Physical layer security: authentication and ciphering using physical unclonable functions on FPGA

Oscar Van Slijpe

Master thesis submitted under the supervision of
Dr. Pr. Jean-Michel Dricot

The co-supervision of
Dr. Pr. Dragomir Milojevic

In order to be awarded the Master's Degree in
Electronics and Information Technology Engineering

Academic year
2022-2023

UNIVERSITÉ LIBRE DE BRUXELLES

Reserve au secretariat

Memoire reussi[1] **OUI NON**

Consultation du memoire/travail de fin d'etudes

Je soussigne

NOM : *VAN SLIJPE*
PRENOM : *OSCAR*
TITRE DU TRAVAIL : *Physical layer security: authentication and ciphering using physical unclonable functions on FPGA*

AUTORISE ~~**REFUSE**~~ la consultation du present memoire/travail de fin d'etudes par les utilisateurs des bibliotheques de l'Universite libre de Bruxelles.

Si la consultation est autorisee, le soussigne concede par la presente a l'Universite libre de Bruxelles, pour toute la duree legale de protection de l'oeuvre, une licence gratuite et non exclusive de reproduction et de communication au public de son oeuvre precisee ci-dessus, sur supports graphiques ou electroniques, afin d'en permettre la consultation par les utilisateurs des bibliotheques de l'ULB et d'autres institutions dans les limites du pret inter-bibliotheques.

Fait en deux exemplaire, **Bruxelles**, le *21/08/2023*
Signature

O.VanSlijpe

[1]Biffer la mention inutile

Abstract

Physical Unclonable Function (PUF) is a promising way to redefine how we implement security in modern systems. Transient Effect Ring Oscillator PUF (TERO-PUF) is a recent method that shows good performance with compact design. In this work, we propose two TERO-PUF implementations on Artix-7 Field-Programmable Gate Array (FPGA), the first using one slice per cell and the second using one Configurable Logic Block (CLB) per cell. The more compact one doesn't reach state of the art performance, while the second gives performance similar to the existing TERO-PUF implementation on other devices. This shows the limitations of the size of this PUF. We make use of Error Correction Code (ECC) to further improve the reliability of the response, and we have added a Secure Hash Algorithm (SHA) to generate a key that can be used for encryption demonstration.

Keywords: Physical Unclonable Function; Transient Effect Ring Oscillator PUF; Field-Programmable Gate Array; Artix-7; Secure hardware design; Cryptographic primitive

Oscar Van Slijpe
Electronics and Information Technology Engineering
Physical layer security: authentication and ciphering using physical unclonable functions on FPGA
2022-2023

List of Acronyms

AES	Advanced Encryption Standard
APUF	Arbiter PUF
BA	Bit-Aliasing
BCH	Bose Chaudhuri and Hocquenghem code
BER	Bit Error Rate
BPUF	Butterfly PUF
BST	Bit-Self Test
BST-APUF	Bit-Self Test Arbiter PUF
CLB	Configurable Logic Block
CRP	Challenge-Response Pair
DD-PUF	Delay Differential PUF
ECC	Error Correction Code
FF-APUF	Feed-Forward Arbiter PUF
FFPUF	Flip-Flop PUF
FPGA	Field-Programmable Gate Array
IC	Integrated Circuit
IoT	Internet of Things
IP	Intellectual property
LSFR	Linear Shift Feedback Register
LUT	Look-Up Table
M-ROPUF	Mux-based Ring Oscillator PUF
ML	Machine Learning
MUX	Multiplexer

NIST	National Institute of Standards and Technology
OC-PUF	Oscillator Collapse PUF
PLL	Phase-locked loop
POWF	Physical One-Way Function
PRNG	Pseudo-Random Number Generator
PUF	Physical Unclonable Function
RE	Reliability
RO-PUF	Ring Oscillator PUF
RWC-SRAMPUF	Read-Write Collision SRAM-PUF
SHA	Secure Hash Algorithm
SPRF	Silicon Physical Random Function
SPUF	Silicon PUF
SR-Latch	Set-Reset Latch
SRAM	Static Random-Access memory
SRAM-PUF	SRAM PUF
TERO	Transient Effect Ring Oscillator
TERO-PUF	Transient Effect Ring Oscillator PUF
UART	Universal Asynchronous Receiver Transmitter
UF	Uniformity
UQ	Uniqueness
VHDL	VHSIC Hardware Description Language
XDC	Xilinx Design Constraints
XOR-APUF	XOR Arbiter PUF

Contents

Chapter 1

Introduction

With the advent of new technologies such as Internet of Things (IoT) or Smart Grid, there is an increasing amount of data being collected and transmitted between devices that could be physically available to anyone, and which sometimes cannot be secured using classical security techniques due to the small computing resources available. Therefore, alternative security methods where a device is able to generate its own security, with a system that has a small footprint compared to its main system, are highly desirable.

One promising candidate, PUF, are physical systems that cannot be replicated identically even if you know exactly how they were created. This means that the output of the function is determined by some intrinsic properties of the system itself, which either cannot be known or cannot be manufactured with sufficient accuracy to give the same responses. Furthermore, if these unknown properties are random, then the response to a given input will also appear to be completely random. The responses are extracted from the system itself and not stored in some kind of internal memory, making them less accessible.

PUFs have been studied in various forms since 2001 [35] and there are some applications using them such as Intellectual property (IP) protection for FPGA [20], authentication protocols for IoT [33, 17, 6] or security for chiplet devices [15].

In this work, we present an implementation of a Transient Effect Ring Oscillator PUF (TERO-PUF) for the first time on Artix-7, with state-of-the-art performance and the possibility to generate a usable key using SHA. The different design possibilities are discussed and two implementations are tested to compare the effect of the size of the TERO cells.

Chapter 2

Physical Unclonable Function: classification and characterisation

In this section, we will discuss in more detail about the typical design of PUF, how they work, and how we can evaluate their performance.

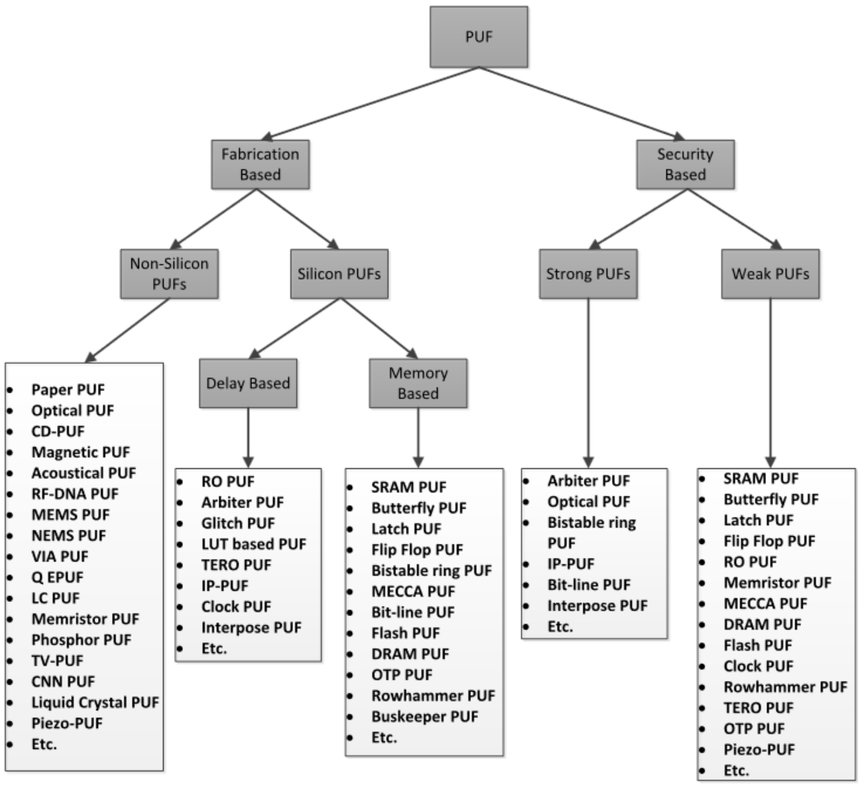


Figure 2.1: PUF classification (from [2])

The PUF concept was introduced in 2001 by [35, 36], first addressed as Physical One-Way Function (POWF), as a comparison to the one-way functions based on number theory used in cryptography. The aim is to exploit randomness in the physical properties of a system to generate a binary sequence. As a first proof of concept, they used a transparent epoxy containing bubbles as the physical medium, which was 'read' by a laser at a specific angle to generate a pattern from the interference between the beam and the epoxy/bubble.

This has led to a variety of different designs which can be classified in two ways, as shown in figure 2.1: how they work physically (manufacturing based) and how they should be used (safety based).

2.1 Fabrication based classification

The non-silicon PUFs are PUFs implemented on systems such as optical, mechanical or magnetic systems. These will not be covered in this study and we will concentrate only on Silicon PUF (SPUF). The first introduction of SPUF was made by [19] in 2002, originally called Silicon Physical Random Function (SPRF). The advantage of such PUFs is the ease with which they can be integrated into other electronic circuits.

In general, electronic circuit models are expected to have some small number of defects in the silicon due to the limited precision of the manufacturing process. The defects are required to have a negligible effect on the device's operation, and this is a constraint that designers consider while designing such circuits. However, these effects can still be exhibited using appropriately designed circuits. These defects cannot be replicated with a similar precision manufacturing process and are often considered to be unknowable (in the sense that to be able to observe them directly one would generally need to destroy the Integrated Circuit (IC)) and appear to be random. This means that they are properties that can be used to implement a PUF on an electronic circuit.

A wide range of methods can be utilised to expose these tiny manufacturing errors and to turn them into usable responses, i.e. binary values. There are two classes of methods based on how they exploit randomness: delay-based PUF and memory-based PUF. Designs using more than one PUF technique are called hybrid PUF.

Delay-based PUFs exploit small variation in the delay of electronic signals between two similar paths. In an ideal scenario, the two paths should produce identical delays, but in reality, there will always be slight differences in delay. These differences are typically small, and their influence on a typical electronic circuit is minimal. Delay-based PUF are electronic circuits designed to amplify these differences to the point where they dominate and determine the output response, ideally without any other external factor influencing the result.

Memory-based PUFs exploit transient state of memory elements during power-up and then observe which stable states they eventually transition to, resulting in the generation of the response.

2.2 Security based classification

A second way of classifying PUF is to separate them on the basis of the size of their Challenge-Response Pair (CRP) space. Some PUFs provide only one possible response. In this case, it can be considered that there is no input since the challenge corresponding to the only response is intrinsic to the PUF itself. Whereas, other PUFs have a large set of Challenge-Response Pair (CRP) spaces. This classification is helpful in determining how a PUF can be utilized for security applications.

Weak PUF are the PUFs with a small CRP space, with the extreme case being a PUF with only one response (the challenge is therefore not required as an input, as it is implicit in the PUF itself). These PUFs must be used in a secure environment and the CRP should not be revealed; otherwise, an attacker could record the entire CRP, compromising the security of the system. Thefore, these PUFs can only be used for applications such as key extraction, truly random number generation, or identification.

Strong PUF refers to PUFs with a large CRP space. In this case, revealing a few CRP does not compromise the PUF because there are still many of them that are unknown (in the ideal case where CRPs are not correlated in any way so that the remaining CRP cannot be predicted). Strong PUFs are ideal for use in authentication processes.

2.3 Example of FPGA-based PUFs

2.3.1 Arbiter PUF

Arbiter PUF (APUF) are the most common and most studied of the PUFs. The concept was introduced in 2002 [19] and many different versions have been created since then. It consists of two paths that travel through successive stages, where the 2 paths have 2 possible configurations depending on the input. The output is generated based on which path is globally the fastest due to the imperfections in the IC. This can be achieved by using successive stages of Multiplexer (MUX) that select which path the signal will take, as shown in figure 2.2. Thus, this is a delay-based PUF, and since the number of CRP doubles for each additional stage, it is a strong PUF.

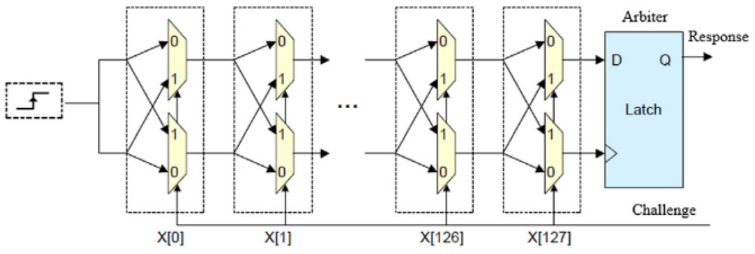


Figure 2.2: Arbiter PUF diagram [2]

One concern with this design is the possibility to model the internal delay of each stage configuration to be able to predict the entire CRP space. A proposed countermeasure for this issue was presented in 2004 [27], where feed-forward signals from the intermediate stage are used as input for some of the following stages. This approach is commonly known as a Feed-Forward Arbiter PUF (FF-APUF).

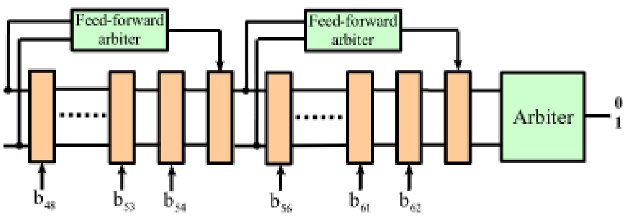


Figure 2.3: FF-APUF [27]

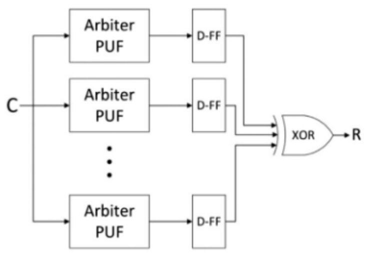


Figure 2.4: XOR-APUF [22]

An alternative approach is to introduce an XOR layer at the output to enhance the confidentiality of the internal properties. This was studied by [37] in 2007, called a XOR Arbiter PUF (XOR-APUF).

Since then, several other features have been developed and tested. To enhance the reliability, [23] has proposed a Bit-Self Test Arbiter PUF (BST-APUF). A more recent example is the work of [1] in 2022, which obfuscates the challenges and adds a majority voting system before the XOR. A comprehensive compilation of Arbiter APUF was done by [22] in 2021.

2.3.2 Ring oscillator PUF

The design proposed by [19] does not directly evaluate the delay difference between two configurable paths. Rather, the signals were looping back into the same configurable path, thus forming an oscillating circuit. The frequency difference served as the measured parameter for determining the output. This was done in order to reduce the influence of the process variation on the output parameter.

