Politecnico di Torino

Department of Control and Computer Engineering (DAUIN)

Master Degree in Computer Engineering

Master Degree Thesis

Design and Development of a Strong Physical Unclonable Function for FPGA devices

Author: Damiano Stochino

Advisor: Paolo Ernesto Prinetto
Co-Advisor: Nicolò Maunero

March, 2022

Abstract

As multifactor authentication has massively spread over almost every security-critical application, electronic devices gained a key role providing a "something only the user has" evidence. Rather than storing a secret key inside a device, as it is usually done, a promising alternative is the employment of so-called Physical Unclonable Functions (PUFs). PUFs exploit physical randomness of a device in order to create a secure challenge-response authentication mechanism. A PUF is therefore a function which associates each challenge to a response, according to some physical and immutable properties of a device.

PUFs can be divided into weak and strong ones. The former are characterized by a large set of challenge-response pairs (CRP) and can be used for both authentication and identification. On the contrary, the latter, given their small CRPs set, are only suitable for identification. Many sources of randomness exist in digital circuits, including, for example, threshold voltages, delays and capacitances. This work is focused on the development of a strong PUF for FPGA devices.

The typical threat model assumes the attacker has the same access level to the device as the developer, which means full physical access and programming capability. Literature shows that some attacks are able to break most of the PUFs that have been known to be strong. Previous work demonstrated that a linear dependency between physical parameters and responses is the main reason for that defeat. However, PUFs having non-linear responses exhibit a better resiliency to that kind of attacks.

It has been shown that the settling time of Bistable Ring, a loop composed of an even number of inverting elements, has an exponential dependency on the threshold voltage of each inverter. Taking that into account, the Bistable Ring is the chosen architecture for this work. Some modifications to the basic design have been made, in order to introduce a challenge-response mechanism and to adapt it to the target FPGA.

Bistable Ring has two stable states only, and, if forced to an unstable state, oscillates for a certain number of times before reaching one of them. With a sufficiently large amount of PUF executions with the same challenge, the corresponding distribution of the number of oscillations can be extrapolated. It has been observed that, even if the number of oscillations can vary a lot among different runs, the distribution itself is sufficiently consistent. On the other hand, changing only one challenge bit leads to a completely different distribution. Such response can be either used directly by an ad-hoc authentication protocol, or converted into a reliable string of bits, through a so-called fuzzy extractor. In order to evaluate the quality of the PUF, rather than using a cryptographic fuzzy extractor, which would mask weaknesses, the Spectral Hashing algorithm by Yair Weiss has been used. Spectral Hashing generates a string of bits which minimizes Hamming distance between codes of similar items. That allows a fair evaluation of the PUF with widely used metrics.

The test system is composed of 12 devices, each provided with eight 64-bits Bistable Rings. All the devices are connected to a host machine running a python testbench, which stores responses into a database. Data have then been analysed offline, computing quality metrics that describe the reliability and the trustworthiness of the designed PUF.

Contents

Chapter 1

Introduction

As multifactor authentication has massively spread over almost every security-critical application, electronic devices gained a key role providing a "something only the user has" evidence. The typical approach is to provide the user with a small, low power and tamper resistant device with a cryptographic key stored inside. The key is inserted in the device during an enrollment phase, usually written in some non-volatile memory. Unfortunately, invasive and semi-invasive attacks [1] are able to extract content from memories, even from volatile ones. Thus, countermeasures have to be taken in order to keep keys secret. These usually resort to expensive anti-tampering techniques [2], such as frequency and voltage sensors, test circuitry destruction, watchdogs, sensor meshes and so on.

