28	25	245.75
Robust test	10	26	22	233.5
Combined test	14	42	35	361.5

Table 7.3: Test validation for the elementary TRNG

	sensitive test usefulness (%)	robust test usefulness (%)
Low attack impact	34.85	≈ 0
Medium attack impact	99.67	83.97
High attack impact	100	99.84

the $[183, 326]$ interval. Table 7.2 shows the hardware utilization results. All proposed hardware designs are synthesized using Xilinx ISE14.7 on Spartan-6 XC6SLX45 FPGA and Synopsys Design Compiler D-2010.03-SP4 to UMC's $0.13\mu m$.1P8M Low Leakage Standard cell Library.

The usefulness of the implementation was experimentally verified using 1024000 sequences. The results are reported in Table 7.3. For high attack impact, both tests have very high usefulness. For medium attack impact, the usefulness is still very high but it is significantly lower for the low attack impact. The measured Type I error is within the expected boundaries (0.93% for the sensitive test and 0% for the robust test).

The robustness of the methodology with respect to process variations was evaluated. The validation platform was implemented on a second FPGA of the same type using the same bitstream. The experiments were repeated and the results showed that the type I error rate remains constant over different FPGAs of the same type. Upon further investigation, it was found that a higher oversampling rate was needed to achieve the same attack impact on a different FPGA.

7.2.3 Case Study II: DC-TRNG

In this section we apply the TOTAL methodology to design online tests for the DC-TRNG on FPGA. The introduction of the DC-TRNG can be found in Section 4.3. In this case study, we use an earlier version of the DC-TRNG

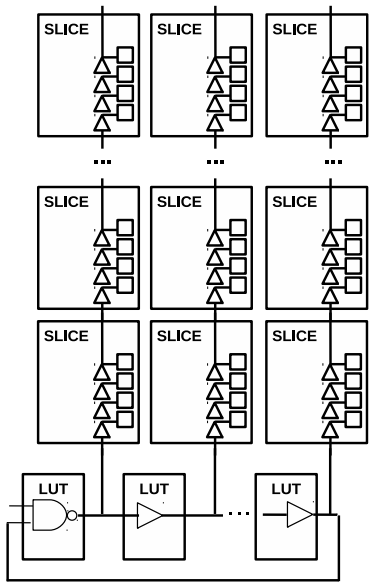


Figure 7.4: The architecture of the DC-TRNG

from [83]. Figure 7.4 shows the architecture of the implemented entropy source. Random jitter accumulated in the ring oscillator is extracted using tapped delay lines built with Xilinx carry-chain primitives. The clock frequency is $100MHz$ and a sample is taken every 32 clock cycles. A shorter accumulation time results in lower entropy per bit.

Design Online Test

The oversampling attack was performed by sampling the entropy source in every clock cycle. The collected data didn't show any detectable weaknesses. The underpower attack, when applied to the original design also didn't result in any significant changes. However, the combination of the two attacks with the voltage reduced to $0.9V$ and a sample taken in each clock cycle resulted in some useful features. The ranking of the features with respect to the sensitive test and the robust test is shown in Table 7.4. The overlapping 3-bit template N_{111} and the auto-correlation coefficient C_1 are the most useful features for the sensitive and the robust test, respectively.

Table 7.4: Top features for the DC-TRNG

sensitive tests		robust tests	
Features	usefulness (%)	Features	usefulness (%)
N_{111}	90	C_1	35
C_2	83	N_{01}	35
N_{1110}	82	N_{10}	35
n_{111}	82	N_{010}	33
N_{0111}	82	r_1	32

Table 7.5: Hardware utilization results for the DC-TRNG

	FPGA			ASIC
	Slices	LUTs	FFs	(GE)
Sensitive test	10	26	24	239.5
Robust test	10	26	22	233.5
Combined test	14	40	34	353.25

Implementation and Validation

Both test implementations are very compact as shown in Table 7.5. Each individual implementation consumes 10 slices on a Xilinx Spartan-6 FPGA and the combined implementation of both tests consumes only 14 slices. The ASIC implementation is also very compact with less than 240 GE for individual implementations and around 350 GE for a combined implementation.

The FPGA implementation was used to evaluate the type I error rate and the usefulness. The measured type I error rate is within the expected boundaries (0.65% for the sensitive test and 0% for the robust test. The measured usefulness is 44.36% for the robust test and 89.4% for the sensitive test.

In this work, we have proposed a design methodology for dedicated, lightweight, low-latency, on-the-fly tests for monitoring the quality of entropy sources. This methodology relies on experimental results to find the optimal statistical features for designing on-the-fly tests. This experiment-oriented approach guarantees that the tests are useful for detecting attacks. We base our work on the assumption that a physical attack on the entropy source will result in changing some global statistical feature of the generated sequence rather than in a complex relation between specific bits. The security of the proposed method depends on the richness of the pool of statistical features and the number of attacks applied during the design phase. The main benefit of the proposed method over using standard statistical tests is the improved performance in terms of latency, area

and test effectiveness. This benefit comes from the fact that the produced tests are tailored for a specific entropy source. Lightweight tests were designed and their effectiveness for attack detection was confirmed by experiments.

7.3 Canary Number Based Design Methodology

In this section, we introduce the concept of canary numbers, to be used in health tests for true random number generators. The content of this section was originally published in [85].

The proposed solution uses canary numbers which are an extra output of the entropy source of lower quality than the other outputs. This enables an early-warning attack detection before the output of the generator is compromised. We illustrate the idea with two case studies, namely ERO-TRNG and DC-TRNG.

The main principle of this methodology is to improve the testability of the noise source by producing two bit streams: raw numbers to be used by the application, and canary numbers to be used solely for testing purposes. Canary numbers have lower statistical quality than the raw numbers and they are more susceptible to changes in operating conditions. For this reason, monitoring canary numbers can be used for an early-warning failure detection, since the statistical quality of the canary numbers drops before the failure affects the raw numbers in a significant way.

In order to generate canary numbers, we propose two generic architectures, namely *the replica based architecture* and *the canary extraction based architecture*. The canary extraction based architecture is illustrated on the example of a DC-TRNG, which is the same DC-TRNG as in Section 7.2.3.

7.3.1 Canary Numbers Based Online Testing

In this work, we aim to improve the trade-offs between test efficiency and latency while using very simple lightweight tests. Rather than tailoring the tests for a particular entropy source, we design the source for improved testability. The central idea of this paper is that the noise source produces 2 bit streams: *the raw numbers* and *the canary numbers*. The canary numbers are not sent to the application, but only to the testing module. The statistical quality of these bits is weakened by design in order to increase the susceptibility to attacks.

We note that similar concepts are used in other areas of security. For example, in software security, canary values are used to prevent buffer overflow attacks and,

in hardware security, a method called *canary logic* [110] is used for detecting fault attacks by deliberately increasing the critical paths of redundant hardware modules. The name originates from the analogy with the role of the canary in a coal mine as an early detection of reduced oxygen levels. The goal is to sacrifice a part of the design that is not important in order to obtain an early warning of the fault and to save the part of the design that is important. The same principle is applied in this work. By applying statistical tests on the canary bits it is possible to detect attacks at an early stage, so that the alarm can be triggered before the entropy of the raw bits drops significantly.

The statistical features (entropy, bias, probabilities of generating a given pattern) of the raw numbers and the canary numbers are functions of platform parameters such as jitter variance, delays of individual logic circuits or noise strength. Usually, only one of the platform parameters is critical for entropy generation. For example the strength of the accumulated jitter is the main contributor to entropy in all jitter based TRNG designs. The attack is performed by altering the platform parameters, for example by reducing the ambient temperature, thereby reducing the amount of jitter below an acceptable level.

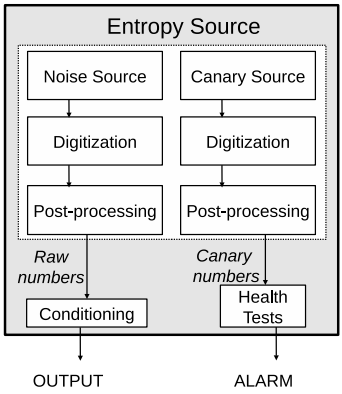
To guarantee the robustness of the design, the entropy of the raw numbers at the operating point should be robust against a small variance of the critical parameter, in other words the slope of the min-entropy estimation H_{raw} should be very small.

High testability is obtained in precisely the opposite situation. For example, a statistical feature measured by the health test should change when the value of the critical parameter changes.

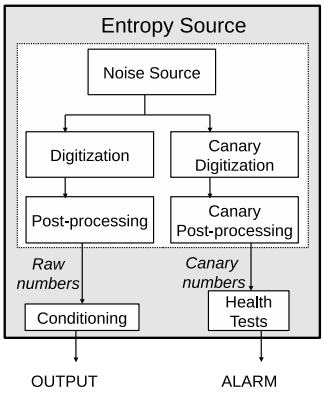
Intuitively, it seems difficult to find a feature with high testability and high robustness at the same time. However, it is possible to re-design the entropy source to produce two bit streams: *raw numbers* with high robustness and *canary numbers* with high testability.

Figure 7.5 shows the two proposed architectures of the entropy source using canary-based testing.

Replica based architecture This architecture (shown in Figure 7.5a) is based on designing a weaker replica of the entropy source for testing purposes. This replica, called *the canary source* is simply a copy of the noise source with different design parameters. Therefore, the replica is designed by exploring the design parameter space to develop a design with high testability. The design parameters of the digitization and the post-processors can optionally also be changed to improve testability.



(a) Replica based architecture.



(b) Canary extraction based architecture.

Figure 7.5: The proposed architectures of the entropy source with canary-based testing.

This architecture can be useful for detecting global attacks (such as underpower or temperature attacks). However, it cannot detect localized attacks that affect only the entropy source. For this reason, this countermeasure should be used only in addition to other online testing techniques.

Canary extraction based architecture This architecture is based on the notion that digitization and post-processing are forms of signal processing designed with the goal of extracting entropy. Our idea is to re-design these components with high testability as a design goal. The design parameters of the digitization block and post-processing block should be tuned to maximize testability rather than min-entropy. New digitization and post-processing techniques are then applied to the same signal (produced by the noise source) that is used to generate raw numbers. The resulting canary bits are used as canary numbers.

Compared to the replica entropy source, this architecture has a higher applicability, since the same noise source is used to produce both the raw numbers and the canary numbers.

7.3.2 Case Study: DC-TRNG

In this section we illustrate the canary extraction based architecture on the example of the DC-TRNG from Section 7.2.3.

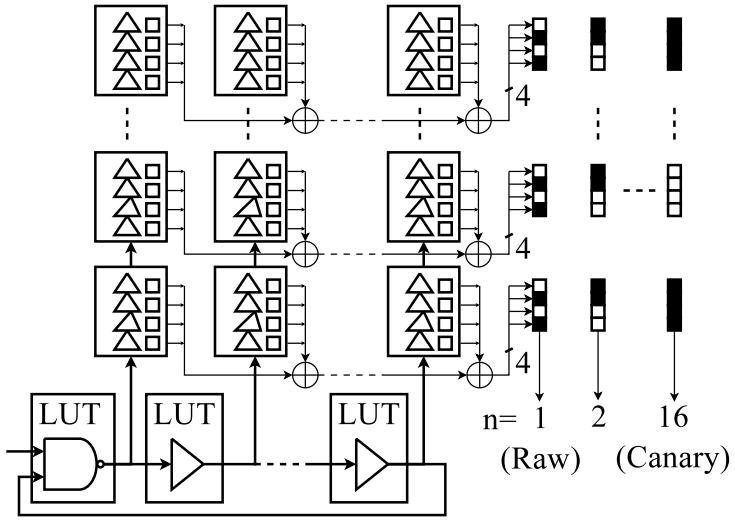
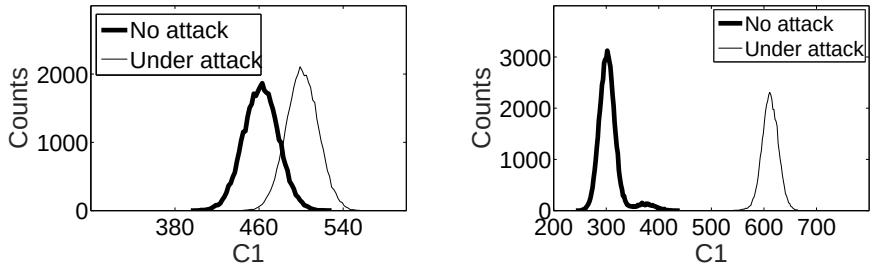


Figure 7.6: Architecture of the DC-TRNG with canary digitizers.

The architecture of this generator is shown in Figure 7.6. The entropy of this TRNG depends on the jitter accumulation time t_A and the step size of the TDC performed in the tapped delay lines t_{step}.

The canary numbers for this TRNG are designed by increasing the value of t_{step}. This is achieved by down-sampling the tapped delay lines. Different priority encoders are used to detect the edge of the signal in different resolutions, which are corresponding to different $n \cdot t_{step}$. To find the most useful canary numbers, down-sampling factors $n = \{2, 4, 8, 16\}$ are examined. $n = 16$ is selected in the end for generating the canary numbers.

In order to verify the proposed design of canary numbers, auto-correlation is used to analyze the bit dependence of the raw numbers and the canary numbers respectively. For every experiment, the auto-correlation value C_1 is computed on a sequence of 1024 bits. The experiment is repeated 100000 times at 30^oC (no attack) and at 0^oC (attack). The histograms of the obtained data are shown in Figure 7.7. The auto-correlation distributions shown in Figure 7.7a are located very close to each other. Even though these distributions are clearly different, it is not possible to design a test to determine the origin of the data based on a single C_1 value. On the other hand, as shown in Figure 7.7b, the auto-correlation distribution of the canary numbers under normal working conditions and under attack are much more distinguishable. One single threshold value can be used to



(a) Auto-correlation distributions of the raw bits under normal operating conditions and under attack.

(b) Auto-correlation distributions of the canary bits under normal operating conditions and under attack.

Figure 7.7: Test results of the carry-chain based TRNG.

detect the origin of the data. The canary numbers clearly manifest a detectable defect at the early attack stage while the raw numbers are still not significantly compromised.

7.4 Summary

This chapter covers our contributions of novel design methodologies for TRNG online test modules. The first approach, called TOTAL, empirically explores the effects of attacks over specific digital noise sources. The principle idea of the second design methodology, canary numbers, is based on the fact that different entropy source modules and extraction modules may have different level of testability. Modules with high testability can be utilized to generate canary numbers, which will be compromised and trigger alarms before the real random numbers have any detectable defects.

Chapter 8

Design example: ES-TRNG

Confucius said, "To know that we know what we know,
and that we do not know what we do not know,
that is true knowledge."

