Technische Universität München
Lehrstuhl für Sicherheit in der Informationstechnik
an der Fakultät für Elektrotechnik und Informationstechnik

Highly Efficient Implementation of Physical Unclonable Functions on FPGAs

Stefan Gehrer

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines **Doktor-Ingenieurs (Dr.-Ing.)** genehmigten Dissertation.

Vorsitzender: Prof. Dr. Holger Boche
Prüfende der Dissertation:
1. Prof. Dr.-Ing. Georg Sigl
2. apl. Prof. Dr.-Ing. Walter Stechele

Die Dissertation wurde am 27. April 2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 14. August 2017 angenommen.

Abstract

Physical Unclonable Functions (PUFs) are an innovative way to use uncontrollable production tolerances of Integrated Circuits (ICs) for the generation of device intrinsic cryptographic keys. Unfortunately, the huge area consumption of many PUF implementations on Field-Programmable Gate Arrays (FPGAs) made them infeasible in real world environments. This work presents a new method for highly efficient usage of Ring Oscillator (RO) PUFs on FPGAs.

The stability of such PUF outputs is essential for the reliability of the cryptographic key. Any bit error in a cryptographic key makes it unusable. Therefore, an extensive stability analysis involving both reversible environmental changes and irreversible aging effects was carried out.

In the first part of this thesis, a basic implementation of RO PUFs on FPGAs is presented. The quality properties such as Hamming Weight (HW) or Hamming Distance (HD) promised a very good usability of RO PUFs on FPGAs. However, the generation of a 2136 bit PUF output consumed 80 % of the available slices on a Xilinx Zynq Z-7020.

In the second part, the complex system of logic and routing resources on FPGAs was analyzed for their usability as entropy sources for PUFs in combination with partial reconfiguration. Different implementations of ROs were reconfigured on the same area of the FPGA to maximize the resource usage. Each of them occupied the same logic block on an FPGA, but used different logic and routing resources inside the block. A partial bit vector response was generated by each implementation. All of them were joined to a larger response vector that could be used to generate a cryptographic key.

The implementation of this method showed that it is possible to decrease the required resources to generate a PUF response with a given length by almost 98 % on a Xilinx Zynq. To achieve this big area shrinkage, every of the eight Lookup Tables (LUTs) within a logic block, and every of the six input pins within a LUT was used for a unique RO implementation. The min-entropy of this reconfigurable PUF was found to be 93 % using different methods such as Context Tree Weighting (CTW), Principal Component Analysis (PCA), and the NIST SP 800-90B entropy test.

Robustness of the PUF against reversible environmental changes and irreversible aging effects is crucial for its reliability. Although the influence of environmental changes is studied in depth, there is still only limited research on aging effects. An accelerated aging test on 28 nm Xilinx Zynq FPGAs was performed for roughly one year in the third part of this thesis. The results were compared to the impact of reversible environmental fluctuations. A set of different designs with distinct types of logical stress was used to amplify the effects of various aging mechanisms. The impact of accelerated life conditions on the frequency of different RO PUFs was measured and analyzed. It was shown that the aging effect is dramatically accelerated with higher temperatures and voltages. The frequency of the ROs was lowered to around 98.6 % of the initial value, after an effective accelerated aging duration of around 60 years. The reliability of the PUFs without error correction was decreased by up to 6 %. Furthermore, the experiments showed that aging effects reduced the reliability in the same order of magnitude as environmental variations, namely voltage and temperature variations.

In the last part of this work, an exemplary application of the partial reconfiguration PUF on FPGAs was shown. The reconfigurable PUF was used with the reliability measurements from the previous chapter to choose an appropriate PUF-based key generation scheme from the literature. The PUFKY algorithm was chosen in this work. The amount of slices that were needed to implement the RO PUF part of the system could be lowered by at least 79.4 % when using the partial reconfiguration PUF. Furthermore, the usage of slices within the CLBs would be much more efficient when using the method presented in this work. A scheme was proposed which allows a secure boot of a PUF-based security module on an FPGA. The security module is able to load encrypted Intellectual Properties (IPs) from an unprotected non-volatile memory, as well as transmit new or modified IPs over-the-air while preserving the confidentiality and authenticity.

Acknowledgments

Foremost, I would like to express my sincere gratitude to Prof. Dr.-Ing. Georg Sigl for supervising this thesis and all members of the Chair of Security in Information Technology for the fruitful discussions. I also thank my second examiner Prof. Dr.-Ing. Walter Stechele.

Being part of the security team at Bosch was an awesome experience. I enjoyed every conversation about work related and unrelated topics, and they greatly contributed to the success of my thesis. A dedicated thank you to Sébastien Léger for being a great supervisor at Bosch, as well as Dr. Jürgen Schirmer for always supporting me as a group leader. Furthermore, I would like to thank Dr. Paul Duplys, Christopher Huth, Hervé Seudié, Dr. Hans Löhr, Robert Szerwinski, Dr. David Förster, and Dr. Jan Zibuschka for all the discussions and the good times in Möglingen and Renningen.

I would like to thank my mother for her endless support, as well as Leah and Jennifer for the dedicated proof-reading of my thesis.

Last but not least, I would like to thank all my friends around the world for always being there for me and making life so much more fun. It would be pretty boring without all y'all!

Contents

Abstract **iii**

Acknowledgments **v**

List of Abbreviations **xvii**

1 Introduction **1**

2 Background **5**

2.1 PUF Concepts 5
2.1.1 History 6
2.1.2 Weak and Strong PUFs 6
2.1.3 Quality Measures 7
2.2 PUF Types 13
2.2.1 Optical PUF 13
2.2.2 Silicon PUFs 15
2.3 PUF-based Key Generation 18
2.3.1 Select 18
2.3.2 Correct 19
2.3.3 Compress 21
2.4 FPGA 21
2.4.1 Overview 21
2.4.2 Structure 22
2.5 Cryptography 23
2.5.1 Symmetric 24
2.5.2 Asymmetric 25

3 Implementing PUFs on FPGAs 27
3.1 Introduction . . . 27
3.2 Theory . . . 28
3.3 Implementation . . . 30
3.4 Experimental Results . . . 36
3.4.1 Area . . . 37
3.4.2 Speed . . . 38
3.4.3 Quality Properties . . . 38
3.5 Conclusion . . . 44

4 Enhancing Efficiency Using PR 45
4.1 Introduction . . . 45
4.2 Prior Work . . . 47
4.3 Reconfigurable PUF . . . 48
4.3.1 Method 1: Using Different LUTs . . . 49
4.3.2 Method 2: Using Different Input Pins . . . 50
4.3.3 Combining Both Methods . . . 52
4.3.4 Analysis of Shared Resources . . . 54
4.4 Implementation . . . 54
4.5 Experimental Results . . . 56
4.5.1 Speed . . . 56
4.5.2 Frequencies . . . 57
4.5.3 Uniformity . . . 58
4.5.4 Bit Alias . . . 60
4.5.5 Uniqueness . . . 60
4.6 Entropy Estimation . . . 63
4.6.1 Context-Tree Weighting . . . 63
4.6.2 Principal Component Analysis . . . 64
4.6.3 NIST SP 800-90B Entropy Test . . . 68
4.7 Conclusion . . . 70

5 Reliability Analysis 71
5.1 Introduction . . . 71
5.2 Prior Work . . . 72
5.3 Aging . . . 73
5.3.1 Aging Mechanisms . . . 73
5.3.2 Stress Design . . . 76
5.3.3 Accelerated Environmental Conditions . . . 78
5.3.4 Implementation and Setup . . . 79
5.3.5 Experimental Results and Discussion . . . 82
5.4 Temperature and Voltage . . . 88
5.4.1 Theory . . . 88
5.4.2 Test Setup . . . 90
5.4.3 Experimental Results . . . 91

5.5 Conclusion 95

6 Key Generation System **97**
6.1 Introduction 97
6.2 SoC Platform 98
6.3 Error Correction & Key Generation 99
6.4 Security Module 101
6.4.1 Enrollment Phase 101
6.4.2 Security Module Boot Procedure 102
6.4.3 IP Loading Procedure 103
6.4.4 IP Storage Procedure 103
6.5 Discussion 105
6.6 Conclusion 106

7 Conclusion **107**

List of Figures

1.1 Schematic of basic PUF behavior 3

2.1 Organization of PUF response data 8
2.2 Uniformity of a PUF response vector 9
2.3 Uniqueness of a PUF response compared to other devices. . . 10
2.4 Reliability of PUF under different environmental conditions . 12
2.5 Bit alias of a single PUF bit over all measured devices 14
2.6 Schematic of a laser PUF 14
2.7 Schematic of a RO PUF 15
2.8 Schematic of an arbiter PUF 16
2.9 Schematic of an SRAM PUF 17
2.10 Generating cryptographic keys from noisy PUF responses . . 18
2.11 Schematic of a fuzzy extractor 20
2.12 Schematic of a modern SoC with PS part and a PL part . . . 22
2.13 Schematic of the implementation of a LUT on an FPGA . . . 23
2.14 Schematic of a symmetric key algorithm 24
2.15 Schematic of a public-key algorithm for encryption 25
2.16 Schematic of public-key algorithm for digital signatures . . . 25

3.1 Schematic and implementation of 3-inverter RO PUF on FPGA 29
3.2 Schematic and implementation of a TFF 30
3.3 Implementation of 3-inverter RO PUF on one slice 31
3.4 Schematic of eight RO PUFs 32
3.5 Biased pair of ROs 33
3.6 Area efficient implementation of 16 bit asynchronous counters 34
3.7 Heatmap of 4272 ROs on the Zynq 39
3.8 Bean plot of the HW on 10 different boards 40
3.9 Bean plot of the Intra-device HD 41

3.10 Histogram of frequency difference between RO pairs 42
3.11 Histograms of bit aliases measured over all boards 43

4.1 Functional principle of partial reconfiguration 46
4.2 Using reconfigurable PUFs for key generation 49
4.3 Two PUF implementations on same CLB using different LUTs 50
4.4 Using different pins of a LUT 51
4.5 Structure of a 3 bit LUT and different implementations of RO 52
4.6 Reconfiguration process on two SoCs 53
4.7 Possible shared resources between different RO implementations 55
4.8 Frequency distribution of all 48 RO designs on one board . . 58
4.9 HW of concatenated response vectors on each SoC 59
4.10 Fractional HW of response vector of each PUF 60
4.11 Bit alias of all concatenated 2136 bit partial PUF responses . 61
4.12 Fractional HD between each partial PUF response 62
4.13 Histogram of fractional HD between each partial PUF response 63
4.14 Colorplot of the principal component coefficients 65
4.15 Separated colorplot of the principal component coefficients . . 66
4.16 Kendall rank correlation coefficient between reconfig. PUFs . 67
4.17 Principal component scores for first six principal components 68

5.1 Bit flip due to the aging of two ROs 72
5.2 HCI effect in a CMOS transistor 74
5.3 NBTI effect in a PMOS transistor 74
5.4 Schematic of TDDB effect 75
5.5 Effect of EM in microelectronics 75
5.6 Differently stressed columns on SoC 76
5.7 DC 0/1 stress and low/high frequency stress designs on LUT 77
5.8 Stress cycle used for aging an FPGA 81
5.9 Accelerated aging test setup 82
5.10 Frequency degradation by different environmental stress . . . 83
5.11 Impact of amount of inverters on frequency degradation . . . 84
5.12 Impact of the electrical stress type on the frequency degradation 84
5.13 Heatmap of frequency degradation by different electrical stress 86
5.14 Impact of the aging process on the HW of the PUFs 86
5.15 Impact of aging process on reliability of the PUFs 87
5.16 Reliability when comparing differently stressed ROs 89
5.17 Absolute frequency change when changing the FPGA voltage 91
5.18 Relative frequency change when changing the FPGA voltage 92
5.19 Intra-device HD when changing the FPGA voltage 92
5.20 Absolute frequency change when changing ambient temperature 93
5.21 Relative frequency change when changing ambient temperature 94
5.22 Intra-device HD when changing the ambient temperature . . 95

6.1 SoC enrollment 102
6.2 SoC boot procedure 103
6.3 SoC boot procedure 104
6.4 IP storage procedure 105

List of Tables

2.1 Notation symbols used for the measurement data. 7

3.1 Resource utilization of a PUF design using 4272 RO PUFs . . 37
3.2 Inter-device HD between each PUF outputs 44

4.1 Configuration of 48 reconfigurable PUF implementations . . . 56

5.1 Effects of different stress types on the frequency of ROs. . . . 78

6.1 Comparison of implementations of key generation schemes . . 100

List of Abbreviations

AES	Advanced Encryption Standard
ASIC	Application-Specific Integrated Circuit
BBRAM	Battery Backed RAM
CA	Certificate Authority
CAN	Controller Area Network
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
CTW	Context Tree Weighting
DES	Data Encryption Standard
DFF	Data Flip-Flop
DSC	Differential Sequence Coding
ECC	Error Correcting Code
ECDH	Elliptic Curve Diffie–Hellman
ECU	Electrical Control Unit
EM	Electro-migration
EMI	Electromagnetic Interference
FF	Flip-Flop
FHD	Fractional Hamming Distance
FHW	Fractional Hamming Weight
FPGA	Field-Programmable Gate Array

GMC	Generalized Multiple Concatenated Code
HCI	Hot Carrier Injection
HD	Hamming Distance
HDL	Hardware Description Language
HKMG	High-κ Metal Gate
HMAC	Keyed-Hash Message Authentication Code
HW	Hamming Weight
IBS	Index-Based Syndrome
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
IP	Intellectual Property
JEDEC	Joint Electron Device Engineering Council
LUT	Lookup Table
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NMOS	N-channel MOSFET
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor
NBTI	Negative-Bias Temperature Instability
mux	Multiplexer
PBTI	Positive-Bias Temperature Instability
PCA	Principal Component Analysis
PCAP	Processor Configuration Access Port
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PL	Programmable Logic
PLD	Programmable Logic Device
PMOS	P-channel MOSFET

PR	Partial Reconfiguration
PRD	Partial Reconfiguration Design
PRNG	Pseudorandom Number Generator
PROM	Programmable Read-Only Memory
PS	Processor System
PUF	Physical Unclonable Function
RM	Reed-Muller-Code
RNG	Random-Number Generator
RO	Ring Oscillator
SCA	Side Channel Attack
SM	Security Module
SoC	System-on-Chip
SRAM	Static Random-Access Memory
TDDB	Time-Dependent Dielectric Breakdown
TFF	Toggle Flip-Flop
TI	Texas Instruments

CHAPTER 1

Introduction

Our modern society is highly characterized by an increasing number of connected electronic devices in areas such as industry 4.0, smart homes, smartphones, and connected cars. The amount of those devices, the transmitted data, and the complexity increases tremendously every year. We rely more and more on the flawless, safe, and secure operation of these devices. But the connectivity of these devices also opens up many possible gateways for adversaries and makes them an interesting target. Furthermore, they potentially carry a lot of sensitive information while being run in an untrusted environment. Therefore, Intellectual Property (IP) and data privacy protection are major requirements for many companies such as IP vendors and manufacturers.

Information security has four main concepts to protect the communication and storage of information:

1. **Confidentiality** ensures that information is not available to unauthorized entities. This security goal is mostly achieved by using encryption of the data, where the encryption and/or decryption is only made possible to authorized entities using cryptographic keys.

2. **Integrity** refers to the absence of any data corruption for the whole life-cycle. Data can be manipulated or corrupted both when stored locally or when using a communication channel. A checksum is mostly used to meet this security goal.

3. **Authenticity** is the assurance that the identity of a subject is the identity claimed or the communicated information is from the source it claims to be from. Authenticity highly relies on integrity. Any manipulation or corruption of authenticated information – and therefore

the tampering of its integrity – should be detected. The proof of authenticity can be made by something that is known (e.g. a password), something you have (e.g. a smartcard), or something you are (e.g. biometrics). Physical Unclonable Functions (PUFs) in particular make use of the latter, as intrinsic information is used. In secure communication, Message Authentication Codes (MACs) and electronic signatures based on Public Key Infrastructures (PKIs) are used to authenticate information and identities.

4. **Availability** refers to the ability to access information whenever it is needed. Systems that store and process information and communication channels have to function correctly at any point in time. Apart from random faults, common attack vectors are denial-of-service attacks.

To protect both carried and transmitted information, cryptography relies on a secret key that has to be brought into and stored inside the system. The key can be used in a symmetric encryption, i.e., both sides are using the same key, to ensure confidentiality of the data. Another possibility is to use it in an asymmetric way with public and private key pairs to ensure integrity, authenticity, and/or confidentiality of the data. This makes knowledge of the cryptographic key an interesting target for adversaries.

The secret key is mostly stored in a non-volatile memory, e.g., flash memory or fuses. Flash memory, however, has the downside of being hard to integrate in modern Complementary Metal-Oxide-Semiconductor (CMOS) processes [WH11, Ini08, Che06]. Fuses on the other side are potentially susceptible to readout by decapsulation and optical analyses. Furthermore, a key management infrastructure is needed, as the key has to be brought into the device. If global master keys are used, losing it on one device would have an impact on every device. Storing a device specific key requires a key management, i.e., they have to be individualized, stored on the device during an enrollment phase, and potentially stored centralized as well. This makes the whole system less flexible and harder to maintain.

PUFs are using a new approach to create a device unique fingerprint or cryptographic key for authenticity, integrity, and confidentiality [PRTG02, GCvDD02, SD07]. The sources of randomness are the unpredictable variations during the manufacturing of Integrated Circuits (ICs). These functions are easy to evaluate, but hard to predict. It should be practically impossible to duplicate them. There is a wide range of types of PUFs, e.g., electronic PUFs, silicon PUFs, and surface PUFs. The idea was introduced by Pappu et al. [PRTG02] as the physical one-way function using an optical system. Nowadays, they are primarily used on ICs. The two mainly used types of PUFs are the delay-based and the memory-based PUF. The latter uses the unpredictable settle state of uninitialized digital memories like Static Random-Access Memorys (SRAMs) [GKST07, HBF09], flip-flops [MTV08],

or latches [SHO08]. Delay-based PUFs rely on the delay variations of routing and logic resources directly. One method to do this is with the arbiter PUF, which introduces a race condition between two or more paths [LLG+04]. Frequency differences between two or more Ring Oscillators (ROs) are used by the RO PUF [GCvDD02, MS11].

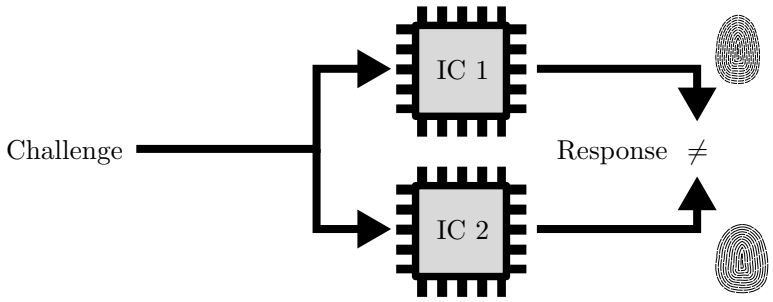


Figure 1.1: Schematic of basic PUF behavior.

Figure 1.1 shows the basic concept of a PUF. The same function is implemented on two different ICs. When those functions are challenged with the same input, each entity will produce a unique response output. It can be seen as a digital fingerprint and be used to authenticate devices or to generate unique cryptographic keys. Device aging and environmental influences such as voltage and temperature might have an influence on the PUF responses. The cryptographic keys have to withstand those variations and remain stable over the lifetime of the device.

The main advantage of silicon PUFs over embedded flash or fuses is their ability to be implemented using the same production process as the main circuit, which possibly saves cost for the device. Furthermore, when using reprogrammable hardware such as Field-Programmable Gate Arrays (FPGAs), the existing structures can be reused as PUFs. FPGAs are ICs that contain many programmable logic blocks, a complex routing system, and other dedicated logic elements. This gives them the ability to configure their hardware after manufacturing. The exact hardware implementation is not fixed as with Application-Specific Integrated Circuits (ASICs), but flexible. This makes them able to be updated in-field, which is important with the modern and fast developing technology. They are an interesting alternative in many modern applications, especially where the quantities are not high enough to justify an ASIC or where the flexibility and updatability is needed. One way to implement PUFs on FPGAs is explained and analyzed in chapter 3.

Most modern FPGAs already contain security features. Unfortunately, most of them are already broken by Side Channel Attacks (SCAs) [MKP11, MBKP11, MKP12, SW12, MOPS13]. SCAs are modern attacks that use information gained by observing the behavior of the physical implementation

of a cryptosystem. This ranges from timing, power consumption, sound, or electromagnetic emanation. Many countermeasures against SCAs are covered by patents. PUFs on the other hand offer a new possibility to implement own cryptographic functions that rely on intrinsic keys.

Implementing PUFs on FPGAs is very costly in terms of resources, as later shown in this thesis. Because the hardware has to be flexible, many resources are wasted by exclusively using them for the implementation of PUFs. Therefore, they are not affordable in most real world scenarios. In chapter 4, a new way to use PUFs more area efficiently on FPGAs is introduced. Instead of using just one PUF on a logic block of an FPGA, multiple PUFs are loaded on the same logic block in this work. By using modern FPGAs with Partial Reconfiguration (PR), the area can be reused. This way, more parts of the complex logic and routing system are used as entropy sources. Due to the reuse of resources, correlation between the PUFs might occur and lower the entropy of the PUF output. Therefore, analysis of the entropy is a main focus in that chapter.

Stability of the generated key is crucial for the usability of PUFs. This stands in contrast to challenge-response systems, where single bit flips are not fatal, as they can just be ignored. Both reversible and irreversible variations can alter the behavior of a PUF. Environmental conditions such as voltage and temperature are reversible variations. The effect of those altered conditions usually disappear once the cause is withdrawn. Device aging on the other hand is an irreversible variation that leads to a permanently changed behavior. The effects of aging on RO-based PUFs on an FPGA are analyzed in chapter 5. Voltage, temperature, and accelerated aging experiments on an FPGA using a climate chamber were done for almost one year. The results are then compared to the effects of reversible environmental conditions.

The generated keys can be used on modern SoCs in many different ways. To give an outlook about the possibilities of PUFs in securing a device and communication, a PUF-based key generation scheme and a security module using the FPGA and processor part of an SoC is presented in chapter 6. The key is generated solely on the FPGA and stays inside the FPGA. This makes it obsolete to get secret keys in or out of the FPGA. A public key can be used to communicate with the outside, whereas the private key is kept inside the device. The system is presented in detail and an estimation of resource usage on FPGAs is given.

CHAPTER 2

Background

This chapter provides background information about the main subjects of this thesis: FPGAs, PUFs, and cryptography.

In the first section 2.1, the basic PUF concepts, a brief overview on the history and the quality measures of PUFs are explained. A selection of important PUF types is presented in section 2.2. The methods of generating cryptographic keys by using PUFs are explained in section 2.3. A short introduction on FPGAs is given in section 2.4, as the PUFs will be implemented on that platform in this thesis. The basic concepts of cryptographic algorithms will be explained in section 2.5.

2.1 PUF Concepts

The definitions of PUFs are not consistent in literature. To have a common understanding for this thesis, the definitions will be explained in this section. A PUF in general is a function that uses production tolerances to generate a device specific response. To use the response in a digital system and make it comparable, the response is usually in a binary form after post processing. The basic principle can be explained with the three words:

Physical A physical entity embodied in a physical structure.

Unclonable The exact implementation cannot be duplicated. However, it might still be possible to model the PUF using, e.g., machine learning.

Function In a mathematical sense, an input is mapped to an output. This is not true for PUFs, as they are usually noisy. Error correction has to

be used to replicate the original output. Furthermore most PUFs ('weak PUFs') are used with a fixed input to, e.g., generate a fixed cryptographic key. Maes et. al propose the term *probabilistic function* as more correct, as an input is mapped with a certain probability to an output value, due to noise.

2.1.1 History

The basic idea of a PUF, without using that exact term, was already introduced by Bauder in 1983 [Bau83] and Simmons in 1991 [Sim91]. Simmons proposed an optical PUF that used reflection patterns to identify strategic arms in arms control treaties. Bauder used a paper PUF for anti-counterfeiting of currencies. Posch et al. used an active coating to protect devices [Pos98]. In 2001, Pappu proposed an optical PUF construction and defined the general concept of PUFs [PRTG02]. However, he used the term physical one-way function. One year later Gassend et al. introduced the silicon PUF [GCvDD02] and used the term physical unclonable function for the first time. This started the research on many different types of PUFs, with a large focus on silicon PUFs. Older concepts were transformed to the new definition of PUFs, new concepts developed and combined.

Companies specialized on PUFs were established shortly after, such as Verayo in 2005 or Intrinsic ID in 2008.

2.1.2 Weak and Strong PUFs

PUFs can be classified by their challenge-response behavior. As explained in the introduction, a PUF usually takes a challenge and answers with a response. A PUF is called a strong PUF if an attacker is unable to guess any response to an unused challenge during the lifetime of a device. This requires a very large challenge-response set to make it impossible for an attacker to just learn all of the pairs. Furthermore, it must be impossible for an attacker to build an accurate model of the PUF behavior. Many strong PUF designs fail the latter requirement, as in those cases, machine learning algorithms are able to model the PUF behavior.

A weak PUF is the term for any PUF that does not meet the requirements of a strong PUF. This can either mean that the PUF constructions were not appropriately designed or it was intentionally designed that way. Generating cryptographic keys is an example for the latter. This extreme case of just one challenge-response pair is not meant to communicate with another entity via challenge-response pairs. A fixed challenge is used to generate a device specific cryptographic key.

The sole purpose of a PUF in this thesis is the generation of a cryptographic key. Hence, only weak PUF constructions will be used and the challenge-response constructions will not be used. This has the advantage

Table 2.1: Notation symbols used for the measurement data.

Symbol	Description
p	Index of a PUF bit within response vector. $1 \leq p \leq P$
P	Length of PUF response vector
d	Index of a device. $1 \leq d \leq D$
D	Number of devices
m	Index of a measurement at a specific time and environmental condition. $1 \leq m \leq M$
M	Number of measurements
$r_{p,d,m}$	Binary response bit p of device d within measurement m
$R_{d,m}$	Response vector of device d at measurement m with $R_{d,m} = \{0,1\}^P$

that any attacks on the strong PUF constructions do not apply to the systems used in this work.

2.1.3 Quality Measures

All PUF constructions share a number of specific properties. The quality of a PUF can be measured in different ways. Distinct sets of properties such as randomness, steadiness, correctness, diffuseness, uniqueness, uniformity, or reliability are used in sometimes different definitions throughout the research on PUFs [HYKS10, SHO08, MCMS10, MGS13]. As there is no standard model that all researchers agree on, the terminology used in this work will be explained in this section.

Organization of PUF response data

The responses of a PUF can be characterized by many different variables. Figure 2.1 shows the dimensions used in this work. The response is assumed to be in a binary form, i.e., the post-processing of analog data will not be covered in this section. Additionally, the notation symbols used in this work are summarized in Table 2.1.

When using a weak PUF for cryptographic key generation, a response vector R with a certain bit length P is needed. Therefore, P PUF structures, each returning a single bit p, have to be implemented. This is represented by the vertical axis. Each of those PUFs is situated on a different location of the device in case of the weak PUF. In a strong PUF scenario it could also be a set of challenge-response pairs.

The index of a device d, with $1 \leq d \leq D$, that the PUF will be used on is represented by the d-axis. Each device should generate a unique cryptographic key, i.e., the PUF response should be unique for every device. The

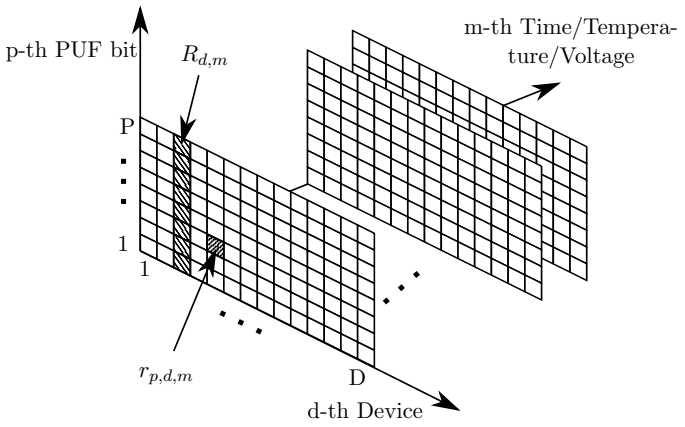


Figure 2.1: Organization of PUF response data. The three dimensions are: 1) PUF bit position within a PUF response vector, 2) the device on which the PUF is used, and 3) the date of the measurement and the corresponding environmental conditions such as voltage and temperature.

responses are measured for D different devices. Each device should have a unique P-bit wide PUF response.

The stability of a PUF response is very important. When they are used to generate cryptographic keys, a single bit flip could corrupt the whole key. To measure the stability of a PUF response, it will be measured multiple times and be compared to the initial measurement. These measurements are represented by the third m-axis. As the device will be used under real-world conditions, a variation of environmental conditions such as temperature and voltage, as well as device aging will be expected. The m-axis represents a set of M different time, temperature, and voltage conditions that the device will be tested with.

The PUF responses have a vector length of P bit. The response of a PUF system with P PUF bits on device d at condition m is defined as:

$$R_{d,m} = r_{1,d,m} \| r_{2,d,m} \| ... \| r_{P,d,m} \tag{2.1}$$

Uniformity

Each PUF system on a device returns a P bit wide vector. Uniformity defines the proportion of ones and zeros in this vector. An ideal, unbiased, statistically random system returns the same amount of ones and zeros. To measure this, the Hamming Weight (HW) is used. To calculate the HW of the response on device d at condition m, all ones of the P bit wide vector

are added up:

$$HW(R_{d,m}) = \sum_{p=1}^{P} (r_{p,d,m}) \tag{2.2}$$

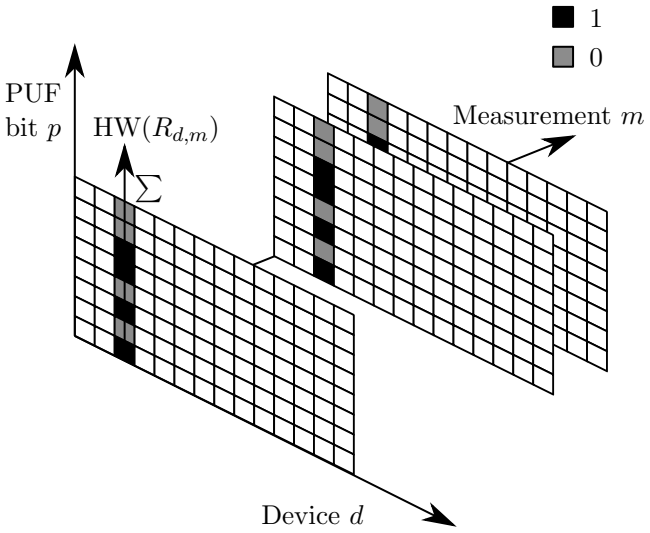


Figure 2.2: Uniformity of a PUF response vector. The distribution of zeros and ones show a bias towards one of those values. The HW is used to measure the uniformity by adding up each single PUF bit within a complete response.

Figure 2.2 shows an unbiased PUF response that produces the same amount of zeros and ones. This equals a HW of $P/2$ for a P bit wide vector. To be independent from the bit width of the response, the HW can be divided by the number of bits to get the Fractional Hamming Weight (FHW):

$$FHW(R_{d,m}) = \frac{HW(R_{d,m})}{P} \tag{2.3}$$

The ideal distribution is a FHW of $\frac{P}{2}/P = 0.5$. Every PUF response vector on any device, at any time, voltage, and/or temperature should always be close to this ideal value to guarantee a uniform distribution of ones

and zeros. This ensures that no fundamental design flaw was made when constructing the PUFs.

Uniqueness

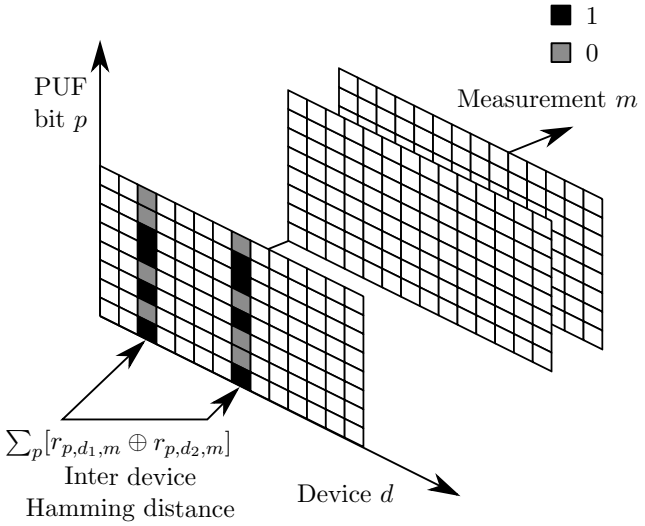


Figure 2.3: Uniqueness of a PUF response compared to other devices. The HD is used to measure the inter-device differences in the responses by calculating the xor of both vectors. Ideally, the two vectors differ in exactly half of their PUF bits.

As the PUFs are used to generate a device unique key, the statistical independence from PUFs on one device to the same PUF on another device has to be tested. This also makes it possible to distinguish chips of the same type from each other, just by using production tolerances. To compare two bit vectors with each other, the Hamming Distance (HD) will be used. The HD is the number of positions in which two PUF responses of the same implementation on two distinct devices, d_1 and d_2, are different:

$$HD(r_{d_1,m}, r_{d_2,m}) = \sum_{p=1}^{P} [r_{p,d_1,m} \oplus r_{p,d_2,m}] \tag{2.4}$$

$$= \mathrm{HW}[R_{d_1,m} \oplus R_{d_2,m}] \tag{2.5}$$

Ideally the two vectors should differ in half of their bits to assume uncorrelated responses, i.e., the HD should be $m/2$. To be independent from

the bit length we divide the HD by the bit length P of the response to get the Fractional Hamming Distance (FHD):

$$FHD(R_{d_1,m}, R_{d_2,m}) = \frac{HD(R_{d_1,m}, R_{d_2,m})}{P} \tag{2.6}$$

The FHD should be close to the ideal value of $\frac{P}{2}/P = 0.5$.

As the results of different devices are compared with each other, this is commonly referred to as inter-device HD.

The uniqueness property takes all possible $\binom{D}{2} = \frac{D(D-1)}{2}$ combinations of inter-device HDs into consideration. It can be calculated as the normalized sum of all possible inter-device HD combinations:

$$U = \frac{2}{D(D-1)} \frac{1}{P} \sum_{d_1=1}^{D-1} \sum_{d_2=d_1+1}^{D} \sum_{p=1}^{P} (r_{p,d_1,m} \oplus r_{p,d_2,m}) \tag{2.7}$$

However, the inter-device HD can only be seen as a necessary condition to assume unique and uncorrelated results. Throughout this thesis, more approaches to rate the uniqueness of a PUF response will be presented, such as correlation analyses and compression algorithms.

Reliability

The stability of a PUF response throughout its lifetime and under all specified environmental conditions of the device is very important. Every bit flip of the response could corrupt a cryptographic key, the challenge response pair, or make the error correction harder. To estimate the stability of PUF responses, the HD is used again as shown in Figure 2.4.