Rather than storing a secret key inside a device, an alternative solution is the exploitation of physical randomness for generating it directly within the device. The idea of using process variations of integrated circuits in order to generate a one-way function has been introduced for the first time by [3]. One-way functions (such as hash functions) are important cryptographic primitives used in a variety of applications. They are characterized by the complexity asymmetry in computing the one-way function and its inverse. Mathematical one-way functions are, for example, the prime numbers multiplication and the modular exponent, which both leads to the common computationally unfeasible problems of the prime numbers factorization and discrete logarithm: it is easy to multiply two prime numbers, but no easy way exists to compute two prime numbers given their product. When physical properties of a device are used to generate a that kind of function, it takes the name of *Physical Unclonable Function*:

- Physical: since they resort on physical properties of a device (typically due to

intrinsic process variations).

- Unclonable: it must be difficult to manufacture a second device with the same behavior of another one (*physically unclonable*), or to predict/compute the outcome of the function without resorting to the physical device (*mathematically unclonable*).

- Function: it must be a function, i.e., it must be described by a set of possible input values (*challenges*), and their corresponding outputs (*responses*) [4]. The set of challenge-response pairs (*CRPs*) fully characterizes the function, and it is unique for each device.

Many sources of randomness exist in digital circuits. For example, [5] exploited random threshold voltage variations of MOSFET as a means of generating a unique binary sequence that can be used for identification. Other common sources of randomness are gate delays and initial values of SRAM cells. A single binary sequence is of course not enough for authentication, since an eavesdropper can easily capture and use it for authenticating himself in place of the legitimate device or user.

In Chapter 2 a brief summary on physical unclonable functions is presented. Firstly, the properties that a PUF must respect are described, as well as metrics to evaluate the PUF quality. Then, a number of proposed PUF implementations are reported. Finally, there is an overview about vulnerabilities and most common attacks against PUFs.

In following chapters, the problem of designing a PUF on an already existing hardware is faced. In particular, this work aims to implement a Physical Unclonable Function for the **SE***Cube*™ device family, a security oriented hardware and software platform.

Distinctly, Chapter 3 describes the platform and the tools used for development, implementation and testing of a physical unclonable function. Chapter 4, instead, goes into the details of designing the PUF for the **SE***Cube*™ FPGA.

Finally, Chapter 5 depicts how data have been collected and reports the results of the analysis.

Chapter 2

Background

The concept of *Physical Unclonable Function* has been formalized for the first time by [6]. The authors also proposed the first physical challenge-response authentication mechanism: a programmable delay path is built by allowing the challenge to select different, but functionally equivalent, sections of the circuit to be stimulated; the response is generated starting from the measured propagation delay, which depends on physical properties of the selected circuit sections.

In the following sections we will see a number of ways for building a such kind of circuit, each with its own advantages and disadvantages.

2.1 PUF properties

Given the general idea behind physical unclonable functions, in this section the characteristics a PUF must satisfy are discussed.

In order to assess the *trustworthiness* of a PUF, the following properties must be verified [7]:

- **Evaluable**: The PUF must produce a response in a limited amount of time. What "limited" means depends on the application. From an abstract point of view, the response must be produced in a polynomial amount of time. However, from a practical point of view, more issues have to be taken into account, including the silicon area and power overheads.
- **Unique**: PUFs shall include information coming from physical random properties of the device. It may happen that some physical properties are shared among all the devices, leading to a biased response. It must be assured that

random properties will be predominant over fixed ones. This means that in some case lot of care must be placed in the symmetry of the design. As an example, a longer wire has for sure a larger capacitance (the physical property) than a much shorter one, independently on the specific device.

- **Reproducible**: Conversely to random number generators, different evaluation of the PUF on the same device with the same challenge must produce a similar response, according to some distance metric. In order for the PUF to be reproducible, random variations must be predominant over the random noise that affects the circuit.
- **Unclonable**: Unclonability is of course the most important property of PUFs. In the Introduction 1 the distinction between physical and mathematical unclonability has already been presented and discussed. Physical unclonability must be valid even for the manufacturer, for this reason it is also called *manufacturer resistance*. A critical requirement for unclonability is the size of the challenge set. A small challenge set would allow an attacker which has physical access to the device, even for a limited amount of time, to read all the possible CRPs.
- **Unpredictable**: Unclonability includes unpredictability. If one can predict the response of a new challenge, starting from a limited set of known CRPs, then the PUF can be mathematically cloned. Predicting responses from unknown challenges includes also so-called *modeling attacks*, which usually involve machine learning techniques.
- **One-way**: Given the response of a PUF, it must be unfeasible to obtain the challenge that generated it.
- **Tamper-evident**: This is not strictly required, but some PUF implementations might include it. When the device is tampered, the PUF function is modified (i.e., the CRPs set is substituted with a new one) in such a way it is no longer able to authenticate the device.