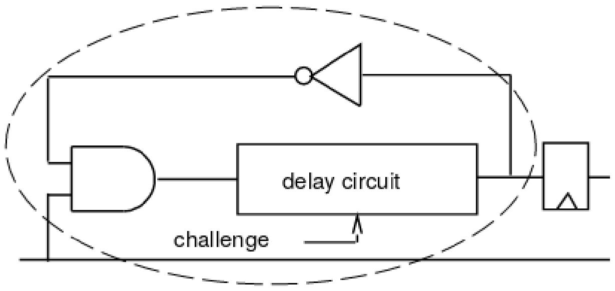


Figure 2.5: Oscillating circuit using APUF for delay [19]

This concept led to the development of another PUF design: the Ring Oscillator PUF (RO-PUF). An oscillating circuit is designed using a basic delay element and replicated to form a collection of oscillators. All oscillators are compared to each other, and a '0' or a '1' is generated based on the oscillator with the fastest frequency.

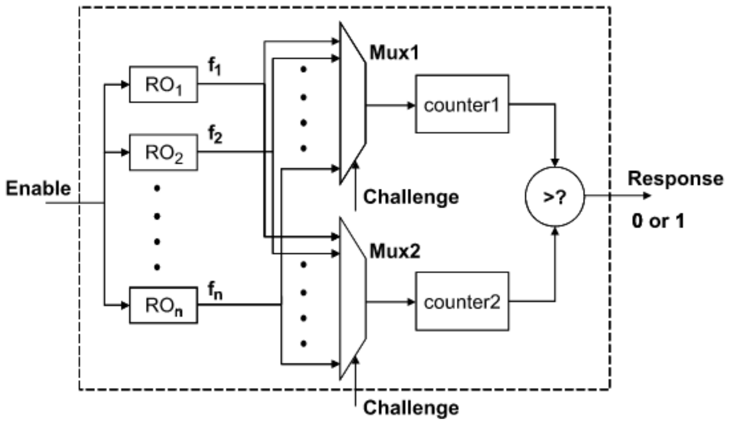


Figure 2.6: RO-PUF diagram [29]

This concept was first studied in 2009 by [29] using a SRAM based FPGA (Spartan 3), then in 2017 by [34] with a flash based FPGA (Microsemi Smart-Fusion2). In both cases, the oscillator circuit is made up of an AND gate (for control) and an odd number of NOT gates (for delay).

In 2022, [41] demonstrated the possibility of replacing the inverter-based oscillator with a MUX-based one, resulting in the creation of the Mux-based Ring Oscillator PUF (M-ROPUF).

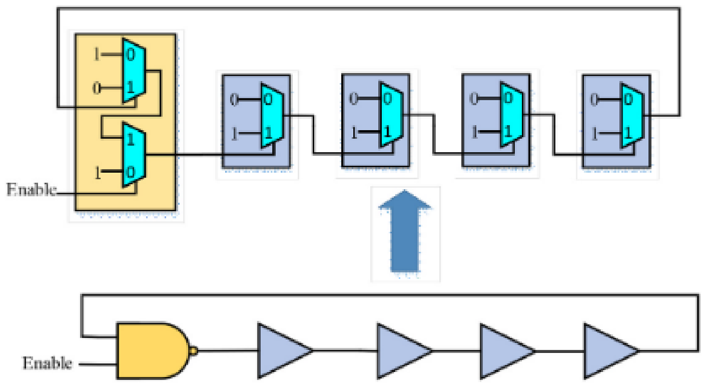


Figure 2.7: M-ROPUF [41]

Similarly to the APUF, RO-PUF are delay-based PUFs. They can be considered weak or strong PUF depending on the oscillator circuit designs as well as the structure used to generate the output from the collection of oscillators.

2.3.3 Transient effect ring oscillator PUF

The RO-PUF concept has been proven to be often very vulnerable to physical attack and frequency analysis by [32] in 2009, [7] in 2010, [5] in 2012 and [8] in 2015. Moreover, [7] also proved that the different oscillator cells of RO-PUF are sensitive to locking phenomena (cells are not independent).

To avoid the locking phenomena and make the frequencies less accessible, a new design using transient effect has been proposed by [9] in 2014.
In Transient Effect Ring Oscillator PUF (TERO-PUF), the cells oscillation occurs in an unstable state and both the stabilisation time and the final number of oscillations are unpredictable. This can be achieved with the circuit represented in figure 2.8, which is an Set-Reset Latch (SR-Latch) with the set and reset signal combined. This design is also sometimes called Oscillator Collapse PUF (OC-PUF)

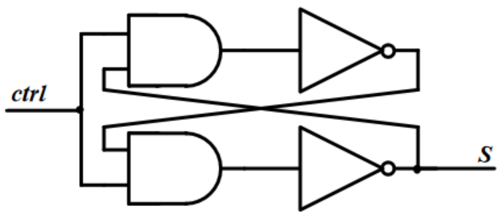


Figure 2.8: TERO-PUF diagram [9]

The first design by [9] was tested on an Altera Cyclone II then by [30] on Altera Cyclone V and Xilinx Spartan 6 FPGAs.

The oscillator circuit has also been build using SR-Latch in [21] and D-Latches in [14], called Delay Differential PUF (DD-PUF).

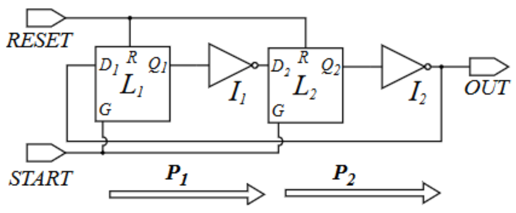


Figure 2.9: DD-PUF diagram [14]

The idea of a programmable delay line using LUT input-output latency was re-introduced in 2018 by [3] on Spartan 3E and a high CRPs implementation

(2^{3N}) has been proposed by [42] in 2020 on an Altera DE2-115.

In 2019, [38] studied the robustness of TERO-PUF design against side channels analysis. While it shows that TERO-PUF can still be vulnerable to some extent, it has also proposed a few countermeasures to minimise it. TERO-PUF still appear as one of the most promising delay based PUF.

2.3.4 SRAM PUF

In parallel to the evolution of delay based SPUF, the memory based SPUF development started with the SRAM PUF (SRAM-PUF) from [20] in 2007.
When an SRAM element is powered up, its initial state is not one of the two stable ones ('1' and '0') but an unstable equilibrium in between. If none of the two stable states is forced, the state in which it will end up is unpredictable and is a function of the exact properties of the silicon.

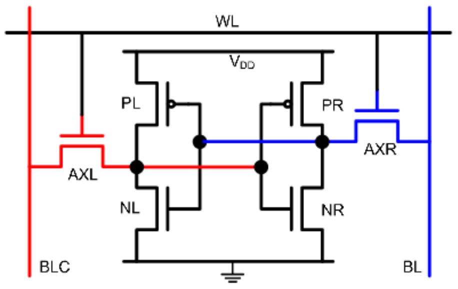


Figure 2.10: SRAM cell [20]

This design has to advantage of using standard SRAM chips, which are typically available in most of electronic board.
There have been multiple implementations since then, such as [10] in 2008, who built a Pseudo-Random Number Generator (PRNG) from it, or [40] in 2008 that used the variation of the PUF due to the temperature to create a temperature sensor with secured data values.

To eliminate the requirement of power up the SRAM each time the PUF needs to be evaluated, [11] proposed in 2022 to introduce a Read-Write Collision SRAM-PUF (RWC-SRAMPUF), violating in purpose the recommended operation of the SRAM.

2.3.5 Butterfly PUF

For SRAM-PUF, the FPGA needs to support uninitialised SRAM memory, which is not always the case. To remove this requirement, [26] introduced their Butterfly PUF (BPUF) in 2008. The concept is the same that the SRAM-PUF but the memory structure used is a cross-coupled latch that can be forced to an unstable state.

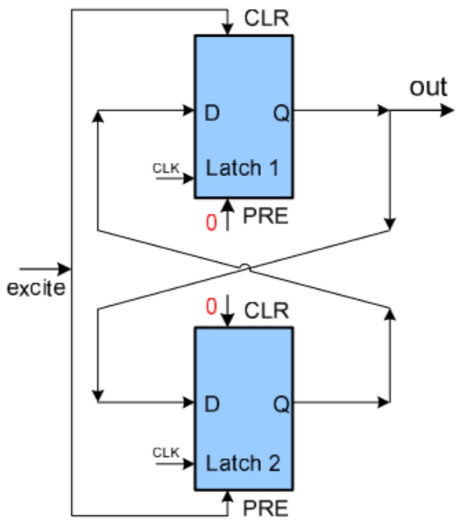


Figure 2.11: BPUF [26]

2.3.6 Flip-Flop PUF

Similarly to the BPUF, the Flip-Flop PUF (FFPUF) aims at removing the requirement on specific hardware from the SRAM-PUF. Introduced in 2008 by [28], this PUF read the power-up value from D Flip-Flop.

2.3.7 Hybrid PUF

It is also possible to combine two or more PUF techniques to have a more complex system that exploits the random defects in multiple ways. This can be useful to obtain a PUF that is more difficult to predict using Machine Learning (ML) techniques. For example, [16] has proposed in 2022 an implementation using both APUF and BPUF together.

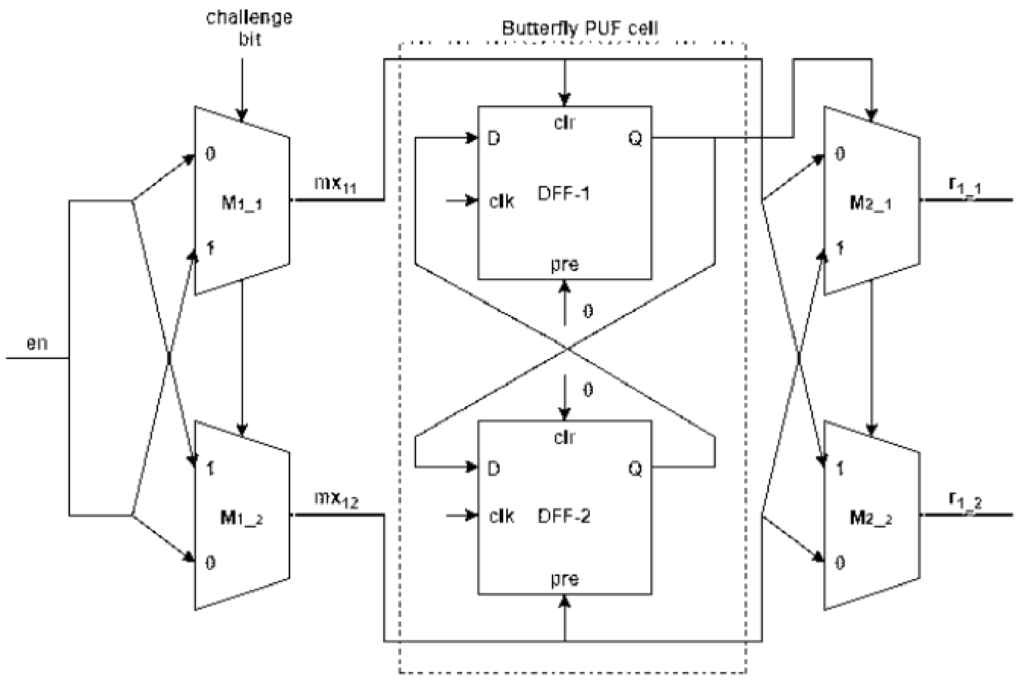


Figure 2.12: Hybrid PUF diagram [16]

2.4 Metrics

To characterise and compare the performance of a PUF, multiple metrics have been used. In this section, we will see the main metrics used to evaluate the performance of PUF for security applications. The metrics that are computed on the set of responses of a single device are called intra-device metrics, and those that characterise the difference between the responses of different devices are called inter-device metrics.

2.4.1 Uniformity

The Uniformity (UF) is a measure of distribution between "0" and "1" bits in the response of a given device (intra-device). Ideally equal to 50% to maximise the entropy.

$$UF = \frac{1}{n}\sum_{i=1}^{n} r_i \times 100\% \tag{2.1}$$

Where n is the size of the response and r_i is the ith bit of it.

2.4.2 Reliability

The Reliability (RE) is a measure of the stability of the response of a given device (intra-device). Ideally equal to 100% meaning that the response's bits are always the same.

$$RE = 100\% - \frac{1}{s}\sum_{i=1}^{s} \frac{\mathrm{HD}\left(X_{ref}, X_i\right)}{n} \times 100\% \tag{2.2}$$

Where s is the number of responses generated, $HD(X_{ref}, X_i)$ is Hamming distance between the ith response generated and the reference response.

The Bit Error Rate (BER) can be obtained using only the right term of the sum.

This can be further improved using Error Correction Code (ECC) or BST as done in [23, 24].

2.4.3 Bit-Aliasing

The Bit-Aliasing (BA) is a measure of how likely a bit value is to be '1' over all the devices (inter-device). This can reveal a bias in the implementation i.e.

the response's bits are influenced by the implementation itself. Ideally, the distribution of bit-aliasing over different devices should follow a normal distribution centred around 50%.

$$BA_j = \frac{1}{k}\sum_{j=1}^{k} r_{i,j} \times 100\% \tag{2.3}$$

Where n is the number of bits of the response, k is the number of devices and $r_{i,j}$ is the ith bit of the jth device response.

2.4.4 Uniqueness

The Uniqueness (UQ) is a measure of how different the responses of different devices are (inter-device). Ideally equal to 50%.

$$UQ = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} \frac{\mathrm{HD}\left(X_i, X_j\right)}{n} \times 100\% \tag{2.4}$$

Where k is the number of devices, $\mathrm{HD}\left(X_i, X_j\right)$ the Hamming distance between the response word of device i and j.

2.5 State of the art

The metrics just described are useful to compare the performance of different implementations. The table 2.1 reports the performance of some of the FPGA implementation describes up to now. Not all studies have used the 4 metrics described here, and the implementations are done using different FPGA. The goal of this table is not to have a complete collection of PUFs implementation, only to have an overview of the typical performance that can be reached currently.

method	**RE**	**UF**	**UQ**	**BA**	**device**	**Ref**
APUF	99.55%	51.84%	46.21%	-	Artix-7	[1]
FF-APUF	90.2%	-	38%	-	ASIC	[27]
XOR-APUF	99.41%	50.73%	48.69%	-	Artix-7	[1]
BST-APUF	99.99%	-	49.1%	50.3%	Artix-7	[23]
RO-PUF	100%	-	45.9%	-	Spartan-3	[29]
RO-PUF	99.19%	51.01	47.86%	51.01%	Artix-7	[13]
BST-RO-PUF	99.99%	46.78%	48.64%	-	Artix-7	[24]
M-ROPUF	-	51.17%	49.52%	54.41%	Kintex-7	[41]
TERO-PUF	97.4%	-	48.5%	-	Spartan-6	[31]
	98.2%	-	47.6%	-	Cyclone-V	[31]
PDL-TERO-PUF	98.8%	-	49.32%	-	Spartan-3	[3]
SRAM-PUF	98.2%	-	-	-	Virtex-7	[40]
RWC-SRAMPUF	98.92%	55.38%	37.36%	46.89%	Artix-7	[11]
FFPUF	99%	49.2%	-	48.96%	Artix-7	[25]

Table 2.1: PUF implementations on FPGA

Chapter 3

Objectives

Different PUFs methods were described in chapter 2. In particular, we saw the evolution of delay-based SPUF from APUF to TERO-PUF. While APUF has been studied multiple times, on different devices with diverse variations, TERO-PUF is more recent and only some devices and variations have been studied until now.