The cryptographic key or the challenge-response pairs of a PUF will be defined under the initial characterization condition m_1. Each subsequent measurement of the PUF will be done under a different condition, due to aging, temperature, or voltage variations. It should however, still result in the same output over the lifetime of a device. As the responses of the same device are compared with each other, the HD for reliability is commonly referred to as intra-device HD. The HD between an initial measurement at condition m_1 to a subsequent measurement at m_2 can be calculated to:

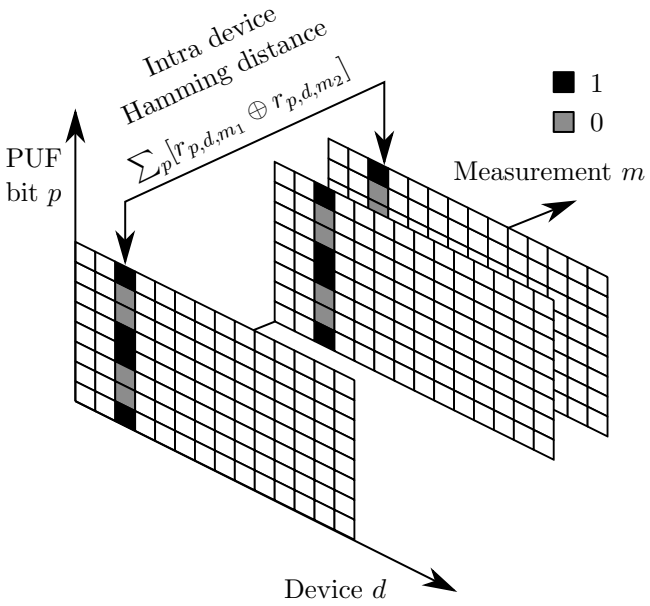


Figure 2.4: The reliability of a PUF under different environmental conditions and age of the device. The intra-device HD is used to measure the occurrence of bit-flips when repeatedly measuring the same PUF on the same device and comparing it to the initial measurement.

$$HD(R_{d,m_1}, R_{d,m_2}) = \sum_{p=1}^{P} r_{p,d,m_1} \oplus r_{p,d,m_2} \tag{2.8}$$

Accordingly the intra-device FHD can be calculated to:

$$FHD(R_{d,m_1}, R_{d,m_2}) = \frac{HD(R_{d,m_1}, R_{d,m_2})}{P} \tag{2.9}$$

As the goal is the absence of any bit-flips within the responses on the same device, the ideal intra-device HD is zero.

Bit alias

While uniformity refers to the bit distribution within the response on one device, the bit alias refers to the distribution of ones and zeros of the same single PUF output bit on different devices. This can be used to identify a badly implemented RO pair within the whole response. A bad bit alias automatically leads to a bad uniqueness as those PUF bits will not differ when measuring the inter-device HD. The bit alias of a PUF bit measured on D devices can be calculated to:

$$HW(R_{p,m}) = \sum_{d=1}^{D} [r_{p,d,m}] \tag{2.10}$$

Accordingly, the FHW can be calculated to:

$$FHW(R_{p,m}) = \frac{HW(R_{p,m})}{D} \tag{2.11}$$

The ideal bit alias is an equal distribution between ones and zeros, which refers to a FHW of 0.5.

2.2 PUF Types

The idea of PUFs is to use intrinsic physical properties to authenticate devices, objects, or systems. This concept is not new, as it has been already widely used with humans, by using, e.g., the fingerprint. A human fingerprint is an intrinsic feature that is given at birth and is (almost) unique for every person. It is easy to measure a fingerprint, but hard to permanently clone a fingerprint to another human being. This basic idea has been transferred to a wide range of non-electrical, electrical, and silicon devices. They share the requirement that they are easy to make, but hard to duplicate or control. A selection of PUF types will be presented in this section.

2.2.1 Optical PUF

PUFs were first mentioned by Pappu [PRTG02] in 2002 as a physical one-way function (POWF). Figure 2.6 shows the construction of such a function. A helium–neon laser is used to radiate a token, in this case a glass sphere.

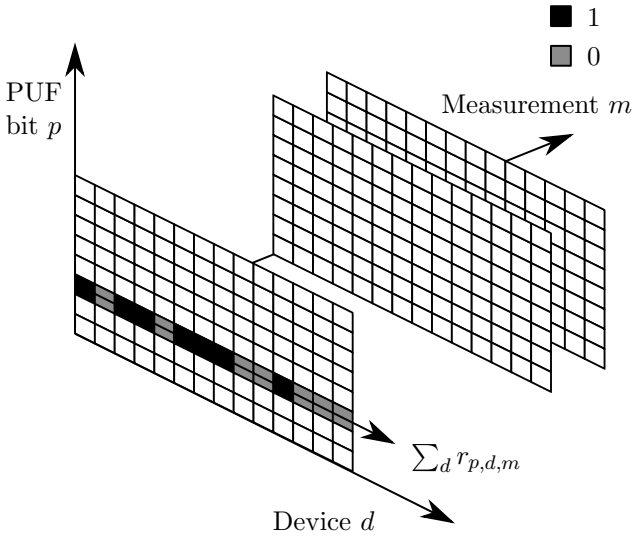


Figure 2.5: Bit alias of a single PUF bit over all measured devices. The bit alias is measured using the HW, i.e., the sum of a single PUF bit over all devices.

The resulting speckle patterns are recorded two-dimensionally. A Gabor transform was used to generate a 2400-bit key.

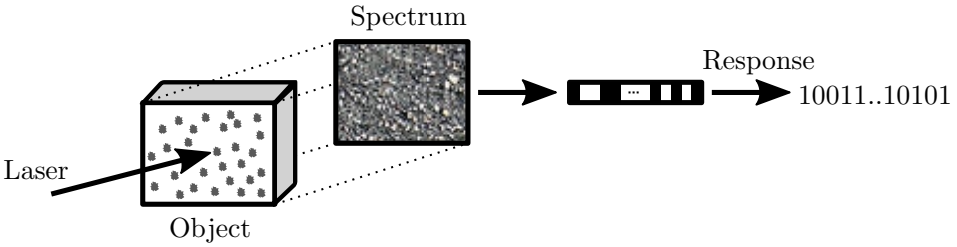


Figure 2.6: Schematic of a laser PUF. The laser hits an object and the resulting spectrum is recorded and converted to a binary response.

The exact orientation of the laser was used to generate different challenges for the function. The challenge orientations and the resulting Gabor hashes were recorded in a database during an enrollment phase. Whenever the device was needed to be verified, the challenge orientation was provided and the response requested. The response was then compared with the one stored in the database to authenticate the device.

Another example for optical PUFs are paper PUFs [CMK05, BSQ10, Kir04]. A variety of possibilities to use paper PUFs exist. An easy way is by scanning the package of a product after manufacturing. Later, the authenticity of the product can be verified by scanning the surface again and comparing it with the initially stored value. This can be interesting for

a low cost counterfeit protection for products that are counterfeited on a regular basis such as pharmaceutical products.

2.2.2 Silicon PUFs

The main focus of research and usage of PUFs is aimed towards silicon PUFs as an intrinsic PUF on ICs. The source of randomness are the variations of digital delays due to production tolerances. The two mainly used types of intrinsic PUFs are the delay-based and the memory-based PUFs. The latter use the unpredictable settle state of uninitialized digital memories like SRAMs [GKST07, HBF09], flip-flops [MTV08], or latches [SHO08]. Delay-based PUFs rely on the delay variations of routing and logic resources directly. One method to do this is the arbiter PUF, which introduces a race condition between two or more paths [LLG+04]. Frequency differences between two or more ring oscillators are used by the ROPUF [GCvDD02, MS11].

Three mainly used PUFs will be presented in detail: the ROPUF, arbiter PUF, and the SRAM PUF.

RO PUFs

ROs were among the first silicon-based PUFs that were proposed to extract entropy from ICs. As shown in Figure 2.7, an odd number of inverters are connected in a ring. This structure will begin to oscillate with a certain frequency f_x if the delay is sufficiently large. The exact frequency of this ring depends on production tolerances of the logic gates and the delay lines. The exact same implementation on two different locations of the chip or on the same location on two different chips will result in slightly distinct frequencies.

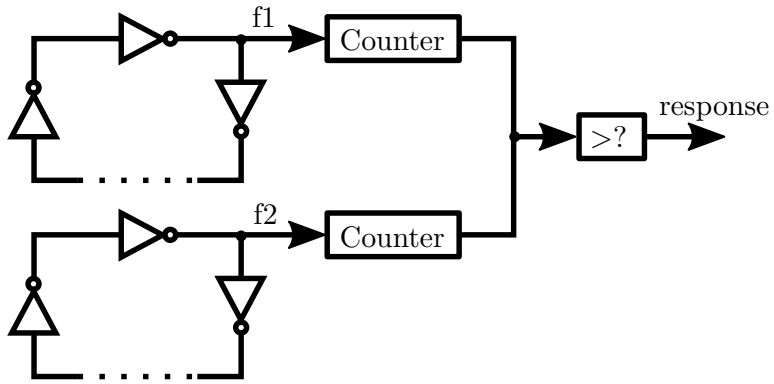


Figure 2.7: Schematic of a RO PUF. The frequency of two ROs is counted and compared with each other to generate one output bit.

As shown in Figure 2.7, this fact can be used to generate bits. Two identical implementations of ROs will oscillate with the frequencies f_1 and f_2 [SD07]. These frequencies will be slightly different due to the production tolerances. Two counters are started and ended at the same time to count the amount of oscillations of f_1 and f_2. The results will be compared and a bit generated according to equation 2.12.

$$r(f_i, f_{i+1}) = \begin{cases} 1, & \text{if } f_i > f_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

Using this technique, k-ROs will produce a $\frac{k}{2}$ bit long response. Other techniques allow the creation of more bits, but they would be correlated, i.e., statistically dependent. By comparing every RO with each other, a total of $\frac{k}{2} = \frac{k \cdot (k-1)}{2}$ combinations is possible. This however produces correlated results. A very fast or very slow RO will be dominant and effectively produce a lot of 1 or 0 bits, that are quickly predictable. If used in the conventional way of using each RO just for one bit generation, then ROs achieve one of the best statistical behaviors [KKR+12].

Arbiter PUF

Arbiter PUFs are delay-based silicon PUFs. They were first introduced by Gassend et. al [GLC+04]. Instead of using frequencies, a race condition is introduced as a measure. Figure 2.8 shows the working principle of an arbiter PUF. The signal starting from the left is racing on two different paths. The Multiplexers (muxes) at the interconnects can either cross the signal paths or leave them straight. They are used to challenge the PUFs. The arbiter circuit at the end of the signal path decides which path was faster, i.e., which signal arrived at first, and produces an according binary response.

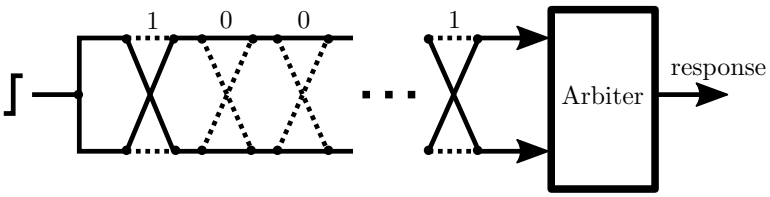


Figure 2.8: Schematic of an arbiter PUF. A signal is racing on two different paths that can be altered by using muxes. An arbiter circuit decides which one of the paths was faster and generates a response bit accordingly.

It is crucial that all paths are designed in perfect symmetry. Unequal length of paths will bias the PUF and lower the entropy dramatically, as the production tolerances are not the main factor of path differences. This makes

their implementation on FPGAs very hard. The exact internal structures of FPGAs are unknown. Therefore, ensuring an absolute identical symmetric implementation is difficult. Another problem occurs when the difference in delay between the two paths, caused by the random silicon process, is very small. This leads either to metastability when both signals arrive at the same time, or an unstable behavior.

When using the arbiter PUF as a strong PUF it is very prone to modeling attacks. The 2^n possible challenges for n muxes have a linear correlation. By using same paths for different challenges, the entropy is lowered. Dominant paths, either very slow or very fast, highly influence the result of multiple challenges. By attacking the challenge-response pairs with machine learning, the response to a given challenge can be calculated without knowing the exact PUF structure. This violates the rule that a PUF should be unclonable.

SRAM PUF

An SRAM is a silicon memory that uses a bistable circuit to store bits. The CMOS implementation is typically built with six transistors. As shown in Figure 2.9, two cross-coupled inverters form the memory cell. Two additional transistors are used for write and read operations. It is considered a volatile memory, as it loses its information after a power down.

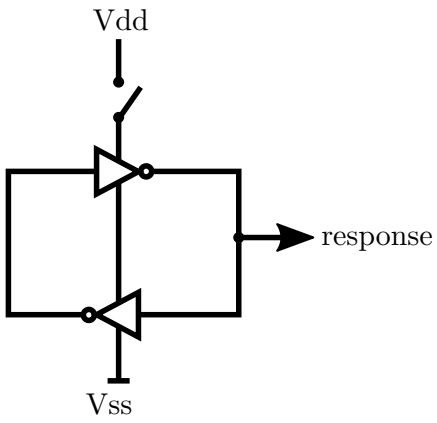


Figure 2.9: Schematic of an SRAM PUF. The cross coupled inverters act as a memory cell and can store information. In case of a PUF, the initial state of the SRAM is used as a response.

The usage of an SRAM as a PUF was first proposed by Guajardo et al. [GKST07]. The bistable settle state of the SRAM is used as a PUF function. Upon enabling the supply voltage, the SRAM will be in a metastable state. As the paths are designed symmetrically, only production tolerances will define the 'stronger' path. This will determine the outcome bit of the

SRAM, i.e., the settle state. As the settle state is unique for every cell, it can be used as a PUF, as long as the SRAM cells are not pre-loaded with an initial value. Unfortunately, the SRAM cells of most FPGAs are preloaded by the hardware and therefore not usable as a PUF. Some research [SGB+10, WG14] on using this method, especially on older devices, exist but will not be covered in this thesis.

2.3 PUF-based Key Generation

Traditional secure key generation had two major premises: a) a good source of randomness to generate unique secure keys and b) a secure non-volatile memory to store the keys in a way that they cannot be read-out by an unauthorized third party. Especially the demand for secure storage becomes increasingly problematic. Embedded flash will be harder and especially costlier to implement in current and future feature sizes. PUFs are a way to fulfill both requirements at once. By using the device unique randomness, a unique key can be generated, and it does not have to be stored, as it can be reliably reproduced during its lifetime.

The PUF output bits cannot be directly used as a cryptographic key. Due to instability of the read-outs, bit flips will occur. They can be additionally enhanced by reversible and irreversible variations. A post-processing is necessary to generate stable, high-entropy cryptographic keys under different environmental conditions and during the complete lifetime of a device. Helper data algorithms are used to generate reliable keys. As shown in Figure 2.10, this process can be separated in three steps: select, correct, and compress [DGSV14].

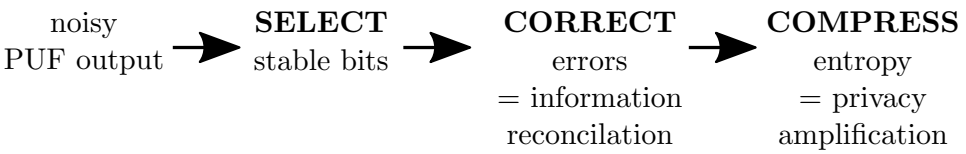


Figure 2.10: The basic steps of generating cryptographic keys from noisy PUF responses.

2.3.1 Select

In this first step only the most reliable bits can be selected. This makes the overall PUF response more stable and robust to environmental changes. When the frequency difference between two ROs is not high enough, a bit flip is very likely. Such PUF bits can be ignored in this first step. The information about the used and unused PUF bits is stored in helper data.

2.3.2 Correct

The second step is error-correction. As the PUF response bits are unstable, this step tries to correct all possible deviations from the initially measured PUF output. Therefore, helper data is generated during a generation step to restore the initial PUF output. This should always lead to a stable error corrected PUF output, that can be used to generate a reliable cryptographic key. Commonly used error-correction schemes are secure sketches [DRS04, DGSV14] like syndrome coding or code-offset. A secure sketch consists of two procedures: Sketch and Recover. Sketching takes a PUF input r and returns a bit vector w. Recover on the other hand takes a noisy PUF output r' and the bit vector w and returns a r'', where the goal is that $r = r''$. The latter is called the correctness property. A second property of secure sketches is the security property. This ensures that if r contains a min-entropy of m, that an attacker cannot recover r with a probability grater than 2^{-m}.

Two examples of secure sketches are the code-offset construction and the syndrome construction. The sketching procedure of the latter calculates a helper data string w as a syndrome to $w := r \cdot H^T$. In the recover procedure a syndrome s is calculated from the noisy PUF output to: $s := r' \cdot H^T \oplus w$. The syndrome decoder is then able to find an error word e such that $s = e \cdot H^T$, with the HW of e being within the correctable error margin. Using the error, the original PUF output can be recovered to $r := r' \oplus e$. Common choices for syndrome constructions are BCH codes [MVHV12], repetition codes [MTV09b], and Reed-Muller codes [MTV09b].

In the code-offset construction [DRS04, MTV09b, BGS+08, MTV09a, vdLSHT10] the sketching procedure XORs a random codeword c with the PUF response r to get $w := r \oplus c$. The recover procedure first calculates a noisy codeword from the noisy PUF output: $c' := r' \oplus w$ and then uses an Error Correcting Code (ECC) to recover the original codeword c.

A fuzzy extractor can be seen as a generalization of a secure sketch and a strong extractor. As PUF outputs are usually not completely uniformly random, a certain min-entropy is guaranteed by a strong extractor. A fuzzy extractor extracts a stable output from a fuzzy source in a generation phase and reproduction phase. Figure 2.11 shows the schematic of a basic fuzzy extractor. At first the generation step is executed. A secure sketch is used to generate a helper data string from the initial PUF response. Accordingly, the response is used in a secure extractor with a random salt to generate the original key R. The random number is stored together with the helper data string as the helper data. The PUF system is now ready to be shipped.

Whenever the PUF system is used in the field, the goal is to regenerate the same key R. Therefore, the noisy PUF measurement r' is used together with the helper data string to regenerate the original response r. Then the regenerated original response is used together with the same random number salt in the extractor again to recover the key R'.

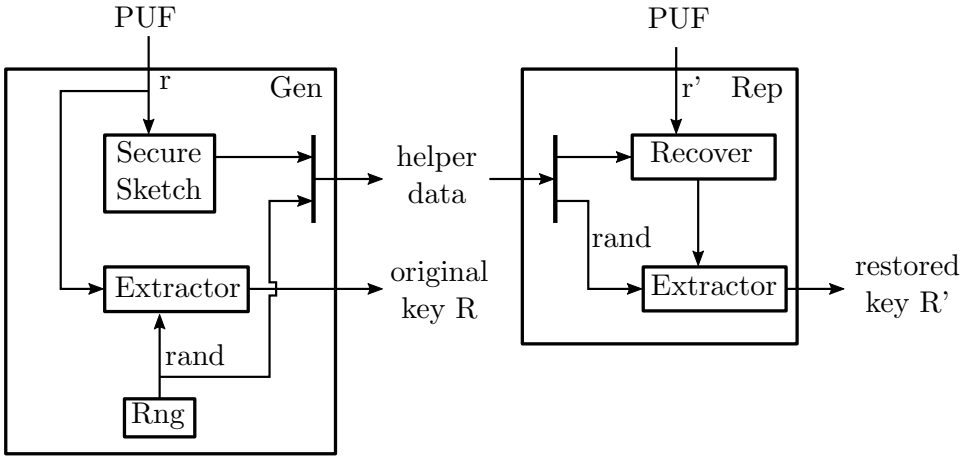


Figure 2.11: Schematic of a fuzzy extractor used for extracting high-entropy and stable keys from a noisy bit source. The generation phase on the left side is performed once to generate the corresponding helper data for the device and PUF. The helper data is then used together with the noisy PUF response to generate a stable key R in-field.

Traditionally, a fixed bit error probability is assumed for many silicon-based PUF responses. This approach however is very pessimistic as many PUF response bits are very robust. A soft-decision [MTV09a, MTV09b] secure sketch takes this into account. In these sketches every PUF response bit has its own error probability and takes an individual likelihood into account. The error correcting capabilities can be improved in comparison to hard-decision sketches. The main advantage is that less PUF bits are needed for the same failure rate and entropy level. On the other hand the decoding effort raises dramatically. When using RO PUFs, the frequency difference between two compared ROs can be used as an estimate for its bit error probability. A commonly used algorithm is a soft-decision maximum-likelihood decoding (SDML), as it offers a good performance, or generalized multiple concatenated (GMC) codes that use, e.g., recursively split Reed-Muller codes.

On the other hand, security criticism was recently raised by Delvaux et al. [DGSV14], that the leakage of soft-decision coding is underestimated. Powerful divide-and-conquer manipulation attacks are presented on some of the used soft-decision coding schemes.

Other presented key-generation algorithms are Index-Based Syndrome (IBS) [YD10] coding, Complementary-IBS coding [HMSS12, HDSMS12], or Differential Sequence Coding (DSC) [HWRL$^+$13, HS14, HYS16]. IBS is a variant of the 1-out-of-n selection, where an index to the most reliable bit matching a codeword is stored. The index is used during reconstruction to select the bits again and reconstruct the codeword. The C-IBS is an

extension of IBS, where not only the index to a PUF bit is stored as helper data, but also to complementary bits. This further increases the stability of the codeword. An ECCs is used to error correct the codeword and reproduce the key.

DSC is similiar to C-IBS, but in contrast to C-IBS, it has a fixed reliability and variable block size. A sequence of PUF outputs is searched for those outputs with an error probability under a predefined threshold. Unlike C-IBS, the block size is not fixed, but the end of one iteration provides the starting point for the next iteration of finding reliable bits.

2.3.3 Compress

In this last step the entropy of a PUF response is compressed, as it is mostly not equal to its bit length. This can be both because of correlations between the PUF bits and possible leakage of the helper data. This step is also commonly referred to as privacy amplification. Most implementations use a lightweight hash algorithm like SPONGENT [MVHV12, HWRL+13] or the Toeplitz hash [BGS+08].

2.4 FPGAs

FPGAs are ICs that are designed to be configured after manufacturing. The basic idea is to have a specific Hardware Description Language (HDL) that can be used to configure the circuit in-field. Programmable logic blocks on the FPGA can be configured as complex logic functions and memory elements can be used to store information. A complex routing system can be configured to route the signals throughout the FPGA. This section contains background information on the structures of modern FPGAs.

2.4.1 Overview

FPGAs were developed from previously used Programmable Logic Devices (PLDs) and Programmable Read-Only Memorys (PROMs). In comparison to PLDs the logic blocks are much more complex. They contain memory elements such as Flip-Flops (FFs), and Lookup Tables (LUTs). Furthermore, the routing possibilities to connect different logic blocks with each other are enhanced. The leading companies pushing FPGAs were the 1983 founded Altera, and two years later in 1985 founded Xilinx. Both companies rely on SRAMs as the main building block of the LUTs. SRAMs are relatively cheap to manufacture, but require an external non-volatile memory to configure the logic blocks at start-up. Microsemi, 1985 founded as Actel, on the other hand relies on flash-based technology. The main advantage of flash-based FPGAs is that it is non-volatile and very robust against single-event upsets.

A recent trend is the combination of FPGAs with microcontrollers or microprocessors to a System-on-Chip (SoC). This allows a combination of flexible software solutions running on the processor and hardware accelerated implemented on the FPGA. Among the most popular devices are the Xilinx Zynq, Altera Cyclone, and MicroSemi's SmartFusion. In this thesis a Xilinx Zynq will be used, but the concepts presented are also transferable to other devices.

2.4.2 Structure

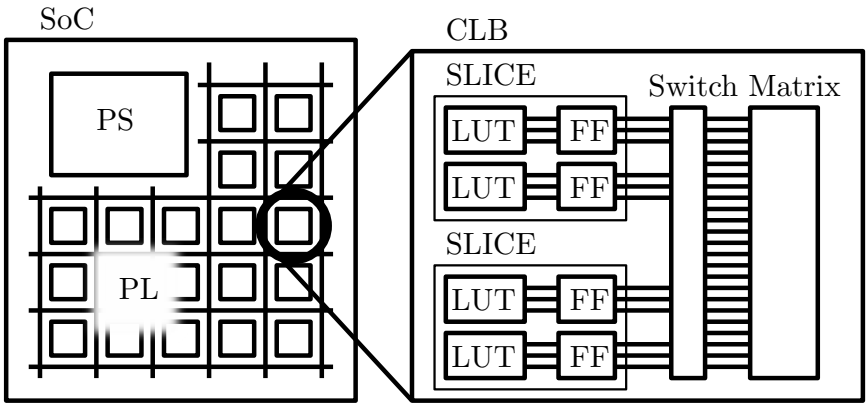


Figure 2.12: Schematic of a modern SoC with PS part and a PL part on the same chip. The right side of the figure shows the basic structure of a CLB containing LUTs, FFs, and a switch matrix.

A schematic of the SoC used in this work is shown in Figure 2.12. The system consists of a hard-wired Processor System (PS) in the upper left part and a Programmable Logic (PL) part. The PL part of the system consists of many Configurable Logic Blocks (CLBs) and a complex routing system. The CLB is built up by several slices and a switch matrix that connects the slices to themselves and to the outside. A slice consists mainly of LUTs to implement logic functions, and FFs to store data. The Xilinx Zynq device, which is used in this work, has two slices per CLB. Each slice consists of four LUTs and eight FFs [Xil14b]. The LUTs can be used as either one 6 bit LUT, or two 5 bit LUTs.

Dynamic partial reconfiguration allows the FPGA part to be partly reconfigured during runtime, without interfering with the rest of the FPGA. This feature can be controlled by various interfaces, e.g., the Processor Configuration Access Port (PCAP) or the Internal Configuration Access Port (ICAP). In this work, we use the PCAP interface, that allows the PS to reconfigure the PL.

Figure 2.13 shows a commonly used implementation of a CLB using N-channel MOSFET (NMOS) pass transistors on an FPGA. This is the mainly

used implementation in modern devices as it consumes less area than using transmission gates [CB13]. Transmission gates on the other hand would have the advantage of providing a more stable signal level. Extra level restorer stages to refresh the signal level would not be needed. Chiasson and Betz [CB13] wrote an interesting work about the advantages of transmission gates on FPGAs and the current usage of pass transistors.

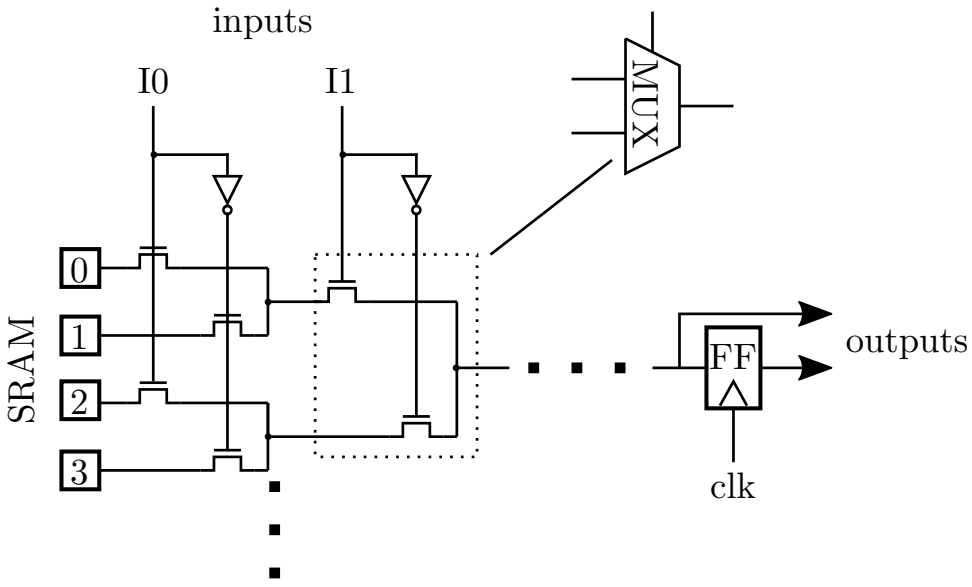


Figure 2.13: Simplified schematic of the implementation of a LUT on an FPGA.

The LUT values are stored in SRAM cells. In this work, 6 bit LUTs are used, i.e., 64 SRAM cells are needed to store the LUT information. The SRAM values of all LUTs are usually written when loading the configuration design of the FPGA from the external non-volatile memory. They do not change their values during runtime unless PR is used. The SRAM values propagate through a mux network that is controlled by the inputs I0 to I5 for a 6 bit LUT. For the sake of simplicity, extra buffer stages and level restorers, which are commonly found in real implementations, are left out in Figure 2.13. The usage of pass transistors is the reason why modern FPGAs have more NMOS than P-channel MOSFET (PMOS) transistors.

2.5 Cryptography

Cryptography is defined as the practice and study of writing and reading secret messages and codes. Two parties, e.g., Alice and Bob try to communicate with each other over an insecure channel. Active and passive third parties should be prevented from reading messages (confidentiality),

manipulating messages (integrity), claiming other identities (authenticity), or disturb the availability of information. Two different approaches to fulfill these requirements are symmetric and asymmetric cryptography. Both concepts will be explained briefly in the following subsections.

2.5.1 Symmetric

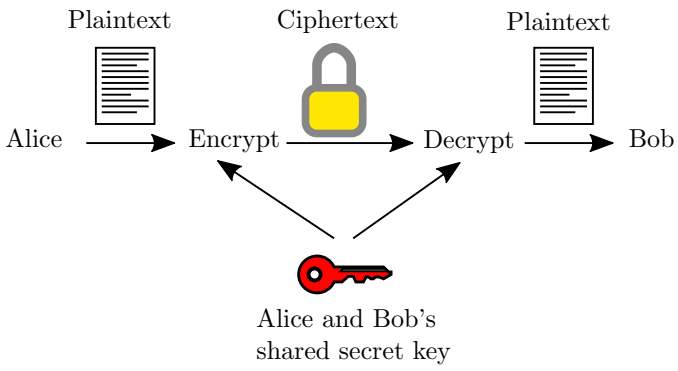


Figure 2.14: Schematic of a symmetric key algorithm. Alice wants to send a secret message to Bob without a third party being able to read the information on the communication channel.

Figure 2.14 shows the basic concept of symmetric cryptography. Symmetric refers to the usage of the cryptographic key. A message has to be delivered securely from Alice to Bob. Alice uses a shared secret key to encrypt the plaintext. After receiving the ciphertext, Bob decrypts the message using the same shared secret key. The state-of-the-art symmetric cryptographic algorithm is Advanced Encryption Standard (AES). It is a block cipher that was developed by Joan Daemen and Vincent Rijmen as the successor of the Data Encryption Standard (DES). By using symmetric keys for encryption the confidentiality of data is ensured.

Another application purpose of symmetric keys are MACs. A MAC is a secure checksum that can be calculated using the plaintext and the symmetric key as input parameters. The MAC is sent together with the plain- or ciphertext to the receiver, which himself can calculate the MAC using the received text and the same symmetric key. The calculated MAC has to be the same as the received MAC. Only parties holding the symmetric key should be able to calculate a correct MAC to a given text. As long as the key is being kept secret between the participating parties, the authenticity and integrity of communicated information can be ensured.

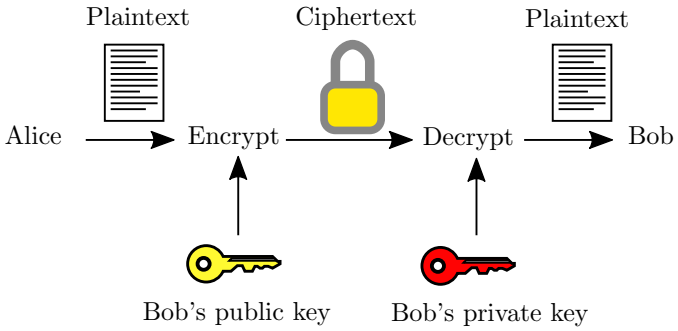


Figure 2.15: Schematic of a public-key cryptography algorithm used for encryption of data.

2.5.2 Asymmetric

Asymmetric key algorithms follow another approach. As shown in Figure 2.15, the keys used for encryption and decryption are different. The recipient of a message has a pair of keys: a private and a corresponding public key. In this case Bob has a private key, which he keeps secret and a public key, that he gives to Alice. Alice can use the public key to encrypt the plaintext. The ciphertext can only be decrypted by holders of the private key, which in this case is Bob. As long as Bob keeps his private key secret and Alice uses the correct public key, the confidentiality of the data is ensured.

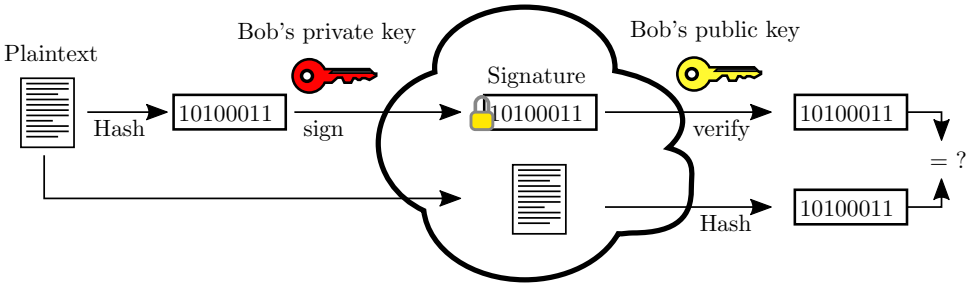


Figure 2.16: Schematic of public-key cryptography algorithm used for digital signatures.

Another possible use case of asymmetric ciphers are digital signatures as shown in Figure 2.16. The private key can be used to sign a message. By using the public key, this signature can be verified by any third party who has the corresponding public key. A valid signature, however, can only be calculated by the owner of the private key. This ensures authenticity and integrity of the data, as long as the private key is being kept secret and the correct public key is used.

In order to ensure the usage of the correct public keys, a PKI is needed.

The PKI can either be centralized with a trusted Certificate Authority (CA) certifying key pairs or be decentralized such as Pretty Good Privacy (PGP).

The state-of-the-art asymmetric algorithms are RSA and elliptic curve cryptography. RSA is named by its inventors Rivest, Shamir, and Adleman. It is based on the practical difficulty of factoring the product of two large prime numbers. Elliptic curve cryptography on the other, relies on the infeasibility of finding the discrete logarithm of random elliptic curve elements with respect to a publicly known base point. They play an integral role in modern authenticated communication, or to establish symmetric keys. Symmetric ciphers usually outperform the asymmetric algorithms, but don't offer the advantages of having two distinct keys.

CHAPTER 3

Implementing RO-based PUFs on FPGAs

The implementation of PUFs on FPGAs is already covered in many research papers. Most of them focus on gaining measurement data to show the properties of different PUF types. Very little research exists about efficiently implementing them on FPGAs. This chapter covers the question if an efficient implementation of RO-based PUFs on FPGAs is possible without using any partial reconfiguration. The experimental data from this chapter is later on used as a reference for quality measures and area efficiency.

The chapter is organized as follows: in section 3.1 a short introduction is given. Section 3.2 covers the theoretical aspects of implementing RO-based PUFs on FPGAs. In section 3.3 a design with 4272 ROs is implemented on a Xilinx Zynq device. The implemented design will be used and measured in section 3.4 to analyze the characteristics of the RO-based PUF with the focus on area, speed, and quality properties. In section 3.5 a conclusion is given with an outlook and motivation to the next chapter.

3.1 Introduction

The SRAM cells on modern FPGAs are initialized at start-up and cannot be used as a PUF. As it is hard to implement exact symmetrical structures on FPGAs — which are needed for implementing own memory-based PUFs such as butterfly PUFs — the focus was set on delay-based PUFs. Because the arbiter PUF is susceptible to machine learning attacks [HMV12], RO-based PUFs will be used in this thesis. The main goal of this work is to create the bit source for a device unique key. Therefore, only PUFs with a fixed challenge are used, i.e., weak PUFs.

The RO PUFs were implemented on a Xilinx Zynq Z7C020. This device

uses CLBs with two slices each. Each slice contains four LUTs. Each LUT can be used as either one 6 bit LUT, or two 5 bit LUTs. Each LUT can use up to two additional FFs to store values. Apart from logic function generation, the slices can also be used for carry chain logic. Carry chain logic can be used for very fast adders, comparators, and counters. The signal path is optimized via a special direct routing to the next slice and CLB. Each slice can be used as a 4 bit carry chain [Xil14a].

RO-based PUFs try to use frequency differences caused by production tolerances to extract entropy from routing and logic. But this frequency also depends on many influences such as noise, reversible variations, irreversible variations, and location within the device. To minimize those influences on the measurement and maximize the effect caused by production tolerances, differential measurements, as explained in Figure 2.7, were used.