It is possible to distinguish and categorize PUFs according to the number of CRPs they can produce [8]:

- **Weak PUFs**. Their challenge-response set is small or even composed of a single response (with no challenge). They can be used for device identification, but are not suitable for authentication. In some cases, they can be adjusted in order to make them strong with minimal modifications.

- **Strong PUFs**. They are characterized by a *huge* set of challenge-response pairs and can be used for authentication. It must be noted that some apparently strong PUFs are vulnerable to attacks that are able to predict any response to any challenge given a small subset of CRPs, as explained in Section 2.4.1.

A simple authentication protocol has been proposed by [9]; in order to assess the authenticity of a device, one can simply record, in the enrollment phase, the PUF response to a chosen challenge, and then compare it later with the one generated in the authentication phase. However, recording a single response is not enough, because an attacker can eavesdrop and use it later in place of the legitimate device.

As a consequence, the general scheme for a PUF authentication protocol shall be based on the following steps:

- **Enrollment**. In a trusted environment, a significant number of challenge response pairs, from the huge set provided by a strong PUF, are extracted and stored in a **secure** database.
- **Authentication**. A trusted verifier selects an **unused** challenge from the database and asks the device for a response. If the response matches with the one stored in the database, then the device is authentic.

It must be stressed the fact that the set of selected challenges must be kept secret. Otherwise, an attacker who has access to the device can reproduce the enrollment phase, and obtain all the responses that may be requested for authentication. In addition, challenges shall never be reused, since an eavesdropper will know their responses in successive authentication sessions.

2.2 Quality metrics

Up to now, the characteristics of a PUF have been presented in an informal way, describing them in a qualitative rather than quantitative way.

In this section, the most relevant quantitative metrics that allow to assess the quality of a PUF are presented. Most of the metrics are based on statistical evaluations, which are not bulletproof, since they might miss some correlation among responses of different devices or challenges.

The authors of [10] summarized the most used metrics for PUF quality assessment:

- **Uniqueness**. Also called *inter-die variation*, it tells how much a PUF can distinctively identify a device. Numerically, it is the average Hamming distance among the responses of k devices to the same challenge. Given an n bits response PUF applied to k devices, the uniqueness U is defined as:

$$U = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(R_i, R_j)}{n} \tag{2.1}$$

  Where R_i and R_j are the responses of two different devices to the same challenge. The ideal value for a large number of devices is $U = 50\%$ (i.e., ideally, the response of a device does not give any information about the responses of other devices).

- **Diffuseness**. The authors of [11] introduced the concept of diffuseness. It consists in the average Hamming distance between responses to different challenges by the same device. The definition is similar to the one of uniqueness:

$$D = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(R_i, R_j)}{n} \tag{2.2}$$

  The difference is that k no longer represents k different devices, but different challenges applied to a single device. The ideal value is still $D = 50\%$

- **Reliability**. It is the complementary of the *intra-die variation*. Ideally, the response of a device to the same challenge should be fixed. In practice, it is susceptible to noise and environmental conditions variations. Supposing to have a nominal response R to a given challenge, it is possible to evaluate the reliability of the PUF by applying the challenge multiple times, possibly under different environmental conditions (e.g., temperature, voltage, etc..). The reliability is given by

$$S = 1 - \frac{1}{k} \sum_{i=1}^{k} \frac{HD(R, R_i)}{n} \tag{2.3}$$

  where k is the number of measurements of the response. The ideal value of the reliability is $S = 100\%$

(2.1), (2.2) and (2.3) are the three main parameters used for PUF evaluation. Uniqueness is strictly bounded with bit-aliasing, a metric proposed by [12] and defined as:

$$\frac{1}{m}\sum_{i=1}^{m} r_{i,t} \tag{2.4}$$

where $r_{i,t}$ is the response bit t of the device i among m devices. It tells if a given bit is biased towards 0 or 1. The ideal value is 50%. If some bias exists, the result of (2.4) will be closer either to 0 or to 1.