Therefore, this thesis proposes a TERO-PUF implementation on Artix-7 and tested on 33 boards (Basys-3), which, as far as we know, is lacking to this day. Furthermore, we aimed at a design that can easily be used to demonstrate the PUF usage for ciphering.

This work follows the one from [13] in 2021 who studied an RO-PUF on the same FPGA and made available the VHDL implementation including the framework developed to control the PUF and communicate between the device and a computer. We reuse the available framework as a starting point to go further in the characterisation of the implementation and the demonstrative possibilities.

The design of the TERO cells and the TERO-PUF itself is described in the chapter 4, followed by the VHDL implementation for Artix-7 on chapter 5.

In chapter 6, the TERO cell's behaviour will be validated and then used in the TERO-PUF implementation. Parameters that are not fixed during design or implementation will have their impact on performance studied and the value that appear as the best will be chosen, with a proposition of explanation when needed. The final performance will be computed using the 33 Basys-3 board, and compared to other PUF. Finally, once the implementation is validated, the usage of features like ECC, Secure Hash Algorithm (SHA) 256 and ciphering will be demonstrated.

Chapter 4

TERO-PUF: Concepts and general design

In this chapter, the concept behind TERO-PUFs will be discussed more in depth. In particular, how the randomness is exhibited using the TERO cells and how to generate the bit response from those them.

As we saw in chapter 2, TERO-PUF is a delay-based PUF. It uses unstable oscillator circuits to exhibit the impact of the random defects in the IC. In the following section, the two main parts of the TERO-PUF operation will be discussed: the TERO cell structure and the method to generate a response from them.

4.1 TERO cells

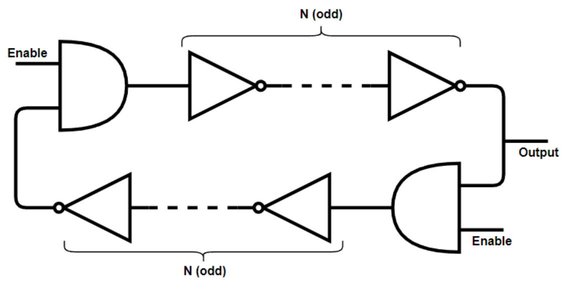
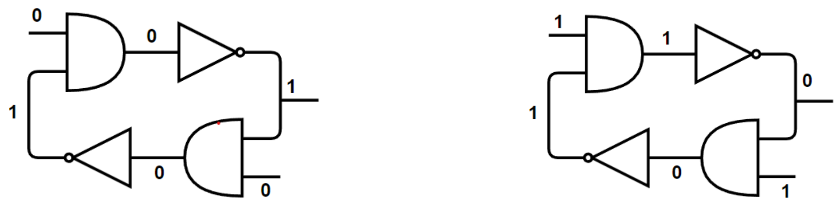


Figure 4.1: TERO cell logical representation

The structure of the TERO cell (represented in figure 4.1) is related to the SR-Latch but where the *set* and *reset* inputs are used simultaneously (the *enable* signal on the diagram). The two opposite AND gates serve as the entry point for the signals when the cell is enabled, and an equal and odd number of NOT gates on each branch create a delay inside the loop. The initial state (when *enable* is low) is stable, as represented in figure 4.2(a).

Once *enable* is set to high, this circuit becomes an oscillator with two signals propagating inside the loop, which toggles the output signal. Due to the imperfect nature of any real IC, one signal will eventually catch up with the other, they will cancel each other, stopping the oscillation and reaching one of the two stable states, represented in figure 4.2(b). The final state depends on which signal catches the other. At this point, the cell will stay in this state until the *enable* signal falls, which will reset the cell to the initial state through the AND gates. The number of NOT gates on each branch (noted N) is used as a parameter to choose the length of the propagation path inside the cell but needs to remain odd.



(a) Initial state (b) Final state

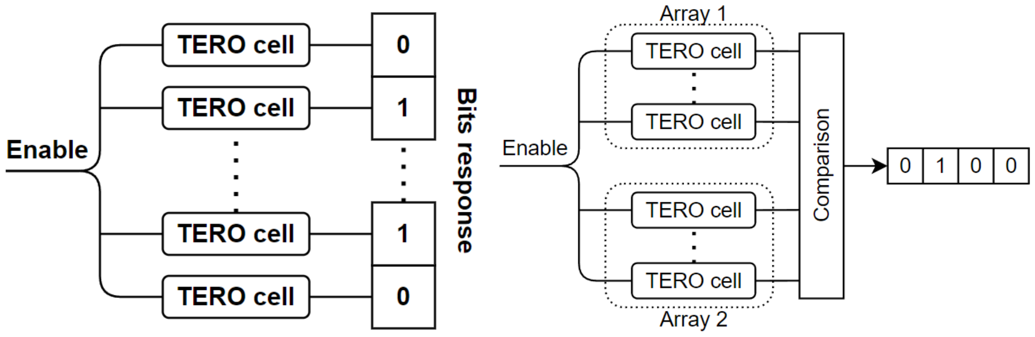
Figure 4.2: TERO cell states (N=1)

The effect of the random defect of the IC on this circuit can be observed in two ways: the final state reached after stabilisation and the number of oscillations reached before this stabilisation. These two properties can be used to generate bits from this structure.

4.2 Response generation

The PUF needs to generate a sequence of bits from the TERO cells, using either the final state or the final number of oscillations. To do so, the simplest method could consist of using the final state of the cell directly as the bit response since it has only two possible states. This is represented in the figure 4.3a.

This has the advantage of being very simple to implement. However, the issue with this technique is that the cells need to be perfectly reliable or the bits would not be stable. It has been shown in [9, 31] that some cells take much more time to stabilise, to the point where they can be considered as "unstable" and will produce an unreliable bit. An additional drawback is that the number of bits that can be generated is equal to the number of TERO cells. Therefore, the resource usage for this method scales linearly with the size of the response.



(a) Direct (b) Comparison

Figure 4.3: Bit generation method

Another approach (represented in figure 4.3b) is to compare the number of oscillations of two cells after stabilisation. A '1' is generated if the first cell has a higher number of oscillations, and a '0' is generated otherwise. This is the method used by [13].

This method is less sensitive to the cell's exact number of oscillations in terms of the reliability of the response as long as the number of oscillations of the different cells is clearly spread. By splitting K cells into two arrays for the comparison, the number of bits that can be generated is equal to $\left(\frac{K}{2}\right)^2$. This scales quadratically with the size of the response, which is better than the first method. One can notice that equality between the two cells will always generate a '0' in this method. This could degrade the uniformity of the response if too many equalities occur and this is something that will need to be tested once the PUF is implemented.

A third design, the gray coding response generation, used in [31], is to subtract the number of oscillations of two cells, apply a Gray coding on this value then select specific bits from this code to create a part of the response. This has the disadvantage that the bits need to be analysed to determine which ones are the more stable in order to select them to generate a chip ID that is reliable. This also adds additional elements to the design compared to the second method.

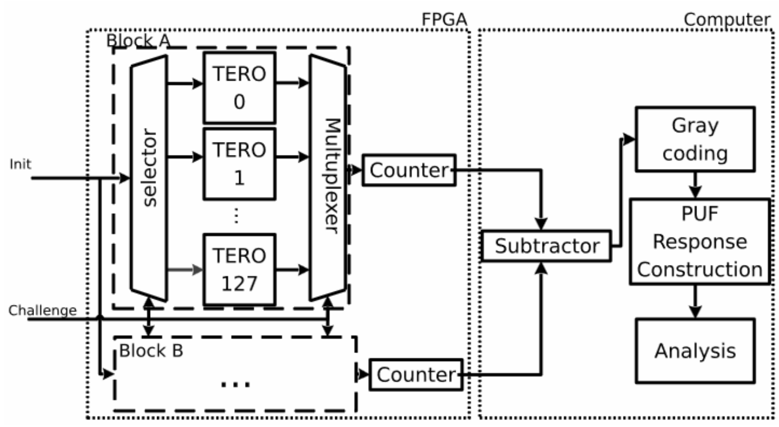


Figure 4.4: Gray coding response generation (figure from [31])

The second method is chosen because it has a small footprint while producing a response with good reliability. Furthermore, it also ensures higher compatibility with the implementation of [13].

To maximise the size of the response, each cell should be compared to every cell of the opposite array. Instead of storing all the possible combinations in memory, [13] has proposed to use a Linear Shift Feedback Register (LSFR) circuit to compute them as needed. Indeed, a such circuit will cycle through the entire possible $2^N - 1$ values exactly once before returning to the initial state (where N is the size of the register). The first half bits of the LSFR can be used as a selection for the first array, and the remaining bits for the second array.

This way, all the possible combinations are covered except the 0-0 one because the state with only '0' is a forbidden state (the LSFR would stay in this state forever), reducing the size of the response bit one. For a large enough number of cells, this impact will become negligible. Moreover, this implementation takes very little hardware resources and scale better than storing all the combination in memory. For those reasons, this LSFR will also be used in this implementation. The new size of the PUF response is therefore equal to $\left(\frac{K}{2}\right)^2 - 1$.

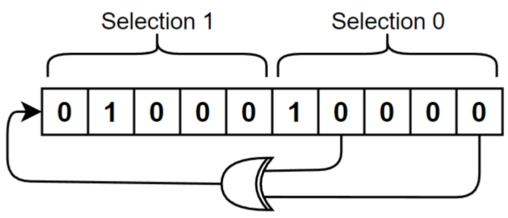


Figure 4.5: 10-bits LSFR

4.3 TERO-PUF operation

We can now fully describe the operation of our TERO-PUF: the TERO cells are composed of 2 AND gates and $2 \times N$ NOT gates (N is odd). K cells are split into two arrays (K need to be even). Each possible pair (except the '0-0' one) is selected in turn using an LSFR. For each pair, the two selected cells are enabled at the same time along with a reference counter connected to the clock, and the number of oscillations is recorded. Once the reference counter reaches the number of clock cycles corresponding to the desired acquisition time, the number of oscillations is compared and one bit of the response is generated. Then the cells and the reference counter are disabled, which reset their state, and the LSFR is shifted to generate the next combination to be selected. $\left(\frac{K}{2}\right)^2 - 1$ bits are generated in this way. The usage of the reference counter imposes that the acquisition time can only be a multiple of the clock period ($\frac{1}{f}$)

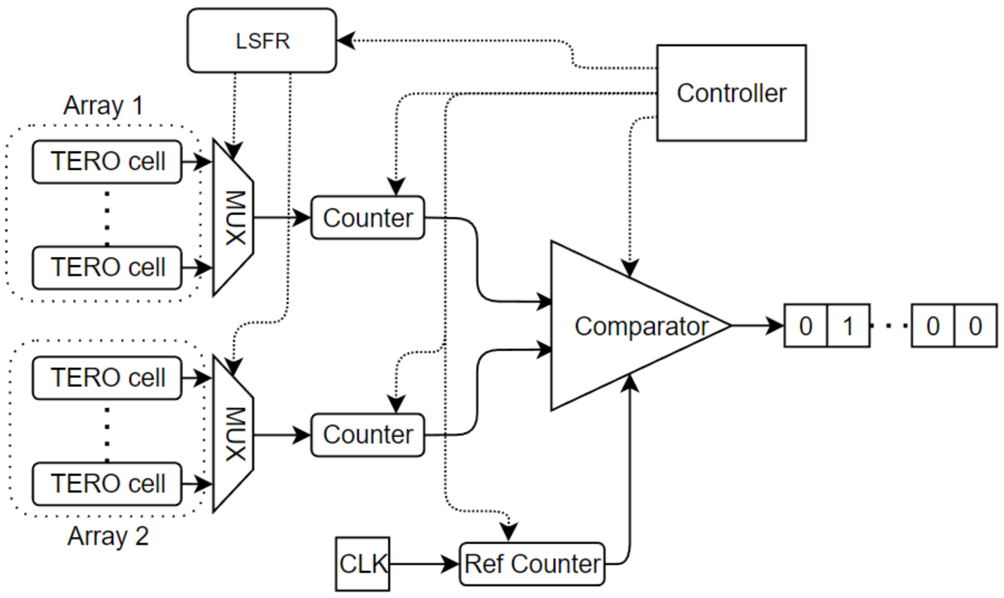


Figure 4.6: TERO-PUF block diagram

Chapter 5

FPGA implementations and characterisation

In this chapter, two VHDL implementations for the ARTIX-7 FPGA will be proposed, based on the concept presented in the previous chapter. The feature developed for demonstrative purposes will also be presented more in-depth. Finally, a report of the FPGA resources usage of the BASYS-3 board that will be used for testing in the next section will be provided.

5.1 Implementation

In this section, we will first study the constraints that need to be respected to apply the proposed design to an FPGA implementation for ARTIX-7. This will add restrictions to the design and will lead us to two possible implementations.

Artix-7 Configurable Logic Block (CLB) a made up of two slice of 4 LUTs, highlighted on green in figure 5.1 and an inter-clb routing block (magenta) for the interface with the global routing of the . The internal routing can be programmed between LUTs as well.

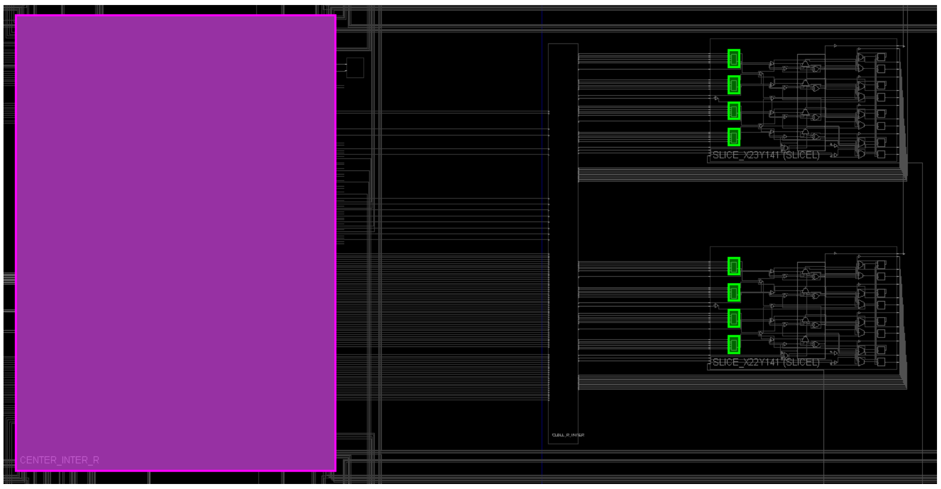


Figure 5.1: Artix-7 CLB

5.1.1 Constraints

The behaviour of the cells is very sensitive, not only to random defects in the IC but to any difference both in terms of implementation (length of the loop, placement, etc...) and external factors (ageing, temperature, voltage). Therefore, to ensure that the PUF response is generated from the random defects with as little bias, the cell implementation should be as identical as possible. To achieve this, the Xilinx Design Constraints (XDC) are used to manually specify some properties of specific parts of the design.