Different possible RO PUF constructions were analyzed for their area consumption, speed, and reliability. It is always a trade-off between those properties when using PUF technology. A high area consumption also leads to a higher cost. Especially in high volume products, this can turn out to be very expensive. Another aspect is speed: when using an Electrical Control Unit (ECU) of a car, tight timing constraints are demanded by the Controller Area Network (CAN) protocol. If a PUF-based key-generation slows down the whole device, those timing constraints might not be reached and the device would not work anymore. On the other hand, a low area PUF that is read-out very fast might lead to a very unstable behavior and an unreliable cryptographic key. Therefore, the requirements have to be analyzed in detail before designing the PUF. The main focus of this thesis is on an area optimized RO PUF implementation with good reliability. Every design was tested for their area usage, speed, uniqueness, uniformity, bit alias, and reliability.

3.2 Theory

Figure 3.1 shows the schematic and an exemplary implementation of an RO PUF on an FPGA. In this case, three inverters are used in the oscillation chain. The first inverter is implemented as a NAND gate to be able to enable and disable the oscillation. Each inverter stage can be implemented by using a single LUT, i.e., a 1 bit LUT for an inverter and a 2 bit LUT for the NAND gate. The complex routing system is used to implement the forward-feeding and loop-feeding of the signals.

Many previous RO PUF implementations used very long inverter chains, to limit the frequency of the ring. A problem that occurs with the very short 3-inverter ring is that no current FPGA can handle the resulting frequencies in the synchronous clock domain. These frequencies usually lie between 500 MHz and 1 GHz, as shown later in this thesis. The downside of using

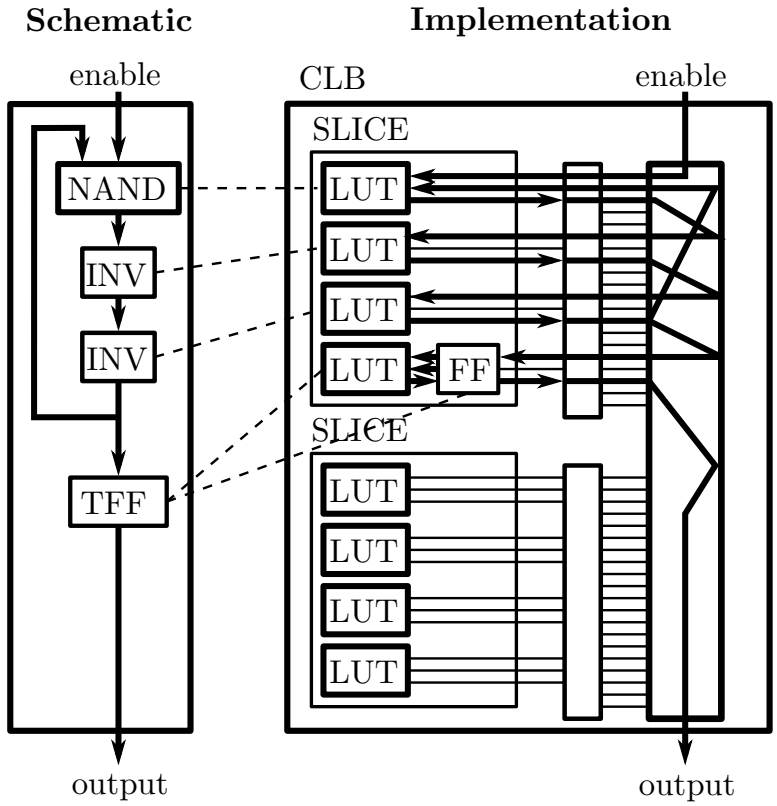


Figure 3.1: Schematic and implementation of a 3-inverter RO PUF on an FPGA. A NAND gate is used to be able to enable and disable the oscillation. A TFF is used to halve the frequency.

large inverter chains is the large area usage without gaining more extractable entropy.

Another approach to lower the frequency is to use one or more Toggle Flip-Flops (TFFs) after the RO. A TFF changes its state ('toggles') on every input clock cycle. Figure 3.2 shows the structure of a TFF. A single Data Flip-Flop (DFF) and an inverter can be used to halve the frequency of an input signal. The high RO output frequency f is used as the clock signal of the DFF to cause an inversion of the output signal. The resulting frequency is exactly $\frac{1}{2}f$. It has to be ensured that the data path delay from the Q output of the DFF, through the inverter, to the D input of the DFF is not larger than the period of the DFF clock, i.e., the RO frequency.

By choosing an appropriate number of TFFs, the frequency can be lowered to an FPGA processable level. The TFF in Figure 3.1 is implemented on the remaining LUT of the upper slice. Again, the LUT is used to implement an inverter and additionally one FF for storing the current value. The output of the FF is then routed outside the CLB and used as the out-

put frequency. Using this technique it is possible to implement a 3-inverter RO-PUF on a single slice. A downside of using TFFs on the Xilinx Zynq is that only one clock domain for memory elements is allowed per slice. This limits the amount of TFFs per CLB to two [Xil14b].

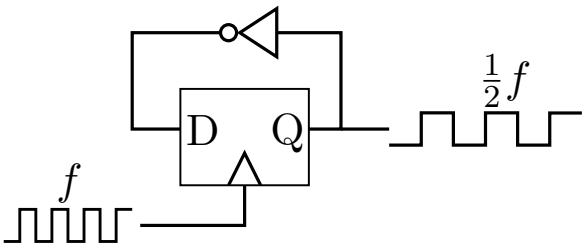


Figure 3.2: Schematic and implementation of a TFF

As explained in Figure 2.7, the frequency of two such chains can then be counted and compared to generate a single PUF bit. Various methods to implement the counters on FPGAs exist. One possibility is to use two synchronous counters that sample the RO frequencies with the global device clock [GCvDD02]. Every detection of a positive signal edge will then increment the counter. This, however, limits the RO frequency to half of the global clock frequency. This problem can be overcome by using TFFs.

Another possible method is the usage of an asynchronous counter [Mer14]. Two ripple counters are fed by the two RO frequencies. The global clock is used as a reference counter to start and stop both measurements at the same moment. The advantage of this method is that higher input frequencies can be processed. As long as the delay line capacity connecting the RO output with the counter is not too long to be charged in time and the first DFF can process the oscillation, very high frequencies can be processed. This also achieves a higher measurement precision, compared to a synchronous measurement. Another advantage of this method is the simple implementation and feasibility in FPGA logic. Most FPGAs have CLBs with integrated carry chains, which can be used to implement the counters very efficiently.

3.3 Implementation

The implementation of RO PUFs on an FPGA requires some considerations. As a synthesis tool always optimizes a circuit for speed and area, a RO with unpredictable behavior would either throw a latch warning or be completely optimized out as it serves no purpose from a tool point of view. Additionally, every pair of ROs that is being compared had to be implemented exactly the same. The placement and routing had to be thoroughly constrained. The usage of slices, logic, pins, and routing was manually fixed to prevent any optimization by the synthesis tool. In earlier works, hard macros were used

in the Xilinx ISE suite. In the newest toolchain, Vivado, this feature was removed. As a replacement, constraints can be defined in the TCL format or the Xilinx proprietary XDC. In this thesis, mainly XDC constraints were used to fix the placement of logic and routing. Each RO implementation should only occupy one CLB and be limited from using outside logic and/or routing.

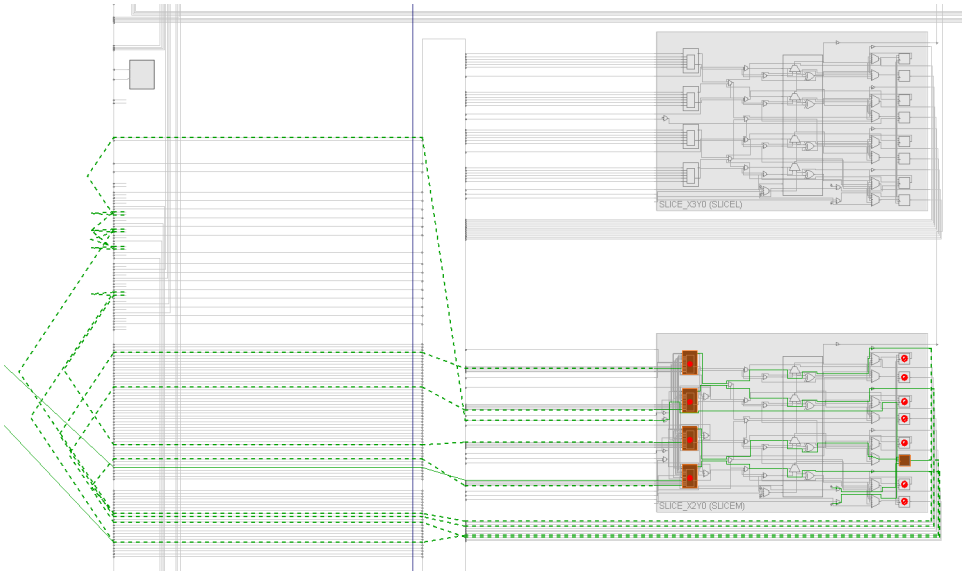


Figure 3.3: RO PUF with three inverters and one TFF implemented on one slice. The only signals entering and leaving the slice and CLB are the PUF enable and output signal.

Figure 3.3 shows the FPGA implementation of the 3-inverter RO PUF presented in Figure 3.1. All relevant signals for the oscillations are dashed and therefore had a fixed routing. Every implementation of this RO PUF was exactly identical. The usage of any logic not belonging to the PUF was prohibited within the CLB to prevent any interference. With this method it was possible to implement a 3-inverter RO PUF on one slice (see Figure 3.1) and a 5-inverter RO PUF on one CLB, i.e., two slices. With seven inverters it was not possible to keep the routing within the CLB. This would create problems with the routing of neighbored ROs and make identical implementations very hard.

By using 3-inverter RO PUFs, it was possible to use the other slice in the CLB for another purpose, e.g., another TFF to lower the frequency even further. One CLB could be used for a RO with three inverters and two TFFs. Timing problems with high frequencies might occur, if the second TFF would not be on the same CLB. Therefore, a second TFF was used right after the first one to divide the initial frequency by four.

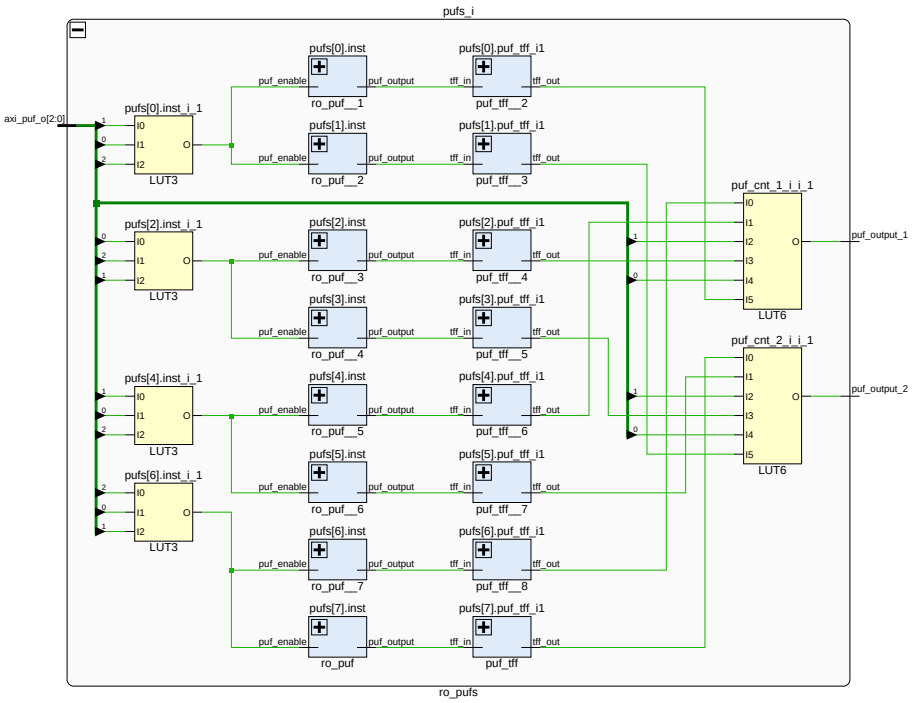


Figure 3.4: Schematic of eight RO PUFs. The yellow rectangles are the input and output muxes. The blue rectangles are the eight ROs with two TFFs each.

Figure 3.4 shows an exemplary schematic of eight ROs. In the first stage, the selected RO pair is enabled, all others are disabled. The blue rectangles are the eight ROs themselves including one TFF, and a second TFF right after the RO block. Two ROs are activated at the same time. Two LUTs serve as muxes to select the correct RO outputs. Very efficient mux implementations are possible on FPGAs [Cha14].

It will be shown in chapter 3.4, that the first implementation of the PUF construction was very unstable. This was caused by random logic being placed on the same CLBs as the ROs and TFFs. Figure 3.5 shows a biased RO pair, caused by interfering logic. Both 3-inverter ROs had the same implementation and routing. But CLB 1 was not only used for the RO implementation (orange boxes), but also for other logic (turquoise boxes). This surrounding logic can have various effects on the frequency of the RO induced by, e.g., electromagnetic emanation or local power drains. CLB 2, on the other hand, was only used for the RO chain. This ultimately led to a lower frequency for the lower RO and therefore a bias towards 0.

This problem was fixed by forcing the second TFF to be placed on the same CLB and restraining any other logic to be placed on the same CLB.

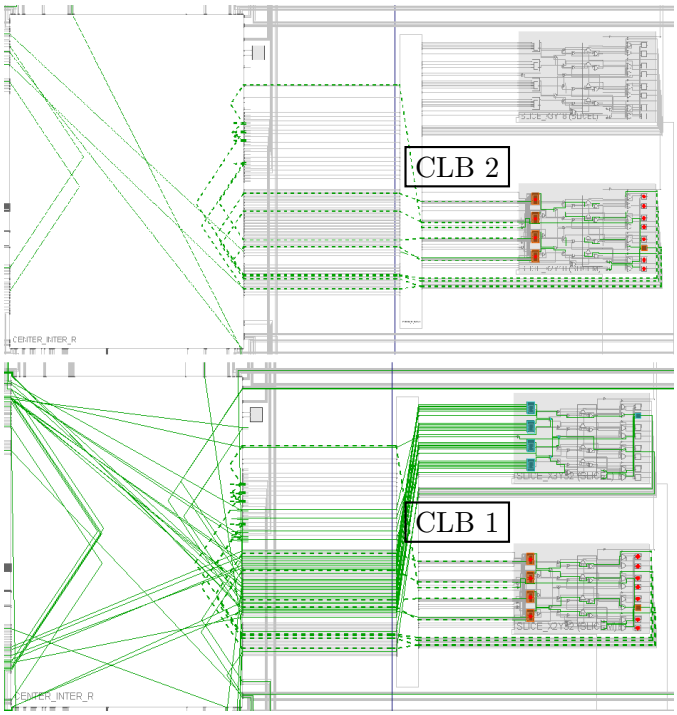
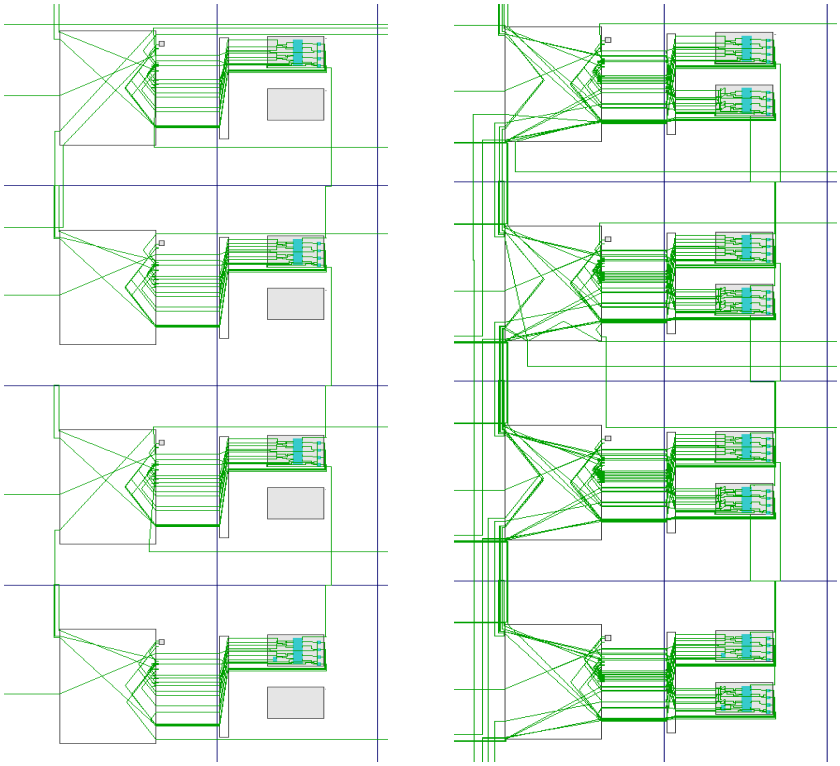


Figure 3.5: Biased pair of ROs. Both 3-inverter ROs (orange) had the same implementation and routing. CLB 1 used its second slice for other logic, whereas CLB 2 left the slice free. This resulted in a lower frequency for the RO on CLB 1 and a bias towards 0.

This, however, raised the area consumption, as another slice was occupied by just one TFF.

The implementation of an asynchronous counter was much more area efficient in the used FPGA technology. Furthermore, only the first flip-flop of the ripple counter ran at the high input frequency, thus making it easier to achieve a design that meets the timing requirements. In Figure 3.6a, four slices were enough to implement the 16 bit version of the counter. By using the 4 bit carry chain element in each slice together with four FFs to store the according counter values, a much better area efficiency than a synchronous counter was achieved. Lower slices of a CLB can only be connected with lower slices of another CLB to a carry chain (and upper slices with upper slices accordingly). Therefore, four CLBs were needed to implement a 16 bit counter. This was of advantage when using two 16 bit counters. The lower slices could be connected to a chain, and the upper slices accordingly. This can be seen in Figure 3.6b. All four CLBs were used efficiently with no space for other logic.

The bit size of the counters is important for the precision of the measurement. The longer the measurement, the more precise, but also the slower



(a) 16 bit Ripple carry counter implemented on four CLBs resp. four slices.

(b) Two 16 bit Ripple carry counters in four CLBs resp. eight slices

Figure 3.6: Area efficient implementation of 16 bit asynchronous ripple carry counters.

it is. Another problem is that the longer the PUF is activated, the hotter it gets. This has an influence on the frequency as shown in previous work [SC06, LBGB00, Zha13]. It was shown by Lopez et al. [LBGB00], that a countermeasure against this is to either let the oscillator run as shortly as possible or as long as possible, because the time-frequency decrease follows a second order exponential curve. Letting the oscillator run too shortly, on the other hand, leads to a high possible measurement error. When measuring a mean frequency f of the ROs for a measurement time t_{meas} a maximum measurement error [Mer14] can be calculated to:

$$e_{max} = \frac{1}{2 \cdot t_{meas} \cdot f} \tag{3.1}$$

Measurements of ROs with various lengths showed that 3-inverter ROs on a 28 nm FPGA had frequencies between 513 MHz and 548 MHz. When using 1-inverter ROs the highest frequency was measured to up to 1300 MHz. As 3-inverter ROs were used in this chapter, a maximum frequency of 600 MHz was assumed. This was important to choose the right amount of TFFs and the necessary bit size of the frequency counter. To lower the maximum frequency of 600 MHz to a constrainable level, two TFFs were used. This lowered the maximum input frequency for the measurement circuit to 150 MHz. A 12 bit counter was used to have enough room for a potentially higher frequency.

A reference counter was used to start and stop the measurement of a RO pair. It was chosen to be 10 bit wide with a frequency of $f_{ref} = 100\,\text{MHz}$. The DFF data path delay of the first TFF was calculated to 1.006 ns, of which 0.642 ns were caused by logic delay and 0.364 ns by route delay. The maximum input frequency for the first TFF was therefore roughly $\frac{1}{1\,\text{ns}} = 1\,\text{GHz}$. With a maximum RO frequency of 600 MHz this design met the requirements.

As additional errors are introduced by, e.g., environment conditions such as temperature and voltage, the measurement error had to be kept as small as possible. An acceptable value could be defined as $e_{max} = 0.05\,\%$ [SC06]. By using TFFs before the measurement, the frequency was lowered to an acceptable value. The input frequency of the measurement circuit was timing-constrained with the highest previously measured frequency, to guarantee a glitch free behavior for all ROs. The minimum runtime of the measurement [Mer14] can then be calculated to:

$$min(t_{meas}) = \frac{1}{2 \cdot e_{max} \cdot f_{min}} = \frac{1}{2 \cdot 0.05\,\% \cdot 128\,\text{MHz}} = 7.8\,\mu\text{s} \tag{3.2}$$

A 10 bit reference counter with a reference frequency of 100 MHz has a runtime of:

$$t_{meas} = \frac{2^{10}}{100\,\text{MHz}} = 10.24\,\mu\text{s} \tag{3.3}$$

Therefore, the minimum runtime requirement was fulfilled.

For each output bit, two RO frequencies had to be measured, along with a reference clock which started and stopped the measurement. For area

efficiency it would be enough to measure only one frequency at a time, store it, and compare it with the next measurement to generate an output bit. On the other hand, implementing a second RO counter would accelerate the measurement time by a factor of two, while the area overhead would be very small with three slices for a 12 bit counter. The two counters for the reference and the RO measurement could be efficiently implemented according to Figure 3.6b.

For this implementation the three counter method was chosen. One counter was used as a reference counter and two counters to count the frequencies of one RO pair. If faster read outs were needed, they could be further sped up by using more counters to measure multiple RO pairs at the same time.

As the resulting PUF output bits are not stable, they cannot be directly used as a key. A key generation scheme is needed. Therefore, more raw bits are needed to be generated than can be used as key bits. Common algorithms need around 12 input bits to generate one key bit [HMSS12, MTV09b]. These works used SRAM PUFs with a bit error probability of around 15 %. More sophisticated methods like the DSC by Hiller et al. [HYS16] decreased the amount to 7.6 input bits per key bit. Hence, an estimation using 12 PUF bits per key bit is very conservative, as RO PUFs showed better results in terms of stability [KKR+12].

The total amount of unprocessed PUF bits r to generate a 128 bit key is therefore:

$$r = 12 \cdot 128\,\text{bit} = 1536\,\text{bit} \tag{3.4}$$

As every bit needs two ROs, an implementation with 3072 ROs was needed.

3.4 Experimental Results

In this section a design using the choices made in the previous section was implemented using the Vivado toolchain and a Xilinx Zynq Z-7020 SoC. This chip consists of two ARM Cortex-A9 Cores as the Processor System (PS) part and an Artix-7 FPGA as the Programmable Logic (PL) part. The PS has access to the external memory and can configure and reconfigure the FPGA. After booting the system, the PS loaded the PUF design from the external memory. The PS was furthermore used to make and control the measurement of the PUFs on the PL part of the board.

To get a statistically more reliable result, the largest possible amount of

ROs was used on the available space. 4272 ROs were implemented on the FPGA area. This resulted in a total amount of 2136 PUF bits, whereas only 1536 bit would have been needed for a 128 bit key. One counter was used for the reference counter and two counters for a pair of ROs. As clock domain crossing occurs when reading out the RO counters with the system clock, additional synchronization logic was implemented.

3.4.1 Area

The area consumption of the design depended on the necessary amount of PUF output bits. For each output bit, two ROs were needed, which consumed one CLB each. These CLBs could not be used for any other logic. As shown in Figure 3.6b, two 16 bit asynchronous counters consumed four additional CLBs. The two TFFs before the measurement stage needed an additional two CLBs.

	RO PUFs	Measurement	Total available	% of total
FF	8544	67	106400	9%
LUT	24908	28	53200	48%
MUX	846	0	39900	2%
Slice	10560	25	13300	80%

Table 3.1: Resource utilization of a PUF design using 4272 3-inverter RO PUFs with three counters to read out the frequencies of the ROs and generate one bit per RO pair. The used SoC was a Xilinx Zynq Z-7020.

Table 3.1 shows the area needed for the implemented design. It can be easily seen that the vast majority of logic resources was used by the 4272 RO PUFs. According to Figure 3.1, three LUTs were needed for every 3-inverter RO. An additional pair of LUT and FF was needed to implement the TFF. Roughly one LUT per RO was occupied by the input mux stage. 348 LUTs and 143 muxes were needed to implement the output mux stage to select the correct RO output frequencies.

The measurement circuit, on the other hand, only consumed 42 FFs and 28 LUTs. Three muxes were needed for each 10 bit, resp. 12 bit counter, and two additional multiplexer for the comparison of the results to generate one output bit.

In conclusion, it can be said that using this RO PUF on existing FPGAs was very area inefficient, which might lead to high cost. Around 80 % of the slices and almost 50 % of the available LUTs were occupied by the PUF structures. The key generation algorithm is not even included in this calculation. Therefore, it might be good for testing purposes, but absolutely not feasible for real life applications. The results will be used in the next

chapter as a reference for area consumption as well as quality measures.

3.4.2 Speed

Another important requirement to PUFs is a fast processing to not hold back the regular IC function for long times. As already calculated, the comparison of two ROs using a 10 bit 100 MHz reference counter took 10.24 µs. As a total of 2136 bit was generated using 4272 RO PUFs a total read-out time of

$$t_{total} = 2136 \cdot 10.24\,\mu s = 21.87\,ms \tag{3.5}$$

was expected.

Simulation of the design confirmed this theoretical approach, where the measurement took 21.98 ms, including some extra wait states causing overhead. The real implementation of the designs suffered from an additional delay due to the read out of the resulting bits to the ARM processor. To measure all 4272 RO PUFs 1000 times, and writing back the results to a text file on the SD card using the PS took 27.05 s. Therefore, the measurement time of one 2136 bit PUF output could be calculated to 27.05 s/1000 = 27.05 ms, which was still a very good result, considering the overhead caused by the interaction with the PS. This could be further improved when processing the output bits inside the FPGA or reading out more than one RO pair at a time.

3.4.3 Quality Properties

The experiments were done on a total of ten different boards: four ZedBoards and six Xilinx ZC702 evaluation boards. Both use the same Xilinx Zynq Z-7020 SoC. Each RO was measured 1000 times. Both the resulting bits of the RO pair comparisons and the raw counter values of each RO were stored. This allowed a better analysis of the data in Matlab. All measurements were performed in a climate chamber at exactly 25 °C and a constant FPGA core voltage of 1.0 V.

Frequencies

The counter values were transformed to frequencies using following equation:

$$f_{RO} = \frac{f_{ref} \cdot 2^{n_{tff}}}{2^{l_{ref}}} \cdot r_{cnt} \tag{3.6}$$

With the reference frequency $f_{ref} = 100\,\text{MHz}$, the number of TFFs in the design $n_{tff} = 2$, the length of reference counter $l_{ref} = 10\,\text{bit}$, this yields an RO frequency f_{RO} of:

$$f_{RO} = \frac{100\,\text{MHz} \cdot 2^2}{2^{10}} \cdot r_{cnt} = 390\,625 \cdot r_{cnt} \tag{3.7}$$

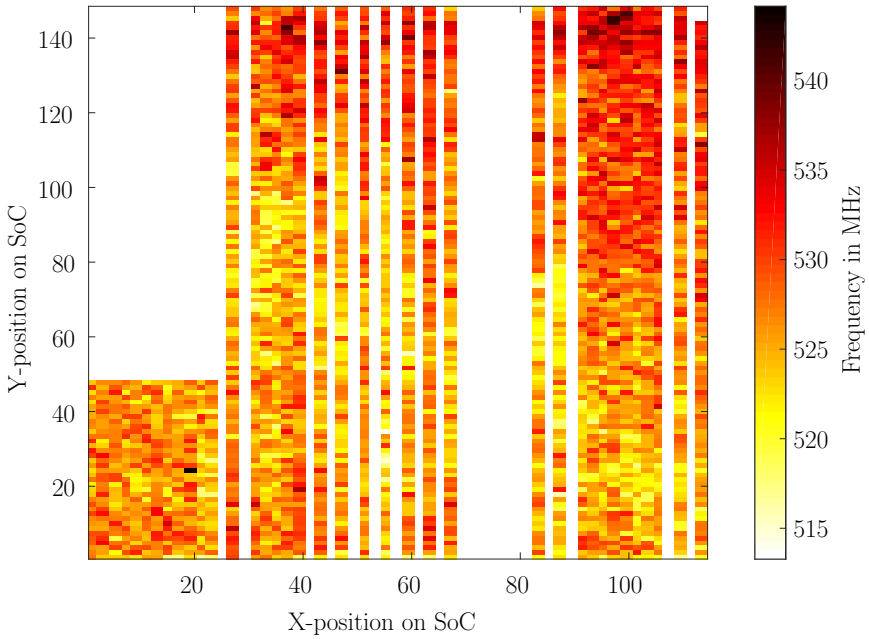


Figure 3.7: Heatmap of the frequencies of each of the 4272 ROs on the Zynq. The PS is located in the upper left corner. Each rectangle represents one RO measurements.

This formula was used to calculate the frequency of each RO on the SoC. The distribution of the frequencies on one SoC is shown in the heatmap in Figure 3.7. Each of the 4272 rectangles represents the frequency of one RO

instance. The PS is situated in the left upper part of the figure. ROs in the middle region that were close to the PS tended to have a lower frequency. Furthermore, the left lower and especially the right upper corner tended to produce higher frequencies. Possible reasons for that effect are diverse: influences from the PS, chip design, manufacturing, different temperatures within the chip, or electromagnetic interference with surrounding logic. This could lead to spatial correlation, when comparing ROs that are not close to each other. Therefore, only neighbored ROs were used to generate bits. It can be seen that neighbored ROs had a more similar and therefore less correlated frequency. The actual frequency difference depended almost entirely on production tolerances. On the other hand, an almost equal frequency might lead to stability problems, as shown in chapter 5.

The mean frequency was measured to 526 MHz with a variance of 16.3 and a standard deviation of 4. This frequency met the requirements of the 12 bit counter. The frequency could be theoretically up to 1.6 GHz before wrapping around the 12 bit counter limit.

Furthermore, it can be seen that the area consumption to produce a 2136 bit key was very high. Almost all of the CLBs on the Zynq were occupied, rendering it almost impossible to place any other logic on the FPGA.

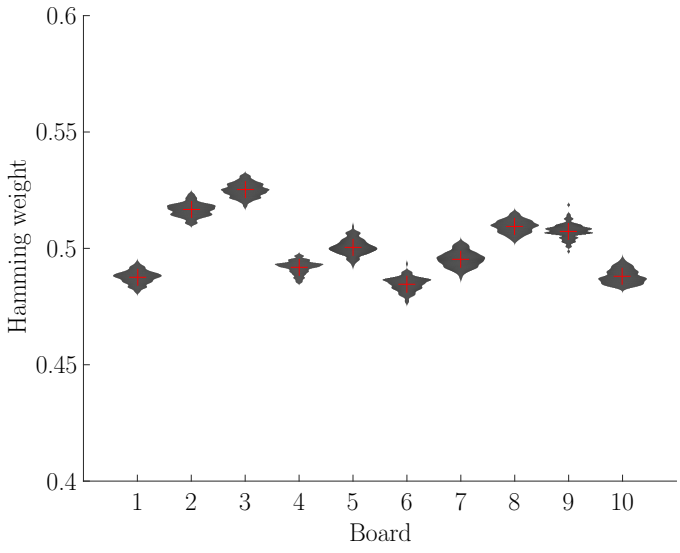


Figure 3.8: Bean plot of the HW of the same 3-inverter RO PUF implementation on 10 different boards.

Uniformity

The uniformity of the PUF responses was measured using the HW. Figure 3.8 shows the HW of the whole 2136 bit response on each Board. To measure the stability of the responses, 1000 consecutive measurements at the same temperature and voltage were taken and visualized in a bean plot. Bean plots visualize multiple probability distributions in one graph. Each bean represents the distribution of the HW of the 2136 bit responses.

It can be seen that within the beans the results show the form of a normal distribution with a small standard deviation. The mean value of each bean, i.e., of each HW on each board, was very close to the ideal value of 0.5. Therefore, an equal distribution of ones and zeros was given. The worst results was measured on board 3 with a mean HW of 0.525. However, this could also be the result of a statistical outlier, as the amount of bits with 2136 was still relatively low.

Stability

The stability of the 1000 consecutive measurements was measured using the intra-device HD. The first measurement on each board was taken as a reference, and then the distance to each other response was calculated.

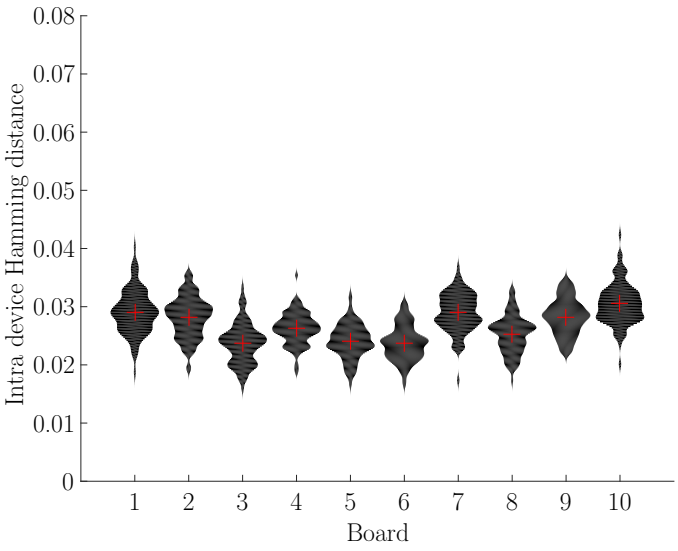


Figure 3.9: Bean plot of the Intra-device HD. The first measurement on each board was taken as a reference value and compared to each of the consecutive measurements.

Figure 3.9 shows the bean plot of the intra-device HD. It can be seen that the mean values of each board were very close to each other. Therefore, the

intra-device HD was not depending on the board and had a value relatively close to the mean over all boards of 2.75 %. This means that almost 3 % of all bits changed during consecutive measurement at the same temperature and voltage on the same board. This is due to very similar frequencies of neighbored ROs. When analyzing the counter values it could be seen that the counters sometimes even had the same output value, and a deviation of around 2 to 3 at an absolute counter value of around 1350 was normal. Compared to other PUF types and even other RO PUF implementations on FPGAs, this was still a very good result [KKR+12]. The distribution of the HD on the same board showed the form of a normal distribution in the bean plot. An in-depth analysis of the reliability under different environmental conditions and aging was performed in chapter 5.

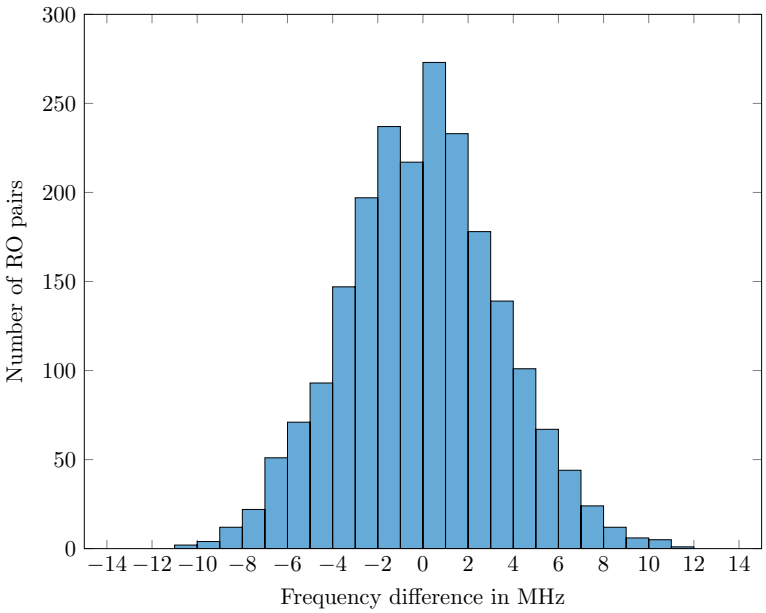
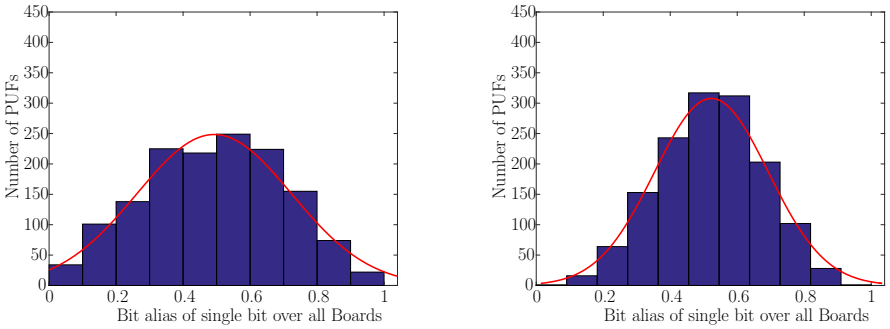


Figure 3.10: Histogram of the frequency difference between each of the 2136 RO pairs on one of the FPGAs. The absolute mean frequency of the ROs was 526 MHz.