-Henry David Thoreau, Walden

Content Source

Yang, B., Rožić, V., Grujić, M., Mentens, N., and Verbauwhede, I. ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling, In *Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2018, volume 3.
Contribution: Main author.

The content of this chapter was originally published at [121]. The proposed TRNG is the culmination of our studies on TRNGs, where the high-resolution sampling, the non-linearity of the delay chain and the stochastic model of the ring oscillator are deployed to design this novel TRNG.

Our goal of this work is to develop an ultra-lightweight, AIS-31 compliant TRNG with conservative entropy estimation. The contributions of this work are the following:

1. We propose a novel, lightweight randomness generator called Edge Sampling based True Random Number Generator (ES-TRNG). ES-TRNG

is a timing jitter based TRNG, suitable for both ASIC and FPGA implementations.

2. We present two design techniques to achieve a better sampling efficiency, while keeping the implementation simple. These two techniques, called variable-precision phase encoding and repetitive sampling, are used for optimizing our ES-TRNG but they possibly have applications in other TRNG designs.

3. We analyze the security of the ES-TRNG using a mathematical model based on the stochastic nature of the timing jitter and the sampling procedure.

4. We implement the ES-TRNG on a commercial FPGA as a case study to prove the design concept and validate the stochastic model.

5. We discuss the dangers of using unrealistic assumptions of the jitter strength in ring oscillator based TRNGs.

8.1 Notation and Definitions

- $f_{\mu,\sigma}(x)$ – The probability density of $\mathcal{N}(\mu, \sigma^2)$:

$$f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} . \tag{8.1}$$

- $\rho_X(x)$ – The probability density of the random variable X defined over the domain $\mathbb{R}$:

$$\int_{-\infty}^{\alpha} \rho_X(x)dx = Pr(X \leq \alpha),\ \forall \alpha \in \mathbb{R}. \tag{8.2}$$

- $\rho_{X|a}(x)$ – The conditional probability density of the random variable X defined over the domain $\mathbb{R}$, given that the event a occurred:

$$\int_{-\infty}^{\alpha} \rho_{X|a}(x)dx = Pr(X \leq \alpha|a),\ \forall \alpha \in \mathbb{R}. \tag{8.3}$$

- Given a set $S = [a_1, b_1) \cup [a_2, b_2) \cup \cdots \cup [a_i, b_i)$, where the $[a_1, b_1) \cdots [a_i, b_i)$ are mutually disjoint real intervals, an integral of function $f(x)$ over S is

the sum of the integral of $f(x)$ over all intervals:

$$\int_S f(x)dx = \sum_{k=1}^{i} \int_{a_k}^{b_k} f(x)dx. \tag{8.4}$$

- Given a set of discrete distributions $F(\Theta)$ defined over the same domain and parameterized by Θ, which is distributed according to the distribution G (referred to as *mixing distribution*), then *the compound distribution* J of $F(\Theta)$ and G is given by integrating parameter Θ against the probability density ρ_G of the mixing distribution G:

$$Pr(J = i) = \int_G Pr(F(\Theta) = i)\rho_G(\Theta)d\Theta\,, \tag{8.5}$$

  where the integration is done across the whole domain of G and i denotes an arbitrary element in the domain of $F(\Theta)$.

- The probability distribution of the sum of two independent random variables is equal to the convolution of their distributions. If $f(x)$ and $g(x)$ are the probability distributions of two independent random variables, their convolution is:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x - \mu)g(\mu)d\mu. \tag{8.6}$$

8.2 ES-TRNG Architecture

In this section, we present the architecture of the ES-TRNG and the design rationale over conventional design criteria. These design criteria include: a compact implementation, a reasonably high throughput, feasibility on various implementation platforms and a low engineering effort. Our ES-TRNG architecture is based on high-precision edge sampling. The randomness source of the ES-TRNG is the timing phase jitter from a free-running ring oscillator.

Two novel techniques are used to improve the throughput and reduce the resource consumption. The first technique is called variable-precision phase encoding. By using a selective high-precision sampling process, this technique enables a compact implementation and a short jitter accumulation time. The second technique is repetitive sampling, which allows multiple samples within a single system clock cycle. Due to the repetitive sampling, the ES-TRNG can obtain a higher throughput. By using a fully digital architecture and not relying

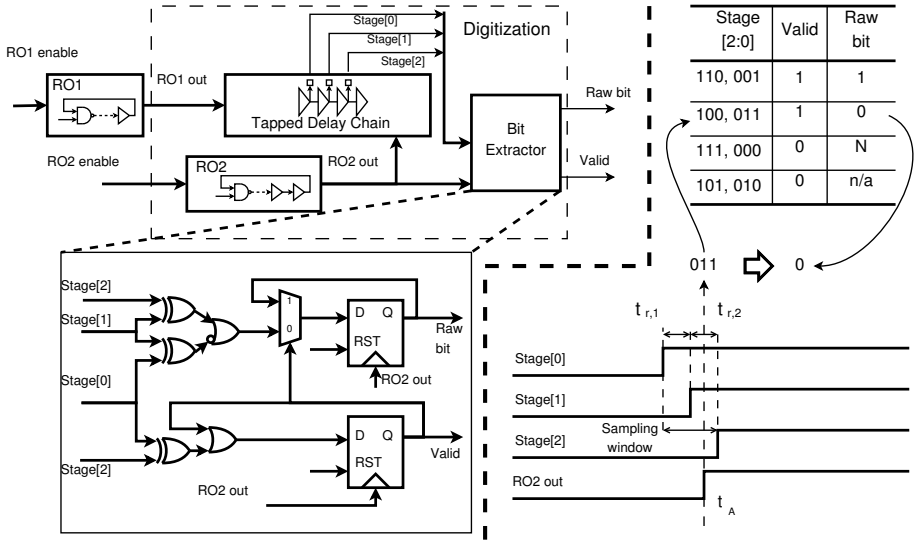


Figure 8.1: The architecture and operation principle of the digital noise source.

on any technology specific components, we obtain a design feasible on a wide range of implementation platforms.

The architecture of the digital noise source is shown in the left part of Figure 8.1. The entropy source denoted by *RO1* is implemented as a free-running ring oscillator with an enable signal. The average period of *RO1* is denoted by T_{01}. The output signal of *RO1* propagates through the *Digitization* module.

The digitization module consists of three main components, namely, a tapped delay chain, a sampling free-running ring oscillator *RO2* and a bit extractor. The average period of *RO2* is denoted by T_{02}. The used tapped delay chain consists of four cascaded delay elements. The first and the last delay elements are used as isolation buffers to provide a similar input and output load for the two delay elements in the middle. These middle elements form a two-stage delay chain. The $0 \rightarrow 1$ delay and the $1 \rightarrow 0$ delay of the first stage in the tapped delay chain are denoted by $t_{r,1}$ and $t_{f,1}$ respectively. Similarly, notations $t_{r,2}$ and $t_{f,2}$ are used for the rising delay and the falling delay of the second stage. The output of this two-stage tapped delay chain is sampled using three FFs (Flip-Flops) at the rising edge of the output signal of RO2.

As depicted on the right part of Figure 8.1, the sampled three-bit signal $Stage[2:0]$ is processed using the *Bit Extractor*. According to the truth table in Figure 8.1, the bit extractor encodes the three-bit input $Stage[2:0]$ to a

Table 8.1: Platform and design parameters.

Platform parameters.	
$t_{r,1}$	The $0 \rightarrow 1$ delay of the 1^{st} stage in the tapped delay chain
$t_{r,2}$	The $0 \rightarrow 1$ delay of the 2^{nd} stage in the tapped delay chain
$t_{f,1}$	The $1 \rightarrow 0$ delay of the 1^{st} stage in the tapped delay chain
$t_{f,2}$	The $1 \rightarrow 0$ delay of the 2^{nd} stage in the tapped delay chain
D	The duty cycle of the free-running RO1
σ_m^2/t_m	The variance of a white-noise jitter accumulated during the measurement time t_m
Design parameters.	
T_{01}	The average free-running RO1 period
T_{02}	The period of the sampling clock RO2
T_{CLK}	The period of the system clock
t_A	Jitter accumulation time

single data bit called *Raw bit* and a strobe signal *Valid*. In the notation used in this truth table, N stands for non-valid value which is not used to generate output bits.

The lower part of Figure 8.1 provides the architecture of the bit extractor. This module has a compact implementation using a single LUT and two FFs on modern FPGAs. The simplicity of the bit extractor also results in a low latency which enables correct operation at high clock frequencies.

8.2.1 Platform and Design Parameters

Table 8.1 summarizes the relevant parameters. Platform parameters are those parameters that are specific to an implementation platform and their values have to be obtained experimentally. Parameter σ_m^2/t_m reflects the strength of the white noise in the timing jitter. Parameters $t_{r/f,1/2}$ are properties of hardware primitives on FPGA fabric. Design parameters of the proposed TRNG are the periods T_{01} and T_{02} of *RO1* and *RO2*, the system clock frequency, and the jitter accumulation time.

8.2.2 Two Novel Techniques

In order to achieve a compact implementation with high throughput, the proposed design utilizes two novel techniques, namely variable-precision phase encoding and repetitive sampling. The central idea is to repeat the sampling

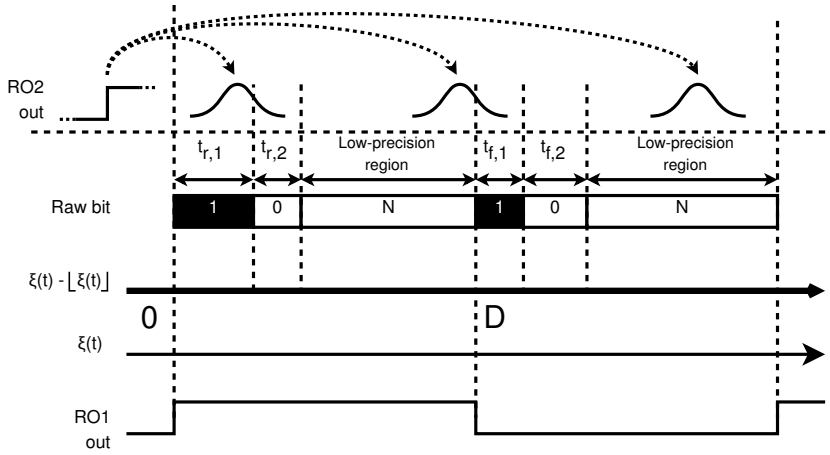


Figure 8.2: Variable-precision phase encoding.

and ignore all samples from the low-precision region (the region where values 000 and 111 are sampled in the tapped delay chain).

Variable-precision Phase Encoding

The variable-precision phase encoding technique is shown in Figure 8.2. This technique is enabled by using both the tapped delay chain and the bit extractor. The bottom part of the figure shows a single period of *RO1*. Symbol $\xi(t)$ denotes the normalized time in the unit of T_{01}. The phase of the oscillator *RO1*, denoted by $\xi(t) - \lfloor \xi(t) \rfloor$, changes periodically increasing from 0 to 1 within a single period.

The position of the captured edge is encoded into a raw bit. Since the delays $t_{r/f,1}$ and $t_{r/f,2}$ of the tapped delay chain are much smaller than the oscillation period T_{01}, the digitization module captures the oscillator phase with high precision around signal edges when the phase value is around 0 or D. This region is called the *high precision region*. The samples from this region are encoded by either 0 or 1 depending on the phase. In the remainder of the cycle, the edge is captured with low precision, i.e. only the correct half-period can be determined from the captured data. This region is denoted by the *low precision region*. The samples from this region are not used (encoded as N).

The sampling of the delay chain is triggered by the rising edge of the signal *RO2 out*. Due to the accumulated timing jitter, the relative sampling position follows a Gaussian distribution. Increased accumulation time t_A leads to a

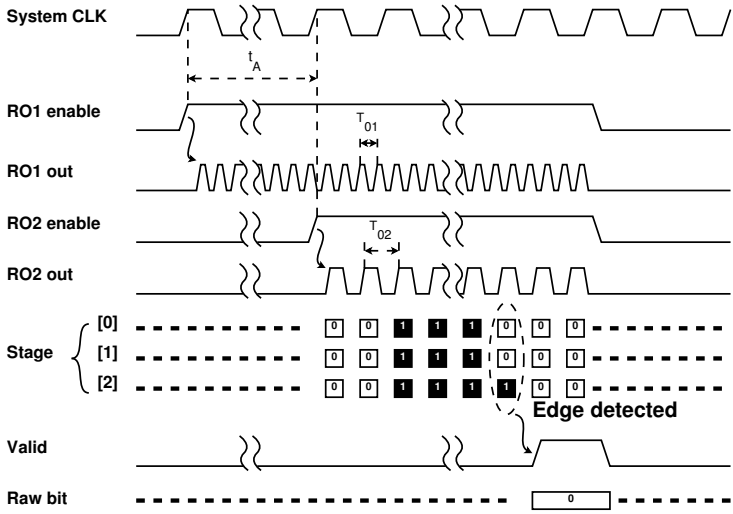


Figure 8.3: Timing diagrams of repetitive sampling.

wider jitter distribution. The expected value of the sampling time is determined by t_A, the number of ignored low-precision samples and the initial phase offset between $RO1$ *out* and $RO2$ *out*. This expected value (the center of the Gaussian distribution) can appear at any position, as indicated at the top of the figure.

We note that using a delay line with more than two elements would extend the region that is sampled with high precision, at the cost of increased area and energy consumption. A design using the precise sampling for all phase values is the DC-TRNG presented in Section 4.3.

Repetitive Sampling

The second technique used in the proposed design is called repetitive sampling. The proposed digitization module has a small critical path, which enables the digital noise source operating at a higher frequency than other components. Repetitive sampling is synchronized to the high-frequency signal $RO2$ *out*, aiming to reduce the time needed to hit the high-precision region, thereby improving the throughput. Once the high-precision region is hit, a *Valid* signal is generated.

The timing diagram of the digital noise source is depicted in Figure 8.3 to illustrate the repetitive sampling. The entropy source *RO1* is reset before generating a bit and then enabled for a period of time t_A. The parameter t_A is

chosen to allow accumulating enough jitter at the entropy source. After time t_A has passed, the sampling oscillator *RO2* is enabled. The $RO1$ *out* signal edge propagating through the tapped delay chain is sampled using $RO2$ *out*. This sampling can be repeated multiple times within a single system clock cycle, because the frequency of a free-running *RO2* can be higher than the system clock frequency available on FPGAs. Once the tapped delay chain stages $Stage[2:0]$ reach the high-precision region, the *Valid* signal will be set, the raw bit will be encoded and *RO1* and *RO2* will be reset.

8.3 Security Analysis

The formal security analysis provides the theoretical guarantees for the quality of the generated bits. These guarantees are achieved by developing an entropy estimator based on the stochastic model of the TRNG. The conservative entropy estimator provides a lower bound on the entropy based on the design parameters and the platform parameters. This estimator is used early in the design stage to guide the choice of the design parameters.