By default, Vivado will try to optimise the design without changing the logical behaviour of the circuit. However, since the cell's exact behaviour is based on random defects, it can not be correctly interpreted by Vivado and any optimisation will most likely break the design. Therefore, the first constraint used is the **DONT_TOUCH** to tell Vivado to not try to optimise the design of the cells so that it remains untouched.

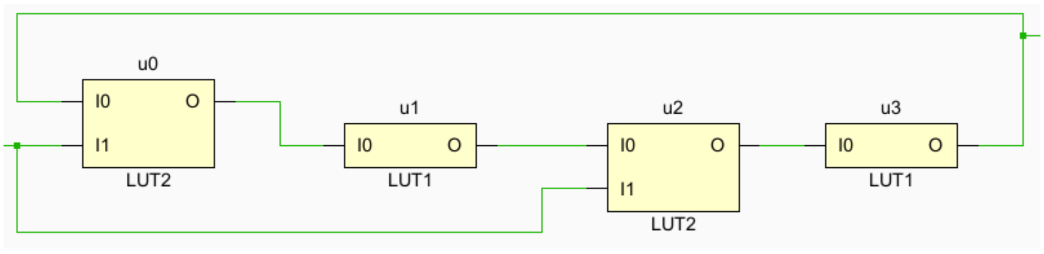
Furthermore, the slices used to implement the cells should be the same type. On the ARTIX-7, there are two main types of slice: SLICEL (only for combinatorial functions) and SLICEM (can additionally implement memory features). There is a higher number of SLICEL available and the TERO cells do not need any memory functionalities. For this reason, only SLICEL are used for the PUF implementation. This is archived using **LOC** constraints.

The routing of the internal loop between the LUTs should also be identical.

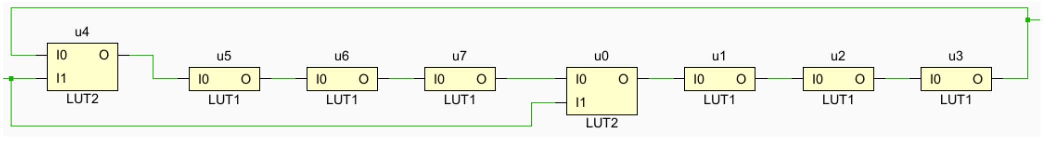
To simplify the routing management, constraining the cells to only one CLB could ensure that the internal loop does not need to use global routing and can stay inside the CLB. The **BEL** is used to fix the exact LUT used for an element and **RLOC** to place elements in slices that are next to each other. It is also possible to assign each input to a specific pin of the LUTs using **LOCK_PINS**, to ensure an identical internal path for all the cells.
More details on how these XDC constraints are used can be found in appendix A.

5.1.2 Cell size

From the routing constraints discussed previously, we know that it is preferable to use only one CLB to implement our cell because it removes the need for inter-CLB routing. On ARTIX-7, CLB contain two slices of 4 LUTs each. The design of the TERO cells imposes to have 2 AND gates and $2N$ NOT gates with N being odd. This restricts our choice of the number of NOT gates to $N = \{1, 3\}$, which correspond to 4 and 8 LUTs respectively. Both cases are studied in parallel and their schematics are shown in figure 5.2. The first one will be noted TERO-4 and the second one TERO-8, due to their LUT usage.



(a) TERO-4 (N=1)



(b) TERO-8 (N=3)

Figure 5.2: TERO cells schematics

Figure 5.3 represent a TERO-8 cell with the eight LUT in orange, one of the internal signal is highlighted in white and the reset in green.

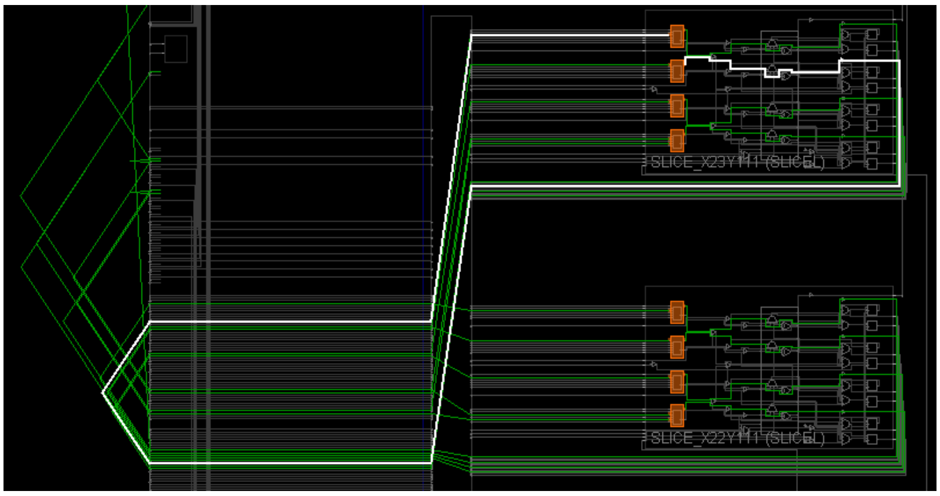


Figure 5.3: TERO-8 cell routing

5.1.3 Number of cells

The initial discussion about the method to generate the response (section 4.2) provided us with a relation between the number of cells and the response size: $\left(\frac{K}{2}\right)^2 - 1$. The size of the PUF response needs to be high enough to be able to study their statistical properties. 2×32 cells are chosen here, which gives a response of 1023 bits.

For the implementation using TERO-8 cells, the number of slices needed is doubled but it is still possible to contain the entire implementation in a single die of the BASYS-3 board used for testing. The figure 5.4 represents the TERO-8 cells regrouped together, with one cell (2 slices) highlighted in green. This left plenty of areas to implement other functionalities in the same device.

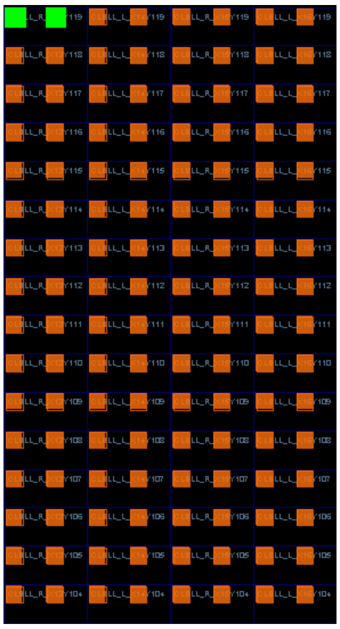


Figure 5.4: Slices usage of TERO-8 cells arrays

5.2 Demonstrative features

To provide a demonstration of a security procedure using the TERO, some additional features are embedded in the implementation: a Bose Chaudhuri and Hocquenghem code (BCH) decoder for error correction and a SHA-256 hashing function

5.2.1 BCH decoder

ECC are good options to improve the reliability of the response. Commonly used in applications such as satellite communication, CD-player or USB flash drive, the BCH code operates using helper data (called syndrome) to correct up to a certain number of random errors. It is represented using 3 parameters $BCH(N, K)$: N is the size of encoded data length and K is the message length. This also fixes the maximal number of errors that can be successfully corrected T, as well as the syndrome's size $N - K$ [18].

The syndrome is computed beforehand from the reference response (during the provision step) and can be either stored on the device or transmitted when needed. The syndrome can be fully revealed since it does not contain any information useful to predict the original response. During the evaluation step, the response of the PUF is checked with the help of the syndrome, and as long as the number of errors is lower than T, it can correct them.

In the implementation of [13], an BCH decoder was added in the FPGA design, and the syndrome was computed externally (using Matlab) and then send to the device when needed. The same BCH implementation is used in this thesis. The parameter has been set to $BCH(255, 171)$, therefore only the first 171 bits of the PUF response are considered. This corresponds to $T = 11$ correctable errors and a syndrome of 84 bits. This is enough to study the improvement in the reliability while having a moderate area usage overhead as we will see in chapter 6.

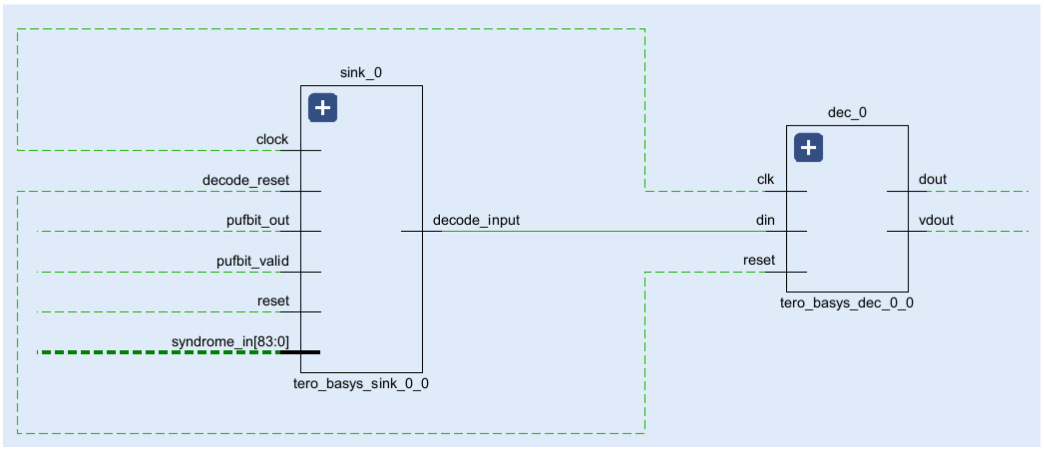


Figure 5.5: BCH(255, 171) schematic

One can notice that due to the way the cell combinations are produced by the LSFR, taking only the first bits of the responses will lead to a non-uniform usage of the cells for those retained bits. This is discussed in the appendix B where we conclude that this does not have a significant impact on the results found in the next section.

5.2.2 SHA-256

A hashing function can be useful to transform the PUF's response into a usable key for security applications. Since one goal of this study was to improve the demonstration part of the framework, it has been chosen to implement this SHA-256 algorithm in the FPGA itself.

There are already VHDL implementations available with open-source licenses online. Therefore, it had been chosen not to re-implement those well-defined operations and to use one of these. The implementation we selected is from the GitHub repository of the user "**batiati**" [4]. The author made available two implementations: one that operates on a single chunk of 512 bits, and the second that operates as a pipeline. We will only need to hash a single key at a time so the first implementation is sufficient.

The SHA-256 hashing operate on a block of 512 bits, that need to be padded from the original input as explained in [39]. First, the input need to be split into blocks of 447 bits maximum. To those blocks, a single **'1'** is added, and then we pad with **'0'** until the block reaches 448 bits. Finally, the 64 remaining bits are the length of the initial input (between 0 and 447) in binary form.

In our case, the input is the response of the BCH decoder. This always has the same size (171 bits) and fits into a single block. The corresponding padded block is represented in table 5.1.

Input	Single **1**	**0** padding	Input size (64 bits)
171 bits	1 bit	276 bits	64 bits
PUF after BCH	**1**	**00..00**	**0000 ... 1010 1011**

Table 5.1: 512 block padding for SHA-256

This gives us the complete SHA-256 block represented in figure 5.6 with first the padding block receiving bit from the BCH decoder, followed by the SHA block. It has been validated it using test vectors from the National Institute of Standards and Technology (NIST) [12] before adding it to the rest of our implementation.

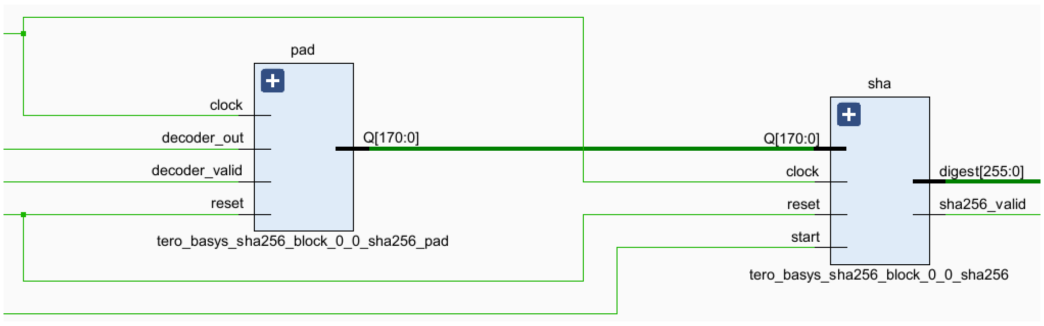


Figure 5.6: SHA-256 block schematic

5.3 FPGA usage

We have now described all elements of our TERO PUF implementations for the ARTIX-7. The first step to characterise it is to analyse the resources used on the boards.

5.3.1 Area

The total area usage of the implementation on a Basys-3 board can be seen in figure 5.7. The slices used for TERO cells are coloured orange because they have a fixed location, while the rest of the used area is light blue.