Figure 3.10 gives a further insight in the counter differences of the RO pairs. The frequency differences of all 2136 RO pairs on one of the FPGAs are plotted. A possibility to decrease the intra-device HD is to ignore all RO pairs with a very small frequency difference during an enrollment phase. These RO pairs would not be used in further measurements. The choice of RO pairs is device specific, as the production tolerances are random. On the other hand, doing this might remove a lot of RO pairs. As seen in the graph, most RO pairs are close to a very small frequency difference. Furthermore, the amount of removed pairs must be known in advance, when a certain



(a) Bias of many bits towards 0. This results in a bad HD and HW and therefore to a bad Uniqueness of the PUF.

(b) Mean value at 0.5, a clearly visible normal distribution and a small standard deviation.

Figure 3.11: Histograms of bit aliases of single bits measured over all boards.

amount of PUF bits are needed. An estimation of the stability is also very important for the choice of a suitable error correction algorithm and its parameters.

Bit alias

The bit alias measures the distribution of ones and zeros of the same RO pair on different boards. As the frequency difference should only be depending on production tolerances, the same PUFs on different boards should produce an equal ratio of ones and zeros. This was used in the previous section to identify the bad design. Figure 3.11a shows the bit alias of the first implementation, where the second slice on the CLB was filled with random logic and was not consistent for every RO. Although the histogram is centered at 0.5 and the mean value is very good with 0.49, the normal distribution is very flat. A higher peak in the middle would be expected. The problem is, that every deviation from 0.5 is bad, no matter if it is towards 0 or 1. The standard deviation was measured to 0.231. This ultimately led to a bad HW and inter-device HD.

Figure 3.11b shows the result of the new design, where the neighbored slice was restrained from using any other logic, and the second TFF was placed in that position. The Gaussian is much more narrow, with a high peak in the middle. Almost no RO pair produced constant zeros or ones on all boards. Most RO pairs had an equal distribution of ones and zeros. A mean value of 0.52 and a standard deviation of 0.176 were calculated.

Uniqueness

To measure the uniqueness of the complete PUF responses, the inter-device HD was used. Therefore, the HD between the PUF responses of each board

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0.00	0.50	0.49	0.49	0.47	0.51	0.48	0.49	0.50	0.48
2	0.50	0.00	0.50	0.50	0.48	0.48	0.50	0.49	0.50	0.49
3	0.49	0.50	0.00	0.50	0.51	0.49	0.50	0.48	0.49	0.49
4	0.49	0.50	0.50	0.00	0.49	0.48	0.49	0.50	0.49	0.50
5	0.47	0.48	0.51	0.49	0.00	0.47	0.48	0.48	0.49	0.49
6	0.51	0.48	0.49	0.48	0.47	0.00	0.50	0.48	0.48	0.48
7	0.48	0.50	0.50	0.49	0.48	0.50	0.00	0.49	0.49	0.48
8	0.49	0.49	0.48	0.50	0.48	0.48	0.49	0.00	0.50	0.49
9	0.50	0.50	0.49	0.49	0.49	0.48	0.49	0.50	0.00	0.47
10	0.48	0.49	0.49	0.50	0.49	0.48	0.48	0.49	0.47	0.00

Table 3.2: Inter-device HD between each 2136 bit PUF output of each of the ten measured boards.

was calculated. Table 3.2 shows the results. It can be seen that every HD was very close to the ideal value of 0.5. The worst value was measured between board 1 and 5, and board 9 and 10 with 0.47. This might be a statistical outlier and was still within an acceptable range considering the relatively small amount of bits for statistical calculations. The mean value of the inter-device HD of the unfixed design was 0.43 and validated the assumption of a bias. The mean value of the new design was 0.492.

3.5 Conclusion

The designed 3-inverter RO PUF fulfilled all requirements to produce unique and stable outputs. The results were in line with previous research, although the RO chains had a much smaller amount of inverters. However, the design still required a very large area. This made it not efficiently usable in real-world scenarios. A solution to overcome this problem will be presented in the following chapter. Furthermore, all measurements were made under the same environmental conditions, i.e., temperature and voltage. Aging of the device was not considered as well. These variations will be further examined in chapter 5.

CHAPTER 4

Enhancing the Efficiency of PUFs on FPGAs Using Partial Reconfiguration

The previous chapter showed the clear downside of using PUFs on FPGAs – the area consumption. To generate a bit vector of sufficient size, that can be used as a reliable key source after an error correction and key generation, almost the whole area of the Zynq Z-7020 was needed. In this chapter a new method is presented that allows RO PUF designs with a much smaller area consumption. The dynamic partial reconfiguration feature is used to extract more entropy of the complex logic and routing resources of the FPGA. Parts of this chapter were published and presented at two conferences [GS14, GS15] and in one journal [GS16].

The chapter is organized as follows: a short introduction and motivation for this new method is given in section 4.1. The prior work to using PR for PUFs on FPGAs is presented in section 4.2. The novel method is then explained in detail in section 4.3. An exemplary implementation of the method on an FPGA is presented in section 4.4 and the experimental results shown and analyzed in section 4.5. A detailed examination of the entropy of the PUF data is done in section 4.6. The conclusion is given in section 4.7.

4.1 Introduction

The main advantage of an FPGA over other ICs is its logic reconfigurability. An intuitive approach to use this feature to overcome the large area consumption of PUFs on FPGAs would be to use a combination of the PS and PL. The PS can load any design on the PL and overwrite it at any time. A key could be generated using one big PUF design, then read back to the PS, and the actual design could be loaded on the PL afterwards. Unfortunately,

this approach has many disadvantages. Firstly, the boot up time of the design is largely affected. In time-critical environments a running design on the PL is needed very fast. Furthermore, the FPGA is completely reset after loading another design on it. This renders parallel processing of other logic impossible. The biggest downside is that the key has to be stored temporarily in the PS and thus increasing its attack surface. The goal of this work is to generate a secure key and store it inside the FPGA without leaking it to the outside. This makes a complete reconfiguration of the FPGA infeasible.

A more sophisticated approach to reuse area on an FPGA is the dynamic partial reconfiguration feature of modern FPGAs. It started as a feature for the large and expensive devices such as the Virtex-4. Throughout generations it also made its way into cheaper devices. Today, all new devices of Xilinx and almost all new devices of Altera support this feature. Dynamic partial reconfiguration allows the FPGA part to be partly reconfigured during runtime, without interfering with the rest of the FPGA. This feature can be controlled by various interfaces, e.g., the PCAPs or the ICAPs. In this work, the PCAP interface, that allows the PS to reconfigure the PL during runtime, was used.

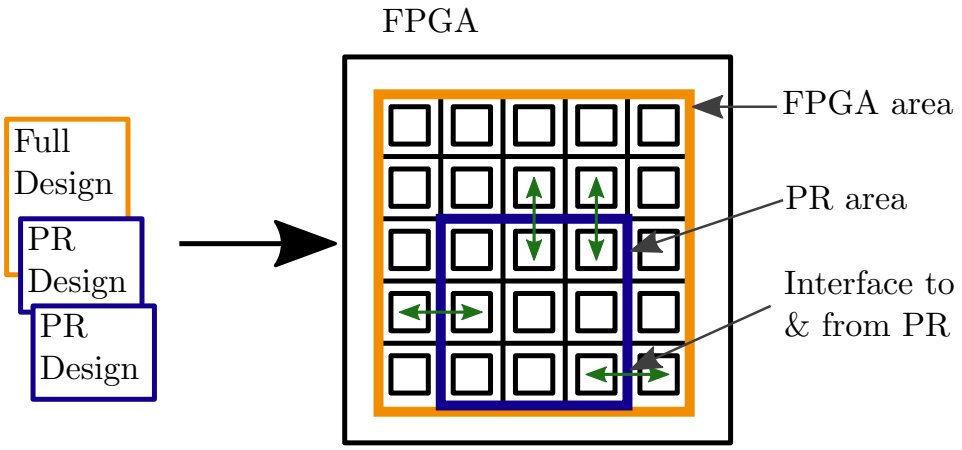


Figure 4.1: Functional principle of partial reconfiguration on a modern FPGA. A full design is used to configure the complete FPGA. Pre-defined parts of the FPGA can be dynamically and partially reconfigured using the PR designs.

Figure 4.1 shows the basic functional principle of PR. A full design is loaded on the FPGA to initialize all logic and routing resources. The complete design is already aware of the PR area and defines it. Afterwards, multiple PR designs can be loaded sequentially on the same predefined area, which is marked by a blue rectangle in Figure 4.1. The communication between the regular FPGA area and the PR area is done through user defined interfaces, that are realized within dedicated CLBs. Both the full design

and PR designs have to be aware of those interfaces. The PR area can be used for different types of logic that are needed for a certain time period. Multiple PR areas can exist on the same FPGA area. The PR design has to match the exact size of the PR area. PR designs can be programmed linearly faster than complete designs, depending on their size [Xil15].

When using partial reconfiguration in combination with PUF-based key generation, the area that was used for the implementation of the PUF and the key generation can be reconfigured with other logic. The area can be reused, while the FPGA keeps running. The key can stay inside the FPGA. Time critical functions can be loaded on the FPGA in parallel to the PUF key generation and already be executed. They would not be interfered with when reconfiguring the PUF area. Unfortunately, the area that is used for the reconfiguration has to have specified interfaces to other logic areas. The partial reconfiguration area is fixed in size and cannot be shrinked or enlarged during runtime of the device. Therefore, any logic that would reuse the PUF area has to be within the same area boundaries.

When using the large design of chapter 3 with partial reconfiguration, the possibilities would still be very limited. A PR area always has to be programmed in a whole and cannot be further split. The whole FPGA would have to be defined as PR area. Thus no other PR area would be possible. The pre-defined interfaces limit the possibilities for logic placement and routing within the PR area dramatically. For time critical designs it might be hard to meet timing constraints or to route all IO pins at all. Furthermore, additional area would be consumed due to those limitations of placement and routing, and result in additional area inefficiency.

Fortunately, the PR feature can be used alternatively to overcome these problems and maintain the flexibility of placement and routing, and the PR feature for other logic. The core idea is having different implementations of the same ROs on one CLB. The designs are then sequentially partially reconfigured. This uses the entropy resources of a CLB more efficiently.

4.2 Prior Work

The idea of reconfigurable PUFs has already been proposed in several publications [KSS+09, MKP09, RJA12, KKL+11]. However, the word "reconfigurable" is always used with a different meaning. Kursawe et al. [KSS+09] propose a constant implementation of a PUF, where its internal configuration is changed by, e.g., irradiation with a laser beam. This reconfiguration creates new challenge-response pairs. The logically reconfigurable PUF is introduced by Katzenbeisser et al. [KKL+11], which uses state-dependent input and output transformations to change the challenge-response behavior for the design on each new state. Rührmaier et al. [RJA12] present the erasable PUF. It can alter the behavior of single challenge-response pairs

irreversibly, while keeping all of the others fixed. Majzoobi et al. [MKP09] use the reconfiguration just for an initial device characterization step. The generated hard to invert input and output networks are used to make model-building attacks more difficult.

The contribution of this work is a PUF system design, which uses sequentially loaded PUFs on the same logic block of an FPGA. Each PUF uses the same netlist, but other routing and logic resources. Unlike a reconfiguration of the PUF itself, a new PUF implementation is reconfigured on the same logic blocks of an FPGA.

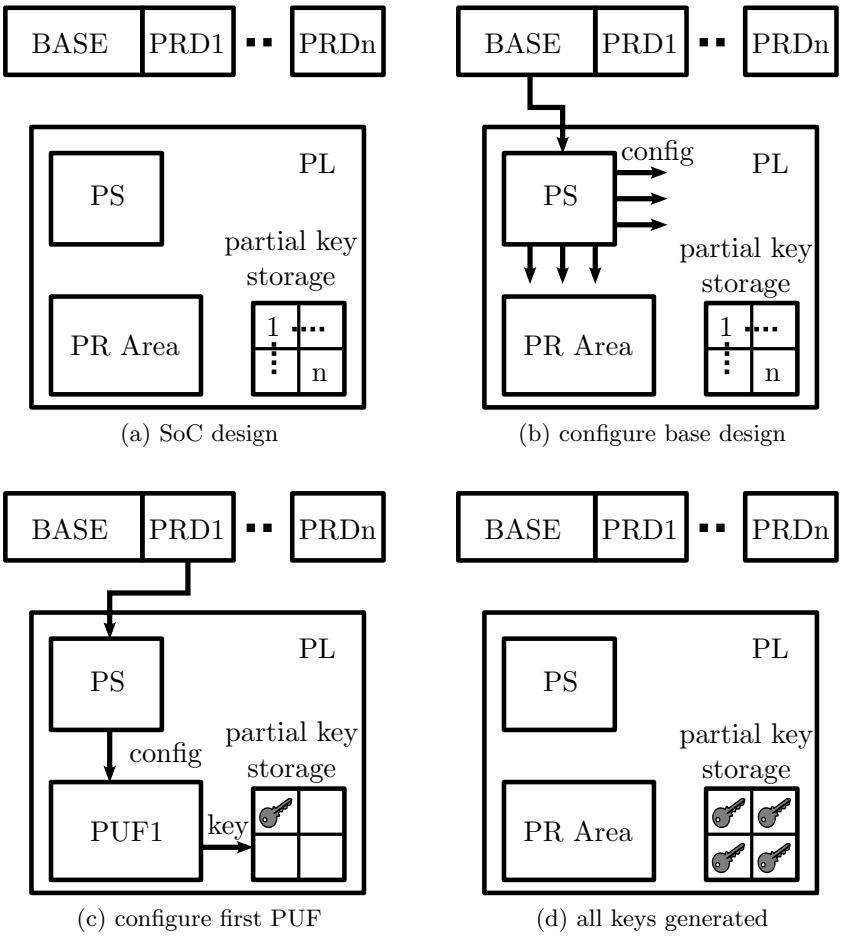
4.3 Reconfigurable PUF

The area usage when implementing a single RO on a CLB is very high, as shown in chapter 3. The solution presented in this chapter uses a smaller 1-inverter RO PUF. Different implementations of the RO PUF are reconfigured as long as enough entropy is left in the CLB. Due to the smaller 1-inverter RO PUF, more variations of implementations are possible.

The proposed system and process of using reconfigurable PUFs is depicted in Figure 4.2. According to Figure 4.2a, the FPGA designs are stored in an external memory. A PR area on the FPGA is defined, as well as a separate partial key storage. The partial key storage can store the results of the n Partial Reconfiguration Designs (PRDs). For reasons of clarity, the resulting bit vector is used synonymously to a key. In reality, a key generation algorithm is needed to convert the unreliable response vector to an actual key [DGSV14].

The PS has access to the external memory and can configure and reconfigure the FPGA. After booting the system, the PS loads the base design from the external memory and configures the FPGA as shown in Figure 4.2b. The PS loads the first partial PUF design from the memory and reconfigures the previously empty PR area with that PUF design, as shown in Figure 4.2c. The PUF design consists of m equally implemented RO pairs. Each RO pair generates a 1 bit result, by comparing their two frequencies. One partial PUF design generates an m bit key, which is stored in the partial key storage. The PUFs are read out; the response vector is generated and stored in the key storage.

After reading out all PUFs and storing the key, the PS reconfigures the PR area with another PUF implementation. Again, an m bit response is generated and stored in the partial key storage. This process can be repeated as long as there is enough entropy in the area left by using different routings and logic resources. After reconfiguring and reading out n PUF implementations, the partial key storage consists of n different m bit wide PUF responses. This process is repeated n times until all partial designs were reconfigured and read out. All of them can now be joined to an $m \cdot n$ bit



(a) SoC design (b) configure base design

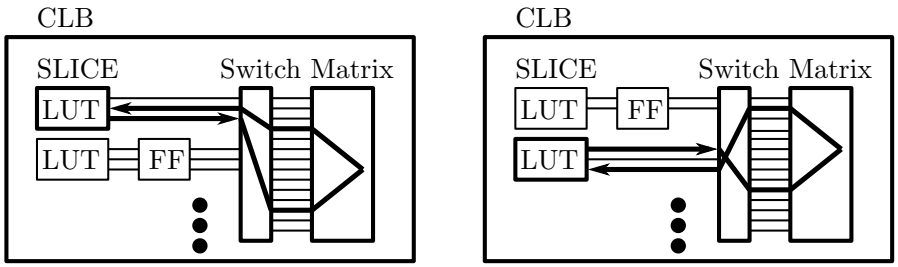
(c) configure first PUF (d) all keys generated

Figure 4.2: Process of using reconfigurable PUFs for key generation on an SoC with an FPGA, base design and partial reconfiguration designs (PRDs) are stored in an external memory.

wide response and be used to generate a cryptographic key. Figure 4.2d shows the filled partial key storage. The generated key can now be used by other parts of the FPGA, e.g., by a trusted platform module (TPM). The key is exclusively generated inside the FPGA and never leaves the FPGA. This increases the security of the system.

4.3.1 Method 1: Using Different LUTs Within the Same CLB

An exemplary reconfiguration is illustrated in Figure 4.3. For the sake of simplicity, only the oscillation ring is depicted, whereas the TFF part is left out. In Figure 4.3a, only the uppermost LUT of one slice is used to imple-



(a) First PUF implementation (b) Second PUF implementation

Figure 4.3: Two possible PUF implementations on the same CLB using different LUTs, showing only the upper slice with one LUT used as a NAND function.

ment the RO. In the case of modern Xilinx FPGAs, this means that seven LUTs remain completely unused. Instead of just using one implementation of the 1-inverter RO, another LUT is used for the second PR design. The second implementation is shown in Figure 4.3b, which uses the same netlist of a 1-inverter RO PUF, but its implementation is distinct in the usage of another LUT and routing. Therefore, the source of entropy is distinct and the frequencies will be different. Due to routing limitations it is not possible to implement these two PUFs at the same time of the same CLB, therefore, we use partial reconfiguration to implement them sequentially. When comparing the frequencies of two ROs, the same implementation has to be used.

It is important to always use different logic and routing resources, otherwise there will be correlation between two PUF implementations and the result will be predictable. When using this method with the Xilinx Zynq, it is possible to implement up to eight different PUFs on one CLB, as there are four LUTs per slice and two slices per CLB. This would already increase the bit response length by a factor of eight.

4.3.2 Method 2: Using Different Input Pins of the Same LUT

After implementing PUFs on every LUT on a CLB, there is still plenty of entropy in the logic block left. Each LUT on a 7-series Xilinx device has six input pins. All of them can be used individually, to implement more partial PUF designs on the same CLB.

Figure 4.4 depicts an exemplary usage of the same LUT for two different 1-inverter RO implementations. The LUT has six input pins, I1 to I6. The first implementation of the RO uses input pin I1. After measuring this RO, a second one can be reconfigured on the same LUT. This time input pin I3 is used for the oscillation ring. By doing this with every input pin, it is possible

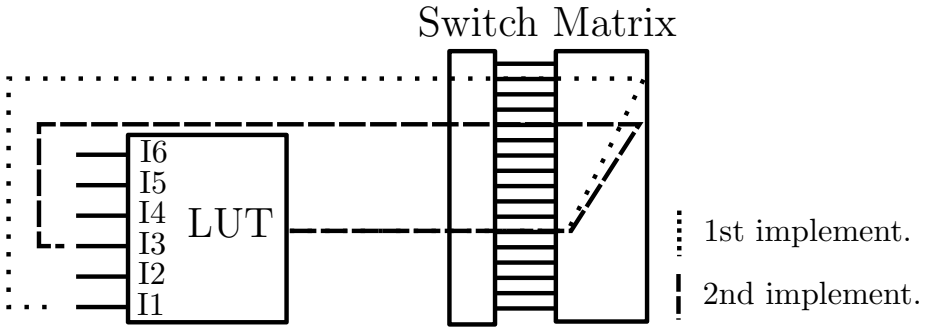
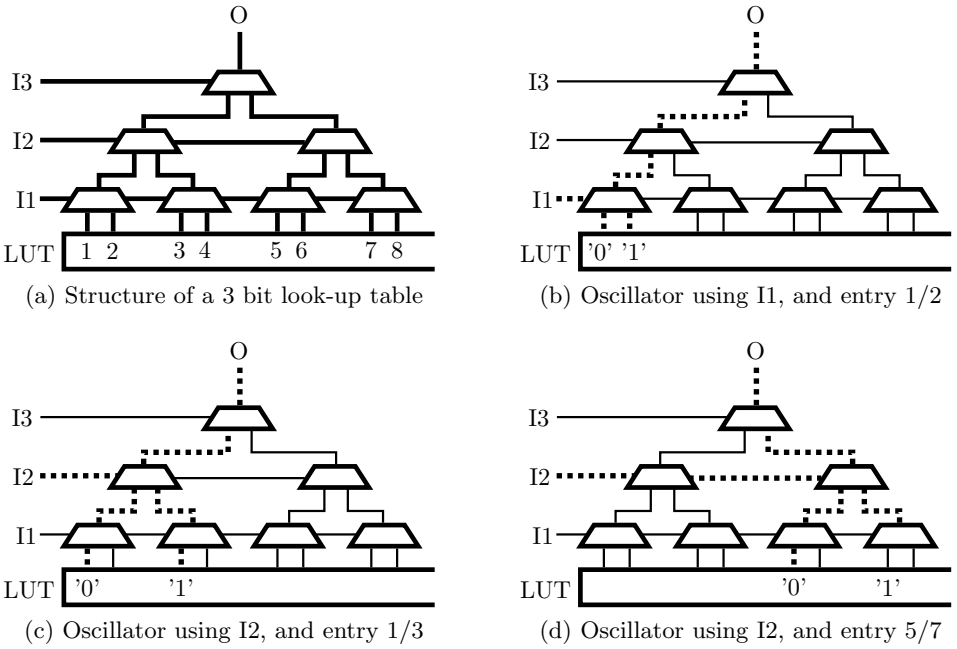


Figure 4.4: Using different pins of a LUT.

to implement six different 1-inverter RO implementations on the same LUT. It can be seen that the routing on the input side is distinct and therefore ensures little correlation. On the other hand, the routing from the output pin of the LUT to the switch matrix remains the same. Therefore, there might be some correlation between the two PUF implementations using the same LUT.

Furthermore, some correlation might be expected inside the LUT. Figure 4.5a shows the basic schematic of a LUT used in an FPGA [Koc13]. For simplicity, a $k = 3$ bit LUT is depicted. The inputs of the first stage multiplexers that are controlled by input I1 are connected to 2^k SRAM configuration cells. Those values are then multiplexed through a tree that is controlled by the input signals. Depending on which input signal is toggling when using it as an oscillator, the delay path length varies. This is important when implementing ring oscillators on LUTs. In this case, only one input pin is constantly toggled; therefore, the closer the input pin multiplexer is to the output signal, the shorter the delay line is and the larger the frequency that is to be expected. As we are already working with high frequencies in the gigahertz range, the usage of pins closer to the SRAM cells and further from the output should always be preferred when possible.

The shared resources and therefore a possible correlation between the PUF outputs can be estimated by analyzing the signal paths through the LUT tree. This concept is depicted in Figure 4.5b to Figure 4.5d. According to Figure 4.5b, the first oscillator is implemented using I1, and the LUT entries 1 and 2. Figures 4.5c and 4.5d show two possibilities to implement another PUF using I2 as the oscillating input. In Figure 4.5c, the LUT entries 1 and 3 are used. Therefore, many logic and routing resources are shared with the first implementation. The better option in this case would be to use LUT entries 5 and 7, as depicted in Figure 4.5d. The implementations would use both the left and the right multiplexer trees. This minimizes their shared resources and uses more sources of entropy.



(a) Structure of a 3 bit look-up table

(b) Oscillator using I1, and entry 1/2

(c) Oscillator using I2, and entry 1/3

(d) Oscillator using I2, and entry 5/7

Figure 4.5: Structure of a 3 bit LUT and different implementations of a RO on this LUT showing the possibilities of resource usage for a greater entropy of reconfigurable PUFs.

4.3.3 Combining Both Methods to Increase the Response Size Per CLB

The previously shown methods can be used together with partial reconfiguration to increase the number of usable RO implementations per CLB. The process of using these different reconfigured designs for key generation is depicted in Figure 4.6. In this example, the first PUF is taken from Figure 4.3a and the second one from Figure 4.3b. One partial PUF design always uses the same RO implementation for all instances. The exemplary design consists of four ring oscillators that have the frequencies $f1$ to $f4$. The exact same design is used on two different SoCs. By comparing the frequencies of neighbored ROs, a response bit $r(f_i, f_{i+1})$ is generated. In this case, both $f1$ and $f2$, as well as $f3$ and $f4$, generate a 1 bit response according to equation 2.12.

By configuring the first PUF on both SoCs, it can be seen that the frequencies are, mostly due to production tolerances, different. The response vector $r_{j,k}$ for the j-th PUF on the k-th SoC can be calculated to:

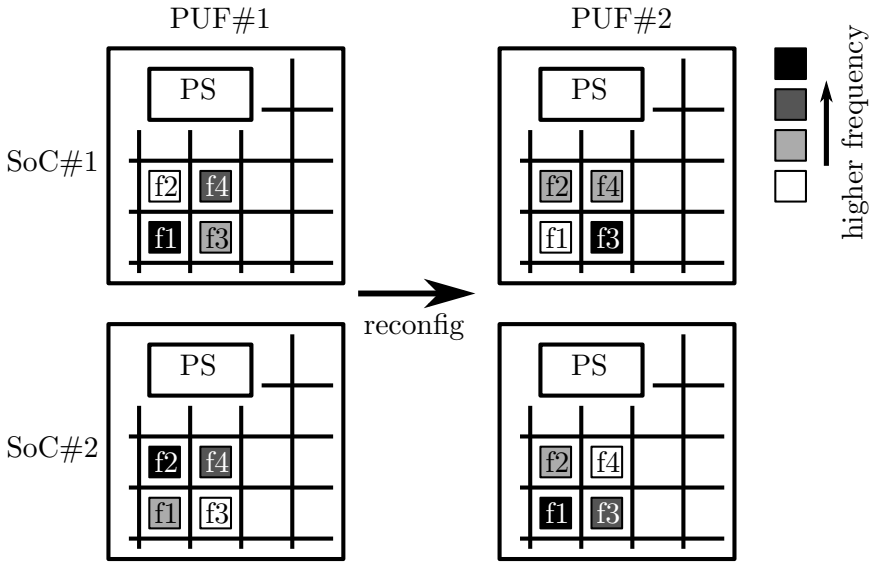


Figure 4.6: Reconfiguration process on two SoCs, showing different frequencies for either the same PUF on two SoCs and different PUF implementations on the same SoC. Darker CLBs indicate higher frequencies of the PUF. In this example, each design creates a 2 bit key, by comparing $f1$ with $f2$, and $f3$ with $f4$.

$$r_{1,1} = 10_b \qquad r_{1,2} = 00_b$$

So far this is the basic concept of using RO PUFs. The contribution of this work is the reconfiguration process that uses the second PUF on both SoCs. As shown in Figure 4.6, the frequencies will change and the response vectors are now:

$$r_{2,1} = 01_b \qquad r_{2,2} = 11_b$$

By making sure that the two different PUF implementations share only few common logic or routing resources, and therefore have little correlation, the responses can be combined. In this case, the response vector for both SoCs can be calculated to:

$$r_{s1} = r_{1,1} \parallel r_{2,1} = 1001_b$$
$$r_{s2} = r_{1,2} \parallel r_{2,2} = 0011_b$$

The response vector is effectively doubled. Only if the implementations have little correlation, the results remain uncorrelated, i.e., depend solely on random production tolerances.

The Xilinx Zynq Z-7020 uses CLBs with eight LUTs and six input pins per LUT. This way it is possible to implement up to $6 \cdot 8 = 48$ unique RO designs. An analysis of possible correlations between these designs will be done in the next subsection and later analyzed using experimental data.

4.3.4 Analysis of Shared Resources Between Implementations

Figure 4.7 shows the physical layout of the CLB used by the Xilinx Zynq Z-7020. This exemplary implementation of a 1-inverter RO PUF uses the bottommost LUT of the lower slice to implement the inverter (B). The routings of the feedback loop and the clock input for the output DFF are highlighted in red. Different points of possible correlations are marked with blue arrows. Arrow *A* highlights possible problems at the switch matrix. Implementing 48 unique RO designs on the same CLB cannot be done without reusing certain routing resources in the switch matrix. The marked node in particular had to be reused by many implementations to make the design routable within the CLB. This can be problematic and cause correlations for all 48 implementations.

Problems that only affect designs that use different input pins of the same LUT are denoted by B and C. While the routing to the input pin of the LUT itself causes no problems, as they are all distinct, possible problems might arise inside the LUT (B). This was already discussed in-depth in Figure 4.5. As the LUT is a 6-to-1 logic function[1], every implementation shares the same output pin routing as well (C).

4.4 Implementation

In this section, an exemplary implementation of the method described in this work is presented. A Xilinx ZC702 evaluation board and a ZedBoard, both featuring the Xilinx Zynq Z-7020, were used for the implementation.

[1]The LUT can also be used as two 5-to-1 functions, but this case is not used here

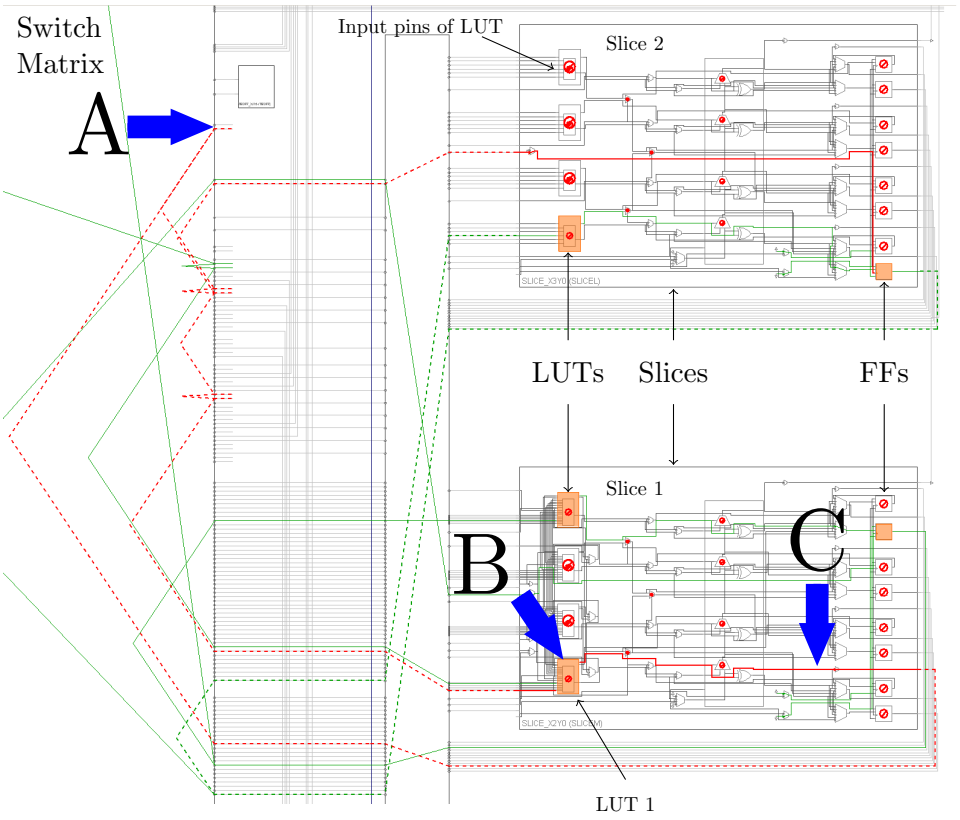


Figure 4.7: Possible shared resources between the different RO implementations within one CLB: A - switch matrix routing of feedback loop, B - routing within the LUT, C - output routing of LUT.

As shown in Figure 4.2a, the SoC was organized with a PR area and a partial key storage. To make sure that each PUF in the same design was implemented identically, the placement of logic and routing was thoroughly constrained. Each implementation was created by using one of the eight LUTs on the CLB and one of the six input pins of the LUT, as shown in Figure 4.3 and Figure 4.4. This way, up to $6 \cdot 8 = 48$ unique RO designs could be implemented. If those designs were not correlated, the response size for the used area could be increased by a factor of 48. The reconfigurable PUF implementations are numbered 1 through 48. According to Table 4.1, each group of six implementations uses the same LUT and consecutively input pin 1 to 6. Therefore, implementation 1 uses input pin 1 of the first LUT, implementation 6 uses input pin 6 of the first LUT, and implementation 48 uses input pin 6 of the eighth LUT of the CLB.

Each of those 48 ROs was then placed 4272 times on the FPGA. Almost all CLBs of the device were used to get a statistically more relevant result. In reality, a far smaller area could be used. Each PUF design itself generated a

Table 4.1: Configuration of 48 reconfigurable PUF implementations according to annotations used in Figure 4.7. Each implementation uses the same CLB but different LUTs within the CLB (see Figure 4.3) and different input pins within the LUT (see Figure 4.4).

		Input pin of LUT					
		1	2	3	4	5	6
LUT of Slice	4	43	44	45	46	47	48
	3	37	38	39	40	41	42
	2	31	32	33	34	35	36
	1	25	26	27	28	29	30
	4	19	20	21	22	23	24
	3	13	14	15	16	17	18
	2	7	8	9	10	11	12
	1	1	2	3	4	5	6

Slice 2

Slice 1

$\frac{4272\,\text{bit}}{2} = 2136\,\text{bit}$ partial key, which was stored in the key storage. By using every of the 48 designs, the key storage had to be $48 \cdot 2136\,\text{bit} = 102\,528\,\text{bit}$ wide.

The width of the counter was extended to 13 bit, as the frequencies of the 1-inverter RO were much higher than the 3-inverter RO.

4.5 Experimental Results

The PUF designs were tested on six Xilinx Zynq ZC702 and four ZedBoards. All measurements were performed in a climate chamber at exactly 25 °C and a constant FPGA core voltage of 1.0 V. The ROs were enabled sequentially. This assured that no surrounding RO could interfere with the measured RO. The raw frequencies were stored by the PS and evaluated with Matlab using equation 2.12. In the analysis of the results, it was important to not only analyze the properties between SoCs, but also between the partial PUF implementations.

4.5.1 Speed

It takes around 44 ms to configure a full FPGA design using the PS [Koh13]. When only configuring a partial design, linearly less time is needed. A full bitstream would reconfigure 6650 CLBs. The reconfiguration time per CLB is therefore roughly $44\,\text{ms}/6650 = 6.62\,\mu\text{s}$. A RO pair consuming two CLBs

would need $2 \cdot 6.62\,\mu s = 13.23\,\mu s$ to be reconfigured. As shown in chapter 3, the time to read out one RO pair was 10.24 µs. The total time that is needed to reconfigure and measure one RO pair can be calculated to:

$$t_{meas} = 2 \cdot 6.62\,\mu s + 10.24\,\mu s = 23.48\,\mu s \tag{4.1}$$

This does not include the reconfiguration of the surrounding logic such as input mux, output mux, and counters. But as shown in chapter 3, these are relatively small compared to the actual ROs. Therefore, a value of 23.5 µs can be used as a rough estimate of the time that was needed to reconfigure and measure one RO pair and generate one PUF output bit. The generation of all 102 528 bit should take $102\,528 \cdot 23.48\,\mu s = 2.4\,s$.

The actual measurement on the SoC took 3.78 s. This included all additional delays caused by, e.g., reconfiguration of additional logic, the PCAP interface, the read out to the PS, and the reading and writing to the SD card. Therefore, a time of $3.78\,s/102\,528 = 36.87\,\mu s$ was needed to reconfigure and measure one RO pair.