8.3.1 Assumptions

The assumptions about the physical processes in the entropy source are the starting point of the security analysis.

In this design the entropy is extracted from the timing jitter of a free-running ring oscillator. Our first assumption is that some amount of white (Gaussian) noise is present in the entropy source. The property of the white noise is that its observations are independent of each other and independent of any other noise source in the system. According to the central limit theorem, the variance of this noise increases linearly with the jitter accumulation time. The platform parameter σ_m^2/t_m is obtained experimentally. Our second assumption is that the value of this parameter is not overestimated. This means that a conservative estimation of this parameter's value must be made during the measurement procedure.

We further assume that other noise sources are present in the entropy source. These include flicker noise (low frequency noise), telegraph noise (popcorn noise) and the global noise from the power supply. None of these sources are exploited in this design because either their observations are correlated, the noise parameters are not measured or the noise source can be manipulated by the attacker. In order to provide a conservative estimation of entropy, the

impact of these noise sources will be treated as deterministic and known to the attacker.

In addition, we assume that the sampled raw bits are independent because the circuit is reset before generating each bit [13]. We note that this doesn't imply that the raw bits are identically distributed.

Finally, we assume that the frequencies of the two free-running ring oscillators and the delays of the elements in the tapped delay line are measured with sufficient precision, such that the measurement error doesn't significantly affect the entropy estimation.

8.3.2 Entropy Source

We will analyze the entropy source using the phase model of the ring oscillator with duty cycle D (in practice $D \approx 0.5$). The phase $\xi \in \mathbb{R}$ of the RO increases over time; integer values i correspond to the rising edges of the output signal and the values $i + D, i = 0, 1, ...$ correspond to the falling edges. The entropy source is reset before generating a new bit, so we assume that the oscillations always start from $\xi(0) = 0$. The phase ξ is affected by both deterministic and white noise sources:

$$\xi(t) = \frac{t}{T_{01}} + \xi_{Deterministic}(t) + \xi_{Gaussian}(t)\,. \tag{8.7}$$

Since the average value of the white noise is zero, the expected value of the phase after time t is

$$E(\xi(t)) = \frac{t}{T_{01}} + E(\xi_{Deterministic}(t))\,. \tag{8.8}$$

The contribution of the deterministic noise sources $E(\xi_{Deterministic}(t))$ cannot be estimated with reasonable precision. Therefore, for entropy estimation we will always consider the worst-case value of this term.

The standard deviation of the phase is greater than the standard deviation of the white noise:

$$\sigma^2(\xi(t)) = \sigma^2\left(\frac{t}{T_{01}} + \xi_{Deterministic}(t) + \xi_{Gaussian(t)}\right)$$

$$\text{(by independence)} \qquad = \sigma^2\left(\frac{t}{T_{01}} + \xi_{Deterministic}(t)\right) + \sigma^2\left(\xi_{Gaussian}(t)\right)$$

$$\geqslant \sigma^2(\xi_{Gaussian}(t))$$

$$= \frac{\sigma_m \cdot t}{T_{01}^2 \cdot t_m} \,. \tag{8.9}$$

This bound on the standard deviation can be computed for any value of t, given the physical and design parameters.

8.3.3 Digitization

The ring oscillator is sampled repeatedly at moments $t_A, t_A + T_{02}, t_A + 2T_{02}, ..$ until a valid sample is detected. We denote the phase at these moments with $X_0, X_1, ...$

$$X_i = \xi(t_A + i \cdot T_{02}) \,,\; i = 0, 1... \tag{8.10}$$

Here we introduce an approximation that will be used for estimating probabilities:

■**Approximation 1:**

$$E(X_i) \approx E(X_0) + i \cdot \Delta\xi \,, \tag{8.11}$$

where $\Delta\xi = T_{02}/T_{01}$. This assumption is justified by the fact that, for sufficiently high frequencies, the impact of the correlated noise is negligible compared to other randomness generating processes. This assumption, while not always explicitly stated, is used in state-of-the-art stochastic models of TRNG designs [37, 43].

For the security analysis of the digitization, we define the normalized platform parameters: $d_{r,1},\ d_{f,1},\ d_{r,2},\ d_{f,2}$ denote the normalized delays of the tapped delay chain ($d_{r,l} = t_{r,l}/T_{01}$ and $d_{f,l} = t_{f,l}/T_{01}$ for $l \in \{1, 2\}$); D denotes the duty cycle of the entropy source.

In order to simplify the following analysis, we denote the standard deviation of the phase at time t_A as $\sigma_{t_A} = \sigma(X_0)$. We use $\sigma_{T_{02}} = \sigma(X_i - X_{i-1})$ to denote the standard deviation of the superimposed phase, which is the result of the white noise accumulated in T_{02}. We would like to note that the white noise accumulated during the time period between X_i and X_{i-1} for any i is independent of each other and also independent of the white noise accumulated in accumulation time t_A.

The sampling circuit maps the phase value into the output bit. We introduce the mapping $s : \mathbb{R} \to \{0, 1, N\}$, where the symbol N denotes that a non-valid

value is detected. Function $s(X)$ can be formally defined as:

$$s(x) = \begin{cases} 1, \text{ for } x \in S_1 \\ 0, \text{ for } x \in S_0 \\ N, \text{ for } x \in S_N \, , \end{cases} \tag{8.12}$$

where S_1, S_0 and S_N are unions of mutually disjoint real intervals defined as:

$$S_1 = \bigcup_{k=-\infty}^{\infty} \Big\{ [k, k + d_{r,1}) \cup [k + D, k + D + d_{f,1}) \Big\}, \tag{8.13}$$

$$S_0 = \bigcup_{k=-\infty}^{\infty} \Big\{ [k + d_{r,1}, k + d_{r,1} + d_{r,2}) \cup [k + D + d_{f,1}, k + D + d_{f,1} + d_{f,2}) \Big\}, \tag{8.14}$$

$$S_N = \bigcup_{k=-\infty}^{\infty} \Big\{ [k + d_{r,1} + d_{r,2}, k + D) \cup [k + D + d_{f,1} + d_{f,2}, k + 1) \Big\}. \tag{8.15}$$

We stress that the sets S_1, S_0 and S_N are mutually disjoint and that:

$$S_1 \cup S_0 \cup S_N = \mathbb{R}. \tag{8.16}$$

For clarity reasons, we introduce the function $g(x) : \mathbb{R} \to \{0, 1\}$.

$$g(x) = \begin{cases} 1, \text{ for } x \in S_N \\ 0, \text{ for } x \in S_1 \cup S_0 \, . \end{cases} \tag{8.17}$$

We note that $g(x)$ is a periodic function with the following property:

For any function $f : \mathbb{R} \to \mathbb{R}$ and any $A \subset \mathbb{R}$:

$$\int_{A \cap S_N} f(x)dx = \int_{A} f(x) \cdot g(x)dx \, . \tag{8.18}$$

A special case of this property is:

$$\int_{-\infty}^{\infty} f(x) \cdot g(x)dx =$$

$$\int_{S_N} f(x)dx = \sum_{k=-\infty}^{\infty} \left[\int_{k+d_{r,1}+d_{r,2}}^{k+D} f(x)dx + \int_{k+D+d_{f,1}+d_{f,2}}^{k+1} f(x)dx \right]. \tag{8.19}$$

In line with these definitions and properties, we start the analysis of the digitization by examining the first sample taken at time t_A.

The phase at the first sampling moment is a normally distributed random variable X_0. The probability density function of this variable is:

$$\rho_{X_0}(x) = f_{E(X_0),\sigma_{t_A}}(x). \tag{8.20}$$

Let Y_i denote the sampled values, $Y_i = s(X_i)$. For convenience, we use the notation $\mu_i = E(X_i) - \lfloor E(X_i) \rfloor$. We can compute the probabilities of Y_0 as:

$$Pr(Y_0 = 1) = \int_{S_1} \rho_{X_0}(x)dx\,, \tag{8.21}$$

$$Pr(Y_0 = 0) = \int_{S_0} \rho_{X_0}(x)dx\,, \tag{8.22}$$

$$Pr(Y_0 = N) = \int_{S_N} \rho_{X_0}(x)dx = 1 - Pr(Y_0 = 1) - Pr(Y_0 = 0). \tag{8.23}$$

To simplify the notation, we let EV_i denote the event $Y_0 = N, \cdots, Y_i = N$. For example, the EV_0 denotes the event $Y_0 = N$. The second sample is only taken when EV_0 occurs. The i_{th} sample is taken when EV_{i-1} occurs.

From Equations (8.19) and (8.23), it follows that:

$$Pr(EV_0) = Pr(Y_0 = N) = \int_{S_N} \rho_{X_0}(t)dt = \int_{-\infty}^{\infty} \rho_{X_0}(t) \cdot g(t)dt. \tag{8.24}$$

For events EV_i, the following property holds:

$$Pr(EV_{i-1}, Y_i = 0) + Pr(EV_{i-1}, Y_i = 1) + Pr(EV_{i-1}, Y_i = N) = Pr(EV_{i-1}). \tag{8.25}$$

An intuitive example of the repetitive sampling is shown in Figure 8.4. The probability density function $\rho_{X_0}(x)$ of the random variable $X_0 = \xi(t_A)$ is located at the left part of the figure. This distribution covers three types of regions under the curve $\rho_{X0}(x)$: the white regions, the black regions and the shaded regions, which correspond to segments of $\rho_{X_0}(X)$ over the sets S_0, S_1 and S_N respectively. The area of the white regions is equal to $Pr(Y_0 = 0)$, while the area of the black regions and the shaded regions are equal to $Pr(Y_0 = 1)$ and

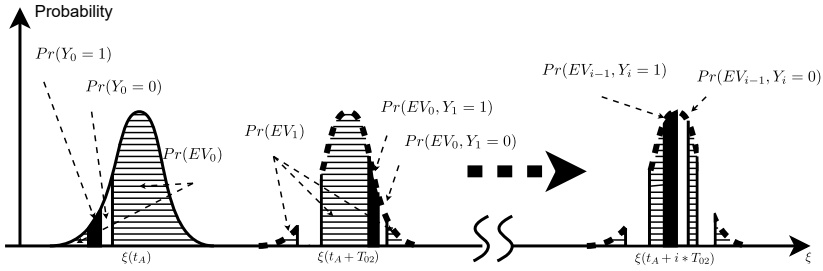


Figure 8.4: The intuitive interpretation of the repetitive sampling.

$Pr(Y_0 = N)$. It is described by Equations (8.21), (8.22) and (8.23). If the realization of X_0 is located at the shaded regions, the second sample is needed.

The middle distribution corresponds to the second sample. The middle distribution also covers three types of regions. The area of the white regions, the black regions and the shaded regions are equal to $Pr(EV_0, Y_1 = 0)$, $Pr(EV_0, Y_1 = 1)$ and $Pr(EV_1)$.

If the event EV_{i-1} occurs, the distribution of the i_{th} sample shown at the right part of the figure can be derived from the distribution of the $(i-1)_{th}$ sample.

Under the **Approximation 1**, Equation (8.10) can be rewritten as:

$$X_i = \xi(t_A) + i \cdot \frac{T_{02}}{T_{01}} + \sum_{k=1}^{i} Z_k = X_0 + i \cdot \frac{T_{02}}{T_{01}} + \sum_{k=1}^{i} Z_k, \tag{8.26}$$

where Z_i denotes a series of random variables defined as:

$$Z_i = \xi_{Gaussian}(t_A + i \cdot T_{02}) - \xi_{Gaussian}(t_A + (i-1) \cdot T_{02}). \tag{8.27}$$

Informally speaking, Z_i is the superimposed component of $\xi(t_A + i \cdot T_{02})$ caused by the white noise accumulated during the time interval $[t_A + (i-1) \cdot T_{02}, t_A + i \cdot T_{02})$. Z_i is a normally distributed random variable $\mathcal{N}(0, \sigma^2_{T_{02}})$.

The goal of the entropy estimation is to compute the binary probability of the raw bits. The computation of this binary probability requires $Pr(EV_{i-1}, Y_i = 1)$. Starting from the $Pr(Y_0 = 1)$ given by Equation (8.21), we first compute $\rho_{X_1|EV_0}(x)$ as:

$$\rho_{X_1|EV_0}(x) = \frac{1}{Pr(EV_0)} \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}(x-\mu) \cdot \rho_{X_0}\left(\mu - \frac{T_{02}}{T_{01}}\right) \cdot g\left(\mu - \frac{T_{02}}{T_{01}}\right) d\mu. \tag{8.28}$$

The proof of the claim in Equation (8.28)

From Equation (8.26), we have:

$$X_1 = X_0 + \frac{T_{02}}{T_{01}} + Z_1, \tag{8.29}$$

where T_{02}/T_{01} is a constant. The random variables X_0 and Z_1 are mutually independent.