2.3 State of the art: proposed implementations

Over the years, a number of possible PUF implementations have been proposed, each resorting to different sources of randomness. In this section, a brief summary of the most interesting proposals is presented.

Despite the large variety of solutions, almost all of them can be grouped in two main categories, according to their structural construction:

- **Delay-based PUFs**: They are based on paths delay measurement, or by the comparison of the delays of couples of paths.
- **Bistability-based PUFs**: The fundamental block is the classical bistable circuit (see Figure 2.3). A bistable circuit is forced to an unstable state and, after a certain amount of time after it is freed, it fells into one of the two stable states, which hopefully depends only on physical variations of the circuit. An accurate attention must be placed on the symmetry of the design, in order to avoid heavily biased responses. A more in depth overview will be presented in Section 2.3.2

2.3.1 Delay-based PUFs

Arbiter PUF

Introduced by [13], it consists of two programmable delay paths. The paths are selected by the challenge bits, through a series of couples of multiplexers (Figure 2.1). Each couple of MUXs represents a delay stage.

Both paths are fed with a rising edge. The final flip-flop works as an arbiter, recognizing the fastest path between the two. Since its challenge set is exponential on the number of stages, it is generally reported as a strong PUF.

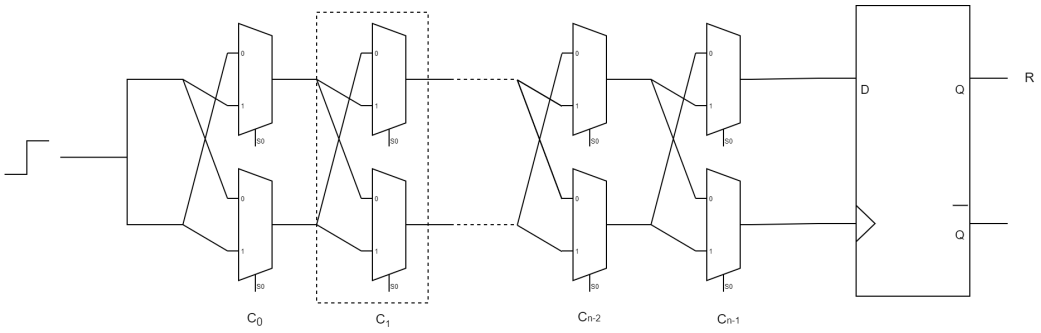

Figure 2.1. Arbiter PUF

Ring oscillator PUF

An alternative for measuring the delay of a path, introduced by [6], is to build a ring composed of an odd number of inverters (Figure 2.2).

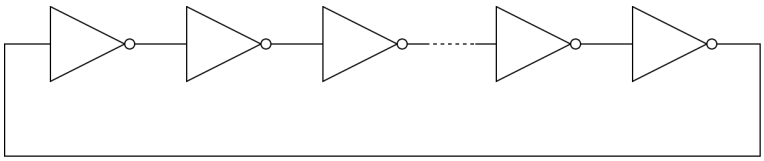

Figure 2.2. Ring oscillator PUF

That kind of circuit is commonly referred as *ring oscillator*, since it is an unstable circuit that oscillates for an undefined amount of time. The oscillation frequency depends on the delay of the entire loop. The ring oscillator in Figure 2.2 is a weak PUF, since only one response is possible. Nonetheless, a challenge response mechanism can be introduced in order to strengthen it, for example as described in [6].

2.3.2 Bistability-based PUFs

This category includes any PUF whose basic block is the classical bistable circuit (Figure 2.3).