4.5.2 Frequencies

Figure 4.8 shows the frequencies of the 48 different RO implementations on one of the tested boards using the configuration shown in Table 4.1. Each bean represents the frequency distribution of all 4272 RO instances. Very high frequencies were measured, due to the fact that only one inverter was used to implement the RO. The frequencies varied from around 600 MHz up to 1.3 GHz depending on the RO implementation. As shown in Figure 4.5 lower input pins were further away from the output pin than higher input pins. Therefore, it can be seen that the first implementations within the same LUT using input pins 1 or 2 tended to have lower frequencies. Additionally, the frequency depends on the routing in the switch matrix. Not every oscillation ring could be directly routed back from the output pin of the LUT to the input pin. The longer routing then resulted in a lower frequency.

Frequencies of around 1.3 GHz as seen in PUF design 29, were the highest that could still offer reliable and stable results. However, frequencies over 1 GHz were theoretically too high for the first TFF, as its data path delay was given to 1 ns by Vivado. These estimations are very conservative, and assume the slowest possible device at the worst operating conditions such as temperature and voltage. Therefore, some ROs over 1 GHz were still working properly. But in fact, while implementing the 48 different designs, a lot of

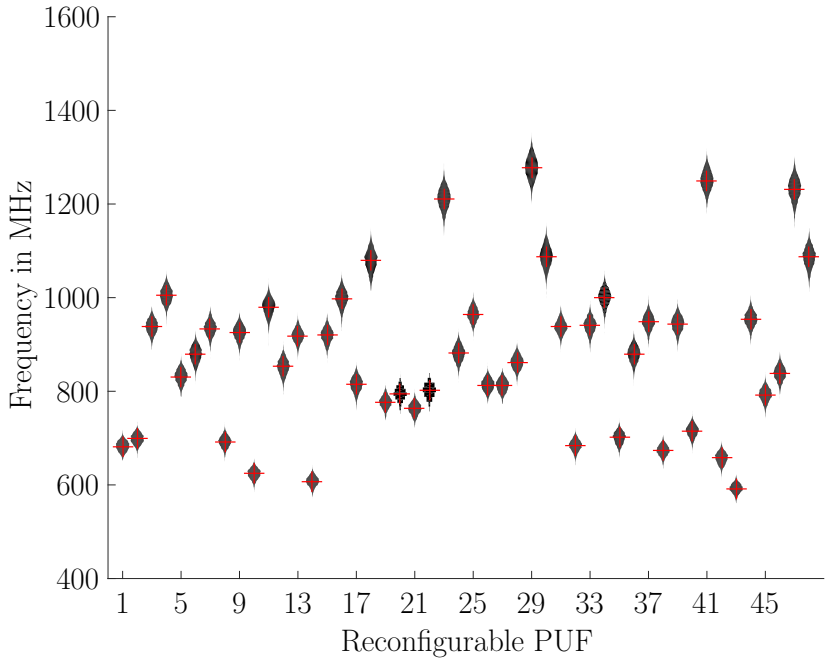


Figure 4.8: Frequency distribution of all 48 RO designs on one board. Each bean is represented by 4272 RO frequencies.

fine tuning had to be done, as some frequencies were too high. This led to ultimately no oscillation at all in some rings, or very unreliable results. As the goal of this work was also to analyze the robustness of PUFs at extreme conditions, ROs over the technical limitations were still tested.

The advantage of the high frequency implementations is that only little routing is shared with other implementations. The lower the frequency, the longer the RO chain, the longer the routing, and therefore, the higher the possibly shared resources. Although the FPGA was operated far out of its specifications with frequencies in the gigahertz range, the ROs and TFFs were still operating reliably. To measure the resulting frequencies reliably, a total of three TFFs was used to lower the maximal frequency to: $1.3\,\mathrm{GHz}/2^3 = 162.5\,\mathrm{MHz}$. Two of these TFFs were placed in the same CLB as the RO to save more area.

4.5.3 Uniformity

Figure 4.9 and Figure 4.10 show the results of the HW for all 2136 bit results of the 48 RO designs on all of the ten tested SoCs.

Figure 4.9 shows the box plot of the fractional HW separated by the SoCs. A boxplot can be a first estimate to make sure that no SoC shows an abnormal behavior. It depicts groups of data through their quartiles. The red line marks the median and 50 % of the data lies within the blue

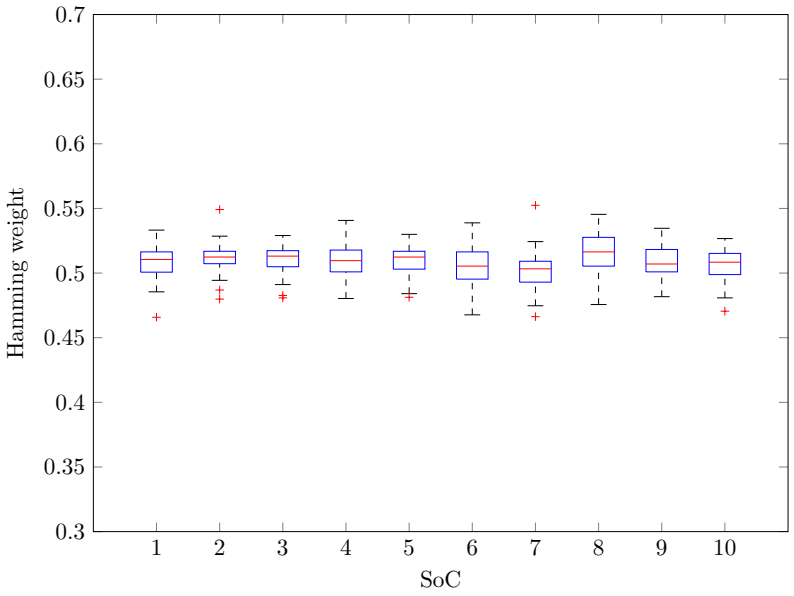


Figure 4.9: Fractional HW of concatenated response vectors on each SoC, mean value over all tested PUFs.

box (25th and 75th percentiles). The two whiskers outside the rectangle correspond to approximately 99.3 % of the data, but are not further away than 1.5 times the distance from the median to either quartile. This makes it possible to spot outliers that are marked by a red cross.

The measurements of 48 partial vectors are represented in each box plot. Every PUF implementation produced a bit vector that is close to the ideal result of 0.5. Some statistical outliers are normal, as the HW of only 2136 bit vectors was plotted. The worst results were found to be 0.55 and 0.46, whereas the mean and median value were always very close to 0.5.

Figure 4.10 shows the box plot of the fractional HW separated by the 48 RO implementations. The box plots are represented by the measurements on the ten different SoCs. This makes it possible to detect bad implementations quickly and improve them. Most of the results were close to the ideal value of 0.5. Some statistical outliers reached a HW of 0.55 and 0.46. The mean and median values on the other hand suggested very good results close to 0.5. No RO implementation resulted in HWs that were far off 0.5.

The uniformity property was therefore given both for all different partial PUF implementations and the concatenated vector on all SoCs. This assured an almost equal distribution of ones and zeros, which is also important for a good HD.

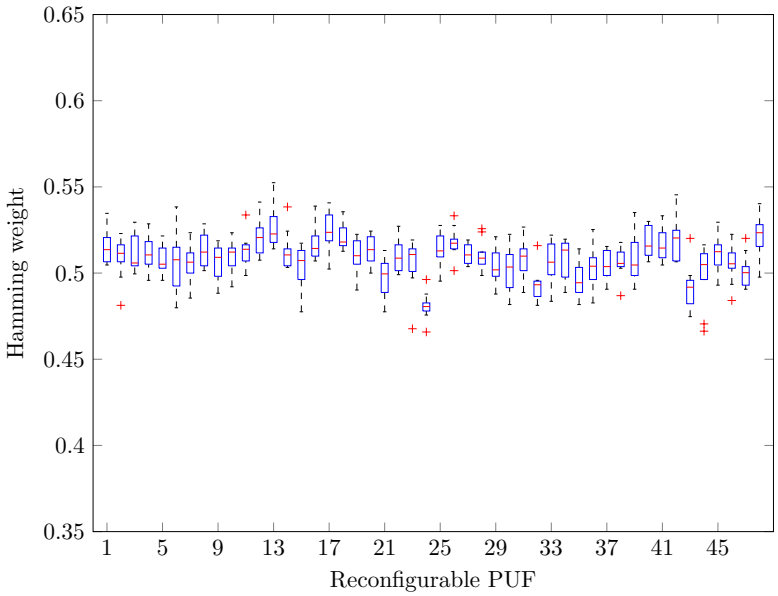


Figure 4.10: Fractional HW of response vector of each PUF, mean value over all tested SoCs.

4.5.4 Bit Alias

Figure 4.11 shows the bit alias of each of the 102 528 bits that are generated by concatenating all 48 RO implementations with 2136 bit each. The alias was then calculated by calculating the mean value of each bit for the measurements on all ten boards. The results are very promising with a mean value of $\mu = 0.509$ and a standard deviation of $\sigma = 0.173$. Both the mean value and the standard deviation are very close to the ideal expected values for a completely random distribution of bits.

4.5.5 Uniqueness

To calculate the inter-device HD between each SoC, the PUF bit vector response of each RO implementation was concatenated to a 102 528 bit wide vector. The fractional HD between each full PUF response of all ten tested SoCs was either 0.48 or 0.49 and thus very close to the ideal results of 0.5. In other words: after concatenating all 48 partial PUF responses, the complete responses could still be used to uniquely distinguish each SoC. Using Equation 2.7 the uniqueness between the SoCs can be calculated to $U_{\text{SoC}} = 0.489$. The important uniqueness property using the inter-device HD was therefore maintained using the method of partial reconfiguration PUFs. The results were still comparable to the ones that were implemented on ASICs by Katzenbeisser et al. [KKR+12].

The inter-device HD should not be used as the only method to guarantee

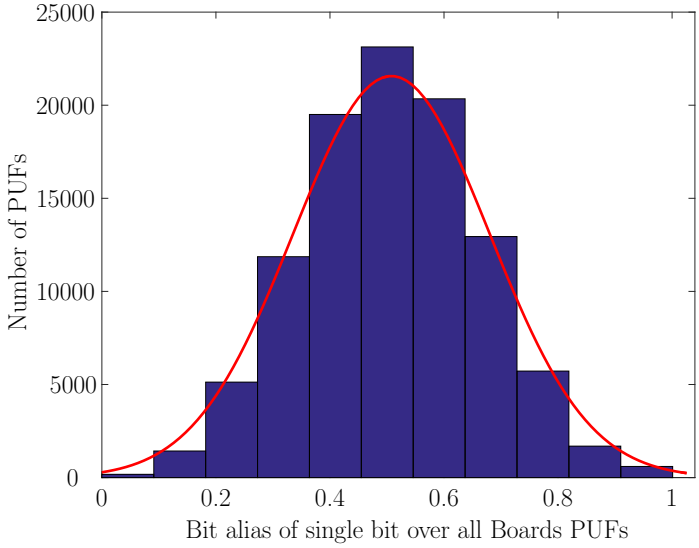


Figure 4.11: Bit alias of all concatenated 2136 bit responses by the 48 RO implementations. The alias was calculated by calculating the mean over the measurements of all ten tested SoCs.

a good uniqueness property. Most of the possible correlation effects when reusing area might stay hidden in the binary representation and simple concatenation of vectors. More dedicated methods have to be used to analyze the ROs for possible correlation effects. A new inter-implementation HD will be used as the HD between different RO implementations. This gives a much deeper insight in possible correlation effects between the different RO implementations that might share logic or routing resources. Every entry on the diagonal equals 0, as it represents the HD of two equal bit vectors.

Figure 4.12 shows the fractional HD between each partial PUF implementation, averaged over all ten SoCs. This plot is important to measure the usability of the method presented in this chapter. Any value too far from 0.5 would suggest a possible correlation between partial RO implementations.

Most of the results are close to the ideal value. As expected, some darker blocks — representing PUFs with lower HD — form around the diagonal. These are RO implementations that only use different pins, but have both the LUT and output routing in common. Therefore, their shared resources are possibly the largest. This leads to a smaller FHD, i.e., they have more than half of their bits in common.

Some blocks are easily recognizable around the diagonal, whereas others are barely visible. This possible correlation could be confirmed by analyzing their implementations and the shared routing and logic resources. This is mainly caused by the effects presented in Figure 4.5, i.e., the routing within

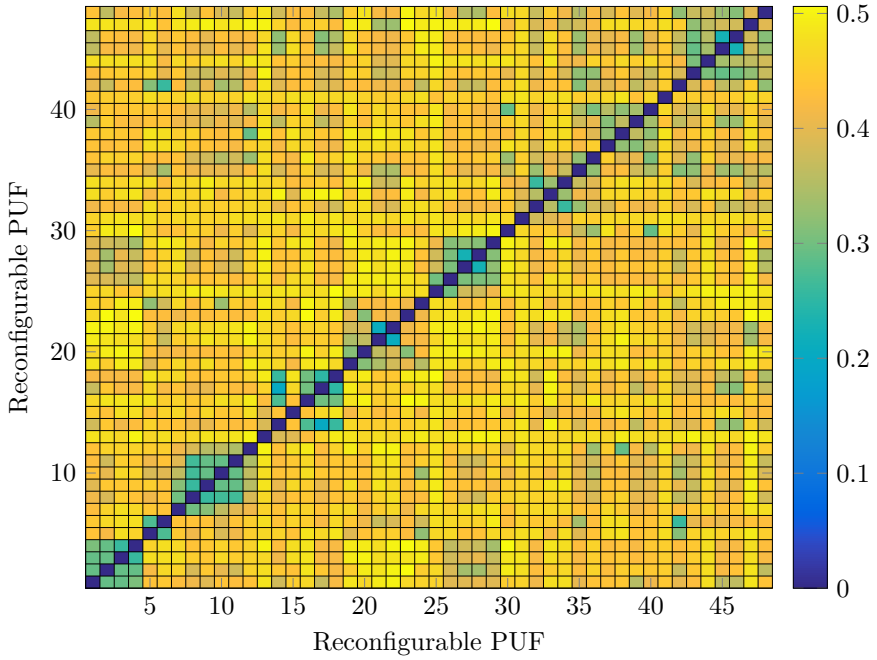


Figure 4.12: Fractional HD between each partial PUF response, mean value over all tested SoCs.

a LUT. Unfortunately, it was impossible to make every RO implementation unique enough to reach an ideal HD of 0.5 for every response. This was mainly due to the fixed output routing of the LUT and limitations of the method shown in Figure 4.5. A very unique implementation was achieved for instance with RO 41, which has an almost perfect HD to every other implementation. This was managed by using a very short routing, which also resulted in a very high frequency as seen in Figure 4.8.

Figure 4.13 shows the histogram of the inter-implementation HD between all partial PUF responses that are plotted in Figure 4.12. It can be seen, that most results lie within a good HD of 0.40 to 0.50. They seem to be uncorrelated to other PUF implementations and can be used to extend the concatenated response bit size. However, as the PUF implementations share some resources, a tendency to produce the same bits between different implementations can be seen by the asymmetry towards a small FHD. Some results are as low as 0.2, which suggest serious correlation between those RO implementations. An estimation of the overall entropy from these results is not trivial. This would be important to quantify the amount of new bits of entropy that are gained by using a PUF that does not have full entropy. The overall uniqueness property for the plots in Figure 4.12 and 4.13 was calculated to $U_{\mathrm{Puf}} = 0.430$.

An easy approach to avoid this problem would be the complete removal

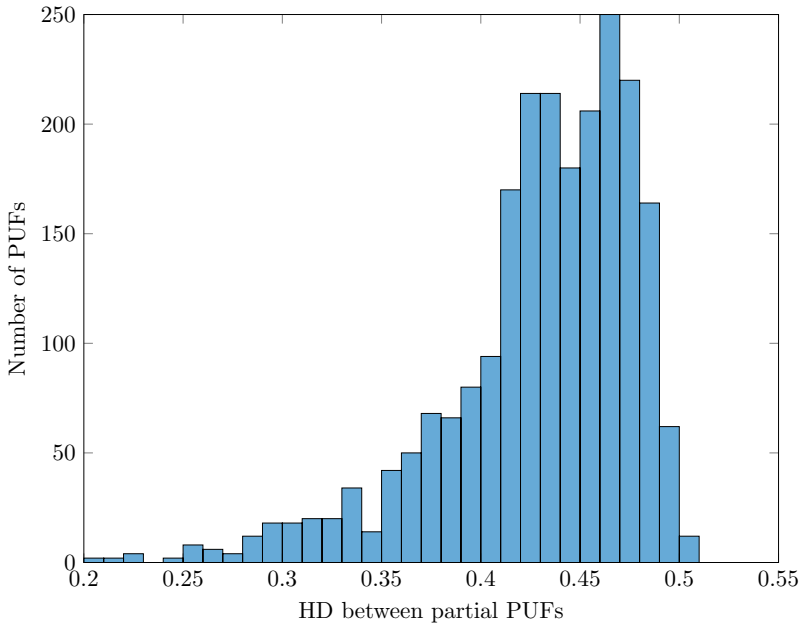


Figure 4.13: Histogram of fractional HD between each partial PUF response.

of the partial PUF responses with a bad HD. But as this also implies losing some entropy, compressing the concatenated response vector sufficiently is the better option.

4.6 Entropy Estimation

Various methods are used in this subsection to estimate a min-entropy for the concatenated responses, including all partial RO implementations. These results can then be used to compress the responses to a high-entropy bit size.

4.6.1 Context-Tree Weighting

Pehl et al. [PPHG14] suggest the use of compression algorithms to estimate the entropy. The usage of Context Tree Weighting (CTW) [WST95] is an approach to estimate an upper bound of the entropy of experimental data [KKR+12]. It offers both theoretical guarantees and good practical performance [BEYY04]. In this work, the CTW compression algorithm by F. Willems [WST95] was used. All 48 individual 2136 bit long partial PUF responses on each of the SoCs were concatenated to a 102 528 bit large bit vector and written in a binary file. To compare the performance of the algorithm, a binary file containing a pseudorandom bit sequence was

generated. The Matlab function *randi* was used with the Pseudorandom Number Generator (PRNG) mrg32k3a.

After compressing both files, the overall file size could not be significantly shrunk. The algorithm needed 8.02 bits per byte to store the pseudorandom bit sequence. As expected, it was not possible to compress the pseudorandom data file and an overhead resulted from storing the tree information. On the other hand, 7.8 bits per byte were needed to store the compressed data containing the PUF responses. This equals a compression ratio of $8/7.8 = 1.03$. Reordering the data brought no improvement in compressibility of the response vectors.

The result suggested a high entropy, but considering the bad HD of some vectors, the algorithm might not find every correlation to compress the data even more. Another problem was the relatively small sample size of PUF responses, which made it harder for the compression algorithm to find correlations.

4.6.2 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical method to simplify datasets by explaining them using a linear combination of linearly uncorrelated variables (principal components) [Jol02]. The first component is defined by having the largest variance and therefore explaining the most variability in the original data. Every following component itself has the largest variance of the remaining components as well, under the constraint that they are orthogonal to their predecessors. The resulting vector set spans an orthogonal basis.

This method has been previously used to analyze the spatial patterns in mean frequency and correlation coefficients [WHP14]. Here, the method is used to detect possible linear dependence between the partial result vectors of different RO implementations.

Matlab offers a very easy way to use PCA with the *pca* function. It returns for an n-by-p matrix, among others, the corresponding coefficients, scores, and an explained vector. The principal component *coefficients* or *loadings*, returned as p-by-p matrix, contain column-wise the coefficients for one principal component. They are sorted in descending order by their explained variance. The principal component *score*, returned as a n-by-p matrix, are the transformed variable values corresponding to a particular data point. The *explained* vector is the percentage of the total variance explained by each principal component. This is important to see, if lower components still have an influence on the final result, or if they can be neglected.

To get more meaningful results, the unmodified RO frequencies were used to calculate the PUF response, instead of using a binary result vector. The 4272 measured frequencies of all ROs were stored in a vector for each of

the 48 implementations. The neighbored frequencies were then subtracted to get a 2136 bit wide vector for each of the 48 implementations. The result was a 48-by-2136 matrix containing the 2136 frequency differences for all 48 implementations. This matrix was then compressed using the PCA, with the 2136 frequency differences as observations and the 48 implementations as features. By using the differential frequency differences, no principal component showed just a mean frequency value, as the mean value should be close to 0.

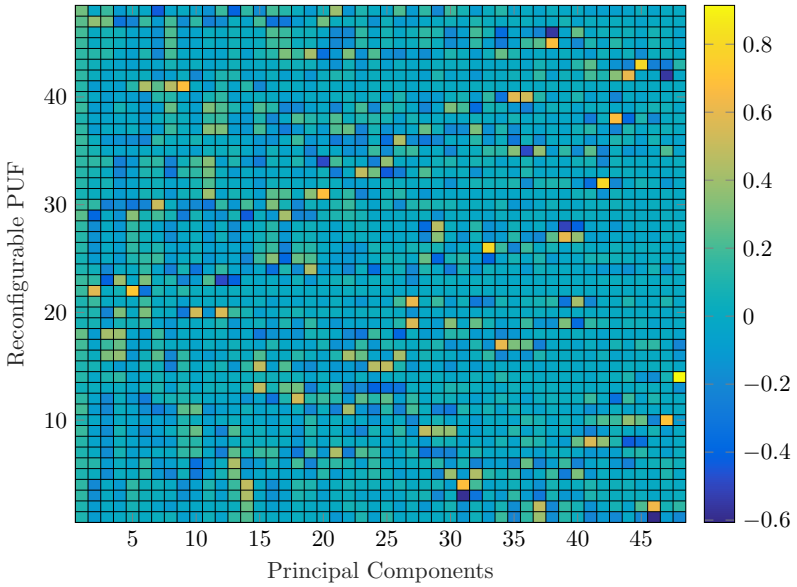


Figure 4.14: Colorplot of the principal component coefficients with the rows containing the influence of the principal components on each RO implementations.

Figure 4.14 shows the resulting 48-by-48 coefficient matrix, containing the influence of each principal component on each RO implementation. It can be seen, that the first principal component had an almost equal influence on all different RO implementations. This could have numerous reasons, such as a slightly biased frequency or a spatial dependence. The following principal components usually had a bigger influence on one or two RO implementations, and almost no influence on the others. This was good for assuming high entropy, as each principal component is almost solely needed to explain the variance of a single RO implementation. No principal component, except for the first one, explained a larger amount of RO implementations, i.e., there was no correlation.

Figure 4.15 shows another visualization of the principal component coefficients. Here, each box contains the influence of one principal components on the 48 RO implementations. The RO implementations are separated in

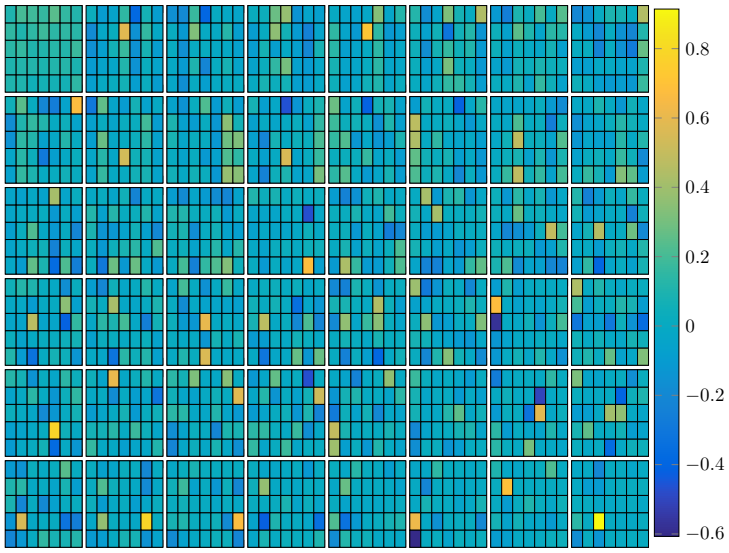


Figure 4.15: Colorplot of the principal component coefficients. Each box contains the influence of one principal component on the 48 RO implementations. The implementations are separated by their use of the eight LUTs (columns) and the six input pins (rows).

each box by their usage of the eight LUTs (columns) and the six input pins (rows). A potential linear dependence of using certain pins and LUTs could be seen by this. Again, it can be seen that the first PC influenced all pins and LUTs equally. Although some boxes show multiple larger influences along the same LUT or pin, no major correlation could be found here either. That means, that no principal component had a influence on multiple LUTs with fixed pins or vice versa.

Listing 4.1 shows the explained vector for each of the 48 principal components in descending order.

Listing 4.1: Explained variance by each of the 48 principal components in descending order

```
21.0724  3.2841  1.5899  0.9000  0.4922  0.2762
8.4100   2.9118  1.4962  0.8383  0.4178  0.2568
7.3911   2.5476  1.4758  0.7716  0.4068  0.2261
5.9685   2.3866  1.2490  0.7034  0.3867  0.2134
5.6104   2.2622  1.1963  0.6258  0.3631  0.1968
4.3027   2.1125  1.1253  0.6004  0.3337  0.1721
3.9845   1.9504  1.0138  0.5736  0.3129  0.1688
3.5728   1.8686  0.9808  0.5567  0.2953  0.1487
```

The first principal component already explained 23.76 % of the variance. A fast convergence of the values towards 0 would be bad for the entropy of the principal components. In that case, a linear combinability would render the lower principal components meaningless. Fortunately, all principal components had an influence on the variance. Equal results were also obtained when using the PCA method with regular PUF results, not using the partial reconfiguration method.

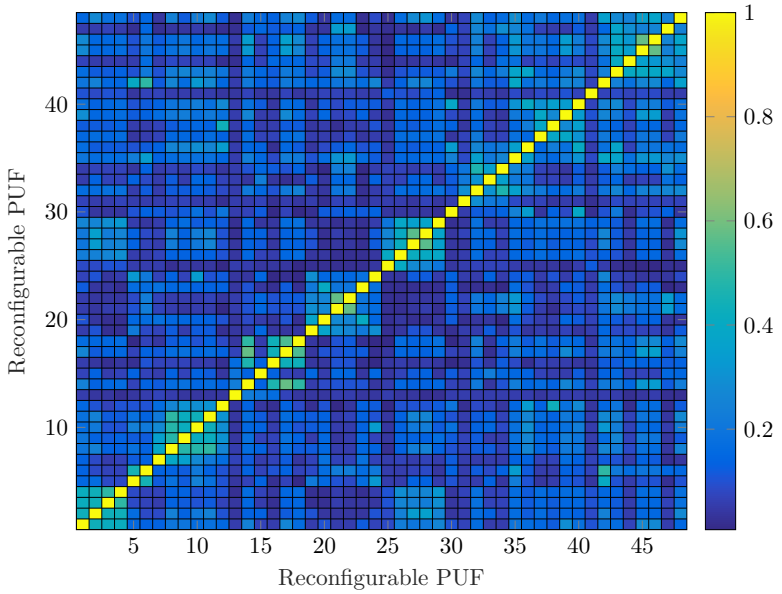


Figure 4.16: Kendall rank correlation coefficient between every reconfigurable PUF implementation for the 2136 measured frequency differences.

Figure 4.16 shows a rank-based statistical method: the Kendall rank correlation coefficient (Kendall's tau) [New02]. In contrast to, e.g., Pearson's correlation, it assesses not only linear, but any monotonic relationship between variables. A value of 1 represents a complete positive or negative correlation between the two variables, whereas 0 stands for the absence of any positive or negative correlation. It is more robust against outliers, that often distort the correlation when using other methods. For more precision, unprocessed frequency differences were used. The boxes around the diagonal are visible, which indicates to the usage of the same LUT for different implementations. Again, RO 41 stood out as a very uncorrelated implementation. Nevertheless, no major correlation could be found here either.

A last possibility is to analyze the principal component scores. Figure 4.17 shows an area plot of the scores. The results for each principal component were mapped on the real local position of the implementation on the FPGA. No spatial dependence of the principal component was visible. Each principal component had a randomly looking influence on the

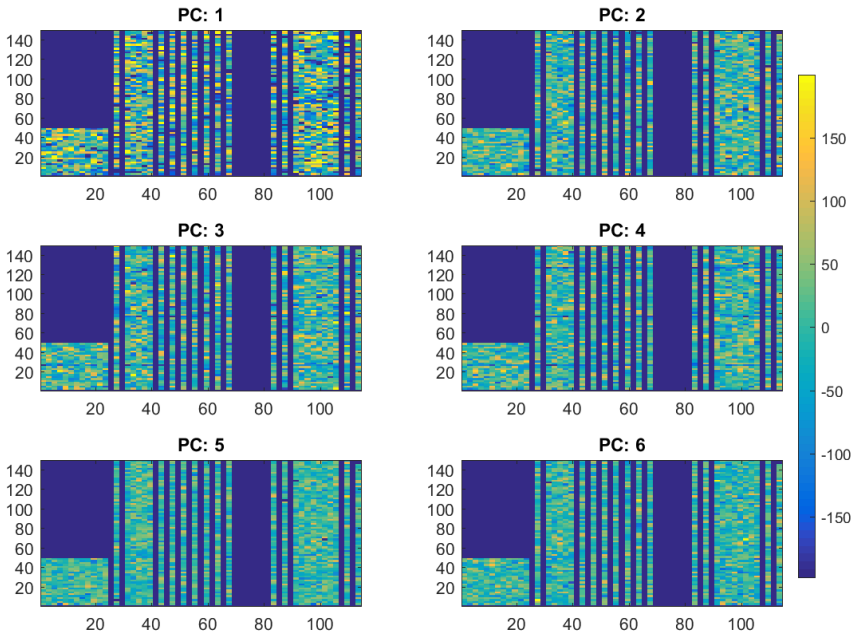


Figure 4.17: Principal component scores for the first six principal components, mapped on the FPGA area.

observations.

As a conclusion, it can be said that the analysis of PCA data is very complicated and not straightforward. Especially, when using a complex dataset as in this experiment, where the frequencies depend on many intrinsic variables. On the other hand, the PCA is a very fast method to detect a linear dependability of data. This does not mean that it is able to detect any statistical dependence, which might be caused by non-linear effects. Fortunately, no major linear dependability between the analyzed RO implementations could be found, which indicates a high entropy.

4.6.3 NIST SP 800-90B Entropy Test

The National Institute of Standards and Technology (NIST) published a recommendation for entropy sources used for random bit generation in SP 800-90B [Bar16] in its second draft. The proposed tests can be used to estimate the min-entropy of an entropy source. In contrast to other Random-Number Generator (RNG) tests, these also work with a much smaller dataset. Most tests base on a principle that make the estimations for smaller data sets more conservative, and they get more precise with a larger data amount. The problem of using RNG tests for PUFs is that usually the available

dataset is relatively small, as PUFs are not constantly producing random values like RNGs.

In accordance to the SP 800-90B guideline, the NIST also maintains a test suite on GitHub[2]. The test suite is split in two groups: IID (independent and identically distributed) and non-IID. As already seen in the previous results, there are correlations in the different RO implementations. Therefore, a non-IID dataset has to be assumed and the *noniid_main.py* was used. The tests are working on binary files with customizable value length. The *noniid_main.py* takes the binary file as input, as well as the bits per symbol. The data is always packed in complete bytes, e.g., 1 bit data require 1 byte in the binary field.

To test the entropy of the RO implementations, a binary file of the resulting bit vector was created. For each board, one binary file containing the 1 bit per symbol result vector was constructed. The results of all 2136 binary results of the 48 implementations were all packed in the same binary file for each of the ten SoCs.

Listing 4.2 shows the results for the analysis of the binary PUF responses. All tests were passed. The test size was correctly evaluated to be 102 528 bit symbols. The min-entropy was estimated to 0.927 bit per 1 bit. This is a very good result and shows that the algorithm was not able to find any major correlation using the binary values. The results of this implementation of the second draft is more conservative than the implementation of the first draft of SP 800-90B, which estimated a min-entropy of 0.934 bit per 1 bit.

Listing 4.2: NIST entropy test results for 2136x48 binary measurements on one of the measured boards.

```
reading 102528 bytes of data
Read in file 34430.bin, 102528 bytes long.
Dataset: 102528 1-bit symbols, 2 symbols in alphabet.
Output symbol values: min = 0, max = 1

Running entropic statistic estimates:
- Most Common Value Estimate: p(max) = 0.513785, min-entropy = 0.960764
- Collision Estimate: p(max) = 0.5, min-entropy = 1
- Markov Estimate: p(max) = 3.89774e-38, min-entropy = 0.970864
- Compression Estimate: p(max) = 0.5, min-entropy = 1
- t-Tuple Estimate: p(max) = 0.526013, min-entropy = 0.926828
- LRS Estimate: p(max) = 0.506003, min-entropy = 0.982781
- MultiMCW Estimate: p(max) = 0.504634, min-entropy = 0.986692
- Lag Estimate: p(max) = 0.510377, min-entropy = 0.970366
- MultiMMC Estimate: p(max) = 0.51243, min-entropy = 0.964574
- LZ78Y Estimate: p(max) = 0.511656, min-entropy = 0.966755
```

The listing also shows the min-entropy estimation for every test. The test that led to the smallest min-entropy was the t-tuple estimate. This method analyzes the frequency of t-tuples (pairs, triples, etc.) in the input data

[2]available at: `https://github.com/usnistgov/SP800-90B_EntropyAssessment`

and estimates an entropy based on their occurrence. These tuples can also overlap. The relatively low min-entropy of this test showed the correlation of the PUF data, that likely occurred due to the reuse of logic and routing. As multiple bits were generated using the same CLB, some patterns might have been created. However, the min-entropy of 0.927 bit is still a very good result, considering the PUF size has been increased by a factor of 48. The collision and compression estimate both showed that each binary value had an equal probability with $p_{max} = 0.5$.

The min-entropy of 0.927 bit per 1 bit input is taken as a conservative guess of the min-entropy for the whole system. By assuming this value, later in post-processing the PUF results, the bit vector size has to be compressed by at least $1 - 0.927 = 7.3\,\%$ to assume a complete entropy for the resulting cryptographic key. In other words, when using the big design used in this experiment, of the 102 528 bit, the cryptographic key could use $102\,528\,\text{bit} \cdot 0.927 = 95\,043\,\text{bit}$ with full entropy.

4.7 Conclusion

A new method to use the area of an FPGA more efficiently for PUFs was presented in this chapter. Instead of using just one PUF implementation, multiple implementations with different logic and routing resource usage were loaded on the same logic block using partial reconfiguration. This way the huge resource consumption of PUFs on FPGAs was compensated by extracting more entropy from the complex logic and routing resources. The presented method was implemented by using 48 different 1-inverter RO PUF implementations using all available LUTs and pins in a CLB. A total of 4272 RO instances were used per implementation and tested on ten SoCs. By concatenating all partial PUF responses, the bit vector size could be increased by a factor of 48. This equaled an area shrinkage of almost 98% to get the same bit size as the straightforward static method.

The HW was always close to the ideal value of 0.5. The inter-device HD between the concatenated bit vectors showed that they can be used as a key source, as all of them had a HD close to 0.5. The inter-implementation HD between the partial RO designs was — as expected — more correlated, but still close to the ideal value of 0.5. The analysis of the bit vectors using CTW showed a compression ratio of only 1.03. The algorithm could not find any major correlation. The PCA was used to find linear dependence. Using this method no major dependability could be found either. When comparing the results to regular RO results, no big difference could be found.

The most conservative method of estimating the min-entropy was using the NIST SP 800-90B test. The min-entropy was found to be 0.93 bit per bit of input data. The entropy of the reconfigurable PUF is therefore rated with 93 %.

CHAPTER 5

Reliability Analysis of PUFs on FPGAs

As ROs are not always run under controlled conditions, but a real world environment, their frequency is affected by physical influences such as temperature, voltage, and Electromagnetic Interference (EMI) from neighbored logic or devices. By operating a device, it also becomes susceptible to aging. The aging process influences the frequency and therefore the PUF behavior as well. The influence of both reversible and irreversible variations were examined in this chapter. Long-term aging, voltage, and temperature experiments have been conducted on real devices and the influence on the reliability of the PUF constructions was analyzed.

This chapter is organized as follows: a short introduction is given in section 5.1. The relevant prior work is presented in section 5.2. An extensive aging examination as the main contribution of this chapter is then presented in section 5.3, followed by the analysis of temperature and voltage influences in section 5.4. A conclusion is given in section 5.5.

First results of this chapter have been released at the ReConFig 2015 [GLS15].