As the first step, from the definition of $g(x)$, we can derive the conditional probability of X_0 given EV_0:

$$\begin{aligned}
\rho_{X_0|EV_0}(x) &= \frac{d}{d\alpha}[Pr(X_0 \le \alpha | EV_0)] \\
&= \frac{1}{Pr(EV_0)} \cdot \frac{d}{d\alpha}[Pr(X_0 \le \alpha, EV_0)] \\
&= \frac{1}{Pr(EV_0)} \cdot \frac{d}{d\alpha}[Pr(X_0 \le \alpha, X_0 \in S_N)] \\
&= \frac{1}{Pr(EV_0)} \cdot \frac{d}{d\alpha}\left[\int_{(-\infty,\alpha]\cap S_N} \rho_{X_0}(x)dx\right] \\
\text{(by Equation (8.18))} &= \frac{1}{Pr(EV_0)} \cdot \frac{d}{d\alpha}\left[\int_{-\infty}^{\alpha} \rho_{X_0}(x) \cdot g(x)dx\right] \\
&= \frac{1}{Pr(EV_0)} \cdot \rho_{X_0}(x) \cdot g(x)\,.
\end{aligned} \tag{8.30}$$

The Z_1 is independent of X_0, thus we have:

$$\rho_{Z_1|EV_0}(x) = \rho_{Z_1}(x). \tag{8.31}$$

We derive the probability density of $X_0 + Z_1$, given the condition EV_0:

$$\begin{aligned}
\rho_{(X_0+Z_1)|EV_0}(x) &= \rho_{Z_1|EV_0} * \rho_{X_0|EV_0})(x) \\
\text{(by Equation (8.31))} &= (\rho_{Z_1} * \rho_{X_0|EV_0})(x) \\
\text{(by convolution)} &= \int_{-\infty}^{\infty} \rho_{Z_1}(x-\mu) \cdot \rho_{X_0|EV_0}(\mu)d\mu
\end{aligned}$$

$$\text{(by Equation (8.30))} = \frac{1}{Pr(EV_0)} \cdot \int_{-\infty}^{\infty} \rho_{Z_1}(x-\mu) \cdot \rho_{X_0}(\mu) \cdot g(\mu) d\mu$$

$$\text{(by Equation (8.27))} = \frac{1}{Pr(EV_0)} \cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}(x-\mu) \cdot \rho_{X_0}(\mu) \cdot g(\mu) d\mu. \quad (8.32)$$

As a final step, we derive the Equation (8.28) as:

$$\begin{aligned}
\rho_{X_1|EV_0}(x) &= \rho_{(X_0+Z_1+\frac{T_{02}}{T_{01}})|EV_0}(x) = \rho_{(X_0+Z_1)|EV_0}\left(x - \frac{T_{02}}{T_{01}}\right) \\
&= \frac{1}{Pr(EV_0)} \\
&\cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}\left(x - \frac{T_{02}}{T_{01}} - \mu\right) \cdot \rho_{X_0}(\mu) \cdot g(\mu) d\mu \\
\text{(by Equation (8.32))} &= \frac{1}{Pr(EV_0)} \\
&\cdot \int_{-\infty}^{\infty} \rho_{Z_1}\left(x - \left(\mu + \frac{T_{02}}{T_{01}}\right)\right) \cdot \rho_{X_0}(\mu) \cdot g(\mu) d\mu \\
&= \frac{1}{Pr(EV_0)} \cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}(x-\mu) \cdot \rho_{X_0}\left(\mu - \frac{T_{02}}{T_{01}}\right) \cdot g\left(\mu - \frac{T_{02}}{T_{01}}\right) d\mu. \quad (8.33)
\end{aligned}$$

An informal but intuitive interpretation of Equation (8.28) is shown in Figure 8.5. The left part of the figure is the probability density function of $\rho_{X_0}(x)$. The black, white and shaded area under the curve correspond to $Pr(Y_0 = 1)$, $Pr(Y_0 = 0)$ and $Pr(EV_0)$ respectively. The second sample is taken, only when the event EV_0 occurs. The shaded region is shifted to the right by T_{02}/T_{01}, which corresponds to the dashed curve. The probability density function $\rho_{Z_1}(x)$ is shifted to positions with probabilities according to the dashed curve. The weighted and shifted results are added together. The result is shown at the right bottom of this Figure 8.5.

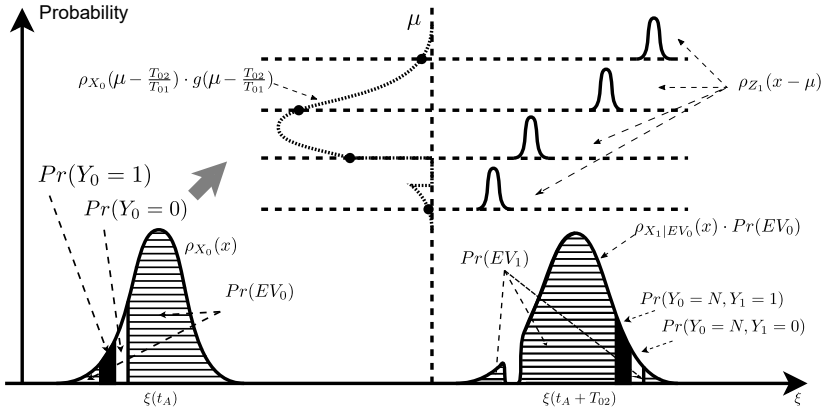


Figure 8.5: The intuitive interpretation of the first sample and the second sample.

Now the conditional probability function $\rho_{X_i|EV_{i-1}}(x)$, $(i > 1)$ can be derived iteratively as:

$$\rho_{X_i|EV_{i-1}}(x) = \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}(x - \mu) \cdot \rho_{X_{i-1}|EV_{i-2}}\left(\mu - \frac{T_{02}}{T_{01}}\right) \cdot g\left(\mu - \frac{T_{02}}{T_{01}}\right) d\mu. \tag{8.34}$$

The formal derivation of Equation (8.34) can be found below.

The proof of the claim in Equation (8.34)

The derivation of Equation (8.34) is similar to Section 8.3.3.

From Equation (8.26), we have:

$$X_i = X_{i-1} + \frac{T_{02}}{T_{01}} + Z_i, \tag{8.35}$$

where T_{02}/T_{01} is a constant. The random variables X_{i-1} and Z_i are mutually independent.

As a first step, we derive $\rho_{X_{i-1}|EV_{i-1}}(x)$ as:

$$\begin{aligned}
\rho_{X_{i-1}|EV_{i-1}}(x) &= \frac{d}{d\alpha}[Pr(X_{i-1} \leq \alpha | EV_{i-1})] \\
&= \frac{1}{Pr(EV_{i-1})} \cdot \frac{d}{d\alpha}[Pr(X_{i-1} \leq \alpha, EV_{i-1})] \\
&= \frac{1}{Pr(EV_{i-1})} \cdot \frac{d}{d\alpha}[Pr(X_{i-1} \leq \alpha, Y_{i-1} = N, EV_{i-2})] \\
&= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \frac{d}{d\alpha}[Pr(X_{i-1} \leq \alpha, Y_{i-1} = N | EV_{i-2})] \\
&= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \frac{d}{d\alpha}[Pr(X_{i-1} \leq \alpha, X_{i-1} \in S_N | EV_{i-2})] \\
&= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \frac{d}{d\alpha}\left[\int\limits_{(-\infty,\alpha]\cap S_N} \rho_{X_{i-1}|EV_{i-2}}(x)dx\right] \\
\text{(by Equation (8.18))} &= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \frac{d}{d\alpha}\left[\int\limits_{-\infty}^{\alpha} \rho_{X_{i-1}|EV_{i-2}}(x) \cdot g(x)dx\right] \\
&= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \rho_{X_{i-1}|EV_{i-2}}(x) \cdot g(x). \qquad (8.36)
\end{aligned}$$

Z_i is independent of EV_{i-1}, thus we have:

$$\rho_{Z_i|EV_{i-1}}(x) = \rho_{Z_i}(x). \qquad (8.37)$$

We derive the probability density of $X_{i-1} + Z_i$, given the condition EV_{i-1}:

$$\begin{aligned}
\rho_{(X_{i-1}+Z_i)|EV_{i-1}}(x) &= (\rho_{Z_i|EV_{i-1}} * \rho_{X_{i-1}|EV_{i-1}})(x) \\
\text{(by Equation (8.37))} &= (\rho_{Z_i} * \rho_{X_{i-1}|EV_{i-1}})(x) \\
\text{(by convolution)} &= \int\limits_{-\infty}^{\infty} \rho_{Z_i}(x-\mu) \cdot \rho_{X_{i-1}|EV_{i-1}}(\mu)d\mu \\
\text{(by Equation (8.33))} &= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \int\limits_{-\infty}^{\infty} \rho_{Z_i}(x-\mu) \cdot \rho_{X_{i-1}|EV_{i-2}}(\mu) \cdot g(\mu)d\mu
\end{aligned}$$

$$\text{(by Equation (8.27))} = \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}(x-\mu) \cdot \rho_{X_{i-1}|EV_{i-2}}(\mu) \cdot g(\mu) d\mu. \tag{8.38}$$

As a final step, we derive the Equation (8.34) as:

$$\rho_{X_i|EV_{i-1}}(x) = \rho_{(X_{i-1}+Z_i+\frac{T_{02}}{T_{01}})|EV_{i-1}}(x) = \rho_{(X_{i-1}+Z_i)|EV_{i-1}}\left(x - \frac{T_{02}}{T_{01}}\right)$$

$$\begin{aligned}\text{(by Equation (8.38))} &= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}\left(x - \frac{T_{02}}{T_{01}} - \mu\right) \\ &\quad \cdot \rho_{X_{i-1}|EV_{i-2}}(\mu) \cdot g(\mu) d\mu \\ &= \frac{Pr(EV_{i-2})}{Pr(EV_{i-1})} \cdot \int_{-\infty}^{\infty} f_{0,\sigma_{T_{02}}}(x-\mu) \cdot \rho_{X_{i-1}|EV_{i-2}}\left(\mu - \frac{T_{02}}{T_{01}}\right) \\ &\quad \cdot g\left(\mu - \frac{T_{02}}{T_{01}}\right) d\mu. \end{aligned} \tag{8.39}$$

Now we can compute probabilities $Pr(EV_i)$ as:

$$\begin{aligned} Pr(EV_i) &= Pr(EV_{i-1}, Y_i = N) \\ &= Pr(EV_{i-1}) \cdot Pr(Y_i = N | EV_{i-1}) \\ &= Pr(EV_{i-1}) \cdot \int_{S_N} \rho_{X_i|EV_{i-1}}(x) dx. \end{aligned} \tag{8.40}$$

Starting with Equations (8.24) and (8.28), we can apply Equations (8.34) and (8.40) iteratively to compute probabilities $Pr(EV_i)$.

8.3.4 Binary Probabilities

Now we can compute the following probabilities:

$$Pr(EV_{i-1}, Y_i = 1) = Pr(EV_{i-1}) \cdot \int_{S_1} \rho_{X_i|EV_{i-1}}(t) dt\,, \tag{8.41}$$

$$Pr(EV_{i-1}, Y_i = 0) = Pr(EV_{i-1}) \cdot \int_{S_0} \rho_{X_i|EV_{i-1}}(t)dt\,. \tag{8.42}$$

$Pr(EV_{i-1}, Y_i = 0)$ can be computed using Equation (8.25), (8.41) and (8.42).

The value of raw bit b is determined by the sequence $(Y_0 Y_1 \cdots Y_j)$ where j is the smallest integer such that $Y_j \neq N$. Therefore, the binary probability of a raw bit can be computed as:

$$Pr(b = 1) = Pr(Y_0 = 1) + \sum_{k=1}^{\infty} Pr(EV_{k-1}, Y_k = 1)\,. \tag{8.43}$$

8.3.5 Entropy Claim

Binary probabilities are computed starting from the platform parameters using Equations (8.9), (8.17), (8.20), (8.24), (8.28), (8.34), (8.41) and (8.43). The only unknown parameter is μ_0 which is equal to the phase of the oscillator at the moment t_A. The value of μ_0 depends on the global noise, low-frequency noises and the operating conditions. For this reason, it cannot be predicted with reasonable precision at design time. The conservative entropy claim is made by examining the effects of μ_0 on the binary probabilities and using the value that results in the lowest entropy. This procedure is shown in the example in Section 8.5.

8.4 Experimental Validation of the Model

We use an experimental approach to validate the proposed stochastic model, i.e. to check if the behavior of the physical TRNG matches the behavior predicted by the model. For this purpose, we implement the proposed design on a Xilinx Spartan-6 FPGA. In this experiment we monitor the number of toggles N_T of *RO2* during the sampling phase before the raw bit is generated.

Before applying the stochastic model, we have to specify the design parameters and measure the platform parameters. Design parameters T_{01} and T_{02} are chosen indirectly by selecting the number of delay elements in *RO1* and *RO2*. We implement *RO1* and *RO2* using a single look-up table and three look-up tables respectively. The periods of *RO1* and *RO2* are then measured using two individual ripple counters, the measurement results are $T_{01} = 2171.8\ ps$ and $T_{02} = 2739.8\ ps$. In regard to other design parameters, the period of the system

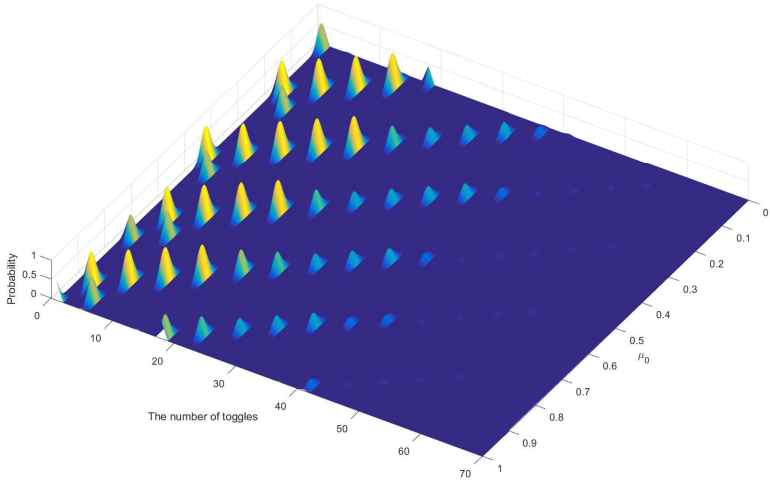


Figure 8.6: Probabilities of toggle numbers under different μ_0.

clock is chosen to be $T_{CLK} = 10\ ns$ and the jitter accumulates for 9 system clock cycles.

We follow the methodologies proposed in Section 3.3 to measure the platform parameters. The obtained propagation delays are $t_{r,1} = 22.25\ ps$, $t_{r,2} = 24.12\ ps$, $t_{f,1} = 35.93\ ps$ and $t_{f,2} = 40.90\ ps$. The duty cycle measurement is $D = 0.43$ and the white-noise jitter strength is $\sigma_m^2/t_m = 0.0029\ ps$.

The number of toggles for a specific value of μ_0 is distributed as follows:

$$Pr(N_T = i) = Pr(EV_{i-1}, Y_i \neq N) = Pr(EV_{i-1}, Y_i = 1) + Pr(EV_{i-1}, Y_i = 0)\,, \tag{8.44}$$

which is easily computed by applying Equations (8.41) and (8.42).

Figure 8.6 shows the predicted toggle distributions for different values of parameter $\mu_0 \in (0, 1)$.

We note that due to a very low jitter strength, the variance of the accumulated jitter at the sampling phase is much lower than the period of *RO1*, i.e. $\sigma_{T_{02}} \ll 1$. A consequence of this low jitter is that the probability $Pr(Y_i = N)$ is very low in cases when μ_i is close to the high-precision region ($\mu_i \approx 0$, $\mu_i \approx D$ and $\mu_i \approx 1$). Conversely, this probability is very high when μ_i is far away from the high-precision phase region (i.e. $\mu_i \approx D/2$ and $\mu_i \approx (1 + D)/2$). We check how this affects the distribution of N_T for the selected oscillator periods. Let's first

observe that $T_{02}/T_{01} \approx 5/4$. Therefore:

$$\mu_{i+1} \approx \mu_i + 1/4 - \lfloor \mu_i + 1/4 \rfloor \,, \tag{8.45}$$

$$\mu_{i+2} \approx \mu_i + 1/2 - \lfloor \mu_i + 1/2 \rfloor \,, \tag{8.46}$$

$$\mu_{i+4} \approx \mu_i \,. \tag{8.47}$$

Since the high-precision regions are close to the beginning of the cycle and the middle of the cycle, we get:

$$Pr(Y_i = N) \approx Pr(Y_{i+2} = N) \,. \tag{8.48}$$

The result is that the N_T distribution for odd values is very different from the distribution for the even N_T values. This effect is observed in the model for any value of μ_0 as well as in the experimental data shown in Figure 8.7a. For clarity, this distribution is presented using two graphs, one for the odd N_T values and one for the even values.