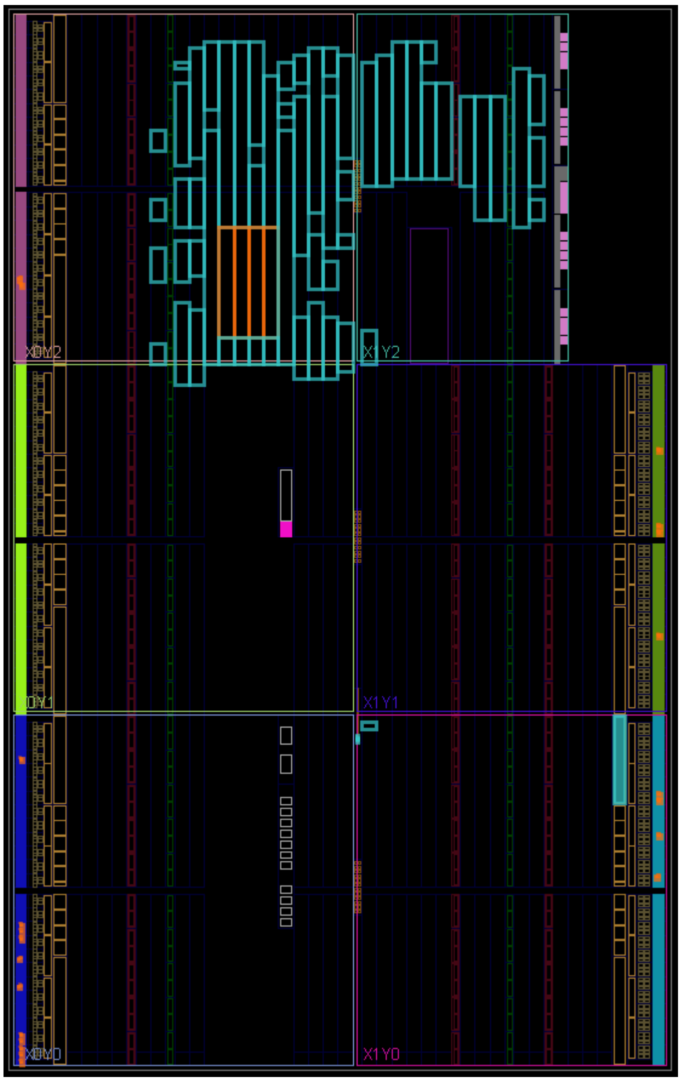
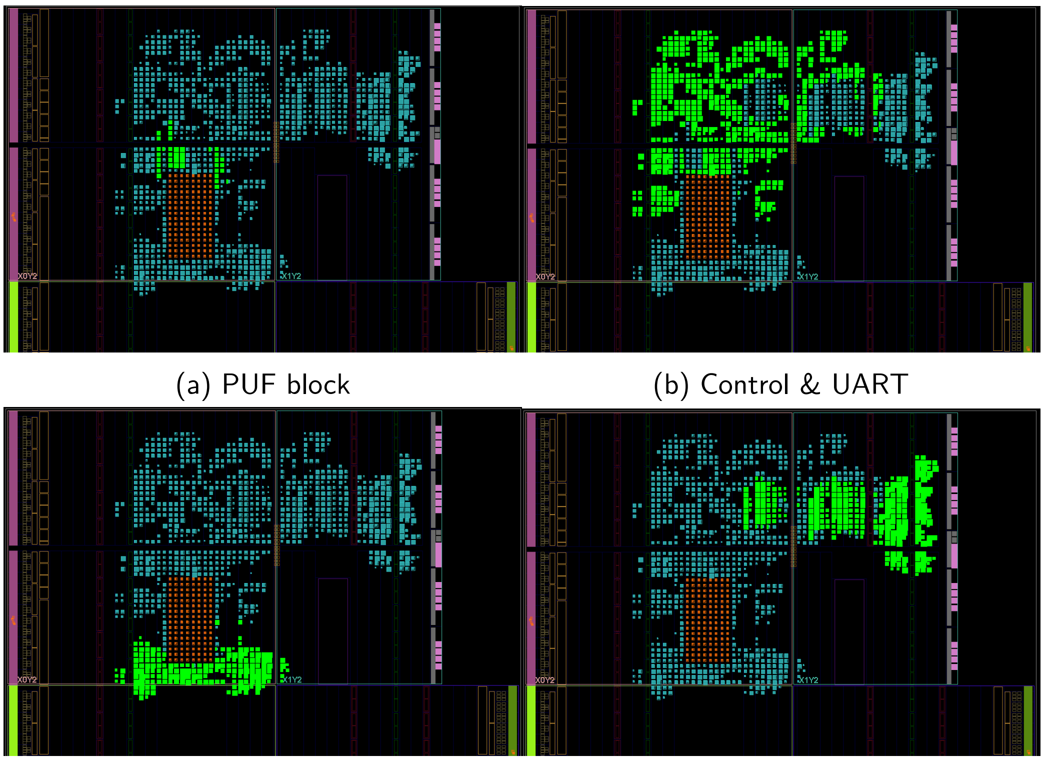


Figure 5.7: Total area usage

The individual blocks can also be highlighted to show the placement made by Vivado. This is done on figure 5.8 for the PUF block (containing the cells and the elements required to generate the response), the control and Universal Asynchronous Receiver Transmitter (UART) block, the BCH decoder and the SHA-256.


(a) PUF block (b) Control & UART

(c) BCH decoder (d) SHA-256

Figure 5.8: Basys-3 usage per block

The primitive statistic for each part of the implementation is reported on the tables 5.2 and 5.3, as well as the total usage reported by Vivado.

Primitive statistic	**LUTs**	**Flop latches**
TERO-4 PUF Block	428 | 11%	162 | 5%
BCH decoder	797 | 20%	981 | 29%
SHA-256	895 | 23%	950 | 28%
Control & UART	1792 | 46%	1308 | 38%
Total	3912	3401

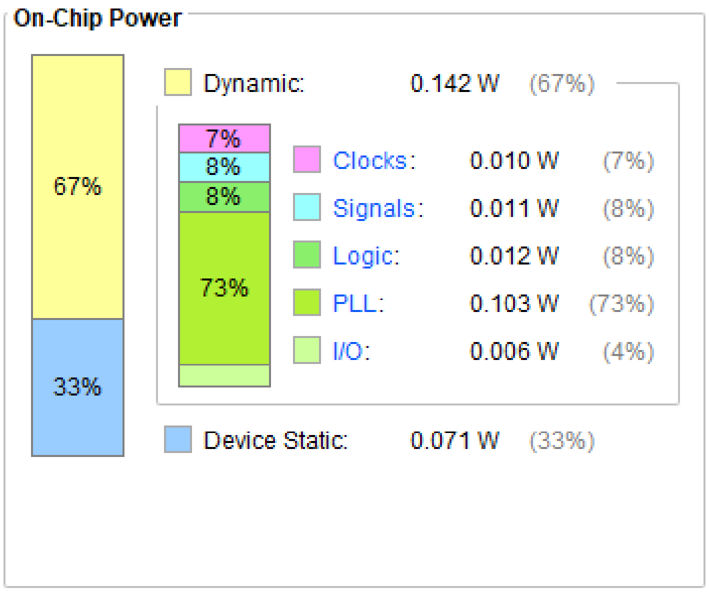
Table 5.2: TERO-4 FPGA resources usage

Primitive statistic	LUTs	Flop latches
TERO-8 PUF Block	684 | 16%	162 | 5%
BCH decoder	797 | 19%	981 | 29%
SHA-256	895 | 21%	950 | 28%
Control & UART	1792 | 43%	1308 | 38%
Total	4168	3353

Table 5.3: TERO-8 FPGA resources usage

The TERO-8 PUF block requires more resources than the TERO-4 one, which makes sense since the TERO-8 cells need twice as many slices then the TERO-4 ones. In both cases, it is the less significant part of the implementations and the control & UART takes most of the space.

5.3.2 Power consumption



🟨 Dynamic:	0.142 W	(67%)
🟪 Clocks:	0.010 W	(7%)
🟦 Signals:	0.011 W	(8%)
🟩 Logic:	0.012 W	(8%)
🟩 PLL:	0.103 W	(73%)
🟩 I/O:	0.006 W	(4%)
🟦 Device Static:	0.071 W	(33%)

Figure 5.9: Power report for TERO-8

The power report from Vivado5.9 indicate an estimated 0.142W of dynamic and an additional 0.071W for the static part such that the total estimated power consumption is 0.213W. We can see that the higher contribution (73%) is the Phase-locked loop (PLL) and that the remaining parts only consume 0.039W (27%).

Chapter 6

PUF results and demonstration

To fully characterise the PUF performance of the two implementations, the following aspects are studied:

- The oscillations of the TERO cells over time.
- The intra-device metrics (uniformity and reliability) for different acquisition times on a single device.
- The inter-device metrics (average uniformity and reliability, uniqueness and bit-aliasing) are computed from the result of 33 devices for a chosen acquisition time.
- The impact of the Error Correction Code (ECC).

Furthermore, the performance found will be compared to other existing TERO-PUF implementations, followed by a description of the demonstration.

All tests are run on Basys-3 with the main clock frequency being 100 MHz. Each data point is averaged over 10 000 samples. The tests were done with an ambient temperature around 20°C and nominal voltage.

The interface between the FPGA and the computer is done via UART and a custom Python module described in appendix C

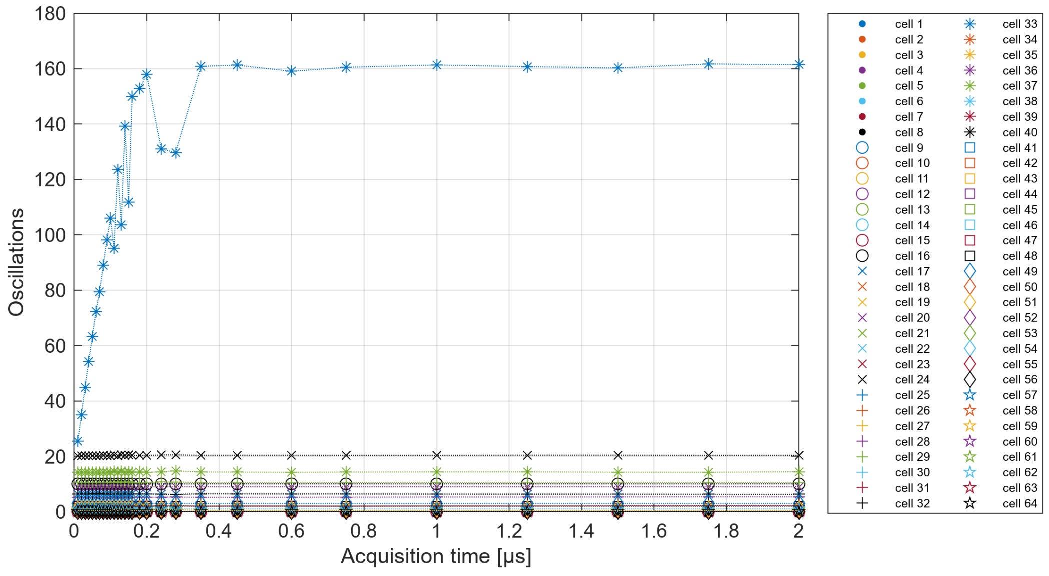
6.1 Cells behaviour and equalities

The value of the counter of each cell is directly sent via UART once the acquisition time is reached, instead of being compared to generate the PUF response. The range of acquisition time to evaluate is chosen from $0.01\mu s$ to $2\mu s$. The expected behaviour is to initially have several oscillations at the beginning, then at some point, reach stability and remain constant.

6.1.1 Oscillations over time

TERO-4

The 64 TERO-4 cell's oscillations are displayed in figure 6.1. The cells stabilise within a single clock cycle ($0.01\mu s$) with a final number of oscillations below 20, except for one cell that only stabilises around $0.4\mu s$ after 160 oscillations.


(a) Full

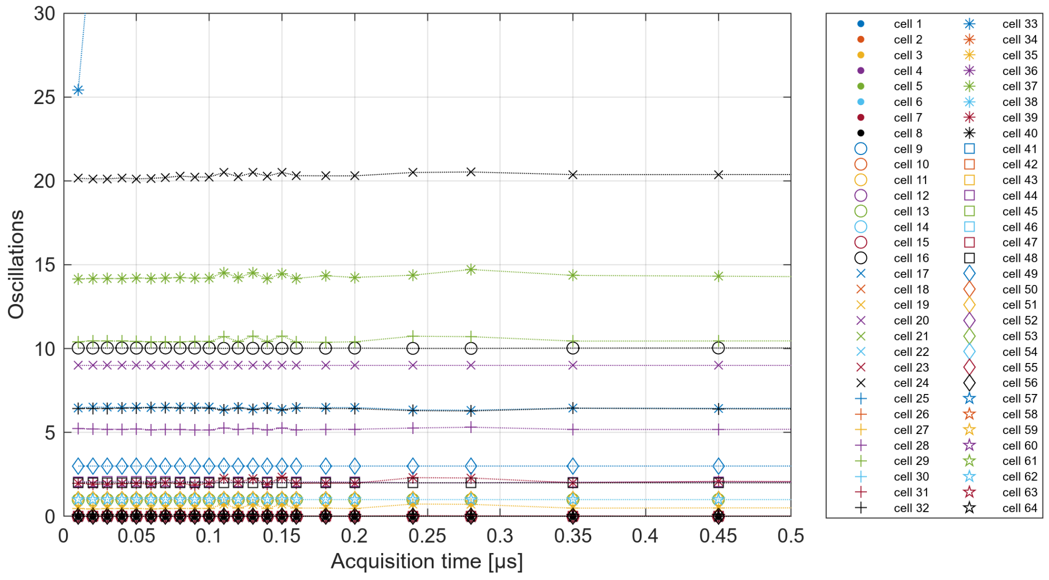
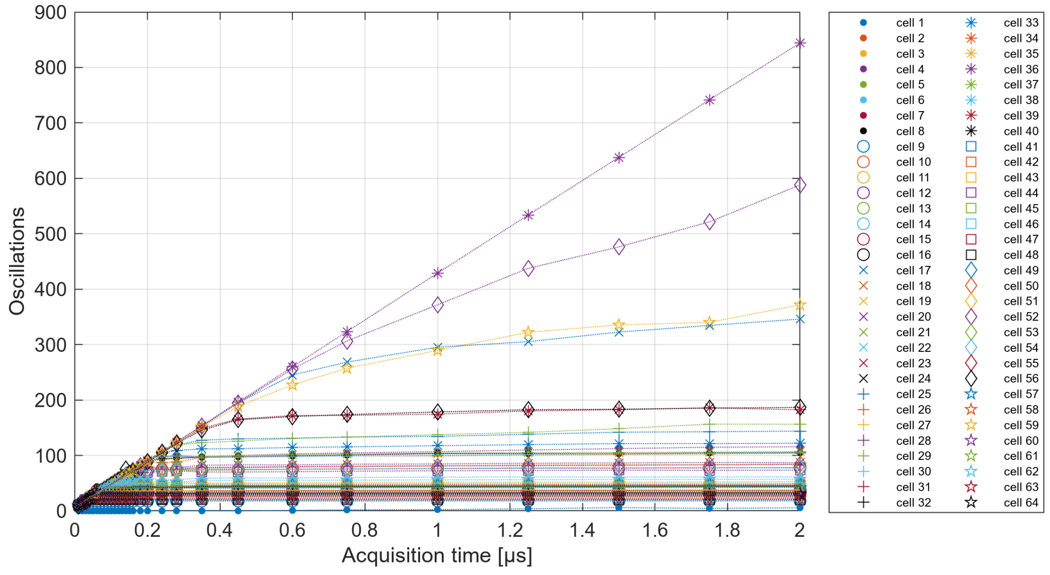

(b) Zoomed

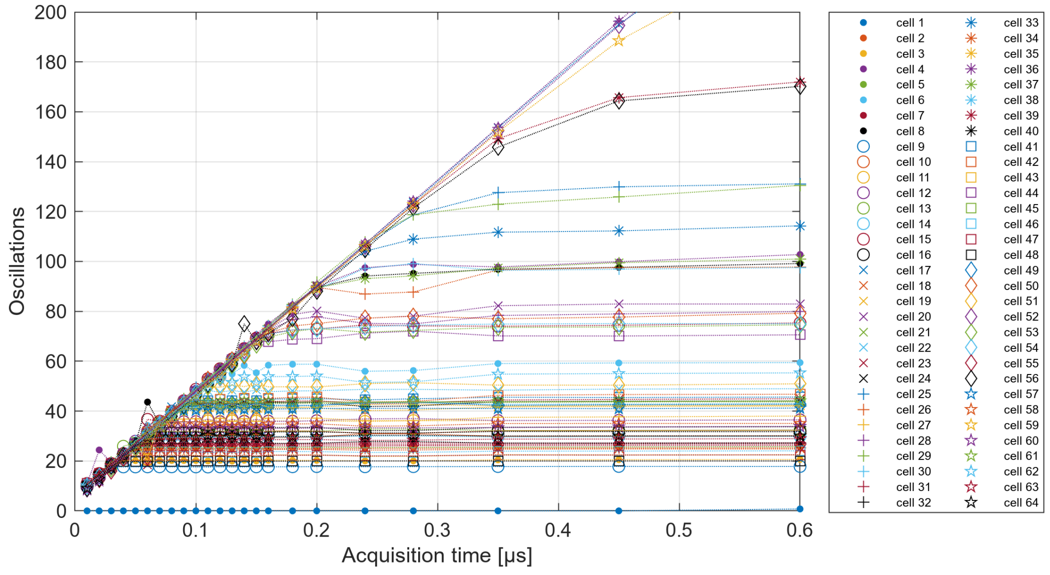
Figure 6.1: TERO-4 cells oscillations over acquisition time

TERO-8

The figure 6.2 represents the oscillation of the 64 TERO-8 cells. The stabilisation time is mostly below $0.5\mu s$. Four cells seem to never really stabilise before $2\mu s$. The final number of oscillations of the stable cells is spread up to 200.