5.1 Introduction

The stability of a generated cryptographic key is crucial for the usability of PUFs. Both reversible and irreversible variations can alter the behavior of the ROs. Environmental conditions, such as voltage and temperature, are reversible variations. The effect of the altered conditions disappear mostly once the cause is withdrawn. Device aging on the other hand is an irreversible variation that leads to a permanently changed behavior of the device.

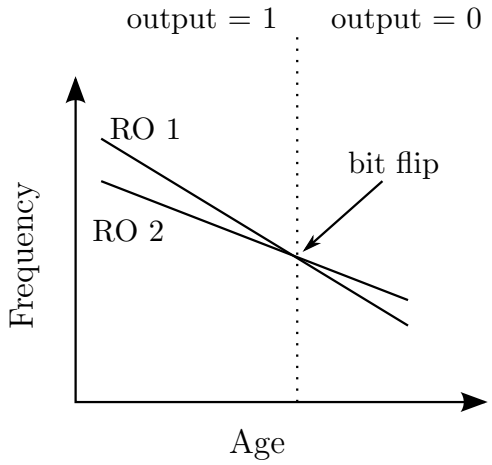


Figure 5.1: Bit flip due to the aging of two ROs.

Measuring the same PUFs on the same device should always yield an intra-device HD close to zero. Figure 5.1 shows a possible effect that can occur due to the aging of a device. In the beginning, the first RO is faster than the second one, thus producing a '1' output. But as the first PUF is aging faster, it becomes slower than the second one throughout the lifetime of the device. This leads to a bit flip and the intra-chip HD rises. The occurrence of this effect is mostly inevitable, but has to be estimated. Any negative effect on the PUF-based key generation has to be prevented.

The effects of both reversible and irreversible variations on the RO frequency and the PUF output will be presented and confirmed with experimental results. The analysis is very important for the usability of RO-based PUFs in real-world environments during a complete device lifetime.

5.2 Prior Work

Much work has been done on the analysis of temperature and voltage influences on PUFs. Almost every paper that presents a PUF implementation on ASICs or FPGAs also includes an analysis of robustness against temperature and voltage fluctuations. Aging on the other hand, has not been covered widely in current publications. No long term evaluation was found.

Katzenbeisser et al. [KKR+12] implemented many different custom PUF designs on ASICs and tested their robustness against variations of supply voltage and temperature.

Stott et al. [SWC10] analyzed the effect of aging on FPGAs. They did both experimental measurements and theoretical simulations. The FPGA was stressed under different electrical conditions. The accelerated life conditions were very high. An overvoltage of almost 80% above the nominal

voltage might lead to other effects than the regular aging mechanisms. Unfortunately, the tests were not PUF specific.

Maes et al. [MRV+12] carried out aging tests on several PUFs that were implemented on their ASIC. The analysis included many different PUF types, but details were missing. Only the fractional HD as a measurement of stability was given in this work.

A more detailed analysis was carried out by Maiti et al. [MS14]. Different FPGAs were tested under various higher voltages and temperatures. The electrical stress was induced by the PUF itself. Therefore, only high frequency stress was applied and no other stress condition was tested. The elevated stress conditions were very high with an overvoltage of 66.7 %.

This work contributes the testing of RO PUFs of various lengths under different elevated aging conditions. This is the first work that was being done with modern 28 nm devices. The FPGA was stressed under five distinct electrical stress types to test their influence on the RO frequency. Additionally, the effects of aging were compared to the ones of reversible environmental changes.

5.3 Aging

In this section, the effects of device aging on the frequency of ROs and the stability of PUF outputs are described. Long term experiments were made to prove the assumptions and describe the severity of aging on PUF stability. Experiments were conducted during a period of over six months. Different types of stress and different types of PUFs were analyzed.

5.3.1 Aging Mechanisms

Four main types of degradation are relevant for modern ICs [WH11]:

1. Hot Carrier Injection (HCI)
2. Negative-Bias Temperature Instability (NBTI)
3. Time-Dependent Dielectric Breakdown (TDDB)
4. Electro-migration (EM)

HCI is the effect that a carrier, which gains sufficient energy, overcomes a potential barrier and is trapped in the oxide layer. A schematic of this principle is shown in Figure 5.3. This results in an alteration of the transistor characteristics typically increasing the threshold voltage. HCI is amplified by a high frequency switching behavior. Hot carriers lead to impact ionization. Hot electrons are injected in the dielectric as gate current. Some

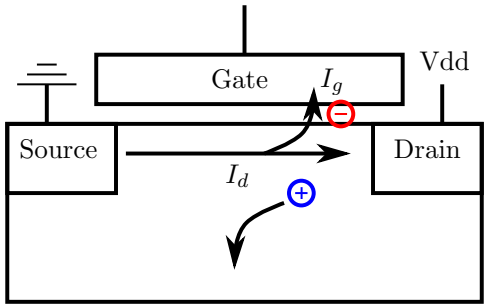


Figure 5.2: HCI effect in a CMOS transistor. A carrier gains sufficient energy to overcome the potential barrier and gets trapped in the oxide layer.

charge carriers can become trapped and change the switching behavior permanently.

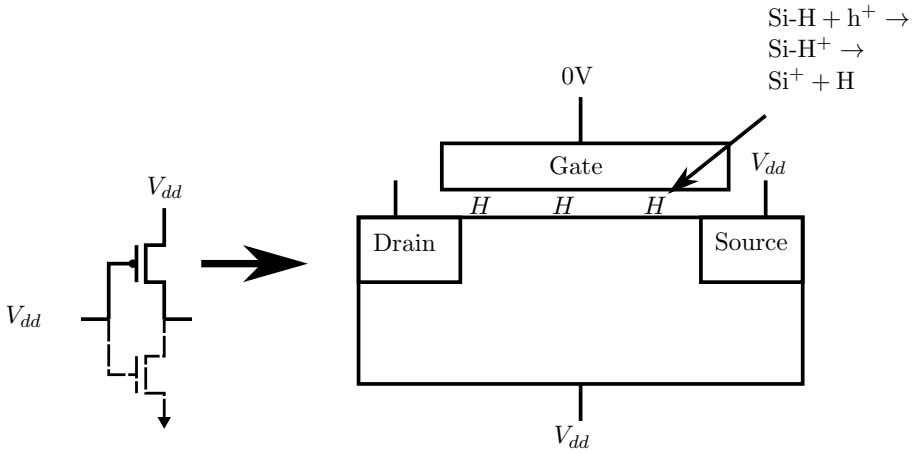


Figure 5.3: NBTI effect in a PMOS transistor. As the holes (h^+) in the inversion layer interact with the Si-H bonds, the electric field breaks the Si-H bonds. H migrates into the substrate, and the remaining dangling bond Si causes a threshold voltage degradation.

NBTI is a static mechanism. As shown in Figure 5.3, dangling bonds at the interface of the channel and oxide layer develop by applying an electric field across the gate oxide. PMOS transistors are mainly susceptible to this type of stress. The impact on the threshold voltage V_{th} is usually higher than with HCI. The equivalent mechanism for NMOS transistors, Positive-Bias Temperature Instability (PBTI), is becoming a bigger concern with the introduction of high-k metal gates. By applying a negative bias at the gate oxide of a PMOS transistor, holes become the majority carriers. Inversion hole induced breaking of Si-H bonds at the Si/SiO2 interface occurs [Ala03].

The generated hydrogen is diffused into the oxide and captured. This effect ultimately decreases I_d and shifts V_{th}.

An additional threshold voltage degradation is caused by the filling of preexisting traps in the dielectric with holes coming from the channel. Removal of stress voltage, however, can empty the traps and recover the threshold voltage degradation [BDBRR12, Ala03].

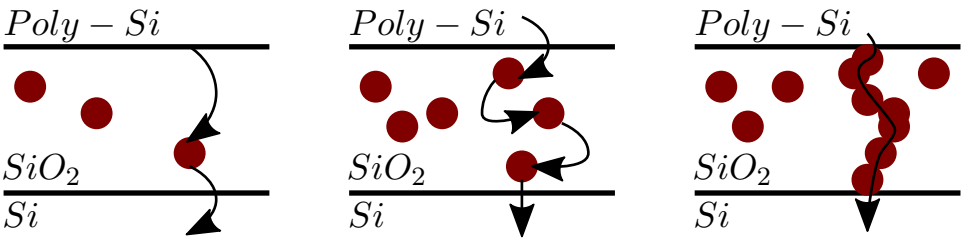


Figure 5.4: Schematic of TDDB effect. An accumulation of trapped charges across the gate oxide form a conductive path.

TDDB is a failure mechanism which is, as shown in Figure 5.4, caused by an accumulation of trapped charges or defects across the gate oxide, while an electric field is applied. A conductive path through the dielectric forms, increases the leakage over time, and finally might even prevent the transistor from switching at all. TDDB is a static mechanism that is only active for PMOS transistors under negative bias and NMOS transistors under positive bias [BDBRR12, Pan09].

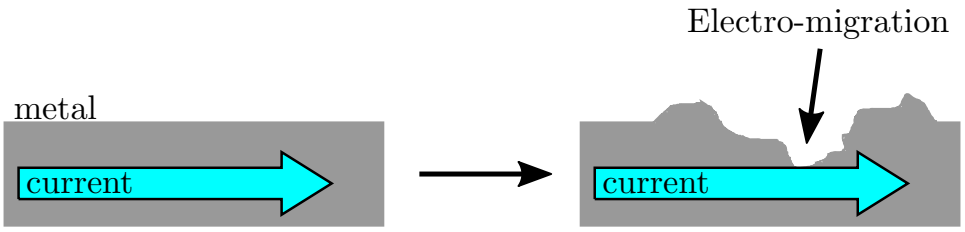


Figure 5.5: Effect of EM in microelectronics. Material is transported by gradual movements of ions. The device behavior can be changed and ultimately lead to failure.

EM is sketched in Figure 5.5. It is an effect in which metal ions migrate over time. High current flows, as present in modern nanoelectronics, can lead to faulty interconnects and ultimately to a device failure. EM is accelerated by a DC behavior, i.e, a low switching frequency, and found to be partly self-healing under frequencies above 10 Hz [Lie13, ZWF+15, TVG+13, TCH93].

5.3.2 Stress Design

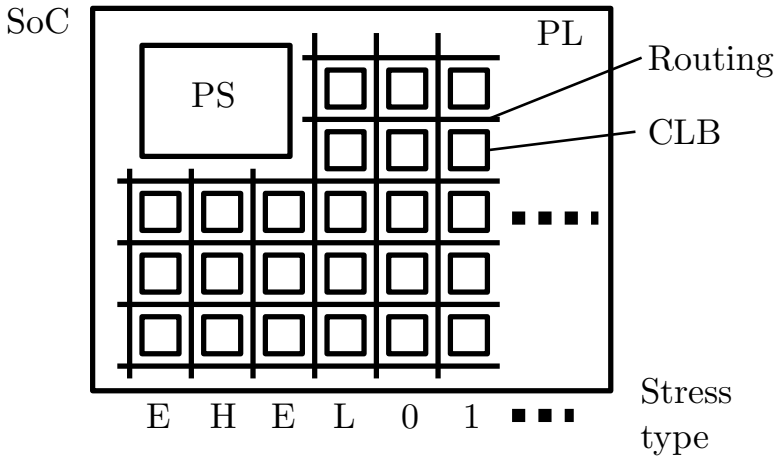


Figure 5.6: SoC with PS and PL part, each column of CLBs was stressed under different conditions, E: empty CLBs ('no stress'), H: high frequency stress, L: low frequency stress, 0: DC 0 stress, 1: DC 1 stress.

In this work an SoC consisting of a PS and a PL was used. The general structure of the SoC was shown in Figure 2.12. In order to age the CLBs, on which the ROs are implemented, in various ways, different electrical stress types were used. As shown in Figure 5.6, the SoC was separated into columns, each of which was stressed with one of five stress types. By having the same type of stress at various positions on the FPGA, it was also possible to measure spatial dependence of the aging.

Five different types of electrical stress were used: *DC 0* stress (constant zero), *DC 1* stress, *low* frequency stress, *high* frequency stress, and *'no'* stress. To lower the impact of the *high* frequency stress on the other stress types, they were surrounded by *'no'* stress columns. After that followed a *low* frequency, a *DC 0* stress, and a *DC 1* stress zone, as shown in Figure 5.6. This pattern was repeated on the whole FPGA. Each CLB in a column was stressed with the same electrical stress.

To understand how the stress designs work, it is important to be aware of the rough structure of a LUT. In Figure 2.13, the implementation of a LUT using NMOS pass transistors was shown. However, the exact implementation of the LUTs are being kept secret by the vendors. This makes modeling a device aging or even explaining the aging effects on FPGAs in detail very hard. It is not possible to directly control the CMOS transistors, which would be needed to induce controlled aging effects, such as NBTI or HCI. Instead of modeling the aging of the LUT, real measurements were performed in this work.

As explained in subsection 2.4.2, the LUT values are stored in an SRAM

and propagated through a mux network that is controlled by the inputs I0 to I5. The frequency measurements shown in Figure 4.8 confirmed that the input pin I0 is closest to the SRAM cells and farthest from the output of the LUT.

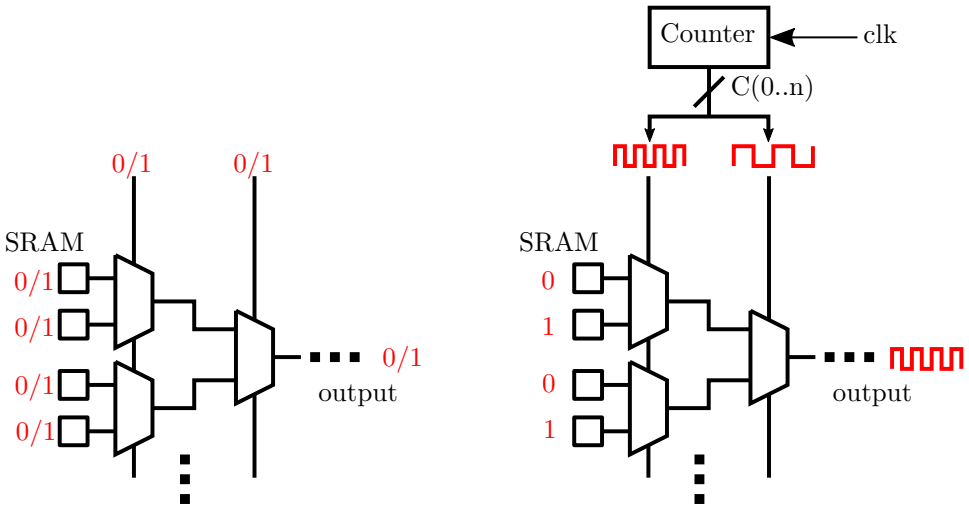


Figure 5.7: DC 0 and DC 1 stress and low and high frequency stress designs on LUT level.

The various stress designs used the SRAM entries and the input signals in distinct ways. Figure 5.7 shows the usage of SRAM entries and input signals for three different stress types. On the left side, the *DC 0* and *DC 1* stress is sketched. In case of DC 0 stress, all SRAM entries as well as all input pins were hardwired to ground. The constant SRAM values were propagated all the way through the LUT. Therefore, the output of this LUT was 0 as well. This however did not ensure that all internal signal lines were also fixed to 0. Due to the inverter sketched in Figure 2.13, some of the NMOS transistors still had an inverted signal applied at their gate. The *DC 1* stress was designed accordingly with the only difference, that all SRAM entries and input pins were hardwired to a high signal level.

The design of the frequency stress is sketched in Figure 5.7 on the right side. The SRAM contained alternating static entries (0,1,0,1...), such that every inversion of the I0 signal led to a bit flip at the output of the LUT. A 6 bit counter was connected to the input signals I0 to I5, where I0 was connected to the least significant bit. Every time the counter was incremented, the output bit of the LUT was changed. Each multiplexer stage toggled its tree with half the frequency of the previous stage. This way, every signal path within the LUT was actually used during the aging process. Different input frequencies *clk*, namely a low and high frequency, can then be used to simulate different types of low and high frequency stress.

Table 5.1: Effects of different stress types on the frequency of ROs.

Stress type	Dynamic effect	Static effect
DC 0	-	strong
DC 1	-	strong
Low frequency	moderate	moderate
High frequency	strong	weak
No stress	-	strong

An additional stress type was the *'no'* stress design. This represented an unused LUT, that was initialized by the synthesis tool. When using Xilinx FPGAs, this usually leads to the same result as setting all SRAM values to DC 0 by hand [SFKP15].

The flip-flops as the memory part of the slices that followed the LUT stage were all equally stressed according to their stress type, i.e., constant stress for DC 0 and DC 1, as well as frequency stress.

The various electrical stress designs enhanced different types of aging effects. Table 5.1 shows the stress types and their degradation influence. As NBTI is still the dominant effect in modern ICs [ZKN+06], a *DC 0* stress, which enables more PMOS transistors, should have more influence on the aging than a *DC 1* stress. *DC 1* stress on the other hand should have a bigger PBTI effect. Both constant effects should show only little HCI effects, as this effect is accelerated by a high frequency. The *high* frequency stress should reveal by far the biggest HCI effect. The *low* frequency stress balances between the dynamic HCI effect as it is still oscillating and the static NBTI effect as it still spends sufficient time in stable signal levels, i.e., in saturation. The impact of the *'no'* stress design was unknown, as it was initialized by the synthesis tool, but should be comparable to the *DC 0* stress.

5.3.3 Accelerated Environmental Conditions

In order to accelerate the degradation of the IC significantly, elevated environmental conditions were used. The two easiest controllable environmental influences are voltage and temperature, which were therefore used in this work to accelerate the aging. In contrast to previous papers [SWC10, MS14] the accelerated conditions were kept within the absolute maximum ratings of the vendor. Operating an IC far out of its specifications could lead to other than the desired aging effects.

The acceleration factor AF_T caused by a higher temperature [Alt16, WH11] can be calculated to:

$$AF_T = e^{\frac{E_a}{k}(\frac{1}{T_{op}} - \frac{1}{T_{stress}})} \tag{5.1}$$

with T_{op} as the normal operation junction temperature, T_{stress} as the stress condition temperature, $E_a = 0.7$ the activation energy, and $k = 8.62 \cdot 10^{-5} eV/K$ the Boltzmann constant.

The acceleration factor due to higher voltage AF_V can be calculated to:

$$AF_V = e^{(\gamma(V_{stress} - V_{op}))} \tag{5.2}$$

with V_{op} as the normal operation voltage, V_{stress} as the elevated voltage, and $\gamma = 2.0$ as the voltage exponent factor [Alt16]. The product of AF_T and AF_V yields the total aging factor.

These formulas are based on the Joint Electron Device Engineering Council (JEDEC) models for semiconductor devices [JED16]. The device specific acceleration factors were taken from similar Altera devices [Alt16], as Xilinx does not have this information publicly available. The acceleration factors are based on modeling and experimental data. Both voltage and temperature have similar effects on HCI and NBTI/PBTI [JED16].

5.3.4 Implementation and Setup

Four 28 nm Xilinx Zynq XC7Z020 were aged in this work. The absolute maximum voltage for the PL part of this SoC was given as 1.1 V, which equals an over-voltage of 10 %. The maximum junction temperature was given as 125 °C [Xil13].

Two ZedBoards containing the XC7Z020 were aged by only using an elevated ambient temperature of 60 °C, which was found to be the maximum ambient temperature under which the board was still working under stress. As the voltage was not modified for theses boards, these aging conditions will be referred to as 'low' stress. The junction temperature under stress was measured to $T_{stress} = 85\,°C = 358\,K$ and the normal junction temperature to $T_{op} = 55\,°C = 328\,K$. This yielded an acceleration factor of:

$$AF_{low} = AF_{T_{low}} = e^{\frac{E_a}{k}(\frac{1}{328\,\mathrm{K}} - \frac{1}{358\,\mathrm{K}})} = 7.96 \tag{5.3}$$

The devices were aged for a test duration of 230 days with several breaks to test the effects of recovery. The effective accelerated aging duration was calculated to:

$$230\,\mathrm{d} \cdot 7.96 = 5.01\,\mathrm{a}. \tag{5.4}$$

As the Xilinx ZC702 allows an easy manipulation of the device voltages, they were used to evaluate very high stress conditions. Two of these devices were aged under the maximum operating conditions of $T_{stress} = 125\,^{\circ}\mathrm{C} = 398\,\mathrm{K}$ and $V_{Stress} = 1.1\,\mathrm{V}$. This yielded an acceleration factor of:

$$AF_{T_{high}} = e^{\frac{0.7}{k}(\frac{1}{328} - \frac{1}{398})} = 77.82 \tag{5.5}$$

$$AF_{V_{high}} = e^{(\gamma(1.1-1))} = 1.22 \tag{5.6}$$

$$AF_{high} = AF_{T_{high}} \cdot AF_{V_{high}} = 95.05 \tag{5.7}$$

For a test duration of 230 days, the effective accelerated aging duration was calculated to:

$$230\,\mathrm{d} \cdot 95.05 = 59.85\,\mathrm{a}. \tag{5.8}$$

The sequence in which the devices were stressed and the PUFs measured was separated in six steps. Figure 5.8 shows a sketch of the used stress cycle. At first a stress design was loaded and the ambient temperature (and voltage if applicable) were elevated. This was kept stable for a six hour stress duration cycle. Afterwards, normal conditions of 1.0 V and 25 °C ambient temperature were applied and blank FPGA design was loaded by the PS.

As the climate chamber and FPGA were still heated up, it was necessary to wait until the device cooled down. Afterwards, the RO frequencies were measured and stored by the PS.

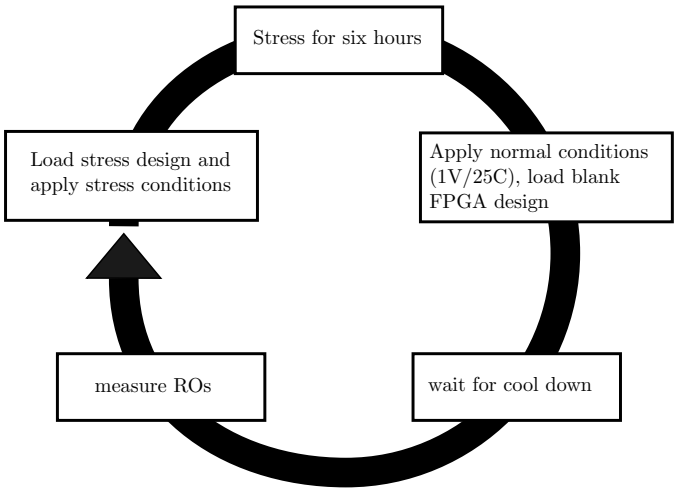


Figure 5.8: Stress cycle used for aging an FPGA

This cycle was performed continuously for 160 days, including two single day breaks to test the effect of regeneration. After 160 days of measurement, the experiment was paused for 100 days to see if the FPGA showed signs of long-term recovery. The FPGA was not used in this period. After that period, the stress experiments were continued normally for another 70 days.

The ambient temperature was controlled by a Heraeus Voetsch VMT 04/16 climate chamber and stayed at the same levels throughout the whole test. The test setup is shown in Figure 5.9. The supervision of the test was performed by the PS part of the SoC. This included programming the FPGA, performing the measurement of the PUFs, storing the test results, changing the voltage of the FPGA, and monitoring the test.

RO PUFs with different amounts of inverters were investigated in this work. RO chains with one, three, and five inverters were implemented (see chapter 3 and 4). This gave an insight whether the length of the RO was important for its reliability. A higher length would lead to a lower frequency oscillation. The PUFs were thoroughly constrained and placed on the device. This guaranteed that each PUF implementation on the FPGA was exactly the same and only the production tolerances were the reason for frequency differences. Each PUF type was placed on every of the 6650 available CLBs on the FPGA. This was achieved by using multiple designs for the same type of PUF and testing them sequentially. As two ROs generate a one bit output, a 3325 bit long bit vector per PUF type was calculated. While measuring the frequencies, only one RO was activated at a time to avoid influences on neighbored ROs. During stress operation, the *low* frequency

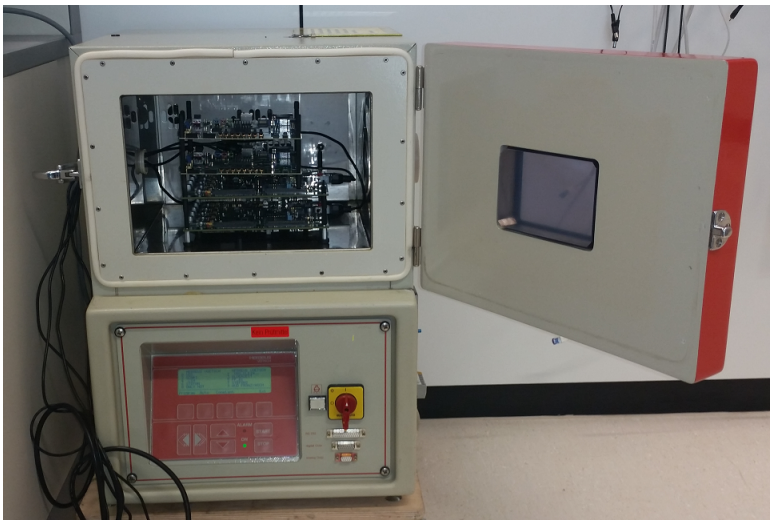


Figure 5.9: Accelerated aging test setup. A Heraeus Voetsch VMT 04/16 climate chamber was used together with two ZedBoards and two Xilinx ZC702 boards, all containing the Zynq XC7Z020 SoC.

was chosen to 100 Hz and the *high* frequency to 300 MHz.

To compare the results of the elevated degradation to the influence of reversible environmental conditions, the frequency of the oscillators at voltages from 0.9 V to 1.1 V and ambient temperatures from −30 °C to 70 °C was also measured.

5.3.5 Experimental Results and Discussion

The experiment was conducted during a timespan of over 330 days. The frequencies of all ROs were measured every six hours and saved by the PS. All data was collected and examined using Matlab.

Figure 5.10 shows the comparison of the boards that were aged using the maximum operation conditions (high V/T), and the ones that were just aged under a higher ambient temperature (low V/T). It can be seen that the aging was significantly accelerated using the higher voltage and temperature. The frequency under high V/T stress dropped to around 98.4 % of the initial value after 330 days, whereas the low V/T stress conditions only led to a decrease to 99.4%. The degradation of the frequency was very fast during the first few days and slowed down towards the end of the experiments.

The regeneration phase of 100 days led to an increase of the frequency of around 0.2 % on all four boards. However, this increase vanished a few days after stressing the devices again. A reversibility of the HCI and NBTI/PBTI effect could not be proven in this experiment by just disabling the stressing. As the devices were cooled down every six hours to measure the RO

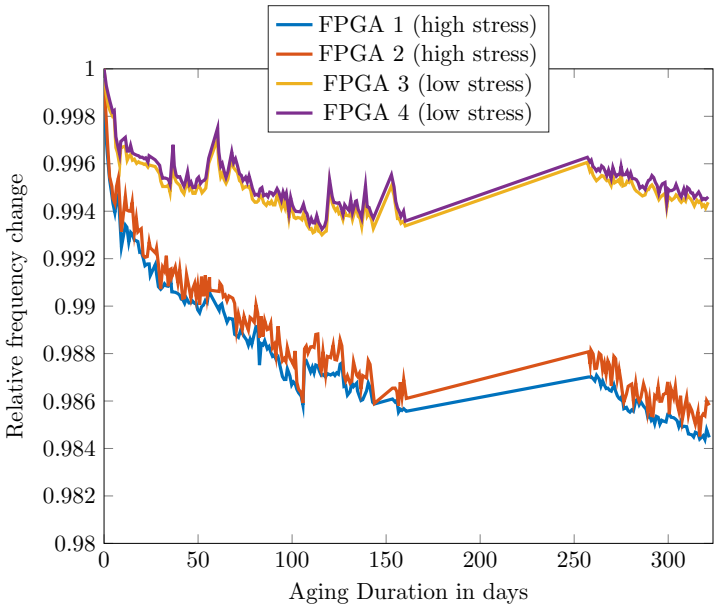


Figure 5.10: Degradation of the frequency of ROs using different environmental stress conditions, median value over all tested ROs.

frequency, the influence of a regeneration phase was not high. The regeneration effects likely occurred during these cool down phases already.

In the following analyses, only the boards aged under high V/T conditions are analyzed.

Figure 5.11 shows the relative frequency change of the different inverter RO PUF types. It can be seen that the degradation process was slightly different for all lengths of ROs. In this case, it was not the shortest PUF that aged the fastest, but the 3-inverter RO PUF. This might be due to the exact implementation of the PUF, which might use more hidden PMOS transistors than the other PUF implementations. It is impossible to analyze this in detail without knowing the exact, secret structure of the FPGA.

Figure 5.12 shows a comparison of the influence of the different electrical stress types for one of the boards that was aged under high stress conditions. In this experiment, the *DC 1* stress led to the lowest frequency change, whereas the *low* frequency stress had the highest impact. The relatively low impact of the *DC 1* stress was consistent with the results of previous research [SWC10]. This type of stress enabled the smallest number of PMOS transistors, which lowered the impact of the NBTI. As no dynamic effects occurred, the impact of HCI was also very low. The susceptibility of NMOS transistors to PBTI is increasing with the usage of high-k gates. Nevertheless, using this 28 nm device, this trend could not be confirmed in this experiment.

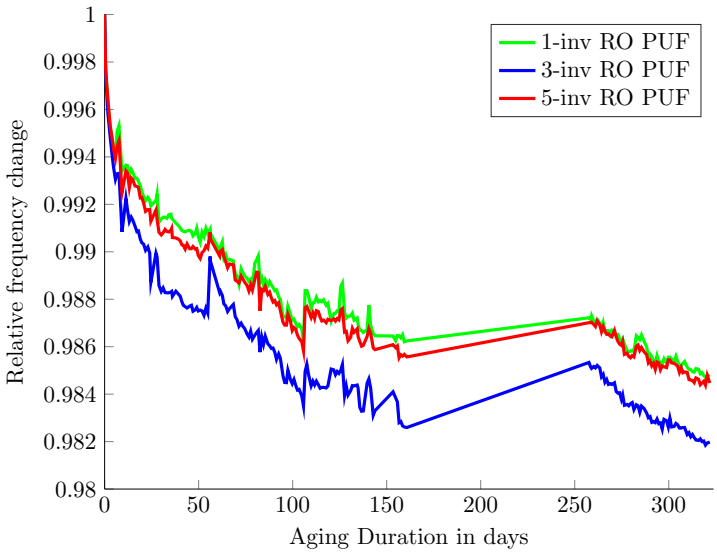


Figure 5.11: Impact of amount of inverters in RO on the frequency degradation, median value over all tested ROs.

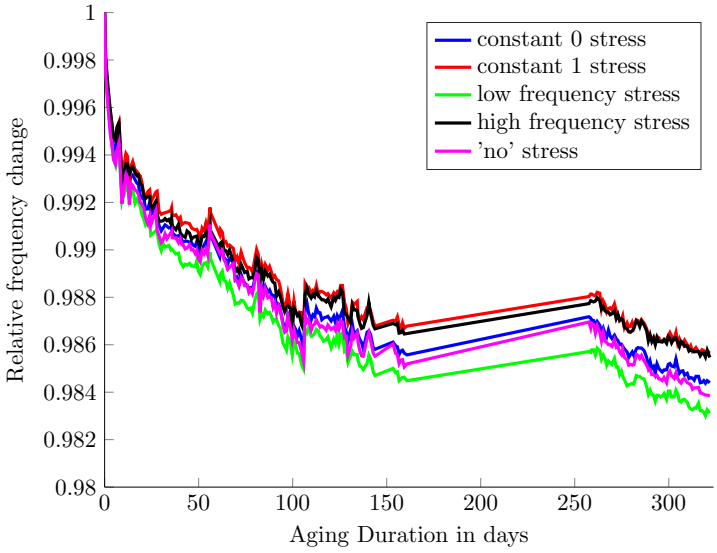


Figure 5.12: Impact of the electrical stress type on the frequency degradation.

The *DC 0* stress enabled more PMOS transistors and therefore had a slightly higher frequency degradation effect than *DC 1* stress. The *low* frequency stress combined both aging effects of dynamic and static stress. It oscillated fast enough to induce HCI effects, and still spent sufficient time in stable '0' and '1' states to enable NBTI/PBTI effects. It is interesting that

the *high* frequency stress showed a smaller impact than the *low* frequency stress. This might be due to the relatively high frequency, where the transistors did not spend much time in stable '1' states, thus lowering the impact of NBTI. An absolute difference of around 0.23 % was measured between the most and least severe electrical stress types. The relatively small difference between *DC 0* and *DC 1* stress could be explained by the structure of the LUT. Due to the inverters, half of the NMOS transistors were enabled and the other half were disabled, no matter which stress type was used. Just the inverters themselves, and the buffers were differently stressed when using *DC 0* and *DC 1* stress.

After recovering for 100 days, the frequency was slightly raised by around 0.15 %. This is in line with other research [MS14] that showed no big healing effects when not using the device. Furthermore, after letting the device run for a few days, the recovered frequency was lowered again and reached its old level after around 24 days.

It has to be noted, that the relative frequency degradation difference between the stress conditions having the highest and lowest impact was just 0.3 %. This can lead to measurement errors, as well as spatial and random effects, to have a large impact on the results, especially because the sample size was relatively small. However, the results were similar for every tested board. For every board the *DC 1* stress had the lowest impact and the *low* frequency had the highest impact on the frequency degradation. As shown in the next paragraph, all stress regions were also visibly distinguishable on a heatmap.

Figure 5.13 shows the frequency degradation of the ROs mapped on the FPGA. The different types of stress are abbreviated with the numbers 0 to 5 and described in the caption of the figure. It can be seen that the columns with *DC 1* stress are the darkest, i.e., they had the lowest frequency change and therefore still had the highest frequency after the aging process. In contrary, the *low* frequency stress columns appear the brightest. Although the absolute difference in frequency change was very small, it is still clearly visible on a map and the areas are easily distinguishable.

It can also be seen that the areas in the upper right corner aged slower than the rest of the chip. In contrary, everything close to the ARM PS in the left upper corner appears brighter and therefore aged faster. The PS itself might be the reason for this, as it was continuously used and heated up the FPGA locally.

To analyze the uniformity, uniqueness, and stability property of PUFs, binary result vectors were created for every data set. The raw frequencies were used in combination with equation 2.12 to create the PUF results and analyze them.

In order to achieve good uniformity and uniqueness of the PUF, a fractional HW close to 0.5 is important. Figure 5.14 shows the HW of the PUF output throughout the aging process. It can be seen that fractional

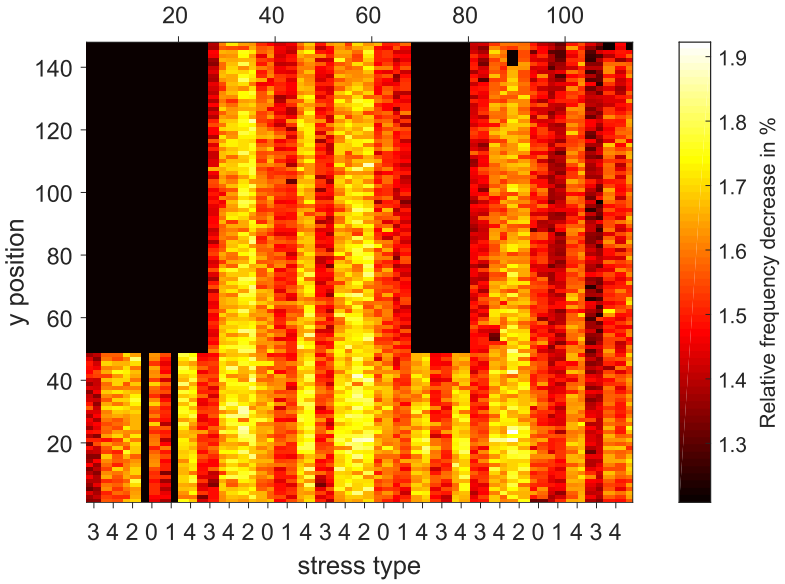


Figure 5.13: Heatmap of the FPGA illustrating the influence of different electrical stress conditions on the frequency degradation. Each column was equally stressed. 0: DC 0, 1: DC 1, 2: low frequency, 3: high frequency, 4: no stress

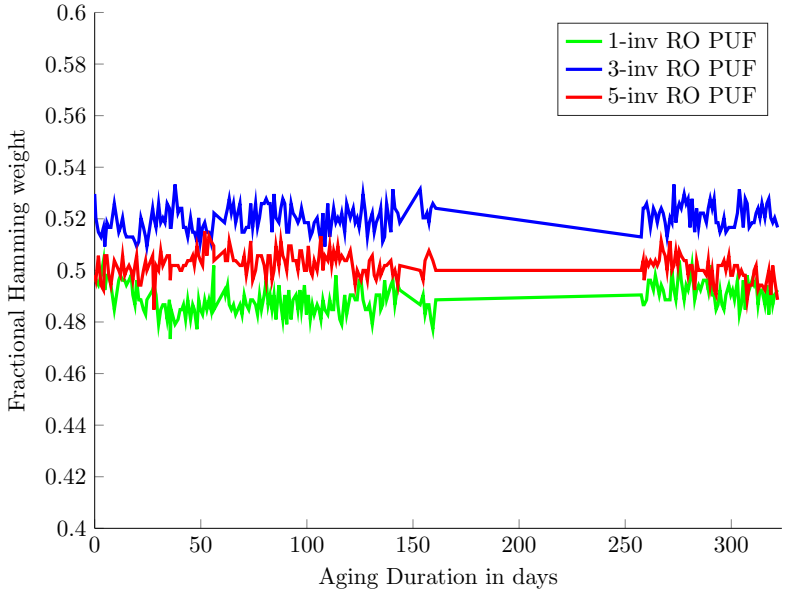


Figure 5.14: Impact of the aging process on the HW of the PUFs.