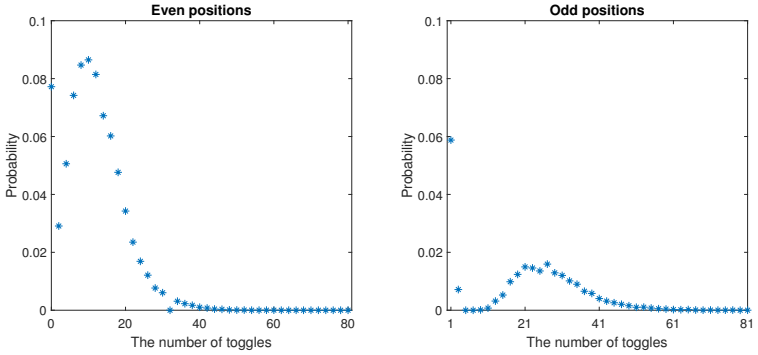
In order to model the distribution of N_T we have to make assumptions about μ_0. In the used experimental setting, it is reasonable to assume that μ_0 follows a Gaussian distribution. We note that we don't make this assumption for computing the entropy, but rather use the worst case value. Therefore, the N_T distribution is computed as the compound distribution of $\mathcal{N}(\mu, \sigma^2)$ and the distribution given by Equation (8.44). For $\mathcal{N}(0.384, 0.09^2)$, the model reasonably approximates experimental data, as shown in Figure 8.7b. We note that $\mathcal{N}(0.384, 0.09^2)$ is unrelated to the platform parameters and is not used for entropy estimation. It is only used to validate the stochastic model.

8.5 Implementation and Security Evaluation

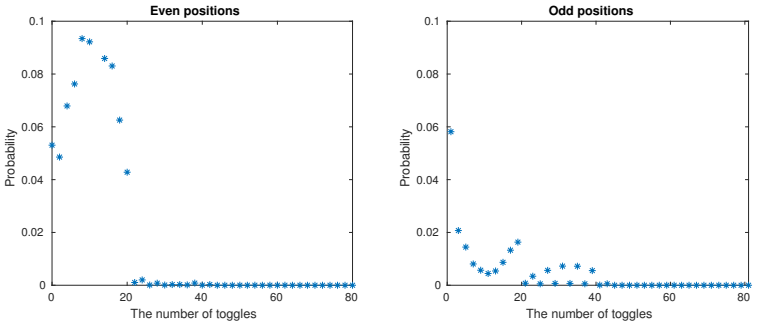
8.5.1 Xilinx FPGA Implementation

In this section, we present a Xilinx Spartan-6 FPGA implementation of the ES-TRNG. In addition to LUTs and sequential elements (flip-flops and latches), this FPGA has high-speed carry chain primitives called CARRY4. These primitives can be configured to work as a tapped delay line.

Figure 8.8 shows the implementation of the core of the proposed digital noise source. RO1 is implemented as a high-speed oscillator using a single LUT, RO2 is implemented using 3 LUTs. A single CARRY4 element is used to implement the tapped delay chain.



(a) Probability distribution of toggles computed from the collected experimental data.



(b) Probability distribution of toggles derived from the stochastic model.

Figure 8.7: Toggles probability distributions predicted by the model and obtained by experiments. Each distribution is shown using two graphs, one for the even values (left) and one for the odd values (right).

The entire core of the TRNG shown in Figure 8.8 is implemented using only one CARRY4 element, 10 LUTs and 5 flip-flops. In addition to this core, the design contains a parity filter for post-processing and a control circuit for setting the enable and reset signals. A system clock of 100 MHz is used in the design.

8.5.2 Application of the Model

In this section, we applied the stochastic model to derive optimal design parameters. The platform parameters measurement is described in Section 8.4. The stochastic model enables us to calculate the Shannon entropy and Min entropy for any value of μ_0 for a specific accumulation time. The conservative

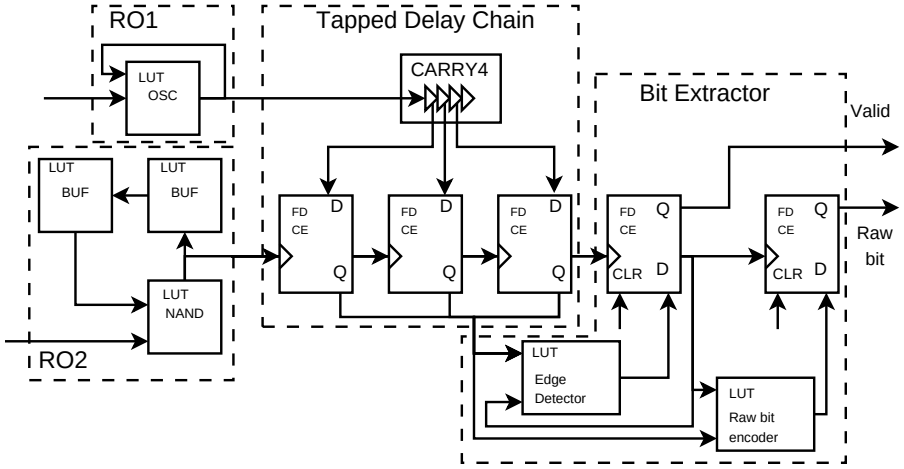


Figure 8.8: FPGA implementation of the digital noise source.

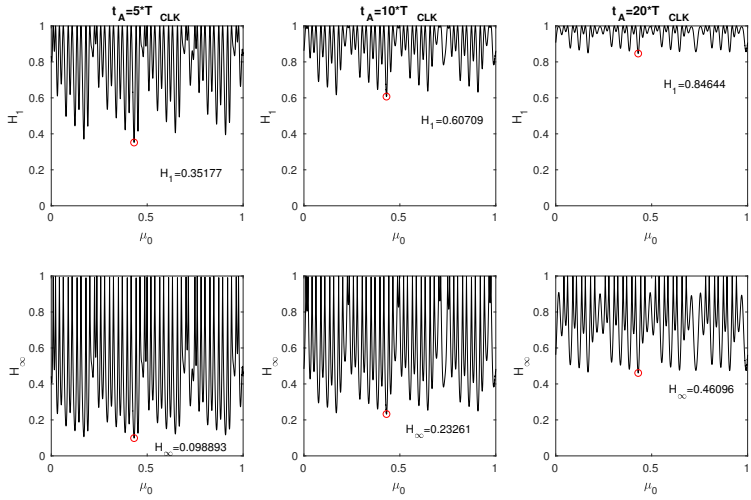


Figure 8.9: The estimated H_1 and H_∞ for different accumulation time.

entropy estimation is made by using the global minimum. This procedure is illustrated in Figure 8.9. This figure shows the entropy estimations for three different values of accumulation time ($t_A = 5 \cdot T_{CLK}$, $10 \cdot T_{CLK}$ and $20 \cdot T_{CLK}$). The global minima are indicated by red dots.

Table 8.2: Model verification

t_A (ns)	H_∞ (Stochastic model)	H_∞ (NIST SP800-90B)	H_1 (Stochastic model)
50	0.099	0.578	0.352
100	0.233	0.661	0.607
200	0.461	0.757	0.846
300	0.548	0.764	0.900

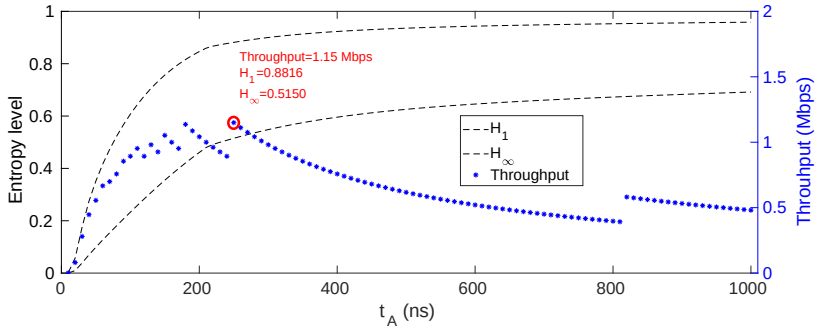


Figure 8.10: The lower bound estimation for H_1 and H_∞, the expected throughput after post-processing for different accumulation times.

To confirm that our entropy estimation is indeed conservative, we applied the entropy assessment procedure from [99] on the collected raw random numbers. The results are summarized in Table 8.2. The results reported by the NIST entropy assessment python package are always higher than the Min entropy estimation derived from the proposed stochastic model. This result is expected because the stochastic model estimation is based on the worst case scenario.

According to the standard [52], a minimal Shannon entropy level of 0.997 per bit is expected in the internal random numbers. This goal is achieved when the bias $\varepsilon_{internal}$ of the internal random numbers is upper-bounded by 3.2%. This Shannon entropy level cannot be achieved by accumulating the jitter less than 1 μs, which leads to a throughput smaller than 1 $Mbps$. However, a higher throughput can be obtained by using a shorter accumulation time followed by a simple algorithmic post-processing. A parity filter of order n_f, which combines n_f consecutive input bits into one output bit using a XOR function, reduces the bias $\varepsilon_{internal}$ to:

$$\varepsilon_{internal} = 2^{n_f-1} \cdot \varepsilon_{raw}^{n_f}\,, \tag{8.49}$$

where ε_{raw} denotes the bias of the raw random numbers.

To optimize the throughput of the ES-TRNG, we computed the minimal n_f required to achieve a Shannon entropy level of 0.997. Figure 8.10 shows the lower bound estimation for H_1 and H_∞ for t_A ranging from 20 ns to 1 μs. For every t_A within this range, we calculate the expected throughput of the internal random numbers after the required n_f-stage parity filter. From the figure, we can see that the throughput increases monotonically when $t_A \leq 110$ ns. Within this region, the required n_f reduces rapidly because of the increasing of estimated Shannon entropy. There are several discontinuous segments on the throughput plot when $t_A > 110$ ns. Each segment corresponds to a different required n_f. When $t_A > 1000ns$, the throughput is always lower than 1 $Mbps$. The global maximum of 1.15 $Mbps$ is obtained at $t_A = 250$ ns where $n_f = 3$. At this point, the H_∞ of the raw random numbers estimated by the stochastic model is 0.515 which is more conservative than the 0.86 derived from the standard [99]. The post-processed data achieves a throughput higher than 1 $Mbps$ with a Shannon entropy level higher than 0.997. For verification, a 10 MB sequence of internal numbers was generated and tested using T0-T5 tests proposed in the AIS-31. The sequence passes all the tests.

8.5.3 Intel FPGA Implementations

In order to show the portability of our design to different FPGA platforms, we have also implemented ES-TRNG on an Intel Cyclone V FPGA. The basic building blocks of this FPGA are adaptive logic modules (ALMs) that contain 8-input flexible LUTs, 2 fast adders and 2 DFFs.

The tapped delay line is implemented using fast carry chains in dedicated adders. Since 1 ALM already contains two carry stages, we would need only 2 ALMs to implement a structure equivalent to the CARRY4 primitive in Xilinx FPGA. However, due to considerable differences of the carry stages' propagation delays, we opted to use the output of every second carry stage. In this way, we obtained more balanced delays of the resulting delay chain stages at the price of increased resource utilization - 4 ALMs instead of 2. To be able to accurately determine the frequency of $RO1$, we implemented $RO1$ with 2 LUTs, while for $RO2$ we used 3 LUTs. The oscillation periods of $RO1$ and $RO2$ were determined with ripple counters and their values are $T_{01} = 1745.68ps$ and $T_{02} = 3020.068ps$ respectively. The bit extractor module is implemented by using a single ALM, thus achieving a compact implementation.

The measurement of the Intel Cyclone V FPGA platform parameters is performed in the same way as for the Xilinx Spartan-6 in Section 5. The propagation delays are calculated as follows: $t_{r,1} = 67.316ps$, $t_{r,2} = 68.316ps$, $t_{f,1} = 52.044ps$ and $t_{f,2} = 50.544ps$. The white-noise jitter strength is

Table 8.3: The comparison with existing TRNG implementations

TRNG types	Stoch. models	Xilinx FPGAs				Intel FPGAs			
		Families	Area	Bit rate [Mbits/s]	Design effort	Families	Area	Bit rate [Mbits/s]	Design effort
Open-loop [7] [23] [24]	Y	Virtex5	n/a LUTs 64 Latches	20	MP MR	Stratix EP1S25	140 Stratix cells (∼14 LABs)	20	MP MR
FIGARO [47] [91]	N	Virtex2	n/a	n/a	n/a	CycloneV	n/a	n/a	MP noMR
BRAM [42]	N	Spartan3	n/a	100	MP noMR	-	-	-	-
CMAFC [60]	N	Virtex5	128 LUTs	2	MP MR	-	-	-	-
DCTRNG [83] [40]	Y	Spartan6	67 slices	14.3	MP noMR	CycloneV	230 ALMs (23 LABs)	11.1	MP MR
			40 slices	1.53					
MERO [107] [88]	Y	Virtex2	n/a	50	n/a	StratixII	9 ALUTs	2.5	n/a
ERO [73]	Y	Spartan6	46 LUTs 19 FFs	0.0042	MP MR	CycloneV	34 LUTs 20 FFs	0.0027	MP MR
COSO [73]	Y	Spartan6	18 LUTs 3 FFs	0.54	MP MR	CycloneV	13 LUTs 3 FFs	1.44	MP MR
MURO [73] [89] [94]	Y	Spartan6	521 LUTs 131 FFs	2.57	noMP noMR	CycloneV	525 LUTs 130 FFs	2.2	noMP noMR
PLL [73]	Y	Spartan6	34 LUTs 14 FFs	0.44	noMP noMR	CycloneV	24 LUTs 14 FFs	0.6	noMP noMR
TERO [73] [105]	Y	Spartan6	39 LUTs 12 FFs	0.625	MP MR	CycloneV	46 LUTs 12 FFs	1	MP MR
STR [73] [17]	Y	Spartan6	346 LUTs 256 FFs	154	MP MR	CycloneV	352 LUTs 256 FFs	245	MP MR
This work	**Y**	**Spartan6**	**10 LUTs+5FFs + (6 LUTs+6FFs) t_A counter**	**1.15**	**MP noMR**	**CycloneV**	**10 LUTs+6FFs + (6 LUTs+6FFs) t_A counter**	**1.067**	**MP noMR**

$\sigma_m^2/t_m = 0.020ps$, while the duty cycle D of $RO1$ is 0.58. After measuring the physical parameters, we determined the design parameters, as previously described. The jitter accumulates for $t_A = 230ns$, while the parity filter has to be of order $n_f = 3$ to obtain a Shannon entropy level of 0.997. The obtained throughput of the ES-TRNG on an Intel Cyclone V FPGA is $1.067Mbps$. We observe that although the white-noise jitter strength on an Intel Cyclone V is higher than on a Xilinx Spartan 6, the propagation delays on a Xilinx Spartan-6 are substantially lower, leading to similar throughputs on both FPGAs.