The bistable circuit is the starting point of sequential logic. It is widely used, in different forms, for building fundamental logic blocks such as latches and flip-flops. One of the most common implementations is composed of two inverters (Figure 2.3), each feeding with its output the input of the other one. If someone overlaps the

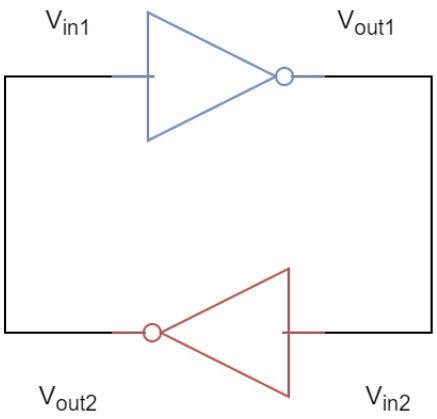

Figure 2.3. Bistable Circuit

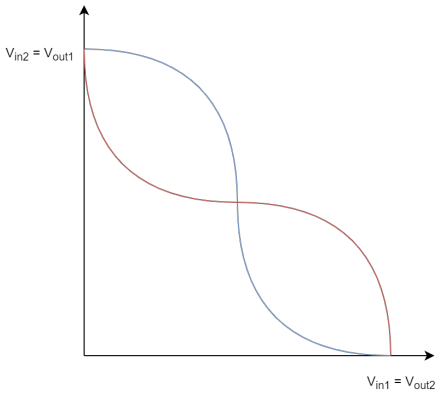

Figure 2.4. Bistable transfer function

transfer functions of the two inverters, obtains the diagram in Figure 2.4, also called *butterfly diagram*. From the diagram, three working points can be highlighted, (the ones in which the transfer functions cross). Two stable points (hence "bistable"), and one metastable point. In sequential logic, the circuit is designed in such a way that can be forced in both stable states alternatively, each state representing a '1' or a '0'. On the contrary, in PUF applications metastability is exploited, as explained in the next paragraphs.

SRAM PUF

Introduced by [14]. An SRAM cell (Figure 2.5 [1]) is actually a bistable that can be forced to any of the two possible states. However, at power-up, when no external signal is activated, its content is initialized with a random value, which depends on intrinsic parameters fluctuations.

The authors of [14] showed that start-up values are constant enough for being used as PUF responses, while the challenge is the address of the target memory word cells. It must be noted that the SRAM PUF is a weak PUF. A memory (and thus the initial values) can indeed be read in a time which is linearly proportional to its size. For the sake of argument, if a memory cannot be read entirely in a feasible amount of time, it cannot be even tested, and thus manufactured. Moreover, most FPGAs force memory content to a known value at power-up, making it difficult to

[1] https://en.wikipedia.org/wiki/File:6t-SRAM-cell.png

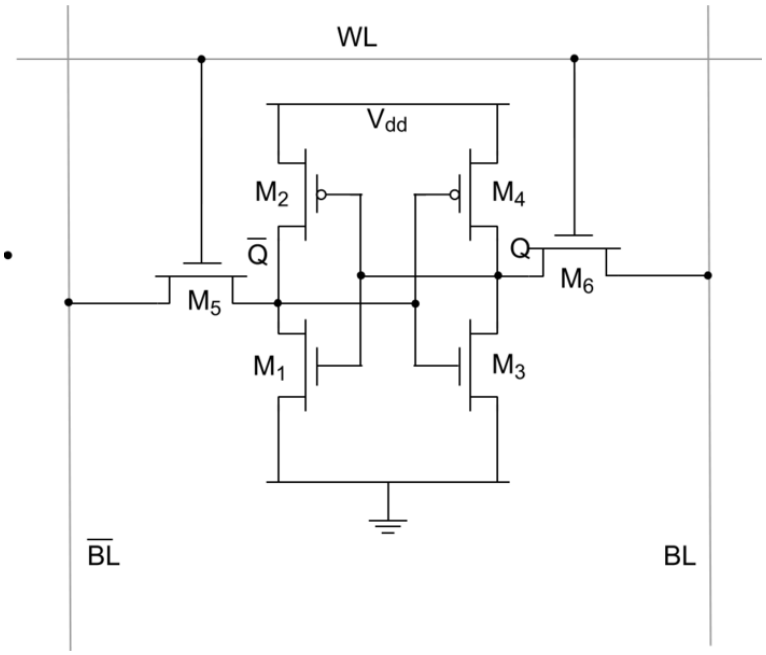

Figure 2.5. SRAM cell

extrapolate responses.