HW remained constantly close to the ideal value 0.5 for all three PUF implementations. The impact of reversible noise was much greater than the

influence of aging.

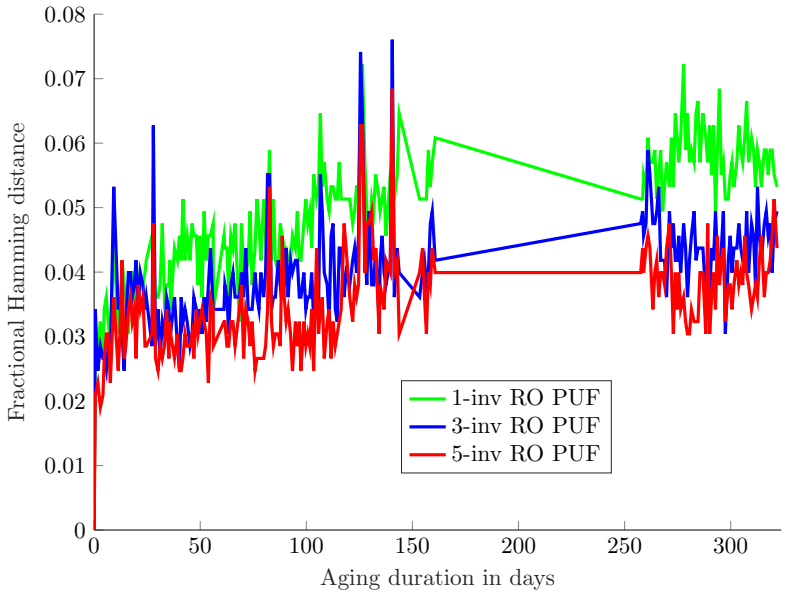


Figure 5.15: Impact of the aging process on the reliability of the PUFs measured with the intra-device HD.

An intra-device HD close to 0 is important for a stable PUF output. Figure 5.15 shows the fractional HD of each measurement referenced to the initial measurement. The fractional HD reached a level of 3% very quickly for all three PUF implementations. After the burn-in phase, it stabilized at a value of around 4 % to 6 %. The 1-inverter RO PUF showed a slightly higher fractional HD than the other implementations. This suggested, that ROs with slower frequencies, due to more involved transistors, led to slightly more stable PUF outputs. The impact of a single irregular transistor with a large delay is compensated. The recovery phase had almost no influence on the stability of the PUF. Some random peaks of the HD occurred for all RO implementations, during which the HD raised by almost 3 % for a single measurement. These peaks could be explained by random noise, such as EMI, temperature fluctuations in the climate chamber, voltage variations, or measurement noise within the FPGA. However, they have to be taken into consideration when designing a key generation algorithm that guarantees stable key outputs.

Neighbored ROs were compared to generate the PUF bits, as shown in Figure 5.6. As those pairs were always in the same stress zones, the stress on the differential measurement was always the same as well. This might not be the case when the PUF area is reused with other logic, as explained in chapter 4. A question that arises from this is: what happens to the stability of the PUF outputs, when two ROs are stressed differently. To analyze

this, the bit generation was now changed to a comparison of two ROs of neighbored stress regions. The spatial effects were kept at a minimum by using RO pairs close to each other. The resulting PUF vectors still showed good results in uniformity with a HW close to 0.5 and uniqueness with inter-device HDs close to 0.5. In the following, the stability of these PUF constructions will be analyzed.

Figure 5.16 shows the intra-device HD throughout the aging process. In comparison to Figure 5.15, it can be clearly seen that the resulting fractional HD to the initial measurement was much higher. Although the absolute frequency difference between the different types of stress was very small, the impact on the reliability when using different stresses for the same PUF seemed to be very high. The larger the frequency differences, the higher the resulting fractional HD. This can be seen, as the fractional HD was higher when comparing one RO from a low and one from a high frequency aging. In contrary, when comparing two ROs from DC 0 and DC 1, the resulting fractional HD was smaller. The maximum measured fractional HD raised from 6 % to almost 10 %. However, these results also have to be taken with a grain of salt. The DC0/DC1 stress zones were closer to each other than the low/high stress zones. This might have influenced the result by introducing spatial effects. Unfortunately, the stress design was not implemented to take this into consideration. Nevertheless, instead of reusing the area for other logic, the results hint that it would be better – from a stability point of view – to apply no stress or hardwire all logic to constant 1. This ensures that the logic being used for the ROs is always stressed the same way.

5.4 Temperature and Voltage

When operating RO PUFs in real world environments, they are prone to constantly changing external noise sources. These reversible variations can range from temperature – caused by device heating or ambient temperature changes – to voltage variations of the device. Robustness of the PUF against such variations is very important. In order to design a key generation scheme properly, the influence of temperature and voltage variations of RO PUFs is analyzed in this section.

5.4.1 Theory

Applying a higher supply voltage to CMOS gates leads to a higher drain current [WH11, Che06]. This effectively charges the capacitive gates of the following transistors faster. With C_L as the capacitive load at the output, β as the PMOS/NMOS gain factor, k as the scaling factor, and V_{DD} as the supply voltage; the rise time t_r as well as the fall time t_f can be approximated to [WH11]:

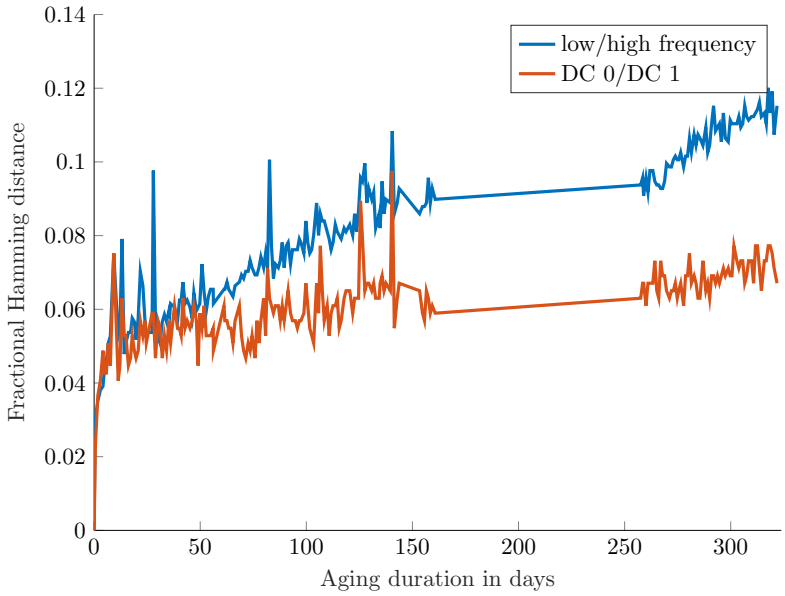


Figure 5.16: Intra-device HD throughout the aging when two ROs of two different stress zones compared.

$$t_r \approx k \cdot \frac{C}{\beta_p V_{DD}} \tag{5.9}$$

$$t_f \approx k \cdot \frac{C}{\beta_n V_{DD}} \tag{5.10}$$

A higher supply voltage decreases both the rise and fall time of a CMOS inverter. Therefore, the propagation delay of a signal is lowered. In the case of an FPGA with pass transistors, as shown in Figure 2.13, the higher voltage leads to faster switching between signal paths within the LUT. This results in a faster frequency of an oscillation ring.

The influence of temperature variations on the RO frequency is more complex, as multiple effects are overlaid. One effect is the decreasing threshold voltage with increased temperatures [WH11, WDDJ71, GWWT12]. This leads to higher drain currents and faster switching times at higher temperatures. Another effect is the influence on the carrier mobility in the silicon, caused by lattice scattering and impurity scattering. In normal temperature ranges, e.g., −30 °C to 120 °C for the FPGAs used in this work, the lattice scattering effect is dominant. It will cause the drain current to decrease with

higher temperatures and thus lowering the switching times of the transistors.

The threshold voltage effect is higher, the smaller the difference between threshold voltage and supply voltage. In larger CMOS processes ($> 32\,\text{nm}$) with classic polysilicon dielectrics, the difference used to be large [HGW+09]. The main effect was the scattering effect, effectively leading to a negative temperature-frequency correlation: the higher the temperature, the lower the frequency. But with the introduction of High-κ Metal Gates (HKMGs), the supply voltage is closer to the threshold voltage. Recent research [Zeg10, WA08, HGW+09, HH15, GWWT12] shows that with HKMG and small supply voltages, the correlation between temperature and frequency can be turned around. Some experiments showed a positive correlation between temperature and frequency for supply voltages of 1 V and below, thus having a larger effect than the scattering.

Another effect that comes into play is local heating by frequencies of the RO. Assuming a negative correlation, the frequency raises with lower temperatures. This, on the other hand, causes a higher power dissipation, which is heating up the RO.

5.4.2 Test Setup

For the temperature analysis, both ZedBoard and Xilinx boards were tested in the climate chamber. The temperature varied between $-45\,°\text{C}$ to $80\,°\text{C}$. The board cold-started from $-45\,°\text{C}$ to ensure the lowest possible temperature inside the SoC. The on-chip condition was measured using the integrated temperature sensor.

The voltage experiments were made using the Xilinx boards. These boards use five Texas Instruments (TI) power switching regulators PTD 08D210W. The TI power controller can be accessed via a PMBus controller, that is connected via I2C to the PS. The power controller allows adjustments of all power lines in the SoC, including the PS and PL voltage[1]. In this experiment, the internal FPGA voltage *VCCINT* was adjusted from 0.9 V to 1.15 V and the frequency of the ROs was measured.

Four distinct RO PUFs were used for both temperature and voltage analysis. Single 1-inverter RO PUFs of three different frequencies were used to analyze the influence on PUFs of different periodicity. The 1-inverter RO PUFs 1 (low frequency), 15 (medium frequency), and 29 (high frequency) of chapter 4, Figure 4.8, were used. Additionally, a 3-inverter RO PUF was measured to test the influence on presumably more stable RO PUFs. The 5-inverter RO PUF was not considered in this test, as routing problems occurred when trying to route the oscillation line directly to an output pin for a secondary frequency measurement using an oscilloscope.

[1]Details can be found under `http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Low+Power+Techniques+part+2+-+Measuring+ZC702+Power+using+TI+Fusion+Power+Designer+Tech+Tip`

5.4.3 Experimental Results

Voltage

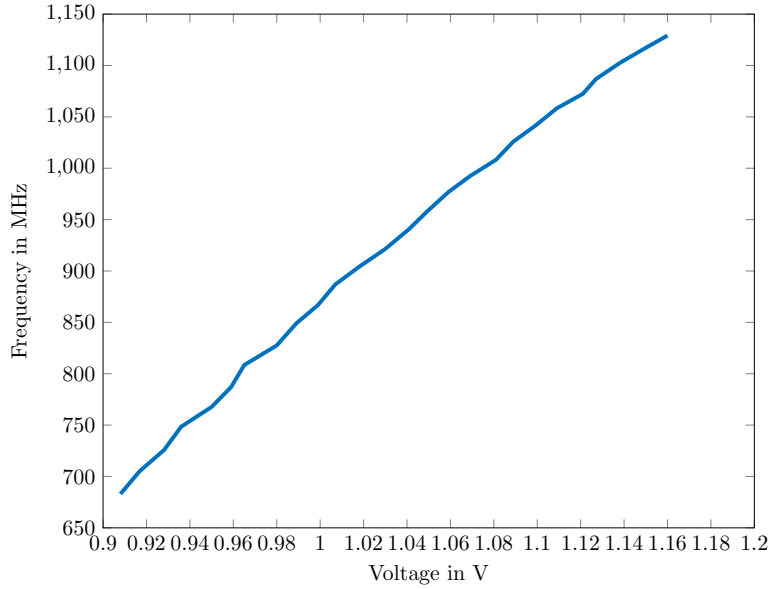


Figure 5.17: Absolute frequency change of the 1-inverter RO PUF 1 when changing the FPGA voltage from 0.9 V to 1.15 V.

Figure 5.17 shows the influence of voltage on the absolute frequency of an RO. The frequency at the recommended voltage of 1 V was around 850 MHz. This frequency changed highly when the voltage was varied. At 0.9V a frequency of only 670 MHz was measured. The maximum frequency was measured to 1130 MHz at 1.15 V. Although the frequency was higher than the normal frequency, the RO was still fully operational and measurable. This showed the robustness of the RO PUF configuration. The frequency had a linear dependence of the voltage.

Figure 5.18 shows the influence of voltage variations on the relative frequency of different RO PUF architectures. The 1-inverter RO PUF 29 had the highest absolute frequency, but it showed the smallest relative frequency change. On the other hand, the 3-inverter RO PUF showed the largest relative frequency change, roughly 10 % higher than 1-inverter RO PUF 29 at the highest voltage. However, the relative frequency change was very similar for all RO PUF implementations.

The influence of voltage variations on the intra-device HD is shown in Figure 5.19. The reference measurement was taken at 1 V, hence a HD of 0. It can be seen how the intra-device HD rises linearly with both an increase and decrease of the voltage. The impact is greater on ROs with a higher frequency. The 1-inverter RO PUF 29 had a fractional HD of 9.2 % at 0.9 V

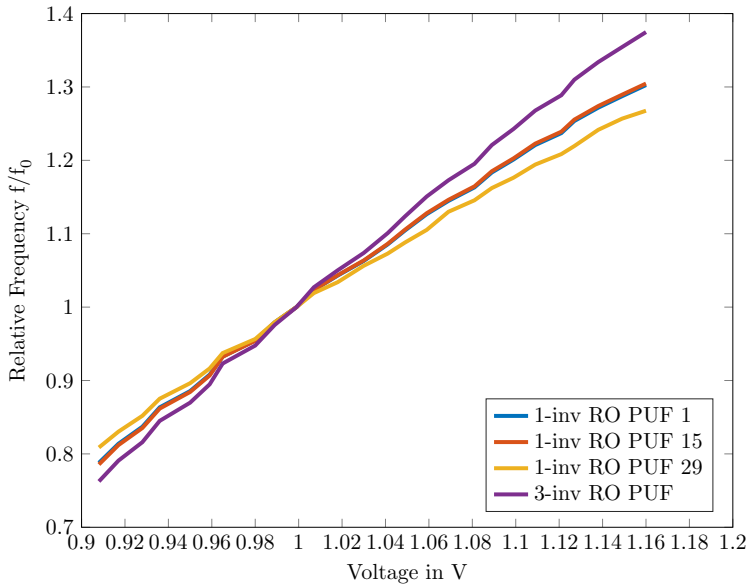


Figure 5.18: Relative frequency change when changing the FPGA voltage for different lengths and speed of PUFs.

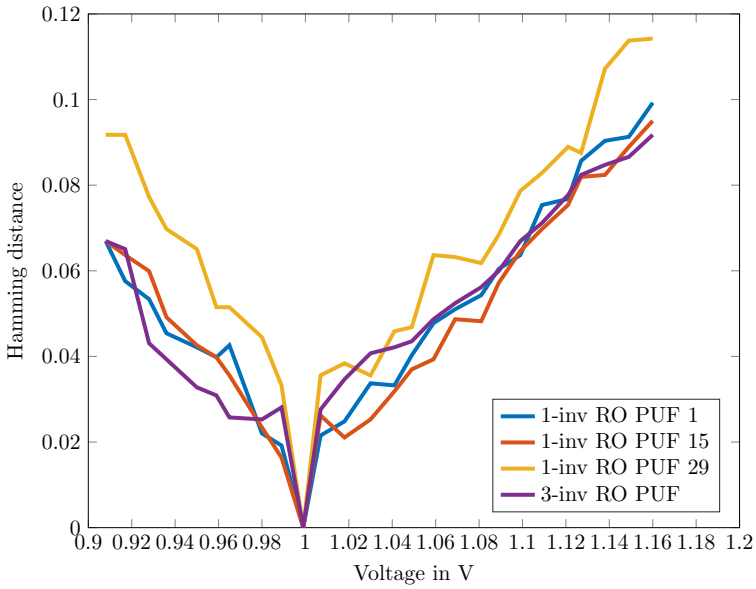


Figure 5.19: Intra-device HD when changing the FPGA voltage for different lengths and speed of PUFs.

and 11.5 % at 1.15 V. The other three tested RO types had a fractional HD of around 6.6 % at 0.9 V and roughly 9.5 % at 1.15 V. For small voltage variations, the intra-device HD is not distinguishable from normal noise of

the output as shown in chapter 3.

Temperature

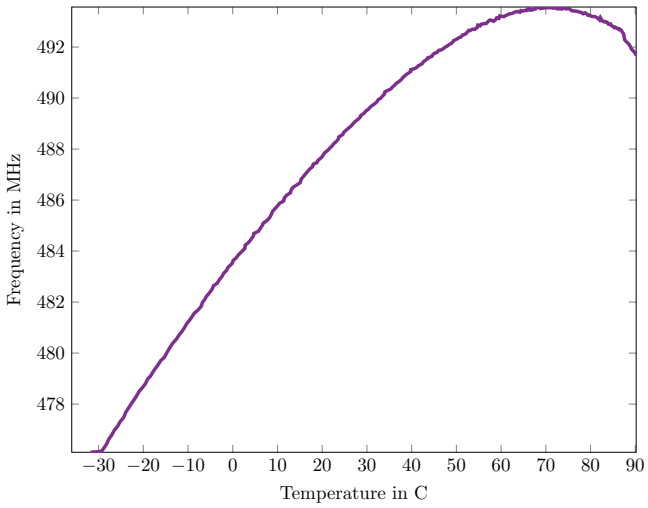


Figure 5.20: Absolute frequency change of the 3-inverter RO PUF when changing the ambient temperature from −45 °C to 80 °C.

Figure 5.20 shows the absolute change of frequency for the 3-inverter RO PUF under different ambient temperatures. The experiment started with the device turned off at the lowest ambient temperature to ensure a cold start. The internal temperature was measured to −31 °C at an ambient temperature of −45 °C. This temperature was well below the specifications, but the the tested device was still working properly. At the standard temperature of 25 °C, the internal temperature was measured to 38 °C in idle mode.

For internal temperatures of −31 °C to 70 °C a positive correlation between temperature and frequency can be seen in Figure 5.20. As the Xilinx Zynq uses HKMG technology, the influence of threshold voltages changes might became greater than the influence of scattering. The lowest frequency was reached at the lowest temperature with 476 MHz. The frequency then increased in form of a negative quadratical curve until an internal temperature of 70 °C, where it reached its maximum of 493.5 MHz. At higher temperatures the correlation started to be negative. From an internal temperature of 70 °C to 90 °C the frequency dropped to 491 MHz. This might be caused by an even more accelerated local heating due to the high frequencies. The impact of the scattering effect became greater and caused the frequency to drop again. As the maximum frequency difference was only 17.5 MHz, the influence of temperature variations on the frequency of ROs was much smaller than the influence of voltage variations.

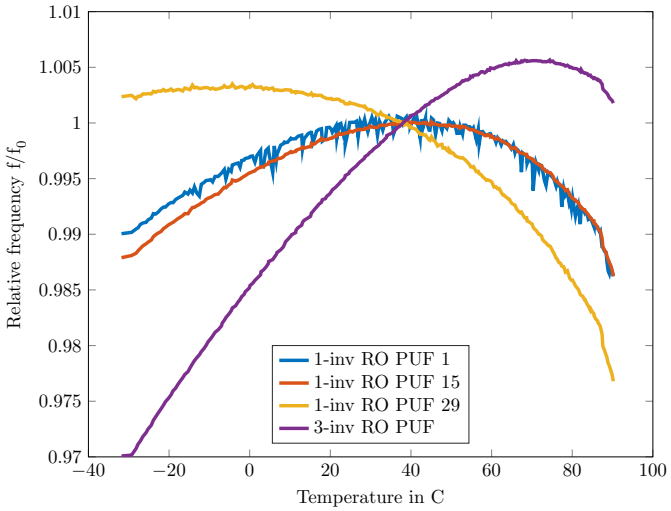


Figure 5.21: Relative frequency change when changing the ambient temperature for different lengths and speed of PUFs.

Figure 5.21 shows the comparison of the correlation between the relative temperature and the frequency for different lengths and speeds of RO PUFs. The 3-inverter RO PUF is the same one that is used in Figure 5.20. It can be seen, that with higher frequencies of the RO PUFs, the correlation between temperature and frequency became more negative. Both the 1-inverter RO PUF 1 and 15 showed a very similar behavior. Their frequency was positively correlated from −31 °C to 40 °C, where it reached its maximum. At higher temperatures, the frequency was negatively correlated. The lowest frequency was reached at the highest internal temperature of 90 °C.

The 1-inverter RO PUF 29 had a constant frequency for internal temperatures of −31 °C to 10 °C, where it also reached its maximum. At higher temperatures the frequency was negatively correlated with the temperature and reached its minimum at 90 °C. The impact of scattering seemed to be higher, the faster the ROs were.

As a conclusion it can be said that the correlation between frequency and temperature was very unpredictable. Multiple temperature effects seemed to influence the frequency when using a device with the 28 nm HKMG technology. As the exact structure of the FPGA is not publicly known, it is hard to model the effects. The empirical measurements showed a very unpredictable behavior. This also supports the idea of a differential measurement using neighbored ROs to generate bits. Their absolute frequency change might be unpredictable but was very similar in relation for every pair during the experiment.

Figure 5.22 shows the dependence of the intra-device HD from the ambient temperature for different PUF types on one of the boards. The initial

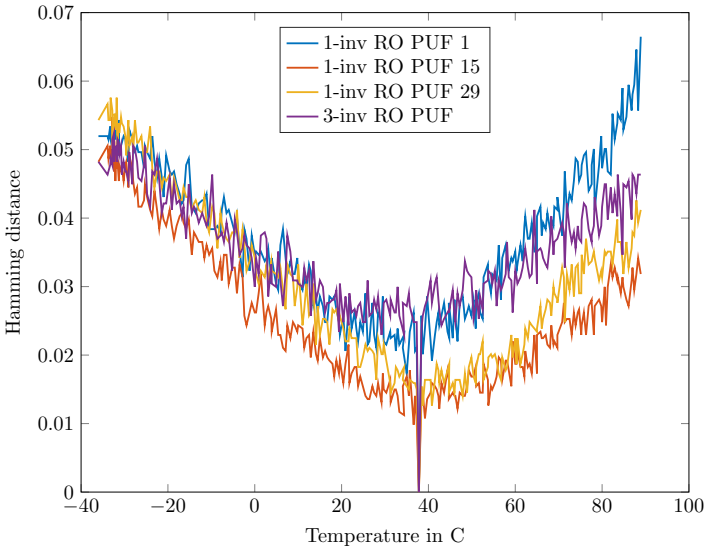


Figure 5.22: Intra-device HD when changing the ambient temperature for different lengths and speed of PUFs. Initial measurement taken at 25 °C ambient temperature.

measurement of all 6650 ROs was taken at 38 °C junction temperature, i.e., 25 °C ambient temperature. This also explains the spike, as only the initial measurement compared with itself has a HD of zero. Every consecutive measurement, even at the same temperature, shows a HD of at least 1.5 % to 2.5 % compared to the initial measurement. The HD raised linearly for both lower and higher temperatures for all RO types. The highest HD was reached by the 1-inverter RO PUF 1 at 90 °C with a HD of 6.6 %. The other PUFs had a HD of 3.2 % to 4.6 % at that temperature. At lower temperatures the results were very similar for all PUFs with HDs ranging from 4.8 % to 5.4 %. The most reliable PUF was the 1-inverter RO PUF 15 in these experiments.

5.5 Conclusion

The impact of reversible and irreversible variations on the quality of RO-based PUFs on 28 nm FPGAs was studied in this chapter. It was shown that one year of accelerated aging, which equaled an effective aging duration of roughly 60 years, lowered the frequency of ROs to around 98.6 %. As the highest frequency degradation occurs during the beginning of the lifetime, a burn-in phase would lower possible instability problems. Low frequency stress was found to cause the highest frequency degradation. Both HCI and NBTI effects are active when CMOS logic is stressed with a low frequency.

The complete aging process caused an intra-device HD of roughly 4 % to

6 %. Stressing RO pairs equally is recommended, as comparing differently stressed ROs with each other caused an intra-device HD of up to 12 %. The uniqueness property remained valid with HWs close to 0.5 and inter-device HDs close to 0.5.

The effects of reversible variations such as temperature and voltage were comparable with aging effects. The impact on the absolute frequency was very high when varying the voltage of the device. The frequency changed by over 20 % for voltages from 0.9 V to 1.15 V. At the same time the intra-device HD reached maximum values of up to 11.5 %. The impact of temperature variations on the absolute frequency was very unpredictable and depended highly on the used RO. This was mainly caused by multiple concurring effects when using the 28 nm HKMG technology. The impact of scattering, threshold voltage changes, and local heating caused by ambient temperature variations had different effects on the RO. Depending on the base frequency and the structure of the PUF, different correlations between temperature and frequency were measured. The impact on the intra-device HD, however, was very comparable for both different PUF implementations and boards, with maximum HDs of 3 % to 6.5 %.

The effects of reversible and irreversible frequency variations can be completely compensated by using error correction algorithms. This ensures a reliable behavior of the PUF throughout the lifetime of the device. The results from this chapter can be used to choose an appropriate error correction algorithm.

CHAPTER 6

PUF-Based Key Generation System

The methods shown in chapter 3 and chapter 4, as well as the results shown in chapter 5 are used in this chapter to design a PUF-based key generation system on an FPGA. As detailed analysis and implementation of key generation schemes are not part of this thesis, well known research results and their FPGA implementation results were used. Furthermore, a key enrollment scheme for the protection of the FPGA bitstream against manipulation, cloning, and reverse engineering is presented. The scheme can be used to securely transmit and store IPs on insecure flash memory. The chapter is organized as follows: In chapter 6.1 a short introduction is given. The used SoC platform with the given security features is presented in chapter 6.2. Using the results from chapter 5, a suitable error correction and key generation algorithm is chosen in chapter 6.3 and the optimized results using the reconfigurable PUF from chapter 4 are presented. A new method to transmit and store IPs securely on FPGA-based SoCs is shown in chapter 6.4. Finally, a conclusion is given in chapter 6.6.

6.1 Introduction

Modern FPGAs such as the 7-series of Xilinx and the V-series of Altera already make use of measures to protect the confidentiality, integrity, and authenticity of bitstreams [San13, DTB+15]. Unfortunately, most encryption schemes were broken with Side Channel Attacks [MKP12, MKP11, MOPS13]. These attacks revealed the encryption keys that were stored inside the device through differential power analysis attacks and the confidentiality of the IPs was voided.

On the other hand, modern FPGA-based SoCs implement secure boot

mechanisms based on public key cryptography, which only protect firmware (PL and PS part) authenticity and integrity. These schemes are, of today, not broken and can be used securely. Based on these mechanisms, a root of trust can be established and utilized for PUF-key-based cryptographic implementations.

The cryptographic functions that are provided by the vendors can only be used for a very limited amount of purposes. These include the encryption and authentication of a very specified protocol to load partitions from memory. User modules and functions cannot access the cryptographic accelerators [San13].

To be more flexible, the FPGA can be configured with own cryptographic functions. Furthermore, the implementations can be hardened against SCAs. The system design and protocol presented in this chapter can be used to design a secure and flexible solution to provide cryptographic functions for the PS and PL of FPGA-based SoCs.

6.2 SoC Platform

Modern SoCs with FPGAs already offer a variety of security features. They can be separated in two categories: the ones that ensure authenticity and the ones that ensure confidentiality. The latter is usually done by encrypting the FPGA bitstream using a symmetric encryption algorithm such as AES. For the Xilinx Zynq an AES-256 was chosen [San13]. The AES key is stored in fuses or Battery Backed RAM (BBRAM). A Keyed-Hash Message Authentication Code (HMAC) provides additional integrity protection of the bitstream using a SHA-256 hash function. The HMAC feature, however, relies on the bitstream encryption, as both the HMAC and the HMAC key are stored in the encrypted bitstream.

Unfortunately, various successful Side Channel Attacks (SCAs) on the symmetric cryptography of both Altera and Xilinx devices [MKP12, MKP11, MOPS13] showed the vulnerability of their implementations. As the HMAC key is stored in the encrypted bitstream, any security feature of the HMAC are also voided. Therefore, the system designed in this chapter does not rely on any symmetric encryption scheme provided by the manufacturer.

To ensure secure boot of the device the Xilinx 7-series also provides a 2048 bit RSA authentication. Every partition, i.e., PL bitstream (.bit files) and PS software (.elf and .bin files) can be authenticated. A SHA-256 hash of the partition is calculated and signed. This asymmetric authentication method is independent from the AES/HMAC encryption/authentication. The main advantage of this method is that the device only has to store a (hash of the) public key, as the private key is only needed for signing the partitions off-site. This makes the key management flexible and secure. It is not important to ensure the confidentiality of the public key, only the

authenticity and integrity. This is realized using eFuses. The complete public key is transmitted with every signed partition and verified against a stored hash of the public key by the SoC. There are no known attacks against the RSA authentication of modern FPGAs.

6.3 Error Correction and Key Generation Algorithm

The main goal of PUFs in this thesis is the ability to generate intrinsic, secure, and stable keys. PUFs face two main problems: a) their outputs are generally noisy (as shown in chapter 5) and b) the entropy is lower and harder to estimate compared to a true RNG (as shown in chapter 4). The resulting bit vector can not be directly used as a cryptographic key. In PUF key generation schemes fuzzy extractors are used to tackle both of these problems by increasing the reliability and compressing the entropy.

As shown in chapter 4 and 5, the PUF bits are not completely uniform and stable. Therefore, a fuzzy extractor is used to enhance the min-entropy of the PUF bits and to ensure a stable output. Many key generation algorithms with FPGA implementations have been proposed [DGSV14]. In the following, four of these schemes are presented with the implementation results taken from the papers.

The PUFKY algorithm by Maes et al. in 2012 [MVHV12] uses a modular design with RO PUFs. A bit error probability of $p_e = 12\,\%$ is assumed by the authors. A raw PUF output of 2226 bit is used together with 2052 bit helper data to generate a full-entropy 128 bit key. The design occupied 1162 slices, of which the main part of 82 % was occupied by the RO PUF itself. As the area consumption of RO PUFs was tackled in chapter 4 of this thesis, the design could be further shrunk. The runtime was measured to 5.62 ms, where the major part of the runtime was needed for the PUF output with 4.59 ms.

A soft-decision helper data algorithm for SRAM PUFs was proposed by Maes et al. in 2009 [MTV09b]. A soft-decision Reed-Muller-Code (RM) based on a Generalized Multiple Concatenated Code (GMC) was used. The design consumed significantly less resources than previously presented designs [BGS+08]. It occupied 237 slices, needed 13 952 bit helper data, a 1536 bit SRAM, and had a runtime of 205 µs.

The C-IBS implementation by Hiller et al. [HDSMS12] used a RM-GMC. 2304 PUF output bits were needed together with 9216 helper data bits to generate a 128 bit key. The design consumed 250 slices. The bit error probability p_e was given with 15 %.

The compressed DSC using convolutional codes by Hiller et al. [HS14, HWRL+13, HYS16] needed 974 PUF output bits and 1108 helper data bits. The design consumed a total of 249 slices. The Seesaw Viterbi decoder

consumed with 10 752 bit very many block RAM bits. The advantages of the DSC are the small amount of required PUF bits and helper data bits. The runtime was roughly 3 times slower than the soft decision algorithm by Maes et al..

Scheme	PUF bits	Slices	Helper data bits
PUFKY [MVHV12]	2226	209	2052
Soft-Decision [MTV09b]	1536	237	13952
C-IBS [HDSMS12]	2304	250	9216
DSC [HYS16]	1224	262	2176

Table 6.1: Comparison of implementations of 128 bit key generation schemes. The PUFKY scheme assumes a bit error probability of 13 % and a desired 128 bit key error probability of 10^{-9}. The other schemes assume a bit error probability of 15 % and a desired 128 bit key error probability of 10^{-6}. The slices do not include the implementations of the PUF instances themselves.

Table 6.1 shows the presented PUF key generation schemes in comparison. For this work the PUFKY algorithm seemed the most appropriate algorithm to balance between area and runtime. As the algorithm was designed for RO PUFs it is also very suitable for this work. The PUFKY algorithm uses a combination of repetition code (REP) and BCH code secure sketches. The repetition code is used to relax the design constraints, whereas the BCH code is used to do the main work. The largest part of slices is needed for the BCH decoder with 112 slices. The decoding time is given with 50 320 cycles. After the REP and the BCH stage, a small and area efficient SPONGENT hashing is used to compress the entropy to a 128 bit key. The hash algorithm consumed 22 slices. The algorithm is fed by 2226 PUF output bits, which contain an estimated 2180 bits of min-entropy.

As shown in chapter 4, a very conservative estimate showed an entropy of 93 % for the presented partial reconfiguration PUF. To generate 2180 bit of entropy, a raw PUF output size of $2180/0.93 = 2344$ bit would be needed. As 48 bit can be generated per pair of CLBs, a total of $\lceil 2 \cdot 2344/48 \rceil = 98$ CLBs would be needed to generate that amount of bits using the partial reconfiguration PUF. This equals $98 \cdot 2 = 196$ slices, as one CLB consists of 2 slices for a XILINX Zynq. The measurement circuit consumes another 25 slices (see Table 3.1). The original PUFKY implementation, on the other hand, needed 952 slices for the RO PUFs. This equals a reduction of necessary slices by 79.4 %. Furthermore, the reconfigurable PUF uses the slices of the CLBs more efficiently. With the RO PUF used in the PUFKY implementation, some slices might not be usable anymore, as the PUF already occupied major parts of the CLB. This fact is not considered when using the raw amount of slices that are occupied by a design.

A main concern of helper data algorithms is their potential entropy leak-

age by public data. There is still a lot of open research about the actual leakage [DGSV14]. However, in comparison to other suggested protocols the PUFKY approach is very conservative as every helper data bit is subtracted from the amount of PUF bits to calculate the final entropy. This allows a higher confidence in the full entropy of the resulting 128 bit key. The fail rate of the PUFKY implementation is given with $p_{fail} \leq 10^{-9}$.

6.4 Security Module

A new security module and protocol using the previously presented reliable PUF-based key generation scheme is proposed in this section. Using this module, the FPGA-based SoC is able to store confidential and authenticated firmware on the (external) device memory, allow secure over-the-air flashing, and allow secure communication using device unique keys. As these IPs contain a lot of valuable knowledge, the protection of their confidentiality is very important. As the symmetric ciphers of most modern FPGAs are broken by SCAs, an own AES has to be implemented and used. The authenticity of the firmware is even more important in modern device protection and secure communication. This is especially true for the PUF bitstream. The confidentiality of this bitstream is not very important, but its authenticity and integrity is. Any attacker being able to alter the bitstream and load own bitstreams on the target FPGA could reroute PUF outputs and simply read out the responses. The attacker should be prevented from using the SoC with any unauthenticated firmware, thus hindering him from characterizing a device.