8.6 Results and Comparison

Table 8.3 shows the comparison to the state-of-the-art TRNG designs for both Xilinx and Intel FPGAs. The second column of the table indicates the

availability of stochastic models for each TRNG design. The utilization of hardware resources is reported in the third column. Only the basic units in the FPGA are reported, such as LUTs, DFFs and Slices. Special dedicated primitives, like PLL building blocks for the PLL-TRNG, are not included here. Among the Xilinx TRNG designs listed in the table, the ES-TRNG achieves the smallest hardware footprint. Our ES-TRNG generates more than 1 *Mbps* internal random numbers with an estimated minimal Shannon entropy level of 0.997. The table also indicates the design effort needed for FPGA implementations: manual placement (MP) and manual routing (MR). MP is critical for some TRNG designs, because the quality of those TRNGs are sensitive to the relative spatial location of their building blocks. For TRNG designs based on identical delays or balanced routing, MR cannot be avoided.

As can be seen in Table 8.3, the ES-TRNG has one of the smallest area consumptions compared to TRNGs implemented on Cyclone V FPGAs. The only TRNG designs with comparable implementation footprints are the COSO-TRNG and the PLL-TRNG [73]. It is worthwhile to note that although the COSO-TRNG has a higher throughput than the ES-TRNG, it requires laborious manual placement and routing, which has to be performed for every target device. On the other hand, the PLL-TRNG does not require any manual placement or routing on Cyclone V FPGAs, but it occupies dedicated PLL modules and provides a 40% lower throughput compared to the ES-TRNG.

Here we would like to point out a problem with the comparison of results in this research domain that became relevant in recent years. Many TRNG designs were developed before the publication of evaluation standards [52] and [99]. Therefore, these old designs and some of the recent ones are not provided with stochastic models that are required today. Without a proper security analysis or by using debunked or simplified assumptions, it is possible to achieve a better hardware performance than if AIS-31 criteria are strictly followed. For example, a designer may base the entropy estimation solely on the results of the statistical tests. If this overestimation is used to guide the choice of design parameters, higher targets for throughput, area and energy are more easily achieved. This has the unfortunate effect that designs with a more rigorous security analysis appear worse in terms of hardware performance when compared to their predecessors.

In particular, in ring oscillator based designs, the jitter strength is often the critical parameter that significantly affects the performance of the final hardware implementation. Some recent works on jitter measurement on FPGAs [37, 58, 120] showed that the strength of the Gaussian noise is lower by an order of magnitude compared to the values that were used in older TRNG designs. For comparison, we carried out the ES-TRNG design procedure assuming that the jitter strength is five time higher than measured (this is similar to the jitter

strength reported in [89]). The obtained design parameters were $t_A = 30\ ns$ and $n_f = 2$ which would result in a throughput of 6.25 $Mbps$.

8.7 Summary

In this chapter, we present a novel true random number generator called ES-TRNG. Two techniques, namely variable-precision phase encoding and repetitive sampling, are used to increase the throughput and the entropy of the proposed generator and to reduce the hardware footprint. The digital noise source is implemented using only 10 LUTs and 5 DFFs of a Xilinx Spartan-6 FPGA, and achieves a throughput of 1.15 Mb/s with 0.997 bits of Shannon entropy. On Intel Cyclone V FPGAs, this implementation uses 10 LUTs and 6 DFFs, and achieves a throughput of 1.07 $Mbps$. The proposed generator is backed up with a security analysis based on the stochastic model of the entropy source.

Chapter 9

Conclusion

The way was long, and wrapped in gloom did seem,
As I urged on to seek my vanished dream.

-Qu Yuan,Li Sao

In this chapter, we revisit the contributions of this PhD thesis while discussing possible short-term and long-term future work.

9.1 Summary of Contributions

This thesis focuses on true random number generators. We have research contributions on each of the TRNG components. All of our contributions are FPGA compatible.

Entropy source We propose an on-chip jitter measurement methodology for TRNG designs that exploits the timing jitter from ring oscillators as the source of randomness. The proposed method is designed to measure the strength of the white noise while filtering out other components of the noise. A conservative estimation of the jitter strength is critical for analyzing the security of jitter-based TRNGs.

Digital noise source We present two novel digital noise sources, namely DC-TRNG and ES-TRNG. The entropy sources of these two TRNGs are based on

timing jitter from free-running ring oscillators. Both the DC-TRNG and the ES-TRNG benefit from the high sampling precision achieved by using the delay chain primitive on FPGAs. However, due to two special techniques utilized by the ES-TRNG, the security analysis of these two TRNGs is significantly different.

Post-processing In this chapter, we analyze an ancient divination algorithm in the context of post-processing. We explore implementation dimensions of a post-processing algorithm called Iterated Von Neumann. A search algorithm is proposed to find the architecture which achieves the optimal throughput under a specific hardware budget.

Online tests We propose two implementations of statistical black-box testing for IID sources and non-IID sources. Two novel methodologies for designing an online test module are introduced. TOTAL is an empirical design method based on analyzing attack effects on the target TRNG. Various design rationales and tradeoffs are also discussed. The canary number methodology paves a different way to design online tests. The testability of the entropy source and the entropy extractor is analyzed to design the canary entropy source and the canary entropy extractor. The principle behind the canary methodology is that with a better testability, the canary entropy source and the canary entropy extractor clearly manifest a detectable defect before the real entropy source and entropy extractor are compromised.

9.2 Short-Term Future Work

In this section we present possible short-term future work, which extends the research work presented in this thesis.

Online tests The TOTAL design methodology can be extended in the following ways:

- In the original paper, the TOTAL methodology was applied to design the online tests for two timing jitter based TRNGs. As a future work, we plan to apply this methodology to other entropy sources.
- Oversampling and underpowering attacks were used as a case study in our original work. We can extend the existing work by including more attack methods, such as electromagnetic injection attacks.

- The usefulness of the TOTAL methodology highly depends on the richness of the statistical feature pool. However, there are unlimited statistical features, in other words, the feature pool can never be complete. This work can be extended by utilizing machine learning algorithms. Golden reference and under-attack datasets can be used as the training set to select useful features.

Digital noise source

- The techniques used in the ES-TRNG design can simply be applied to other TRNGs to improve the efficiency of entropy extraction. Benefiting from the high sampling precision, the throughput of other timing jitter based TRNGs can be vastly improved.
- A unified model for a ring oscillator based digital noise source can be derived to comprehend the sampling procedure. An initial step to proceed this work is comparing existing jitter based TRNGs including the PLL-TRNG, the MURO-TRNG, the DC-TRNG and the ES-TRNG.

Attack A bad implementation of a TRNG can potentially introduce unwanted vulnerabilities to a secure system. The frequency of free-running ring oscillators may leak side-channel information of its surrounding designs on the same device. There is a trend of implementing a TRNG and a PUF in a unified design with shared hardware components. For such systems, the statistical properties of the generated random numbers can potentially be used to guide an attack on the result of the PUF.

9.3 Long-Term Future Research Directions

Identification of the requirements of true randomness Many applications claim that a "good" random number generator is required for their security or performance. However, some of these applications do not clearly specify their requirements: a random sequence with good statistical properties, or the unpredictability of the random numbers. A long-term future work is to identify random number requirements for different applications, such as threshold implementations, gate-level masking and Gaussian sampling.

A universal post-processing algorithm. Post-processing algorithms in general are the subject of long-term future work. How to extract the entropy from

the noise source is critical to design a secure and efficient TRNG. There are unanswered questions such as:

- How to quantify and then reduce the entropy loss during post-processing? Can we use different algorithms rather than an invertible compression function?
- Is it possible to provide the verifiability of TRNGs, in other words, can users verify the unpredictability of the used TRNG, without putting trust in designers and semiconductor foundries? Such verifiability exists in QRNGs. According to [20], a violation of the CHSH inequality supported by experiments can be seen as an experimental confirmation that genuinely quantum randomness can be obtained.

Random number distribution in an IoT system. With the ongoing adoption of the IoT, it has become crucial for designers to regard security as a critical design requirement of IoT-related devices. However, it is still a challenge to integrate a TRNG into a low-end IoT device, such as a sensor node. One possible solution to this challenge is to implement TRNGs on gateways of the system, which have less implementation constraints. Distributing random numbers to individual IoT devices for different security applications is the challenge of this long-term future work.

Philosophical Questions

Does true randomness really exist?

Isn't true randomness just a result of insufficient observations and inadequate measures?

Bibliography

[1] Acosta, A. J., Bellido, M. J., Valencia, M., Barriga, A., and Huertas, J. L. Fully digital redundant random number generator in cmos technology. In *Solid-State Circuits Conference, 1993. ESSCIRC '93. Nineteenth European* (Sept 1993), vol. 1, pp. 198–201.

[2] Allini, E. N., Petura, O., Fischer, V., and Bernard, F. Optimization of the pll configuration in a pll-based trng design. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)* (March 2018), pp. 1265–1270.

[3] Arora, M. *The art of hardware architecture: Design methods and techniques for digital circuits*. Springer Science & Business Media, 2011.

[4] Barker, E., Kelsey, J., and Secretary, J. B. Nist draft special publication 800-90b recommendation for the entropy sources used for random bit generation, 2012.

[5] Baudet, M., Lubicz, D., Micolod, J., and Tassiaux, A. On the security of oscillator-based random number generators. *Journal of Cryptology 24*, 2 (Apr 2011), 398–425.

[6] Bellido, M. J., Acosta, A. J., Valencia, M., Barriga, A., and Huertas, J. L. Simple binary random number generator. *Electronics Letters 28*, 7 (March 1992), 617–618.

[7] Ben-Romdhane, M., Graba, T., and Danger, J.-L. Stochastic model of a metastability-based true random number generator. In *Trust and Trustworthy Computing* (Berlin, Heidelberg, 2013), M. Huth, N. Asokan, S. Čapkun, I. Flechais, and L. Coles-Kemp, Eds., Springer Berlin Heidelberg, pp. 92–105.

[8] Bernard, F., Fischer, V., and Valtchanov, B. Mathematical model of physical rngs based on coherent sampling. *Tatra Mountains Mathematical Publications 45*, 1 (2010), 1 – 14.

[9] Bernstein, D. J., Chang, Y., Cheng, C., Chou, L., Heninger, N., Lange, T., and van Someren, N. Factoring RSA keys from certified smart cards: Coppersmith in the wild. In *Advances in Cryptology - ASIACRYPT* (2013), pp. 341–360.

[10] Bochard, N., Bernard, F., Fischer, V., and Valtchanov, B. True-randomness and pseudo-randomness in ring oscillator-based true random number generators. *International Journal of Reconfigurable Computing 2010* (2010).

[11] Bock, H., Bucci, M., and Luzzi, R. An offset-compensated oscillator-based random bit source for security applications. In *Cryptographic Hardware and Embedded Systems - CHES 2004* (Berlin, Heidelberg, 2004), M. Joye and J.-J. Quisquater, Eds., Springer Berlin Heidelberg, pp. 268–281.

[12] Brown, G. W. History of rand's random digits.

[13] Bucci, M., and Luzzi, R. Design of testable random bit generators. In *Cryptographic Hardware and Embedded Systems – CHES 2005* (Berlin, Heidelberg, 2005), J. R. Rao and B. Sunar, Eds., Springer Berlin Heidelberg, pp. 147–156.

[14] Callegari, S., Rovatti, R., and Setti, G. Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos. *IEEE Transactions on Signal Processing 53*, 2 (feb 2005), 793–805.

[15] Chen, W., Che, W., Bi, Z., Wang, J., Yan, N., Tan, X., Wang, J., Min, H., and Tan, J. A 1.04 μW Truly Random Number Generator for Gen2 RFID tag. In *2009 IEEE Asian Solid-State Circuits Conference* (nov 2009), IEEE, pp. 117–120.

[16] Cherkaoui, A., Fischer, V., Aubert, A., and Fesquet, L. A self-timed ring based true random number generator. In *Proceedings - International Symposium on Asynchronous Circuits and Systems* (may 2013), IEEE, pp. 99–106.

[17] Cherkaoui, A., Fischer, V., Fesquet, L., and Aubert, A. A Very High Speed True Random Number Generator with Entropy Assessment. *Cryptographic Hardware and Embedded Systems (CHES)* (2013), 179–196.

[18] Ching, I. The yi king [the book of changes], tr. by james legge, 1822.

[19] Chor, B., Goldreich, O., Hasted, J., Freidmann, J., Rudich, S., and Smolensky, R. The bit extraction problem or t-resilient functions.

In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (Oct 1985), pp. 396–407.

[20] Clauser, J. F., Horne, M. A., Shimony, A., and Holt, R. A. Proposed experiment to test local hidden-variable theories. *Phys. Rev. Lett. 23* (Oct 1969), 880–884.

[21] Coppock, W. R., and Philbrook, C. A mathematical and physical analysis of circuit jitter with application to cryptographic random bit generation. *MQP Report* (2005).

[22] Corporation, R. *A million random digits with 100,000 normal deviates*. Free Press, 1955.

[23] Danger, J. L., Guilley, S., and Hoogvorst, P. Fast true random generator in fpgas. In *2007 IEEE Northeast Workshop on Circuits and Systems* (Aug 2007), pp. 506–509.

[24] Danger, J. L., Guilley, S., and Hoogvorst, P. High speed true random number generator based on open loop structures in fpgas. *Microelectron. J. 40*, 11 (Nov. 2009), 1650–1656.

[25] De Roover, C., and Steyaert, M. A 500 mV 650 pW random number generator in 130 nm CMOS for a UWB localization system. In *ESSCIRC 2010 - 36th European Solid State Circuits Conference* (sep 2010), IEEE, pp. 278–281.

[26] Debian Security Advisory. DSA-1571-1 openssl – predictable random number generator. `https://www.debian.org/security/2008/dsa-1571`. 2008.

[27] Dichtl, M. How to predict the output of a hardware random number generator. In *Cryptographic Hardware and Embedded Systems - CHES 2003* (Berlin, Heidelberg, 2003), C. D. Walter, Ç. K. Koç, and C. Paar, Eds., Springer Berlin Heidelberg, pp. 181–188.

[28] Dichtl, M. Bad and good ways of post-processing biased physical random numbers. In *Fast Software Encryption* (Berlin, Heidelberg, 2007), A. Biryukov, Ed., Springer Berlin Heidelberg, pp. 137–152.

[29] Dichtl, M., and Golić, J. D. High-speed true random number generation with logic gates only. In *Cryptographic Hardware and Embedded Systems - CHES 2007* (Berlin, Heidelberg, 2007), P. Paillier and I. Verbauwhede, Eds., Springer Berlin Heidelberg, pp. 45–62.