(a) Full



(b) Zoomed

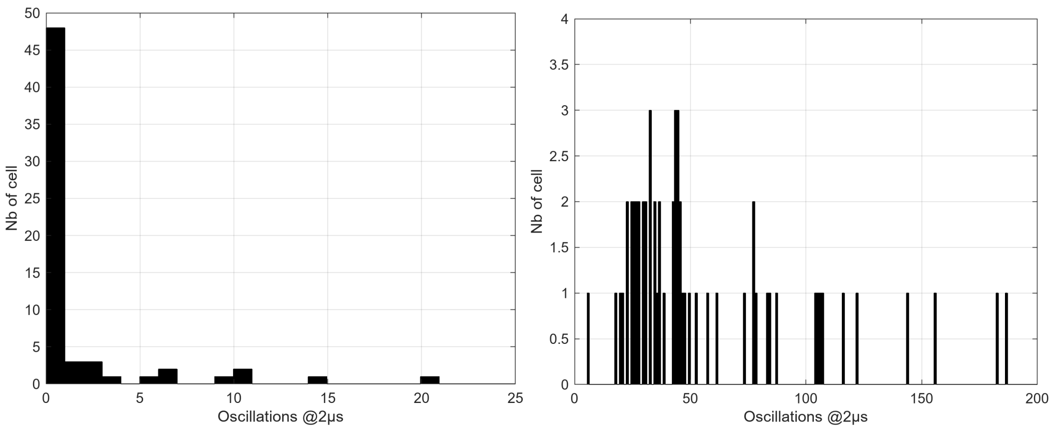
Figure 6.2: TERO-8 cells oscillations over acquisition time

Figures 6.2a and 6.2b confirm that most of the cells do stabilise and only a few are unstable as expected. If the unstable cells become an issue for the implementation, we could imagine changing the physical location of the unstable cells on the FPGA until all cells are stable. However, this would require a calibration step and this will not be studied in this thesis.

6.1.2 Final state

The final state of the cells (i.e. the number of oscillations after stabilisation) is difficult to see on the previous graphs. Here we display the final state of each cell at $2\mu s$ as a histogram on figures 6.3. The cells that are considered unstable from the previous section are excluded for more visibility.

For the TERO-4, this reveals that 48 of the 64 cells have a final number of oscillations equal to zero, meaning that they do not oscillate a single time. For the TERO-8, the final number of oscillations is more spread than for the TERO-4, with at most 3 cells with the same number.



(a) TERO-4 (b) TERO-8

Figure 6.3: TERO cells - number of oscillations at $2\mu s$

6.1.3 Equality's

In the PUF implementation, the bits responses are generated by comparing the number of oscillations of two cells. Therefore, a high proportion of the cell having the same number of oscillations will lead to a high number of equalities in the process. As discussed in the section 4.2, the equality results in a response that is always the same and therefore needs to be avoided as much as possible.

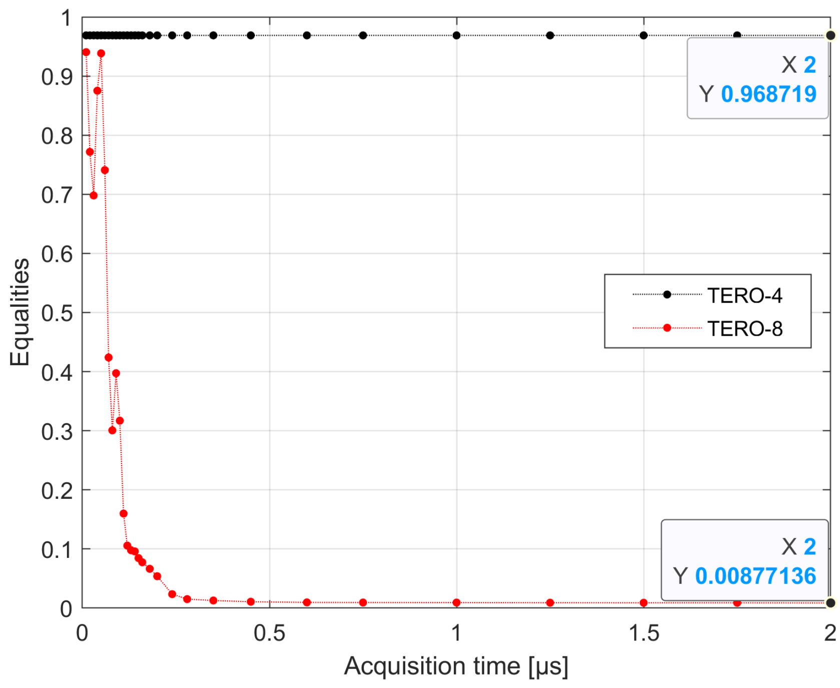


Figure 6.4: TERO cells oscillations equality's over acquisition time

Comparing each pair of cells similarly than for the response generation, we can find the proportion of equalities, which is shown in figure 6.4. The TERO-4 cells comparison raises an equality for 96.8% of the cells combination, for any acquisition time. The TERO-8 cells however produced less than 1% of equality after $0.4\mu s$ and almost 0% at $2\mu s$. It is also interesting to observe the beginning where there is a high number of equality which decreases rapidly. This is coherent since this is the moment where the cells start to stabilise, each one at a time, and therefore the number of oscillations diverges for each cell.

From this test, TERO-4 cells seem to not be suitable due to a high proportion of the cells that do not oscillate, while the TERO-8 cells behave as expected. However, it is possible that another device would produce a completely different behaviour and it could simply be due to unfortunate circumstances.

6.2 Intra-device performances and acquisition time determination

This tests studies the intra-device metrics (Uniformity and Reliability) of the PUF for different acquisition times (still between $0.01\mu s$ to $2\mu s$). This provides a first look at the PUF performances on a single device, before the final characterisation with the inter-device metrics. The goal is also to choose an adequate acquisition time to have the best PUF properties for the final test.

6.2.1 Uniformity

As defined in the chapter 2, Uniformity (UF) is the measure of the response's bit repartition between '0' and '1' and is ideally equal to 50%. This metric is displayed in figure 6.5 for both TERO-4 and TERO-8. The uniformity of the TERO-4 PUF stays steady at 3% while the TERO-8 oscillates shortly then rises and stabilises to 53.0% around $0.3\mu s$.

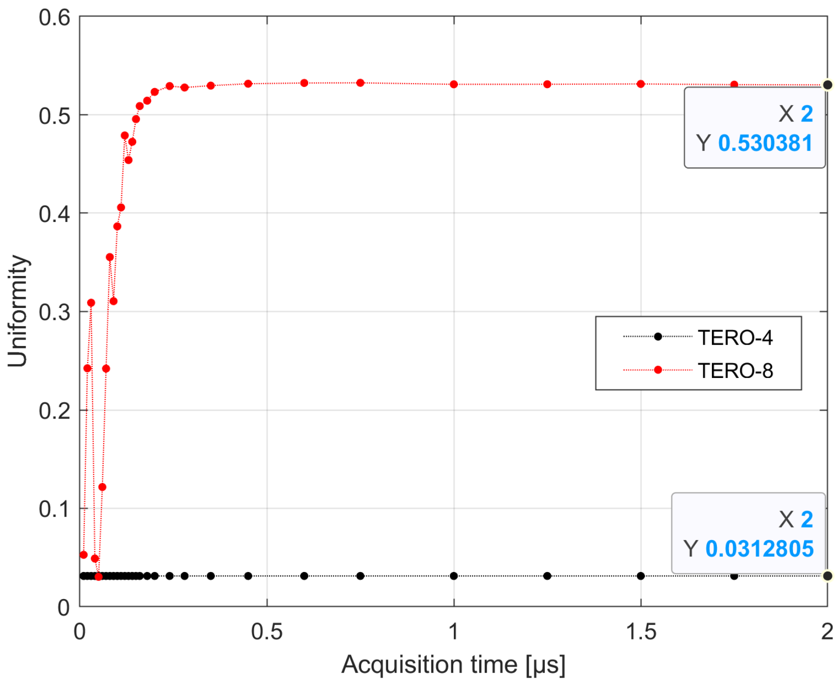


Figure 6.5: PUF intra-uniformity

It was expected from the previous test about cell oscillations that the uniformity of the TERO-4 would be poor, due to the high number of equalities. From the 96.8% of equality found, we know that 96.8% of the generated bits are '0'. If we assume that the remaining 3.2% of the bit are perfectly uniform, this would induce that $96.8\% + 3.2\% \times 50\% = 98.4\%$ of the bits are '0', which corresponds to a uniformity of 1.6%. The 3% found is higher than this prediction, indicating that there is more '1' on the remaining bits than expected. Nevertheless, this proves that the TERO-4 uniformity is extremely low, at least for this specific device.

For the TERO-8, the oscillation of the uniformity at the beginning of the figure 6.5 is also expected from the oscillation of the equalities at the same acquisition time. Once it is stabilised, the uniformity exceeds 50% by 3%, indicating that there is slightly more '1' than '0' in the bits response.

In both case, there is more '1' than expected, which could be either due to this specific device or to the implementation. This will be determined by the inter-device characterisation (6.3).

6.2.2 Reliability

The Reliability (RE) is the measure of the stability of each bit over multiple samples and is ideally equal to 100%. This metric is shown in figure 6.6. The TERO-4 reliability is constantly at 100% while the TERO-8 reliability oscillates initially between 99 and 93%, then stabilises at 97.7% after $0.4\mu s$.

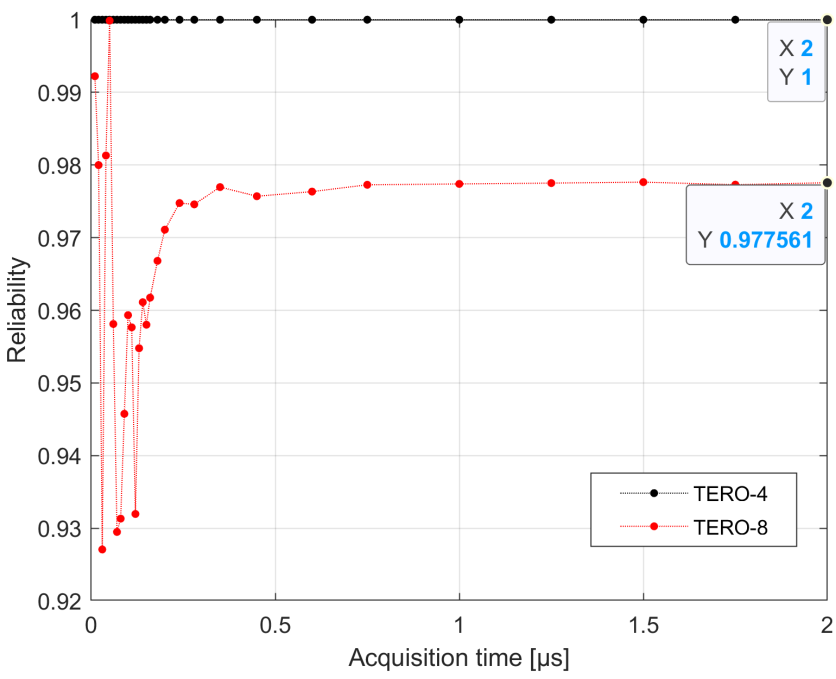


Figure 6.6: PUF intra-reliability

The TERO-4 constant reliability is coherent with the fact that 63 of the 64 stabilise immediately, and therefore the response does not change anymore with the acquisition time. The fact that it is at 100% reveals that the cell's final states are completely stable for this device over the 10 000 samples captured.

Once the TERO-8 cells have stabilised, the reliability found is comparable to the state of the art, but additional error correction techniques could increase the stability depending on the target application.

When we look at both the uniformity and reliability of the implementations, we can confirm that the TERO-4 does not provide good enough performance on this specific device while TERO-8 seems to produce correct performance. This also shows that an acquisition time higher than $0.5\mu s$ does not provide any benefit to the results from this device. We choose $1\mu s$ for the acquisition time used in the final tests, by assuming that the variation of this result between the devices is smaller than $0.5\mu s$.

6.3 Inter-device performances

The previous performance estimations are done on a single board and it is important to study the variation of those performances over multiple devices. The responses produced by different devices should be very unique and we can also check a systematic bias in the implementation for specific bits. In this test, the two TERO PUFs are uploaded on 33 BASYS-3 boards and the responses are recorded for an acquisition time of $1\mu s$. The device used in the previous test is at index 1.

6.3.1 Inter-device uniformity

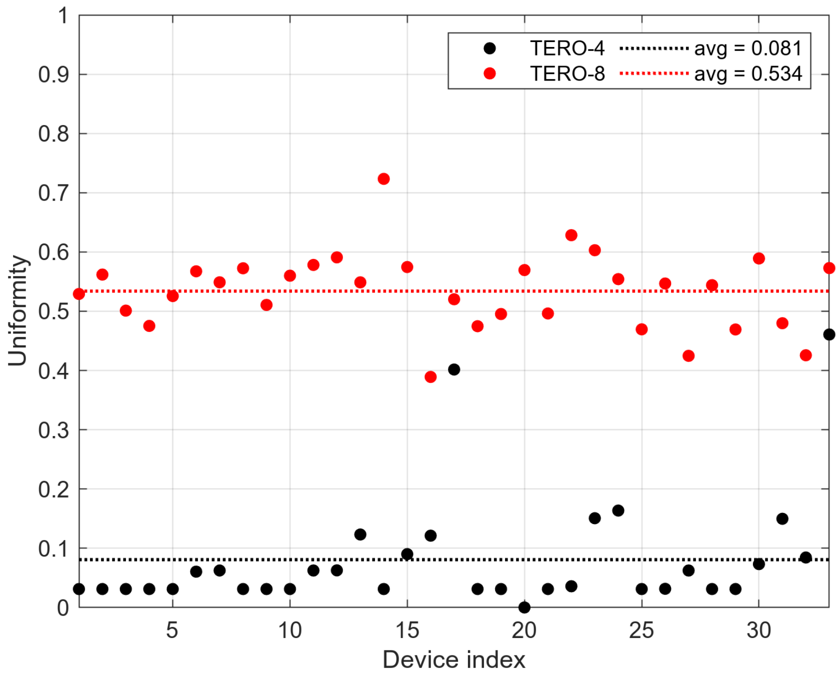


Figure 6.7: Inter-device uniformity's

The TERO-4 uniformities of the 33 boards are represented in the figure 6.7, with the dotted line for the average. 2 devices produce a uniformity higher than 40%, while most of them stay below 8.0% average. This proves that the poor uniformity found in the previous test (device at index 1) is not due to bad luck in the choice of the device but rather comes from the TERO-4 cells implementation itself.