Butterfly PUF

Introduced by [15], it is composed of a single bistable circuit, for a single response bit (in case more response bits are needed, the structure can be replicated multiple times). Its name comes from the picture generated by the bistable state function (Figure 2.4).

The principle is the same of SRAM PUF. In this case, a bistable circuit is assembled in logic hardware, or exploiting for example FPGA structures. The bistable circuit must be forced in an unstable state, i.e., both outputs either to '0' or '1'. This can be achieved when the device is powered-off (the same as SRAM PUF), or by adding some logic, for example transforming the NOT gates in NOR ones (Figure 2.6).

When the *preload* signal is high, both nets are forced to '0'. When the *preload* signal is released, NOR gates behave as inverters, and the bistable circuit goes to an unstable state, which is quickly abandoned, since both inverters starts forcing the a '1' on their outputs (and thus on their inputs). Eventually, after an oscillatory phase, a stable state is reached. The specific final state ("01" or "10") depends on

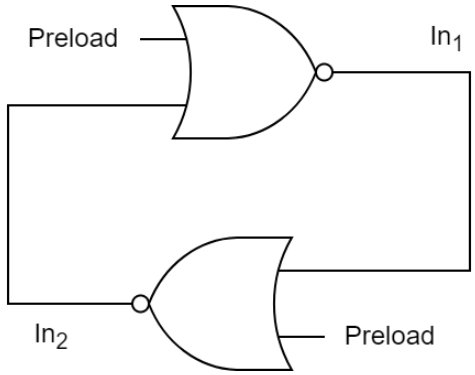

Figure 2.6. Bistable with preload feature

small asymmetries of the circuit, that **may** come from physical variations.

From a practical point of view, the authors of [15] implemented the bistable circuit using two latches rather than two inverters, due to the difficulties introduced by the construction of a combinational loop in VHDL, since they are generally rejected by the synthesizer (the same problem has been faced and solved while implementing the FPGA PUF designed for this Thesis). The authors also highlighted the need of a perfect symmetry of the structure. In fact, since the final state depends on circuit asymmetries, if a large one is introduced in the design, no physical variation can overturn it. As an example, if one of the two wires connecting the NOR gates in Figure 2.6 is much longer than the other one, it will always have a larger capacitance with respect to the other. Therefore, it will be for sure slower to rise-up, and the final state would be with high probability the one with a '0' in that net, leading to a poor uniqueness.

For that reason, the authors of [15] had to manually route the nets connecting the two latches. It must be noted that building a symmetric circuit is not straightforward inside an FPGA, since, usually, no physical information about the FPGA layout is provided by the vendor. The butterfly PUF, as introduced by [15], is a weak PUF. However, it offers the basic idea behind a more sophisticated circuit called *bistable ring*.

Bistable ring

Introduced by [16], it is a generalization of the classical bistable circuit. The basic idea is to build a loop with an even number of inverters (Figure 2.7).