[30] Drutarovský, M., Šimka, M., Fischer, V., and Celle, F. A Simple PLL-Based True Random Number Generator for Embedded Digital Systems. *COMPUTING AND INFORMATICS 23*, 5-6 (2004), 501–515.

[31] Elias, P. The efficient construction of an unbiased random sequence. *Ann. Math. Statist. 43*, 3 (06 1972), 865–870.

[32] Favi, C., and Charbon, E. A 17ps time-to-digital converter implemented in 65nm fpga technology. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2009), FPGA '09, ACM, pp. 113–120.

[33] FIPS, P. 140-1. *Security Requirements for Cryptographic Modules 11* (1994).

[34] Fischer, V. A closer look at security in random number generators design. In *Constructive Side-Channel Analysis and Secure Design* (Berlin, Heidelberg, 2012), W. Schindler and S. A. Huss, Eds., Springer Berlin Heidelberg, pp. 167–182.

[35] Fischer, V., Bernard, F., Bochard, N., and Varchola, M. Enhancing security of ring oscillator-based trng implemented in fpga. In *2008 International Conference on Field Programmable Logic and Applications* (Sept 2008), pp. 245–250.

[36] Fischer, V., and Drutarovský, M. True random number generator embedded in reconfigurable hardware. In *Cryptographic Hardware and Embedded Systems - CHES 2002* (Berlin, Heidelberg, 2002), B. S. Kaliski, ç. K. Koç, and C. Paar, Eds., Springer Berlin Heidelberg, pp. 415–430.

[37] Fischer, V., and Lubicz, D. Embedded evaluation of randomness in oscillator based elementary trng. In *Cryptographic Hardware and Embedded Systems – CHES 2014* (Berlin, Heidelberg, 2014), L. Batina and M. Robshaw, Eds., Springer Berlin Heidelberg, pp. 527–543.

[38] Gimenez, G., Cherkaoui, A., Frisch, R., and Fesquet, L. Self-timed Ring based True Random Number Generator: Threat model and countermeasures. In *2017 2nd International Verification and Security Workshop, IVSW 2017* (jul 2017), IEEE, pp. 31–38.

[39] Grujić, M., Rožić, V., Yang, B., and Verbauwhede, I. A closer look at the delay-chain based trng. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (May 2018), pp. 1–5.

[40] Grujić, M., Yang, B., Rožić, V., and Verbauwhede, I. Towards inter-vendor compatibility of true random number generators for fpgas. In

2018 Design, Automation Test in Europe Conference Exhibition (DATE) (March 2018), pp. 1520–1523.

[41] Güneysu, T. True random number generation in block memories of reconfigurable devices. In *2010 International Conference on Field-Programmable Technology* (Dec 2010), pp. 200–207.

[42] Güneysu, T., and Paar, C. Transforming write collisions in block rams into security applications. In *2009 International Conference on Field-Programmable Technology* (Dec 2009), pp. 128–134.

[43] Haddad, P., Fischer, V., Bernard, F., and Nicolai, J. A physical approach for stochastic modeling of tero-based trng. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2015), Springer, pp. 357–372.

[44] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, J. A. Mining your ps and qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2012), Security'12, USENIX Association, pp. 35–35.

[45] ID Quantique SA. Quantis AIS 31 certified random number generator (RNG). *https://www.idquantique.com/random-number-generation/products/quantis-ais-31/* (Accessed:04-30-2018).

[46] Intel. Cyclone IV Device Handbook, March 2016.

[47] J. D. J. Golić. New methods for digital generation and postprocessing of random data. *IEEE Transactions on Computers 55*, 10 (Oct 2006), 1217–1229.

[48] Jennewein, T., Achleitner, U., Weihs, G., Weinfurter, H., and Zeilinger, A. A fast and compact quantum random number generator. *Review of Scientific Instruments 71*, 4 (apr 2000), 1675–1680.

[49] Jun, B., and Kocher, P. The Intel random number generator. *Cryptography Research Inc. white paper 27* (1999), 1–8.

[50] Kendall, M. G., and Smith, B. B. Randomness and random sampling numbers. *Journal of the royal Statistical Society 101*, 1 (1938), 147–166.

[51] Kerckhoffs, A. La cryptographic militaire. *Journal des sciences militaires* (1883), 5–38.

[52] Killmann, W., and Schindler, W. A Proposal for: Functionality classes for random number generators . BDI, Bonn.

[53] Kohlbrenner, P., and Gaj, K. An embedded true random number generator for fpgas. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays* (New York, NY, USA, 2004), FPGA '04, ACM, pp. 71–78.

[54] Koralov, L., and Sinai, Y. G. *Random Variables and Their Distributions. In: Theory of Probability and Random Processes.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 3–23.

[55] Lacharme, P. Post-processing functions for a biased physical random number generator. In *Fast Software Encryption* (Berlin, Heidelberg, 2008), K. Nyberg, Ed., Springer Berlin Heidelberg, pp. 334–342.

[56] Lenstra, A. K., Hughes, J. P., Augier, M., Bos, J. W., Kleinjung, T., and Wachter, C. Public keys. In *Advances in Cryptology – CRYPTO 2012* (Berlin, Heidelberg, 2012), R. Safavi-Naini and R. Canetti, Eds., Springer Berlin Heidelberg, pp. 626–642.

[57] Lozach, F., Ben-Romdhane, M., Graba, T., and Danger, J. L. Fpga design of an open-loop true random number generator. In *2013 Euromicro Conference on Digital System Design* (Sept 2013), pp. 615–622.

[58] Lubicz, D., and Bochard, N. Towards an oscillator based TRNG with a certified entropy rate. *IEEE Transactions on Computers 64*, 4 (2015), 1191–1200.

[59] Ma, Y., Lin, J., Chen, T., Xu, C., Liu, Z., and Jing, J. Entropy Evaluation for Oscillator-Based True Random Number Generators. In *CHES* (2014), pp. 544–561.

[60] Majzoobi, M., Koushanfar, F., and Devadas, S. Fpga-based true random number generation using circuit metastability with adaptive feedback control. In *Cryptographic Hardware and Embedded Systems – CHES 2011* (Berlin, Heidelberg, 2011), B. Preneel and T. Takagi, Eds., Springer Berlin Heidelberg, pp. 17–32.

[61] Marin, E., Singelée, D., Yang, B., Verbauwhede, I., and Preneel, B. On the feasibility of cryptography for a wireless insulin pump system. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2016), CODASPY '16, ACM, pp. 113–120.

[62] Marin, E., Singelée, D., Yang, B., Volski, V., Vandenbosch, G. A. E., Nuttin, B., and Preneel, B. Securing wireless neurostimulators. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2018), CODASPY '18, ACM, pp. 287–298.

[63] Markovski, S., Gligoroski, D., and Kocarev, L. Unbiased random sequences from quasigroup string transformations. In *Fast Software Encryption* (Berlin, Heidelberg, 2005), H. Gilbert and H. Handschuh, Eds., Springer Berlin Heidelberg, pp. 163–180.

[64] Marsaglia, G. The marsaglia random number cdrom including the diehard battery of tests of randomness, 1995. *URL http://www. stat. fsu. edu/pub/diehard* (2008).

[65] Marsaglia, G. DIEHARD Test suite. *Online: http://www.stat.fsu.edu/pub/diehard/* (Accessed:08-01-1998).

[66] Massari, N., Gasparini, L., Tomasi, A., Meneghetti, A., Xu, H., Perenzoni, D., Morgari, G., and Stoppa, D. 16.3 A 16×16 pixels spad-based 128-mb/s quantum random number generator with -74db light rejection ratio and -6.7ppm/°c bias sensitivity on temperature. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)* (Jan 2016), pp. 292–293.

[67] Menninga, H., Favi, C., Fishburn, M., and Charbon, E. A multi-channel, 10ps resolution, FPGA-based TDC with 300MS/s throughput for open-source PET applications. In *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2011 IEEE* (Oct 2011), pp. 1515–1522.

[68] Nyquist, H. Thermal Agitation of Electric Charge in Conductors. *Physical Review 32*, 1 (jul 1928), 110–113.

[69] Pareschi, F., Setti, G., and Rovatti, R. A Fast Chaos-based True Random Number Generator for Cryptographic Applications. In *2006 Proceedings of the 32nd European Solid-State Circuits Conference* (sep 2006), IEEE, pp. 130–133.

[70] Peres, Y. Iterating von neumann's procedure for extracting random bits. *The Annals of Statistics 20*, 1 (1992), 590–597.

[71] Petrie, C., and Connelly, J. A noise-based IC random number generator for applications in cryptography. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 47*, 5 (may 2000), 615–621.

[72] Petura, O., Mureddu, U., Bochard, N., and Fischer, V. Optimization of the PLL based TRNG design using the genetic algorithm. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (may 2017), IEEE, pp. 1–4.

[73] Petura, O., Mureddu, U., Bochard, N., Fischer, V., and Bossuet, L. A survey of AIS-20/31 compliant TRNG cores suitable for FPGA devices. In *FPL* (2016), pp. 1–10.

[74] Picek, S., Mariot, L., Yang, B., Jakobovic, D., and Mentens, N. Design of s-boxes defined with cellular automata rules. In *Proceedings of the Computing Frontiers Conference* (New York, NY, USA, 2017), CF'17, ACM, pp. 409–414.

[75] Picek, S., Sisejkovic, D., Rozic, V., Yang, B., Jakobovic, D., and Mentens, N. Evolving cryptographic pseudorandom number generators. In *Parallel Problem Solving from Nature – PPSN XIV* (Cham, 2016), J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, Eds., Springer International Publishing, pp. 613–622.

[76] Picek, S., Yang, B., and Mentens, N. A search strategy to optimize the affine variant properties of s-boxes. In *Arithmetic of Finite Fields* (Cham, 2016), S. Duquesne and S. Petkova-Nikova, Eds., Springer International Publishing, pp. 208–223.

[77] Picek, S., Yang, B., Rozic, V., and Mentens, N. On the construction of hardware-friendly 4×4 and 5×5 s-boxes. In *Selected Areas in Cryptography (SAC)* (Cham, 2016), R. Avanzi and H. Heys, Eds., Springer International Publishing, pp. 161–179.

[78] Picek, S., Yang, B., Rozic, V., Vliegen, J., Winderickx, J., De Cnudde, T., and Mentens, N. Prngs for masking applications and their mapping to evolvable hardware. In *Smart Card Research and Advanced Applications* (Cham, 2017), K. Lemke-Rust and M. Tunstall, Eds., Springer International Publishing, pp. 209–227.

[79] Rahman, M. T., Xiao, K., Forte, D., Zhang, X., Shi, J., and Tehranipoor, M. TI-TRNG: Technology independent true random number generator. In *Design Automation Conference (DAC)* (New York, New York, USA, 2014), ACM Press, pp. 1–6.

[80] Rényi, A. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics* (Berkeley, Calif., 1961), University of California Press, pp. 547–561.

[81] Robson, S., Leung, B., and Gong, G. Truly random number generator based on a ring oscillator utilizing last passage time. *IEEE Transactions on Circuits and Systems II: Express Briefs 61*, 12 (Dec 2014), 937–941.

[82] Rodriguez-Vazquez, A., Espejo-Meana, S., Huertas, J. L., and Martin, J. D. Analog building blocks for noise and truly random number generation in CMOS VLSI. *Solid-State Circuits Conference, 1990. ESSCIRC '90. Sixteenth European* (1990), 225–228.

[83] Rozic, V., Yang, B., Dehaene, W., and Verbauwhede, I. Highly efficient entropy extraction for true random number generators on fpgas. In *Proceedings of the 52Nd Annual Design Automation Conference* (New York, NY, USA, 2015), DAC '15, ACM, pp. 116:1–116:6.

[84] Rožić, V., Yang, B., Dehaene, W., and Verbauwhede, I. Iterating von neumann's post-processing under hardware constraints. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (May 2016), pp. 37–42.

[85] Rožić, V., Yang, B., Mentens, N., and Verbauwhede, I. Canary Numbers: Design for Light-weight Online Testability of True Random Number Generators. In *In NIST RBG Workshop* (2016), p. 5.

[86] Rožić, V., Yang, B., Vliegen, J., Mentens, N., and Verbauwhede, I. The monte carlo puf. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)* (Sept 2017), pp. 1–6.

[87] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., and Vo, S. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special-Pub:800-22 NIST.

[88] Santoro, R., Sentieys, O., and Roy, S. On-the-fly evaluation of fpga-based true random number generator. In *2009 IEEE Computer Society Annual Symposium on VLSI* (May 2009), pp. 55–60.

[89] Schellekens, D., Preneel, B., and Verbauwhede, I. Fpga vendor agnostic true random number generator. In *2006 International Conference on Field Programmable Logic and Applications* (Aug 2006), pp. 1–6.

[90] Schneier, Bruce. Random Number Bug in Debian Linux - Schneier on Security. *https://www.schneier.com/blog/archives/2008/05/random_number_b.html* (Accessed:07-23-2018).

[91] Schramm, M., Dojen, R., and Heigl, M. Experimental assessment of firo- and garo-based noise sources for digital trng designs on fpgas. In *2017 International Conference on Applied Electronics (AE)* (Sept 2017), pp. 1–6.

[92] Shannon, C. E. A mathematical theory of communication. *Bell System Technical Journal 27*, 3, 379–423.

[93] Stefanov, A., Gisin, N., Guinnard, O., Guinnard, L., and Zbinden, H. Optical quantum random number generator. *Journal of Modern Optics 47*, 4 (jul 2000), 595–598.

[94] Sunar, B., Martin, W. J., and Stinson, D. R. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on Computers 56*, 1 (Jan 2007), 109–119.

[95] Suresh, V. B., Antonioli, D., and Burleson, W. P. On-chip lightweight implementation of reduced nist randomness test suite. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (June 2013), pp. 93–98.

[96] Tippett, L. *Random Sampling Numbers. Arranged by L.H.C. Tippett, Etc.* [Tracts for Computers. no. 15.]. 1927.

[97] Tkacik, T. E. A Hardware Random Number Generator. *Cryptographic Hardware and Embedded Systems* (2002), 450–453.

[98] Tokunaga, C., Blaauw, D., and Mudge, T. True random number generator with a metastability-based quality control. *IEEE Journal of Solid-State Circuits 43*, 1 (Jan 2008), 78–85.

[99] Turan, M. S., Barker, E., Kelsey, J., McKay, K., Baish, M., and Boyle, M. Recommendation for the Entropy Sources Used for Random BitGeneration. NIST Special Publication 800-90B.

[100] Turing, A. M. Alan Turing's Manual for the Ferranti Mk. I.

[101] Uchida, K., Tanamoto, T., Ohba, R., Yasuda, S., and Fujita, S. Single-electron random-number generator (rng) for highly secure ubiquitous computing applications. In *Digest. International Electron Devices Meeting,* (Dec 2002), pp. 177–180.