The TERO-8 uniformities in figure 6.7 reveal that the device's uniformity from the previous test (53.0%) was really representative of the 53.4% average uniformity for this implementation. However, this also shows that this metric variation of this metrics between 39% and 72% depending on the device used.

6.3.2 Inter-device reliability

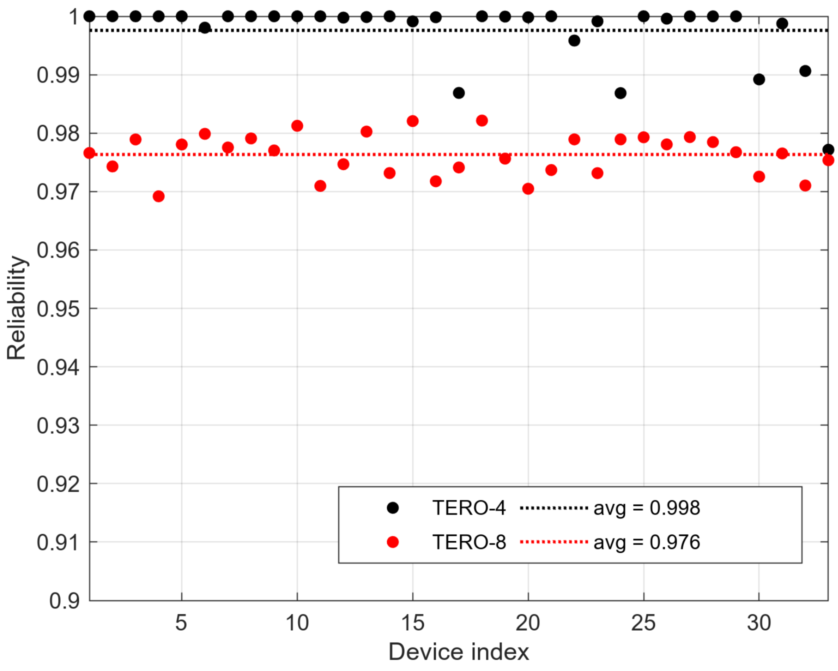


Figure 6.8: Inter-device reliability's

Figure 6.8 represent the TERO-4 reliability's. This indicates that most devices produce near-perfect reliability (99.8% average). This also reveals that the device used in the previous test (index 1) was one of the devices that have perfect reliability over the 10 000 samples, but that is not always the case and a few devices produced a reliability below 99%.

The TERO-8, displayed in figure 6.8, confirms that the value found on the previous test (97.7%) was a good representation of the average (97.6%) for the 33 devices. Furthermore, we can see that all the values are within a variation of 1%.

6.3.3 Bit-Aliasing

Bit-aliasing is the measure of how likely is a specific bit to be '1' over all the devices. Ideally, this would be equal to 50% for all the bits. However, since we only tested 33 devices, instead of looking only at the average, we should compare the distribution of the bit-aliasing values to the ideal distribution i.e. a binomial distribution of probability 50% over 33 experiments (represented in green in figure 6.9).

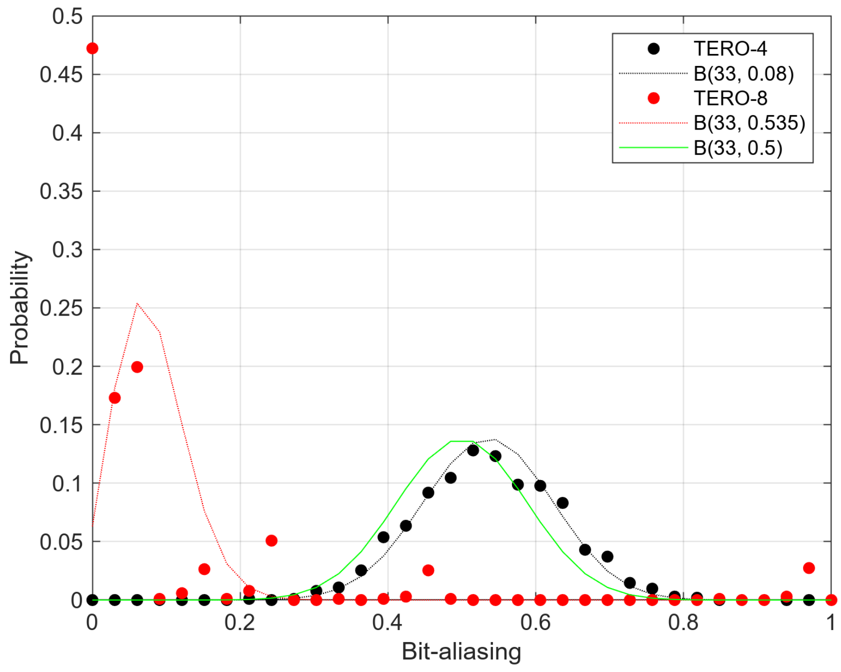


Figure 6.9: Inter-device bit-aliasing

We already know that only 8.0% of the TERO-4 bits are '1' for the average uniformity. Figure 6.9 reveals that almost 50% of the bits are completely biased toward 0, meaning that they are always '0' in all the 33 devices tested. The average bit-aliasing is 8.0% but the TERO-4 distribution does not follow a binomial distribution for this probability either. From the previous test, we know that the large number of bits equal to '0' in the device at index 1 is due to the large number of cells that do not oscillate. If we suppose that this is also the case in the other device, then this would mean that at least part of the cell that does not oscillate seems to always be the same regardless of the device. This indicates a bias in the implementation itself for specific cells.

The TERO-8 bit-aliasing distribution (figure 6.9) follows closely a binomial distribution centred around its average of 53.5%. This indicates that all the bits seem to give different response depending on the device, with a small overall bias of 3.5% toward '1' which correspond almost to the 3.4% excess of uniformity. This means that no bit is systematically at the same value but that there seems to be a global tendency for the cells in one array to produce a higher number of oscillations than the cells in the other array.

6.3.4 Uniqueness

The uniqueness is the measure of how different the responses of the different devices are in terms of hamming distance, with an ideal value of 50%. The results for this metric are on the table 6.1.

	Uniqueness
TERO-4	8.2%
TERO-8	49.4%

Table 6.1: Uniqueness

The TERO-4 responses only produce an 8.2% of uniqueness, which is understandable since we know from the bit-aliasing that a large part of the bits response is always '0'. For the TERO-8 however, the responses have a uniqueness of 49.42% over the 33 devices tested, which is close to the ideal value.

	Uniformity	Reliability	Bit-aliasing	Uniqueness
TERO-4	8.1%	99.8%	8.0%	8.2%
TERO-8	53.4%	97.6%	53.5%	49.4%

Table 6.2: Inter-device performances

It appears clearly that the TERO-4 implementation does not provide the desired performances in terms of uniformity, bit-aliasing and uniqueness. This is most likely due to the cells that systematically stabilise without any oscillation. We therefore consider this implementation to not be usable and it will not be discussed in the rest of this chapter.

6.4 Error correction

The TERO-8 average reliability is 97.63%, which, depending on the target application, may not be good enough. We will therefore evaluate the improvement brought by the ECC described in 5.2.1.

The responses of the ECC implementation are recorded for the same acquisition time ($1\mu s$) over the 33 BASYS-3 boards in the same condition as the previous test. The table 6.3 contains all the inter-device metrics of the initial TERO-8 implementation ("Full"), the reduced version of it with only 171 bits ("Reduced") and the response of the BCH decoder for error correction ("ECC").

TERO-8	Response's size	RE	UF	UQ	BA
Full	1023 bits	97.6%	53.4%	49.4%	53.5%
Reduced	171 bits	97.7%	53.4%	49.9%	54.4%
ECC	171 bits	99.9%	53.5%	49.9%	54.4%

Table 6.3: TERO-8 performances without and with ECC

First, we can observe that the performances of the reduced version (without ECC) are very similar to the full one. Therefore, any improvement due to the ECC on the reduced response can be supposed to be equally effective on an entire response.

Moreover, when we compare the reliability between the reduced response and the ECC one, it goes from 97.71% to 99.90%, which is an improvement. At the same time, the other performances remain substantially the same.

As expected, using this technique, we significantly increase the performance of the TERO-8 without any downside other than the additional FPGA resources used (discussed in 5.3.1). Depending on the target application, reliability higher than 99.9% can still be required. In this case, a BCH supporting a higher number of errors should be used.

6.5 Comparison with existing implementations

The performance produced by the TERO-8 implementation are comparable to both other TERO-PUF implementation (table 6.4) and other PUF techniques using an Artix-7 for implementation (table 6.5), with only the ECC version giving a reliability in line with the literature.

method	RE	UF	UQ	BA	revice	Ref
TERO	98.3%	-	48%	-	Cyclone-II	[9]
TERO	99.99%	-	46.7%	-	Altera DE2	[9]
TERO	97.4%	-	48.5%	-	Spartan-6	[31]
	98.2%	-	47.6%	-	Cyclone-V	
PDL-TERO	98.8%	-	49.32%	-	Spartan-3	[3]
RAW (full)	**97.6%**	**53.4%**	**49.4%**	**53.5%**	**Artix-7**	**This**
ECC	**99.9%**	**53.5%**	**49.9%**	**54.4%**		**work**

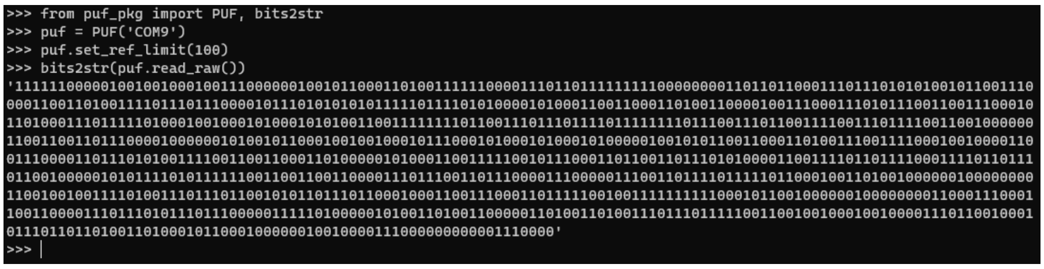
Table 6.4: TERO-PUF implementations on FPGA

method	RE	UF	UQ	BA	Ref
APUF	99.55%	51.84%	46.21%	-	[1]
XOR-APUF	99.41%	50.73%	48.69%	-	[1]
BST-APUF	99.99%	-	49.1%	50.3%	[23]
ROPUF	99.19%	51.01	47.86%	51.01%	[13]
BST-ROPUF	99.99%	46.78%	48.64%	-	[24]
RWC-SRAMPUF	98.92%	55.38%	37.36%	46.89%	[11]
Flip-Flop PUF	99%	49.2%	-	48.96%	[25]
RAW (full)	**97.6%**	**53.4%**	**49.4%**	**53.5%**	**This**
ECC	**99.9%**	**53.5%**	**49.9%**	**54.4%**	**work**

Table 6.5: PUF implementations on Artix-7

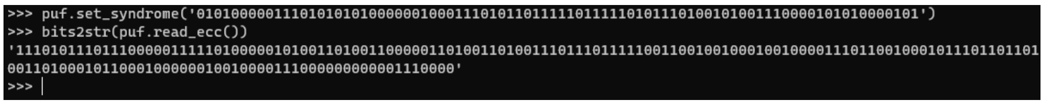
6.6 Demonstration

Using the python module (appendix C) and a USB cable for UART communication between the FPGA and a computer, we can demonstrate the different responses of the implementation: the raw response (figure 6.10), the response after ECC (figure 6.11) and the SHA key generated (figure 6.12).



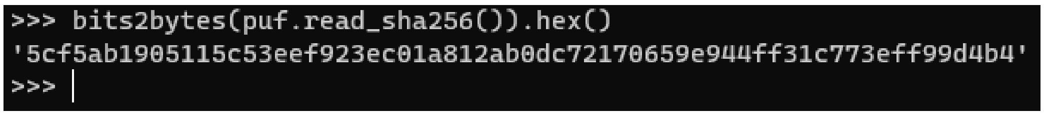
```
>>> from puf_pkg import PUF, bits2str
>>> puf = PUF('COM9')
>>> puf.set_ref_limit(100)
>>> bits2str(puf.read_raw())
'1111110000010010010001001110000000100101100011010011111100001110110111111111100000000110110110001110111010101001011001110
000110011010011110111011100001011101010101011111011110101000010100011001100011010011000010011100011101011100110011100010
1101000111011111101000100100010100010101001100111111110110011101110111101111111101110011101100111001110111100110010000000
11001100110111000010000000101001011000100100100010111000101000101000101000000100101011001100011010011100111100010010000110
011100001101110101001111001100110001101000001010001100111110010111000110110011011101010000110011110110111100011110110111
011001000000101011110101111110011001100110000111011100110111000011100000111001101111011111101100010011010010000000100000000
11001001001111010011101110110010101101110110001000110011100011011111100100111111111100010110010000000100000000110001110001
100110000111011101011101110000011111010000001010011010011000001101001101001110111011111100110010010001001000001110110010001
0111011011010011010001011000100000001001000011100000000000001110000'
>>> |
```

Figure 6.10: Demonstration raw response



```
>>> puf.set_syndrome('0101000001110101010100000010001110101101111101111110101110100101001110000101010000101')
>>> bits2str(puf.read_ecc())
'111010111011100000111110100000010100110100110000001101001101001110111011111100110010010001001000001110110010001011101101101
0011010001011000100000001001000011100000000000001110000'
>>> |
```

Figure 6.11: Demonstration ECC response



```
>>> bits2bytes(puf.read_sha256()).hex()
'5cf5ab1905115c53eef923ec01a812ab0dc72170659e944ff31c773eff99d4b4'
>>> |
```

Figure 6.12: Demonstration SHA-256 key

Chapter 7

Conclusion

TERO-PUF is a promising technique that needs further research to cover more devices and possible variations. In particular, to our knowledge, there is no existing implementation of TERO-PUF on Artix-7 documented in the literature, therefore, we have proposed an implementation.

During the design of the PUF (Chapter 5), two parameters were identified: the size of the TERO cells and the acquisition time. The discussion on the size of the cell led to two implementations: TERO-4, with TERO cells using 4 LUTs, and TERO-8, with 8 LUTs.