The most important requirement of the SoC platform is therefore a public-key-based secure boot mechanism in which the internal boot code validates the signature of the next firmware/bitstream part. This feature is offered by most modern FPGA SoCs.

The proposed security module offers the following features:

1. an enrollment in a secure environment of the vendor
2. a secure boot procedure including the FPGA bitstream
3. an IP loading procedure, and
4. an IP storage procedure

Any protected IP can be stored on untrusted non volatile memory, as it is encrypted and authenticated.

6.4.1 Enrollment Phase

The system is enrolled in a secure environment, where only authorized staff has access to workstations and its strictly separated IT network. Figure 6.1

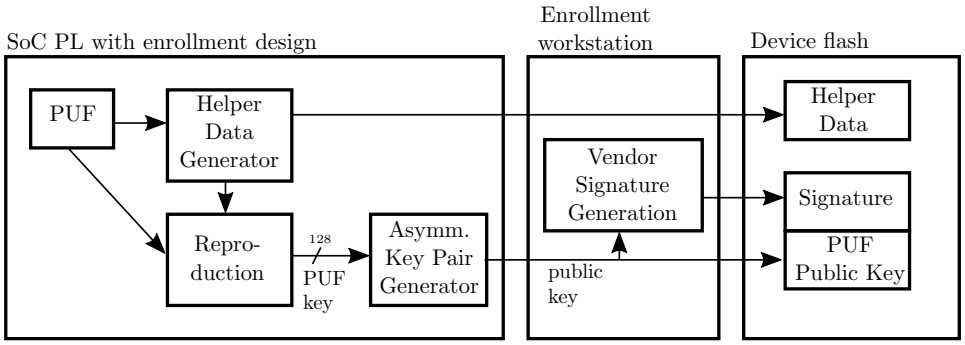


Figure 6.1: SoC enrollment in a secure environment of the vendor.

shows the enrollment process. An enrollment bitstream is used to generate the PUF response. It is then used within the device to generate the helper data. The PUF response is used together with the helper data to generate a reliable and uniform key. This key can be used, e.g., as a salt for a PRNG to generate a private and public key pair. The key length should be at least 2048 bit to be sufficient until 2030. The private key never leaves the FPGA, only the public key does [Bar16].

The PUF public key is signed on an external enrollment workstation to ensure that the device was enrolled by the vendor. The signature is then stored together with the public key and the helper data on the device flash memory.

The public key of the system vendor respectively a hash of the key is burned in the device fuses. The fuses that enforce secure boot and disable JTAG are set as well. The Security Module (SM) firmware that contains, e.g., the PUF is signed with the system vendor private key. Both signature and SM firmware are stored in the device flash memory. The device can then be used in a non-secure environment, as only vendor signed firmwares can be executed.

6.4.2 Security Module Boot Procedure

Figure 6.2 and 6.3 show the boot procedure of the SoC. The PS bootloader loads (1a) the SM firmware together with the according certificate, including the vendor public key, from the device flash memory. The hash of the vendor public key is read from the internal fuses and compared with a hash from the provided vendor public key (1b) and used to verify the certificate of the SM. If the public key and/or the certificate is not correct, the system enters a fail-secure state. Otherwise, the SM including the PUF is loaded (2) into the PL. If the reconfigurable PUF is used, the PS loads the partial implementations sequentially on the PL and the intermediate results are stored within the PL. The complete PUF response is used together with the

helper data to reconstruct the device unique private key and stored together with the PUF public key inside the PL. The private key never leaves the PL part of the SoC.

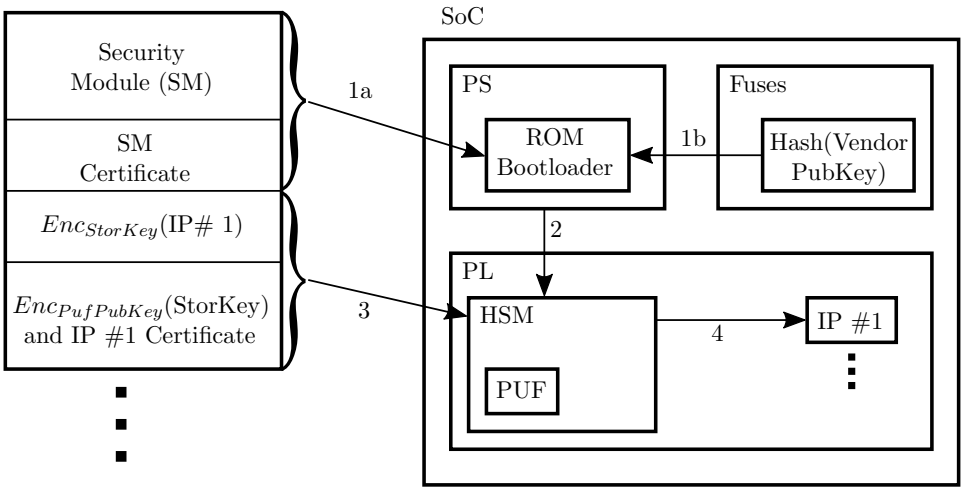


Figure 6.2: SoC boot procedure with the non-volatile memory on the left side and the SoC on the right side.

If the reconfigurable PUF is used together with a very robust ECC, the complete area could now be reused by other IPs. Otherwise, the area used for the ROs should be reconfigured with a blank design according to Figure 5.12 and Figure 5.16. After the boot procedure, a device unique PUF-based public/private key pair is available inside the PL. The SM can be used as a root of trust.

6.4.3 IP Loading Procedure

After establishing a root of trust in the PL using the PUF-based key generation, the encrypted and signed IP cores can be loaded from the flash memory. As shown in Figure 6.2, the SM (3) loads the encrypted IP, the corresponding storage key, which is encrypted with the PUF public key, and the IP certificate. At first, the certificate of the encrypted IP is verified to ensure authenticity and integrity of the partition using the PUF public key. The PUF private key is then used to decrypt the storage key, which itself is then used to decrypt the IP. The IP can then be programmed (4) to the FPGA using partial reconfiguration.

6.4.4 IP Storage Procedure

A secure communication with the vendor's server can be used to update IPs or load new IPs over-the-air. Figure 6.4 shows a scheme for secure IP

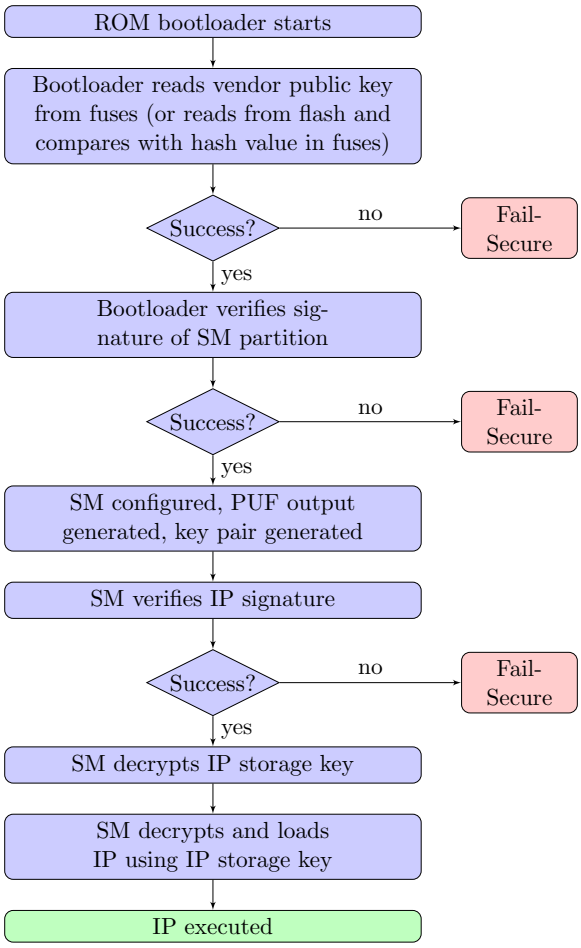


Figure 6.3: SoC boot procedure.

transmission and storage on the device. In order to set up a secure communication with the vendor's server, the device sends its own PUF public key together with the vendors signature to the server. The server verifies the signature to ensure the authenticity of the communication partner and its PUF public key. An ephemeral symmetric transport key is used to protect the confidentiality of the transmitted IP. The transport key is encrypted with the PUF public key and transmitted to the device.

When a secure communication with the vendor's server is set up, the IP can be encrypted with the transport key and transmitted to the PL part of the device. The device decrypts the IP with the transport key and stores it in the device flash memory using a random storage key. The storage key is encrypted with the PUF public key and also stored in the device flash memory. The encrypted IP is signed with the PUF private key in order to ensure authenticity and integrity. No plain text leaks outside of the PL

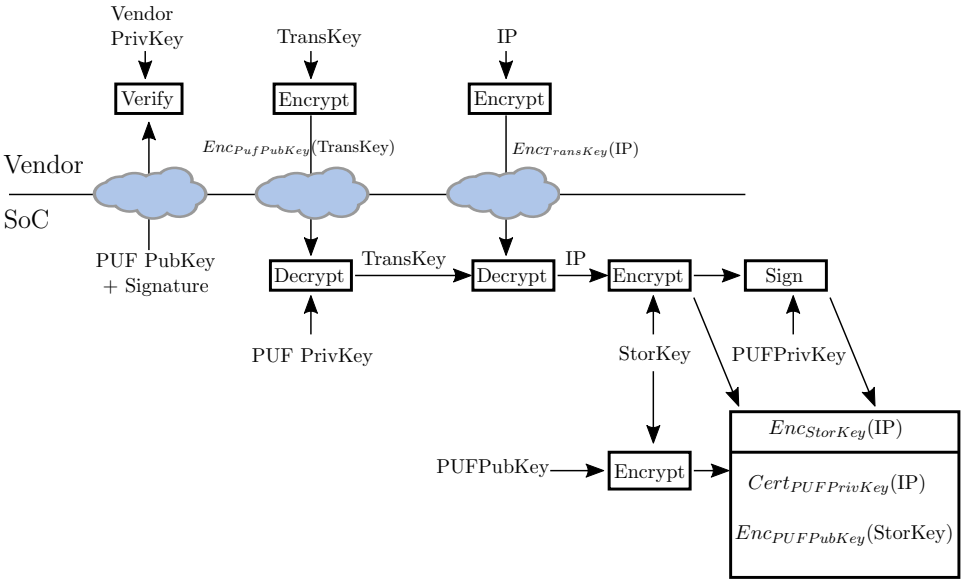


Figure 6.4: IP storage procedure using symmetric transport and storage keys, as well as the asymmetric PUF generated key pair.

during this procedure.

6.5 Discussion

The presented scheme can only be seen as an exemplary application of a PUF-based key generation on an FPGA. Some security problems might arise in the schemes, depending on the use case. One example is the re-usage of the PUF-based asymmetric key pair for both decryption and signature verification. This is not an ideal design standard. Only if an attacker could not use the FPGA as an oracle in a chosen-ciphertext attack, most attacks would not be possible. However, especially with future attacks being more sophisticated, the requirements to the security level have to be thoroughly analyzed to justify this design choice. A possible alternative would be the generation of two asymmetric keys, one for each purpose. This would require a longer PUF key size to provide sufficient entropy for the key generation. Alternatively, an AES-GCM could be used to sign and encrypt an IP inside an FPGA using a symmetric, PUF-based key. If the goal is the usage of smaller keys, an elliptic-curve-based algorithm like Elliptic Curve Diffie–Hellman (ECDH) in combination with a signature verification could be used as a symmetric key agreement protocol. An in-depth analysis of these protocols is not part of this thesis. Hence, the proposed scheme should always be analyzed for the security needs of the desired protocol.

6.6 Conclusion

In this chapter an exemplary application of the highly efficient implementation of PUFs on FPGAs was shown. The measured quality properties of chapter 5 were used to choose an appropriate PUF-based key generation scheme. The PUFKY algorithm was used in this work. The amount of slices that were needed to implement the RO PUF part of the system could be lowered by at least 79.4 % when using the partial reconfiguration PUF presented in chapter 4. The actual reduction of occupied CLBs and area is even bigger, as the slices of the CLBs are used much more efficiently.

As the implementation of symmetric encryption algorithms on most modern FPGA-based SoCs is broken, a new scheme was introduced. This scheme allows the secure boot of an authenticated SM that is able to generate a device specific asymmetric key pair. This key pair can be used to load encrypted IPs from an unprotected device flash memory. Furthermore, a procedure was proposed that allows the transmission of modified or new IPs over-the-air from the vendor. All IPs are transmitted and stored encrypted, such that the confidentiality of the IPs is preserved. Furthermore, all IPs are authenticated to ensure that only verified firmwares and partitions can be run on the device.

CHAPTER 7

Conclusion and Future Work

In this thesis it was shown how to efficiently implement RO PUFs on FPGAs using partial reconfiguration. All PUF implementations were tested on their quality properties. An extensive examination of the reliability of the PUFs was done by performing accelerated aging tests over a period of around one year, and complemented with voltage and temperature variation tests.

In the first part of the thesis, a basic 3-inverter RO PUF system was implemented on a 28 nm FPGA. All tested quality properties such as bit alias, uniformity, stability, uniqueness, and read-out time showed very promising results. The intra-device HD was measured to 3 %. The inter-device HD and the HW of the bit vectors were very close to an ideal value of 0.5. The downside of this design was its huge area consumption. The generation of 2136 raw PUF bits consumed almost 80 % of all available slices on a Xilinx Zynq Z-7020.

In the second part of this thesis, the problem of the high area consumption was tackled by introducing the partial reconfiguration PUF. All available LUTs in a CLB, as well as all available input pins of the LUTs were used for separate 1-inverter RO designs. The designs were reconfigured one after another using partial reconfiguration. Each result was stored inside the FPGA. Finally, all partial bit vectors were concatenated to a large bit vector. Using this method, the raw PUF output could be increased by a factor of 48 by using all eight LUTs of a CLB and six input pins of a LUT. In other words, the required resources to generate a PUF response with a given length could be decreased by 98 %. Instead of consuming 80 % of the available slices to generate 2136 PUF bits, only roughly 2 % would be needed using this method.

A design was implemented that generated a total of 102 528 raw output

bits using the same area as the design in the first part. The design was tested on a total of ten Xilinx Zynq Z-7020. The read-out time, including measurements, reconfiguration, and write-back of all 102 528 bits was measured to 3.78 s. The HWs of the partial and concatenated bit vectors were very close to an ideal value of 0.5. As expected, some of the partial bit vectors on the same board were correlated due to shared resources between the partial RO designs. The HD between those correlated designs was in a few cases as low as 0.2, but mostly close to the ideal result of 0.5. The inter-device HD of the concatenated partial bit vectors was always very close to 0.5.

The entropy of the concatenated bit vectors was shown to be good using three tests: CTW, PCA, and the NIST SP 800-90B entropy test. CTW showed that the bit vectors were incompressible. The algorithm was not able find any compressible correlations. The PCA showed that each partial RO design contributed to the final result and could not be linearly combined by using any other partial RO designs. The NIST SP 800-90B, a very young entropy test that works on small data sets, measured a min-entropy of 0.927 bit per raw PUF bit. This is a very promising result, considering that the total bit vector size could be increased by a factor of 48.

In the third part of this work, the reliability of RO PUFs on FPGAs was analyzed. The first section covered the aging behavior of ROs on FPGAs. Different aging designs, each accelerating different kinds of aging mechanisms, were implemented on an FPGA. They were then run inside a climate chamber with accelerated aging conditions, i.e., higher voltage and temperature. The average RO frequency on a device that was effectively aged for almost 60 years was lowered by 1.6 %. The HW was not affected by the aging and stayed close to the ideal value of 0.5. At the same time, the aging process caused an intra-device HD of up to 6 %. The experiments also showed that it is very important to compare equally stressed ROs with each other. Otherwise, the intra-device HD reached values up to 12 %.

In the second section, the influence of temperature and voltage variations was tested. A maximum of 11.5 % intra-device HD was measured when changing the voltage from 1 V to 1.15 V. Voltage variations showed a linear effect on the frequency of the ROs. Temperature variations, on the other hand, had a very unpredictable impact on the frequency. This could be explained by the usage of the 28 nm HKMG technology. The intra-device HD caused by temperature variations showed an almost linear behavior with maxima of 5 % at −38 °C ambient temperature and 6 % at 90 °C.

In the last part, a PUF-based key generation system was introduced. A suitable error correction and key generation algorithm was chosen from literature and the possible area reduction calculated. When using the PUFKY algorithm, the CLB usage could theoretically be reduced by 79.4 %. Furthermore, the partial reconfiguration PUF would have a more efficient usage of the slices in the CLBs and hence a better usability of the remaining area. Afterwards, a security module was presented as an outlook for the secure

boot and secure storage possibilities of an FPGA-based SoC using the stable and area-saving partial reconfiguration PUF. The system is able to enroll a device during manufacturing, securely boot the SoC, load encrypted IPs from an unprotected non-volatile memory, and securely store IPs transmitted over-the-air.

Future Work This thesis showed the efficiency and usability of RO PUFs on FPGAs. In future work, other types of PUFs could be implemented with the partial reconfiguration method presented in chapter 4. The sources of entropy could even be extended by using different routing possibilities within the CLB. This, on the other hand, would increase the correlation between those partial RO designs. A problem that could be tackled by a higher compression (privacy amplification) of the raw output bits. Other methods to analyze the entropy of the resulting bit vectors could be used. The work on entropy estimation of relatively small data sets is still very young and not much prior work exists. The methods presented in this work could be used as a basis and extended by other methods.

The accelerated aging tests that were done in chapter 5 could be extended by more stress designs that would accelerate other types of aging mechanisms. Furthermore, an inverse stress could be applied on the stressed logic, which might invert some aging mechanisms. A larger number of devices should be tested for more reliable results. Due to the relatively small climate chamber, only four devices were aged in this work.

The error correction and key generation algorithm PUFKY presented in chapter 6 could practically be implemented with the partial reconfiguration PUF. After the implementation of a key generation algorithm, the security module could be implemented in future efforts to demonstrate the usability of the proposed secure system design.

Bibliography

[Ala03] M. A. Alam, "A Critical Examination of the Mechanics of Dynamic NBTI for PMOSFETS," in *IEEE International Electron Devices Meeting 2003*, Dec. 2003, pp. 14.4.1–14.4.4.

[Alt16] Altera, "Reliability Report MNL-1085," 2016. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/rr/rr.pdf

[Bar16] E. Barker, "Recommendation for Key Management Part 1: General," National Institute of Standards and Technology, Tech. Rep. NIST SP 800-57pt1r4, Jan. 2016.

[Bau83] D. W. Bauder, "An Anti - Counterfeiting Concept for Currency Systems," Sandia National Labs, Tech. Rep., 1983.

[BDBRR12] P. F. Butzen, V. Dal Bem, A. I. Reis, and R. P. Ribas, "Design of CMOS logic gates with enhanced robustness against aging degradation," *Microelectronics Reliability*, vol. 52, no. 9–10, pp. 1822–1826, Sep. 2012.

[BEYY04] R. Begleiter, R. El-Yaniv, and G. Yona, "On Prediction Using Variable Order Markov Models," *Journal of Artificial Intelligence Research*, vol. 22, no. 1, pp. 385–421, Dec. 2004.

[BGS+08] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 181–197.

[BSQ10] P. Bulens, F.-X. Standaert, and J.-J. Quisquater, "How to Strongly Link Data and its Medium: The Paper Case," *IET Information Security*, vol. 4, no. 3, pp. 125–136, Sep. 2010.

[CB13] C. Chiasson and V. Betz, "Should FPGAs Abandon the Pass-Gate?" in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Sep. 2013.

[Cha14] K. Chapman, "Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources, Xilinx XAPP522," Oct. 2014. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp522-mux-design-techniques.pdf

[Che06] W.-K. Chen, *The VLSI Handbook, Second Edition (Electrical Engineering Handbook)*. Boca Raton, FL, USA: CRC Press, Inc., 2006.

[CMK05] Y. Chen, M. K. Mihçak, and D. Kirovski, "Certifying Authenticity via Fiber-Infused Paper," *ACM SIGecom Exchanges*, vol. 5, no. 3, pp. 29–37, Apr. 2005.

[DGSV14] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2014.

[DRS04] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," in *Advances in Cryptology - EUROCRYPT*. Springer, 2004, pp. 523–540.

[DTB+15] R. Druyer, L. Torres, P. Benoit, P. V. Bonzom, and P. LeQuere, "A Survey on Security Features in Modern FPGAs," in *Proc. Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2015.

[GCvDD02] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proc. Conference on Computer and Communications Security*. Washington, DC, USA: ACM, 2002, pp. 148–160.

[GKST07] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Vienna, Austria: Springer-Verlag, 2007, pp. 63–80.

[GLC+04] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas, "Identification and Authentication of Integrated Circuits: Research Articles," *Concurrency and Computation: Practice & Experience - Computer Security*, vol. 16, no. 11, pp. 1077–1098, Sep. 2004.

[GLS15] S. Gehrer, S. Leger, and G. Sigl, "Aging Effects on Ring-Oscillator-Based Physical Unclonable Functions on FPGAs," in *Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec. 2015.

[GS14] S. Gehrer and G. Sigl, "Reconfigurable PUFs for FPGA-based SoCs," in *Proc. International Symposium on Integrated Circuits (ISIC)*, Dec. 2014, pp. 140–143.

[GS15] S. Gehrer and G. Sigl, "Using the Reconfigurability of Modern FPGAs for Highly Efficient PUF-Based Key Generation," in *Proc. Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, Jun. 2015.

[GS16] S. Gehrer and G. Sigl, "Area-Efficient PUF-Based Key Generation on System-on-Chips with FPGAs," *Journal of Circuits, Systems and Computers*, vol. 25, no. 01, Jan. 2016.

[GWWT12] M. Gag, T. Wegner, A. Waschki, and D. Timmermann, "Temperature and On-Chip Crosstalk Measurement Using Ring Oscillators in FPGA," in *Proc. International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2012, pp. 201–204.

[HBF09] D. Holcomb, W. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, Sep. 2009.

[HDSMS12] M. Hiller, F. De Santis, D. Merli, and G. Sigl, "Reliability Bound and Channel Capacity of IBS-Based Fuzzy Embedders," in *Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, Jun. 2012, pp. 213–220.

[HGW+09] S. J. Han, D. Guo, X. Wang, A. C. Mocuta, W. K. Henson, and K. Rim, "Reverse Temperature Dependence of Circuit Performance in High- /Metal-Gate Technology," *IEEE Electron Device Letters*, vol. 30, no. 12, pp. 1344–1346, Dec. 2009.

[HH15] J. Hussein and M. Hart, "Lowering Power at 28 nm with Xilinx 7 Series Devices, Xilinx WP389," 2015. [Online].

Available: https://www.xilinx.com/support/documentation/white_papers/wp389_Lowering_Power_at_28nm.pdf

[HMSS12] M. Hiller, D. Merli, F. Stumpf, and G. Sigl, "Complementary IBS: Application Specific Error Correction for PUFs," in *Proc. International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2012.

[HMV12] G. Hospodar, R. Maes, and I. Verbauwhede, "Machine Learning Attacks on 65nm Arbiter PUFs: Accurate Modeling Poses Strict Bounds on Usability," in *Proc. International Workshop on Information Forensics and Security (WIFS)*. IEEE, Dec. 2012, pp. 37–42.

[HS14] M. Hiller and G. Sigl, "Increasing the Efficiency of Syndrome Coding for PUFs with Helper Data Compression," in *Proc. Conference on Design, Automation & Test in Europe (DATE)*, ser. DATE '14. Belgium: European Design and Automation Association, 2014, pp. 71:1–71:6.

[HWRL+13] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner, and G. Sigl, "Breaking Through Fixed PUF Block Limitations with Differential Sequence Coding and Convolutional Codes," in *Proc. International Workshop on Trustworthy Embedded Devices (TrustED)*. New York: ACM, 2013, pp. 43–54.

[HYKS10] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs," in *Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec. 2010, pp. 298–303.

[HYS16] M. Hiller, M. D. Yu, and G. Sigl, "Cherry-Picking Reliable PUF Bits With Differential Sequence Coding," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 2065–2076, Sep. 2016.

[Ini08] K. Iniewski, *Circuits at the Nanoscale: Communications, Imaging, and Sensing*. CRC Press, Sep. 2008.

[JED16] JEDEC, "Failure Mechanisms and Models for Semiconductor Devices," Sep. 2016. [Online]. Available: http://www.jedec.org/sites/default/files/docs/JEP122H.pdf

[Jol02] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed., ser. Springer Series in Statistics. Springer-Verlag New York, 2002.

[Kir04] D. Kirovski, "Toward an Automated Verification of Certificates of Authenticity," in *5th ACM Conference on Electronic Commerce*, ser. EC '04. New York, NY, USA: ACM, 2004, pp. 160–169.

[KKL+11] S. Katzenbeisser, n. Koçabas, V. v. d. Leest, A.-R. Sadeghi, G. J. Schrijen, and C. Wachsmann, "Recyclable PUFs: Logically Reconfigurable PUFs." *Journal of Cryptographic Engineering*, pp. 177–186, 2011.

[KKR+12] S. Katzenbeisser, n. Kocabaş, V. Rožić, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Leuven, Belgium: Springer-Verlag, 2012, pp. 283–301.

[Koc13] D. Koch, *Partial Reconfiguration on FPGAs - Architectures, Tools and Applications*. New York, NY: Springer, 2013.

[Koh13] C. Kohn, "Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices, Xilinx XAPP1159," Jan. 2013. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1159-partial-reconfig-hw-accelerator-zynq-7000.pdf

[KSS+09] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Skoric, and P. Tuyls, "Reconfigurable Physical Unclonable Functions - Enabling Technology for Tamper-resistant Storage," in *Proc. International Symposium on Hardware-Oriented Security and Trust (HOST)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 22–29.

[LBGB00] S. Lopez-Buedo, J. Garrido, and E. Boemo, "Thermal Testing on Reconfigurable Computers," *IEEE Design Test of Computers*, vol. 17, no. 1, pp. 84–91, Jan. 2000.

[Lie13] J. Lienig, "Electromigration and Its Impact on Physical Design in Future Technologies," in *Proceedings of the 2013 ACM International Symposium on International Symposium on Physical Design*, ser. ISPD '13. New York, NY, USA: ACM, 2013, pp. 33–40.

[LLG+04] J. Lee, D. Lim, B. Gassend, G. Suh, M. van Dijk, and S. Devadas, "A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications," in *Symposium on VLSI Circuits*, Jun. 2004, pp. 176–179.

[MBKP11] A. Moradi, A. Barenghi, T. Kasper, and C. Paar, "On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs," in *Proc. Conference on Computer and Communications Security.* ACM, 2011, pp. 111–124.

[MCMS10] A. Maiti, J. Casarona, L. Mchale, and P. Schaumont, "A Large Scale Characterization of RO-PUF," in *Proc. International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 94–99.

[Mer14] D. Merli, "Attacking and Protecting Ring Oscillator Physical Unclonable Functions and Code-Offset Fuzzy Extractors," Doctoral dissertation, Technische Universität München, München, 2014.

[MGS13] A. Maiti, V. Gunreddy, and P. Schaumont, "A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions," in *Embedded Systems Design with FPGAs*, P. Athanas, D. Pnevmatikatos, and N. Sklavos, Eds. New York, NY: Springer New York, 2013, pp. 245–267.

[MKP09] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Techniques for Design and Implementation of Secure Reconfigurable PUFs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 1, pp. 5:1–5:33, Mar. 2009.

[MKP11] A. Moradi, M. Kasper, and C. Paar, "On the Portability of Side-Channel Attacks – An Analysis of the Xilinx Virtex 4, Virtex 5, and Spartan 6 Bitstream Encryption Mechanism," 2011. [Online]. Available: https://eprint.iacr.org/2011/391

[MKP12] A. Moradi, M. Kasper, and C. Paar, "Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures: An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism," in *Proc. 12th Conference on Topics in Cryptology.* San Francisco, CA: Springer-Verlag, 2012.

[MOPS13] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski, "Side-Channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix Ii: Facilitating Black-Box Analysis Using Software Reverse-Engineering," in *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA).* Monterey, California, USA: ACM, 2013, pp. 91–100.

[MRV+12] R. Maes, V. Rozic, I. Verbauwhede, P. Koeberl, E. Van der Sluis, and V. van der Leest, "Experimental Evaluation of Physically Unclonable Functions in 65 nm CMOS," in *Proc. of the ESSCIRC*. IEEE, 2012, pp. 486–489.

[MS11] A. Maiti and P. Schaumont, "Improved Ring Oscillator PUF: An FPGA-friendly Secure Primitive," *Journal of Cryptology*, vol. 24, no. 2, pp. 375–397, 2011.

[MS14] A. Maiti and P. Schaumont, "The Impact of Aging on a Physical Unclonable Function," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 9, pp. 1854–1864, Sep. 2014.

[MTV08] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic PUFs from Flip-Flops on Reconfigurable Devices," in *Proc. Benelux Workshop on Information and System Security (WISSec)*, vol. 17, 2008.

[MTV09a] R. Maes, P. Tuyls, and I. Verbauwhede, "A Soft Decision Helper Data Algorithm for SRAM PUFs," in *Proc. International Symposium on Information Theory (ISIT)*, Jun. 2009, pp. 2101–2105.

[MTV09b] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 332–347.

[MVHV12] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Leuven, Belgium: Springer-Verlag, 2012, pp. 302–319.

[New02] R. Newson, "Parameters Behind "Nonparametric" Statistics: Kendall's Tau, Somers' D and Median Differences," *Stata Journal*, vol. 2, no. 1, pp. 45–64, 2002.

[Pan09] Panasonic, "Failure Mechanism of Semiconductor Devices - T04007be-3," 2009. [Online]. Available: http://www.semicon.panasonic.co.jp/en/aboutus/reliability.html

[Pos98] R. Posch, "Protecting Devices by Active Coating," *Journal of Universal Computer Science*, vol. 4, no. 7, pp. 652–668, 1998.

[PPHG14] M. Pehl, A. R. Punnakkal, M. Hiller, and H. Graeb, "Advanced Performance Metrics for Physical Unclonable Functions," in *Proc. International Symposium on Integrated Circuits (ISIC)*. IEEE, Dec. 2014, pp. 136–139.

[PRTG02] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[RJA12] U. Rührmair, C. Jaeger, and M. Algasinger, "An Attack on PUF-Based Session Key Exchange and a Hardware-Based Countermeasure: Erasable PUFs," in *Proc. International Conference on Financial Cryptography and Data Security*. Gros Islet, St. Lucia: Springer-Verlag, 2012, pp. 190–204.

[San13] L. Sanders, "Secure Boot of Zynq-7000 All Programmable SoC, Xilinx XAPP1175," 2013. [Online]. Available: https://www.xilinx.com/support/documentation/application_notes/xapp1175_zynq_secure_boot.pdf

[SC06] P. Sedcole and P. Cheung, "Within-Die Delay Variability in 90nm FPGAs and Beyond," in *IEEE International Conference on Field Programmable Technology, 2006. FPT 2006*, Dec. 2006, pp. 97–104.

[SD07] G. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proc. Design Automation Conference (DAC)*, 2007, pp. 9–14.

[SFKP15] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "FPGA Trojans Through Detecting and Weakening of Cryptographic Primitives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1236–1249, Aug. 2015.

[SGB+10] O. Sander, B. Glas, L. Braun, K. Müller-Glaser, and J. Becker, "Intrinsic Identification of Xilinx Virtex-5 FPGA Devices Using Uninitialized Parts of Configuration Memory Space," in *Proc. International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2010, pp. 13–18.

[SHO08] Y. Su, J. Holleman, and B. Otis, "A Digital 1.6 pJ/bit Chip Identification Circuit Using Process Variations," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 69–77, Jan. 2008.

[Sim91] G. Simmons, "Identification of Data, Devices, Documents and Individuals," in *Proc. 25th Annual International Carnahan*

Conference on Security Technology. IEEE, Oct. 1991, pp. 197–218.

[SW12] S. Skorobogatov and C. Woods, "In the Blink of an Eye: There Goes Your AES Key," 2012. [Online]. Available: https://eprint.iacr.org/2012/296

[SWC10] E. Stott, J. S. Wong, and P. Y. Cheung, "Degradation Analysis and Mitigation in FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL).* IEEE, Aug. 2010, pp. 428–433.

[TCH93] J. Tao, N. Cheung, and C. Hu, "Metal Electromigration Damage Healing Under Bidirectional Current Stress," *IEEE Electron Device Letters*, vol. 14, no. 12, pp. 554–556, Dec. 1993.

[TVG+13] A. Thaduri, A. K. Verma, V. Gopika, R. Gopinath, and U. Kumar, "Reliability Prediction of Semiconductor Devices Using Modified Physics of Failure Approach," *International Journal of System Assurance Engineering and Management*, vol. 4, no. 1, pp. 33–47, Mar. 2013.

[vdLSHT10] V. van der Leest, G.-J. Schrijen, H. Handschuh, and P. Tuyls, "Hardware Intrinsic Security from D Flip-Flops," in *Proc. ACM Workshop on Scalable Trusted Computing (STC).* ACM, 2010, pp. 53–62.

[WA08] D. Wolpert and P. Ampadu, "Normal and Reverse Temperature Dependence in Variation-Tolerant Nanoscale Systems with High-K Dielectrics and Metal Gates," in *Nano-Net.* Springer, 2008, pp. 14–18.

[WDDJ71] R. Wang, J. Dunkley, T. A. DeMassa, and L. F. Jelsma, "Threshold Voltage Variations with Temperature in MOS Transistors," *IEEE Transactions on Electron Devices*, vol. 18, no. 6, pp. 386–388, Jun. 1971.

[WG14] A. Wild and T. Güneysu, "Enabling SRAM-PUFs on Xilinx FPGAs," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014.

[WH11] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston: Addison Wesley, 2011.

[WHP14] F. Wilde, M. Hiller, and M. Pehl, "Statistic-Based Security Analysis of Ring Oscillator PUFs," in *Proc. International Symposium on Integrated Circuits (ISIC)*, Dec. 2014, pp. 148–151.

[WST95] F. Willems, Y. Shtarkov, and T. Tjalkens, "The Context-Tree Weighting Method: Basic Properties," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 653–664, May 1995.

[Xil13] Xilinx, "Zynq-7000 All Programmable SoC Overview," Dec. 2013. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[Xil14a] Xilinx, "7 Series FPGAs Configurable Logic Block User Guide," Nov. 2014. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[Xil14b] Xilinx, "Zynq-7000 All Programmable SoC Technical Reference Manual," Feb. 2014. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[Xil15] Xilinx, "Vivado Design Suite User Guide - Partial Reconfiguration," Nov. 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug909-vivado-partial-reconfiguration.pdf

[YD10] M.-D. Yu and S. Devadas, "Secure and Robust Error Correction for Physical Unclonable Functions," *Design Test of Computers, IEEE*, vol. 27, no. 1, pp. 48–65, 2010.

[Zeg10] B. V. V. Zeghbroeck, *Principles of Semiconductor Devices and Heterojunctions*, 1st ed. Upper Saddle River, N.J.; London: Prentice Hall, May 2010.

[Zha13] S. Zhang, "Delay Characterization in FPGA-Based Reconfigurable Systems," Master Thesis, Universität Stuttgart, 2013.

[ZKN+06] S. Zafar, Y. Kim, V. Narayanan, C. Cabral, V. Paruchuri, B. Doris, J. Stathis, A. Callegari, and M. Chudzik, "A Comparative Study of NBTI and PBTI (Charge Trapping) in SiO2/HfO2 Stacks with FUSI, TiN, Re Gates," in *Proc. Symposium on VLSI Technology.* IEEE, 2006, pp. 23–25.

[ZWF+15] C. Zhou, X. Wang, R. Fung, S. J. Wen, R. Wong, and C. H. Kim, "High Frequency AC Electromigration Lifetime Measurements from a 32nm Test Chip," in *2015 Symposium on VLSI Technology (VLSI Technology)*, Jun. 2015, pp. T42–T43.