[102] Valtchanov, B., Aubert, A., Bernard, F., and Fischer, V. Modeling and observing the jitter in ring oscillators implemented in FPGAs. In *DDECS* (2008), pp. 158–163.

[103] Vanhoef, M., and Piessens, F. Predicting, decrypting, and abusing wpa2/802.11 group keys. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, 2016), USENIX Association, pp. 673–688.

[104] Varchola, M., and Drutarovsk, Y. New fpga based trng principle using transition effect with built-in malfunction detection. In *International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices-CryptArchi. Prague (Czechia)* (2009), Citeseer, pp. 150–155.

[105] Varchola, M., and Drutarovsky, M. New high entropy element for fpga based true random number generators. In *Cryptographic Hardware and Embedded Systems, CHES 2010* (Berlin, Heidelberg, 2010), S. Mangard and F.-X. Standaert, Eds., Springer Berlin Heidelberg, pp. 351–365.

[106] Vaskova, A., López-Ongil, C., Jiménez-Horas, A., Millán, E. S., and Entrena, L. Robust cryptographic ciphers with on-line statistical properties validation. In *2010 IEEE 16th International On-Line Testing Symposium* (July 2010), pp. 208–210.

[107] Vasyltsov, I., Hambardzumyan, E., Kim, Y.-S., and Karpinskyy, B. Fast digital trng based on metastable ring oscillator. In *Cryptographic Hardware and Embedded Systems – CHES 2008* (Berlin, Heidelberg, 2008), E. Oswald and P. Rohatgi, Eds., Springer Berlin Heidelberg, pp. 164–180.

[108] Veljković, F., Rožić, V., and Verbauwhede, I. Low-cost implementations of on-the-fly tests for random number generators. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)* (March 2012), pp. 959–964.

[109] von Neumann, J. Various techniques used in connection with random digits. In *Monte Carlo Method* (Washington, D.C.: U.S. Government Printing Office, 1951), A. Householder, G. Forsythe, and H. Germond, Eds., National Bureau of Standards Applied Mathematics Series, 12, pp. 36–38.

[110] Wachs, M., and Ip, D. Design and integration challenges of building security hardware ip. In *Proceedings of the 52Nd Annual Design Automation Conference* (New York, NY, USA, 2015), DAC '15, ACM, pp. 177:1–177:6.

[111] Walker, J. HotBits: Genuine Random Numbers.

[112] Walker, S., and Foo, S. Evaluating metastability in electronic circuits for random number generation. In *Proceedings IEEE Computer Society Workshop on VLSI 2001. Emerging Technologies for VLSI Systems* (May 2001), pp. 99–101.

[113] Williams, F. C., and Kilburn, T. The university of manchester computing machine. In *1951 International Workshop on Managing Requirements Knowledge* (Dec 1951), pp. 57–57.

[114] Wold, K., and Petrović, S. Behavioral model of trng based on oscillator rings implemented in fpga. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems* (April 2011), pp. 163–166.

[115] Wold, K., and Petrović, S. Security properties of oscillator rings in true random number generators. In *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)* (April 2012), pp. 145–150.

[116] Wold, K., and Tan, C. H. Analysis and enhancement of random number generator in fpga based on oscillator rings. In *2008 International Conference on Reconfigurable Computing and FPGAs* (Dec 2008), pp. 385–390.

[117] Xilinx Inc. 7 Series FPGAs Configurable Logic Block User Guide (UG474).

[118] Xilinx Inc. Spartan-6 FPGA Configurable Logic Block User Guide (UG384).

[119] Yalcin, M. E., Suykens, J. A. K., and Vandewalle, J. True random bit generation from a double-scroll attractor. *IEEE Transactions on Circuits and Systems I: Regular Papers 51*, 7 (July 2004), 1395–1404.

[120] Yang, B., Rožić, V., Grujić, M., Mentens, N., and Verbauwhede, I. On-chip jitter measurement for true random number generators. *2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)* (2017), 91–96.

[121] Yang, B., Rožić, V., Grujić, M., Mentens, N., and Verbauwhede, I. ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling. In *Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* (2018), pp. 37–42.

[122] Yang, B., Rožić, V., Mentens, N., Dehaene, W., and Verbauwhede, I. Embedded hw/sw platform for on-the-fly testing of true random number generators. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)* (March 2015), pp. 345–350.

[123] Yang, B., Rožić, V., Mentens, N., Dehaene, W., and Verbauwhede, I. Total: Trng on-the-fly testing for attack detection using lightweight hardware. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)* (March 2016), pp. 127–132.

[124] Yang, B., Rožić, V., Mentens, N., and Verbauwhede, I. On-the-fly tests for non-ideal true random number generators. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (May 2015), pp. 2017–2020.

[125] Yang, K., Fick, D., Henry, M. B., Lee, Y., Blaauw, D., and Sylvester, D. A 23Mb/s 23pJ/b fully synthesized true-random-number generator in 28nm and 65nm CMOS. In *Digest of Technical Papers - IEEE International Solid-State Circuits Conference* (feb 2014), vol. 57, IEEE, pp. 280–281.

[126] Yoo, S.-K., Karakoyunlu, D., Birand, B., and Sunar, B. Improving the robustness of ring oscillator trngs. *ACM Trans. Reconfigurable Technol. Syst. 3*, 2 (May 2010), 9:1–9:30.

[127] Yu, M.-D., Verbauwhede, I., Devadas, S., and M'Raihi, D. A noise bifurcation architecture for linear additive physical functions. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on* (2014), IEEE, pp. 124–129.

[128] Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., and Verbauwhede, I. Rectangle: a bit-slice lightweight block cipher suitable for multiple platforms. *Science China Information Sciences 58*, 12 (Dec 2015), 1–15.

Curriculum

Bohan Yang was born on June 10th 1986 in Xianyang, China. He obtained a Bachelor's degree in Automation from Xi'an Jiaotong University, Xi'an, China in 2008. Then he continued to pursue his graduate study under the KU.Leuven-Tsinghua dual degree Master's programme, and received Master's degree from both KU Leuven and Tsinghua University in 2012. In October 2012, he joined the COSIC research group at the KU Leuven, Belgium as a PhD student.

List of publications

International Journals

[1] B. Yang, V. Rožić, M. Grujić, N. Mentens and I. Verbauwhede, ES-TRNG: A High-throughput, Low-area True Random Number Generator based on Edge Sampling, In *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, Issue 3, 2018.

[2] M. Grujić, V. Rožić, B. Yang, and I. Verbauwhede, Lightweight Prediction-Based Tests for On-Line Min-Entropy Estimation, In *IEEE Embedded Systems Letters*, 9(2), pp. 45-48, 2017.

[3] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, RECTANGLE: A Bit-slice Lightweight Block Cipher Suitable for Multiple Platforms, In *SCIENCE CHINA INFORMATION SCIENCES*, (22), 2015.

International Conferences and Workshops

[1] B. Yang, V. Rožić, M. Grujić, N. Mentens, and I. Verbauwhede, On-chip jitter measurement for true random number generators, In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, IEEE, 6 pages, 2017, (nominated for best paper).

[2] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, TOTAL: TRNG On-the-fly Testing for Attack detection using Lightweight hardware, In *Design, Automation and Test in Europe (DATE)*, IEEE, pp. 127-132, 2016, (nominated for best paper in automation category).

[3] B. Yang, V. Rožić, N. Mentens, and I. Verbauwhede, On-the-Fly Tests for Non-Ideal True Random Number Generators, In *IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, pp. 2017-2020, 2015.

[4] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, Embedded HW/SW Platform for On-the-Fly Testing of True Random Number Generators, In *Design, Automation and Test in Europe (DATE)*, IEEE, pp. 345-350, 2015.

[5] D. Šijačić, J. Balasch, B. Yang, S. Ghosh and I. Verbauwhede, Towards Efficient and Automated Side Channel Evaluations at Design Time, In *7th Internation Workshop on Security PROOFS for Embedded Systems (PROOFS)*, 2018

[6] M. Madau, M. Agoyan, J. Balasch, M. Grujić, P. Haddad, P. Maurine, V. Rožić, D. D. Singelée, B. Yang, and I. Verbauwhede, The impact of pulsed electromagnetic fault injection on true random number generators, In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, J. Daemen, and L. Sauvage (eds.), IEEE, 6 pages, 2018

[7] M. Grujić, V. Rožić, B. Yang, and I. Verbauwhede, A Closer Look at the Delay-Chain based TRNG, In *IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 5 pages, 2018.

[8] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. Van Battum, I. Verbauwhede, M. Wakker, and B. Yang, Design and Testing Methodologies for True Random Number Generators Towards Industry Certification, In *International IEEE European Test Symposium - ETS*, IEEE Computer Society, 10 pages, 2018.

[9] M. Grujić, B. Yang, V. Rožić, and I. Verbauwhede, Towards Inter-Vendor Compatibility of True Random Number Generators for FPGAs, In *Design, Automation and Test in Europe (DATE)*, IEEE, 4 pages, 2018.

[10] E. Marin, D. Singelée, B. Yang, V. Volskiy, G. Vandenbosch, B. Nuttin, and B. Preneel, Securing wireless neurostimulators, In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, ACM, pp. 287-298, 2018.

[11] T. Ashur, J. Delvaux, S. Lee, P. Maene, E. Marin, S. Nikova, O. Reparaz, V. Rožić, D. Singelée, B. Yang, and B. Preneel, A Privacy-Preserving Device Tracking System Using a Low-Power Wide-Area Network (LPWAN), In *16th International Conference on Cryptology and Network Security*, CANS 2017, Lecture Notes in Computer Science, Springer-Verlag, 22 pages, 2017.

[12] V. Rožić, B. Yang, J. Vliegen, N. Mentens, and I. Verbauwhede, The Monte Carlo PUF, In *27th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, 6 pages, 2017.

[13] S. Picek, L. Mariot, B. Yang, D. Jakobovic, and N. Mentens, Design of S-boxes Defined with Cellular Automata Rules, In *Proceedings of the 17th Conference on Computing Frontiers (CF)*, M. Becchi, and F. Palumbo (eds.), ACM, pp. 409-414, 2017.

[14] S. Picek, B. Yang, V. Rožić, J. Vliegen, J. Winderickx, T. De Cnudde, and N. Mentens, PRNGs for Masking Applications and Their Mapping to Evolvable Hardware, In *Smart Card Research and Advanced Applications (CARDIS)*, Lecture Notes in Computer Science 10146, K. Lemke-Rust, and M. Tunstall (eds.), Springer-Verlag, pp. 209-227, 2017.

[15] Y. Cao, V. Rožić, B. Yang, J. Balasch, and I. Verbauwhede, Exploring Active Manipulation Attacks on the TERO Random Number Generator, In *59th Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE, pp. 273-276, 2016.

[16] S. Picek, D. Sisejkovic, V. Rožić, B. Yang, D. Jakobovic, and N. Mentens, Evolving Cryptographic Pseudorandom Number Generators, In *Parallel Problem Solving from Nature-PPSN XV*, Lecture Notes in Computer Science 9921, Springer-Verlag, 10 pages, 2016.

[17] S. Picek, B. Yang, V. Rožić, and N. Mentens, On the Construction of Hardware-friendly 4X4 and 5X5 S-boxes, In *Selected Areas in Cryptography (SAC)*, Lecture Notes in Computer Science, Springer-Verlag, 12 pages, 2016.

[18] D. Šijačić, A. B. Kidmose, B. Yang, S. Banik, B. Bilgin, A. Bogdanov, and I. Verbauwhede, Hold Your Breath, PRIMATEs Are Lightweight, In *Selected Areas in Cryptography (SAC)*, Lecture Notes in Computer Science, Springer-Verlag, 20 pages, 2016.

[19] S. Picek, B. Yang, and N. Mentens, A Search Strategy to Optimize the Affine Variant Properties of S-boxes, In *International Workshop on the Arithmetic of Finite Fields (WAIFI)*, Lecture Notes in Computer Science, Springer-Verlag, 16 pages, 2016.

[20] E. J. Marinissen, P. Cockburn, P. Hsieh, C. Huang, M. Konijnenburg, Y. Zorian, J. Delvaux, V. Rožić, B. Yang, D. Singelée, I. Verbauwhede, C. Mayor, and R. Van Rijsinge, IoT: Source of Test Challenges, In *International IEEE European Test Symposium (ETS)*, IEEE Computer Society, pp. 1-10, 2016.

[21] V. Rožić, B. Yang, W. Dehaene, and I. Verbauwhede, Iterating Von Neumann's Post-Processing under Hardware Constraints, In *9th IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, IEEE, pp. 37-42, 2016.

[22] S. Picek, D. Sisejkovic, D. Jakobovic, L. Batina, B. Yang, D. Šijačić, and N. Mentens, Extreme Pipelining Towards the Best Area-performance Trade-off in Hardware, In *Progress in Cryptology (AFRICACRYPT)*, Lecture Notes in Computer Science 9646, Springer-Verlag, pp. 147-166, 2016.

[23] R. De Clercq, R. De Keulenaer, B. Coppens, B. Yang, K. De Bosschere, B. De Sutter, P. Maene, B. Preneel, and I. Verbauwhede, SOFIA: Software and Control Flow Integrity Architecture, In *Design, Automation and Test in Europe (DATE)*, IEEE, pp. 1172-1177, 2016.

[24] E. Marin, D. Singelée, B. Yang, I. Verbauwhede, and B. Preneel, On the Feasibility of Cryptography for a Wireless Insulin Pump System, In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, ACM, pp. 113-120, 2016.

[25] V. Rožić, B. Yang, W. Dehaene, and I. Verbauwhede, Highly Efficient Entropy Extraction for True Random Number Generators on FPGAs, In *52nd Design Automation Conference (DAC)*, IEEE, pp. 116:1-116:6, 2015.

[26] R. Chaves, G. Di Natale, L. Batina, S. Bhasin, B. Ege, A. Fournaris, N. Mentens, S. Picek, F. Regazzoni, V. Rožić, N. Sklavos, and B. Yang, Challenges in Designing Trustworthy Cryptographic Co-Processors, In *IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, pp. 2009-2012, 2015.

Conferences and Workshops without Proceedings

[1] V. Rožić, B. Yang, N. Mentens, and I. Verbauwhede, Canary Numbers: Design for Light-weight Online Testability of True Random Number Generators, In *NIST RBG Workshop*, 2016.

[2] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, RECTANGLE, A Bit-slice Ultra-Lightweight Block Cipher Suitable for Multiple Platforms, In *NIST Lightweight Cryptography Workshop 2015*, 15 pages, 2015.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
COMPUTER SECURITY AND INDUSTRIAL CRYPTOGRAPHY
Kasteelpark Arenberg 10 box 2452
B-3001 Leuven
bohan.yang@esat.kuleuven.be
https://www.esat.kuleuven.be/cosic/ and http://byang.xyz