The acquisition time is not fixed to the design and has been chosen to have the best performance for the experimental result (Chapter 6). The two implementations metrics were studied on a single device, in terms of cell behaviour and performance, for different acquisition times. We found that the TERO-4 implementation produced too many non-oscillating cells and appeared to be unusable (at least on this device), while the TERO-8 worked as expected.

Inter-device analysis was performed for TERO-4 and TERO-8 on 33 Basys-3 boards for a $1\mu s$ acquisition time. The results confirms that the TERO-4 is not usable on most of the devices while the TERO-8 implementation delivers performance in line with other PUF and TERO-PUF implementations in literature (table 6.4 and table 6.5).

We also demonstrated the use of the SHA-256 and the full demonstration using the Python module.

A meaningful addition to the demonstration would be to have an Advanced Encryption Standard (AES) block in the implementation and to do text (de)-

ciphering on the FPGA. This way, the generated key does not have to leave the device, reducing a potential leak.
PUF are generally sensitive to factors such as temperature or voltage, and it is important to study their impact on performance to characterise the conditions under which they can be used. In addition, [38] has studied side-channel possibilities on TERO-PUF and proposed some countermeasures, that could be added to the design used here.
Finally, we choose in this work to limit the TERO cells to one CLB to simplify the routing and maintain a compact implementation. Larger cells could lead to different performances and should be studied.

Bibliography

[1] N. Nalla Anandakumar, Mohammad S. Hashmi, and Muhammad Akmal Chaudhary. "Implementation of Efficient XOR Arbiter PUF on FPGA With Enhanced Uniqueness and Security". In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 129832–129842.

[2] N. Nalla Anandakumar, Mohammad S. Hashmi, and Mark Tehranipoor. "FPGA-based Physical Unclonable Functions: A comprehensive overview of theory and architectures". In: *Integration* 81 (2021), pp. 175–194.

[3] Amir Ardakani, Shahriar B. Shokouhi, and Arash Reyhani-Masoleh. "Improving performance of FPGA-based SR-latch PUF using Transient Effect Ring Oscillator and programmable delay lines". In: *Integration* 62 (2018), pp. 371–381.

[4] Rafael Batiati. *VHDL SHA2-256*. original-date: 2021-04-22. 2021. URL: `https://github.com/batiati/VHDL_SHA2-256` (visited on 05/19/2023).

[5] Pierre Bayon et al. "Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator". In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by Werner Schindler and Sorin A. Huss. Red. by David Hutchison et al. Vol. 7275. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 151–166.

[6] Ygal Bendavid et al. "IoT Device Security: Challenging "A Lightweight RFID Mutual Authentication Protocol Based on Physical Unclonable Function"". In: *Sensors (Basel, Switzerland)* 18.12 (2018). Place: Switzerland Publisher: MDPI, pp. 4444–.

[7] Nathalie Bochard et al. "True-Randomness and Pseudo-Randomness in Ring Oscillator-Based True Random Number Generators". In: *International Journal of Reconfigurable Computing* 2010 (2010), pp. 1–13.

[8] L. Bossuet, P. Bayon, and V. Fischer. "An Ultra-Lightweight Transmitter for Contactless Rapid Identification of Embedded IP in FPGA". In: *IEEE Embedded Systems Letters* 7.4 (2015), pp. 97–100.

[9] Lilian Bossuet et al. “A PUF Based on a Transient Effect Ring Oscillator and Insensitive to Locking Phenomenon”. In: *IEEE Transactions on Emerging Topics in Computing* 2.1 (2014), pp. 30–36.

[10] Shuai Chen, Bing Li, and Cenjun Zhou. “FPGA implementation of SRAM PUFs based cryptographically secure pseudo-random number generator”. In: *Microprocessors and Microsystems* 59 (2018), pp. 57–68.

[11] Ihsan Cicek and Ahmad Al Khas. “A new read-write collision-based SRAM PUF implemented on Xilinx FPGAs”. In: *Journal of Cryptographic Engineering* (2022).

[12] Information Technology Laboratory Computer Security Division. *Example Values - Cryptographic Standards and Guidelines | CSRC | CSRC*. CSRC | NIST. 2016. URL: `https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values` (visited on 05/24/2023).

[13] Brent De Weerdt. “Implementation of physical unclonable functions on FPGA circuits”. PhD thesis. Universite Libre de Bruxelles, 2021. 56 pp.

[14] Riccardo Della Sala, Davide Bellizia, and Giuseppe Scotti. “A Novel Ultra-Compact FPGA PUF: The DD-PUF”. In: *Cryptography* 5.3 (2021), p. 23.

[15] Aleksa Deric and Daniel Holcomb. “Know Time to Die - Integrity Checking for Zero Trust Chiplet-based Systems Using Between-Die Delay PUFs”. In: *IACR transactions on cryptographic hardware and embedded systems* 2022.3 (2022). Publisher: Ruhr-Universitat Bochum, pp. 391–412.

[16] K. N. Devika and Ramesh Bhakthavatchalu. “FPGA implementation of programmable Hybrid PUF using Butterfly and Arbiter PUF concepts”. In: *Journal of Physics: Conference Series* 2312.1 (2022). Publisher: IOP Publishing, p. 012033.

[17] Mohammad Ebrahimabadi et al. *An Attack Resilient PUF-based Authentication Mechanism for Distributed Systems*. 2022. arXiv: `2206.07019[cs]`.

[18] Jurgen Freudenberger, Daniel Nicolas Bailon, and Malek Safieh. “Reduced complexity hard and soft-input BCH decoding with applications in concatenated codes”. In: *IET Circuits, Devices & Systems* 15.3 (2021), pp. 284–296.

[19] Blaise Gassend et al. “Silicon Physical Random Functions”. In: (2002).

[20] Jorge Guajardo et al. “FPGA Intrinsic PUFs and Their Use for IP Protection”. In: *Cryptographic Hardware and Embedded Systems - CHES 2007* 4727 (2007). Ed. by Pascal Paillier and Ingrid Verbauwhede. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science, pp. 63–80.

[21] Bilal Habib, Jens-Peter Kaps, and Kris Gaj. “Implementation of efficient SR-Latch PUF on FPGA and SoC devices”. In: *Microprocessors and Microsystems* 53 (2017), pp. 92–105.

[22] Mohammed El-Hajj et al. “A taxonomy of PUF Schemes with a novel Arbiter-based PUF resisting machine learning attacks”. In: *Computer Networks* 194 (2021), p. 108133.

[23] Zhangqing He et al. “A Highly Reliable Arbiter PUF With Improved Uniqueness in FPGA Implementation Using Bit-Self-Test”. In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 181751–181762.

[24] Zhangqing He et al. “A highly reliable FPGA-based RO PUF with enhanced challenge response pairs resilient to modeling attacks”. In: *IEICE Electronics Express* 18.20 (2021), pp. 20210350–20210350.

[25] Sajid Khan et al. “A symmetric D flip-flop based PUF with improved uniqueness”. In: *Microelectronics Reliability* 106 (2020), p. 113595.

[26] Sandeep S. Kumar et al. “Extended abstract: The butterfly PUF protecting IP on every FPGA”. In: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST). Anaheim, CA, USA: IEEE, 2008, pp. 67–70.

[27] J.W. Lee et al. “A technique to build a secret key in integrated circuits for identification and authentication applications”. In: *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)* (2004), pp. 176–179.

[28] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. “Intrinsic PUFs from flip-flops on reconfigurable devices”. In: (2008).

[29] Abhranil Maiti and Patrick Schaumont. “Improved Ring Oscillator PUF: An FPGA-friendly Secure Primitive”. In: *Journal of Cryptology* 24.2 (2009), pp. 375–397.

[30] Cedric Marchand, Lilian Bossuet, and Abdelkarim Cherkaoui. *Design and Characterization of the TERO-PUF on SRAM FPGAs*. Pages: 139. 2016. 134 pp.

[31] Cedric Marchand et al. “Implementation and Characterization of a Physical Unclonable Function for IoT: A Case Study With the TERO-PUF”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.1 (2017), pp. 97–109.

[32] A. Theodore Markettos and Simon W. Moore. “The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators”. In: *Cryptographic Hardware and Embedded Systems - CHES 2009*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 317–331.

[33] Abdulaziz Al-Meer and Saif Al-Kuwari. *Physical Unclonable Functions (PUF) for IoT Devices*. 2022. arXiv: `2205.08587[cs]`.

[34] Ugo Mureddu et al. “Efficient design of Oscillator based Physical Unclonable Functions on Flash FPGAs”. In: 2017 IEEE 2nd International Verification and Security Workshop (IVSW). Thessaloniki: IEEE, 2017, pp. 146–151.

[35] Ravikanth Pappu et al. “Physical One-Way Functions”. In: *Science* 297 (2001), pp. 2026–2030.

[36] Ravikanth Pappu et al. *Physical One-Way Functions*. 2002, pp. 2026–2030.

[37] G Edward Suh and Srinivas Devadas. “Physical Unclonable Functions for Device Authentication and Secret Key Generation”. In: (2007), p. 6.

[38] Lars Tebelmann, Michael Pehl, and Vincent Immler. “Side-Channel Analysis of the TERO PUF”. In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by Ilia Polian and Marc Stottinger. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 43–60.

[39] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. Federal Information Processing Standard (FIPS) 180-4. U.S. Department of Commerce, 2015.

[40] Mohammad Usmani. “Applications Of Physical Unclonable Functions on ASICS and FPGAs”. Publisher: University of Massachusetts Amherst. PhD thesis. University of Massachusetts Amherst, 2018. 87 pp.

[41] Liang Yao et al. “M-RO PUF: A portable pure digital RO PUF based on MUX unit”. In: *Microelectronics Journal* 119 (2022), p. 105314.

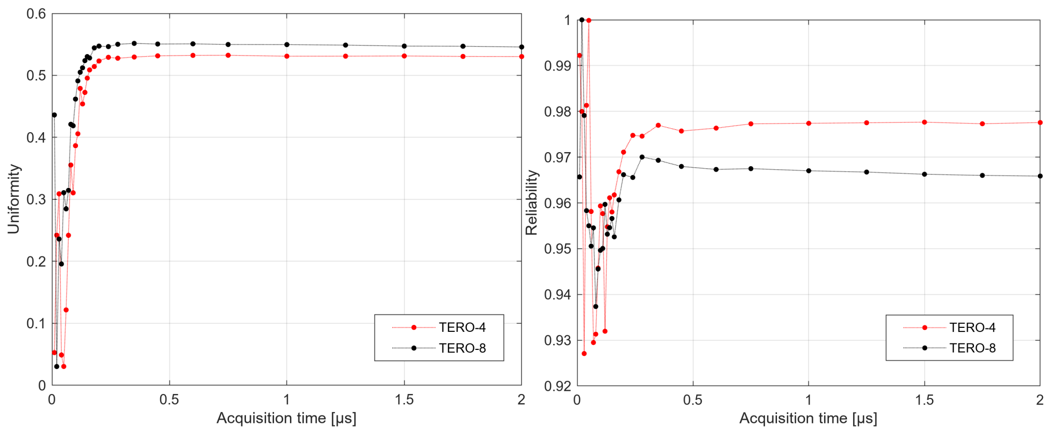
[42] Yi Zhang et al. “A Highly Reliable Strong Physical Unclonable Function Design Based on FPGA”. In: *Journal of Physics: Conference Series* 1619.1 (2020), p. 012003.

Appendices

A XDC constraints

Using the BEL constrain, the exact LUT of a CLB can be specify for a given part of the TERO cell. Additionally, the PIN LOCK constrain provides control on the exact pin of the LUT used for a given signal. Those two constrains can help to build TERO cells that are identical.

On figure 1, we can observe the impact of those constrains on uniformity and reliability for a given device. The red points correspond to the design without the BEL and PIN LOCK constrains while the black one used both of them.



(a) Uniformity (b) Reliability

Figure 1: TERO-8 performance improvement using locked pin

B LSFR combinations distribution

The LSFR is used to generate the combinations of cells to compare instead of storing them in memory. It ensures that all the combinations are covered exactly once except the '0-0' combination never generated, providing a uniform usage of each cell. However, this does not ensure that the cell usage stays as uniform as possible during the generation process, and it can be seen in figure 2 that it is indeed not completely uniform when only a part of the combination is generated.

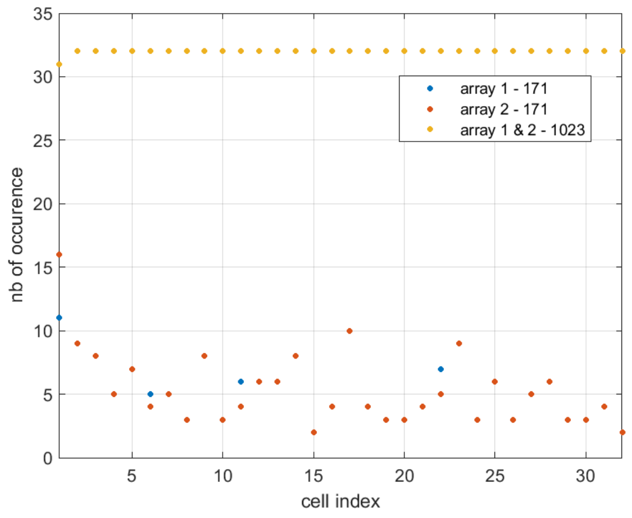


Figure 2: LSFR10 cells selection for only 171 bits

This means that some cells are more used than others for comparison in this case. If those cells have an extreme final state (high or low number of oscillations in regard to the other), the comparisons with those cells will almost always generate the same bit. Since this cell is used more often, this could have a negative impact on the uniformity of the response.

TERO-8	Response size	Uniformity	Reliability	Bit-aliasing	Uniqueness
Full	1023 bits	53.4%	97.6%	53.5%	49.4%
Reduced	171 bits	53.4%	97.7%	54.4%	49.9%

Table 1: TERO-8 performances for full and reduced response

However, as we can see on the table 1, this does not seem to have a significant impact on average on the TERO-8 implementation.

C Python module

A Python module has been written to implement the UART interfaces in an easy-to-use method way. It used the **Pyserial** package for UART communication and **Numpy** to retrieve the data.

The PUF device is represented by the **PUF** class, which requires the port for the UART to be instantiated. We can also specify the baudrate and the initial value for the reference counter (to select the acquisition time).

Each interface with the PUF corresponds to a method:

- **set_ref_limit(limit: int)**: update the limit of the reference counter, to change the acquisition time. The possible values are [1 -> 65536].
- **read_raw()**: return the raw PUF response (1023 bits).
- **read_ecc()**: return the PUF response after the BCH decoder (171 bits). The syndrome needs to be previously set using **set_syndrome(syndrome: str)**.
- **read_sha()**: return the sha256 key (256 bits).

For each reading operation, a second method has been created to easily record multiple samples for the same test. The return of those methods has their own class **SamplesReturnData** that stores the responses samples and computes the corresponding reference response and uniformity. This could easily be extended to also compute the reliability.

A second point that could be improved would be to compute the syndrome from the raw PUF response directly on this module instead of using Matlab. It could therefore automatically do the provision step at start then send the syndrome each time it is needed.