A such kind circuit has only two stable states:

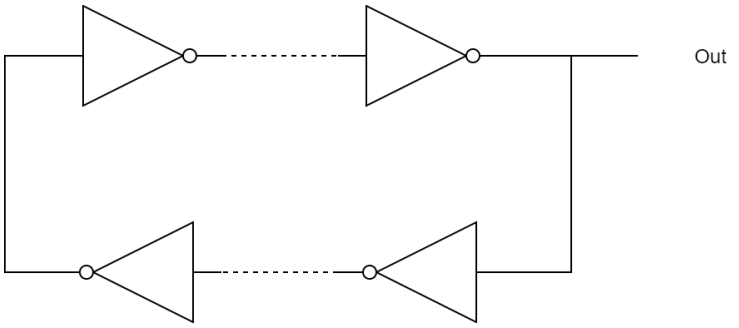

Figure 2.7. Bistable ring

- '0101..0101', that we call *5 state*
- '1010..1010', that we call *A state*

The names *5 state* and *A state* come from their hexadecimal representations, that, in case the number of inverters is a multiple of 4, are only composed of 5 and A digits respectively.

In order to transform the bistable ring into a strong PUF and to make it evaluable, couple of transformations are required, which are summarized in Figure 2.8. First of all, NOT gates are transformed either in NOR or NAND gates and a *preload* signal is added. This allows to force the bistable ring into an unstable state (all ones or all zeros), and then release it to measure the response corresponding with the final state of the bistable. Then, in order to insert a challenge mechanism, all the NOR/NAND gates are duplicated, allowing each challenge bit to select one gate from each couple. The Bistable Ring as described in Figure 2.8 is a strong PUF, since it provides an exponential number of challenges with respect to its size.

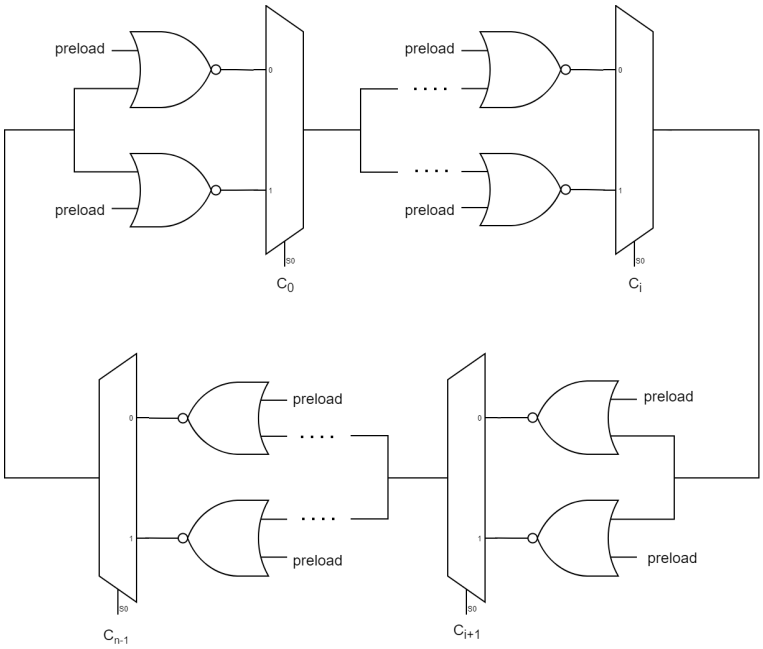

Figure 2.8. Bistable ring PUF

2.4 PUF weaknesses and attacks

In order to evaluate PUFs weakness and possible attacks, it is necessary first to define the attacker and threat model. As observed by [17], the established attack model for strong PUFs includes full physical access and the possibility to apply arbitrary challenges, as well as to read responses. Given that, since in this work the goal is to design a strong PUF for FPGA devices, the following assumptions are taken:

- The attacker has physical access to the device for an undefined, but limited, amount of time
- The circuit embedding the PUF is not resorting to any anti-tampering technique. This allows the attacker to eavesdrop responses.
- The attacker has the possibility to program any aspect of the device, including firmware of MCUs and bitstream of FPGAs. Therefore, the attacker can present to the PUF any arbitrary challenge. This point is important because makes the concept of *Controlled Physical Unclonable Function* useless, as explained below.

The authors of [18] proposed a general solution to protect PUFs from modeling attacks (see Section 2.4.1). They introduced the concept of *Controlled Physical Unclonable Function*. A CPUF is defined as *a PUF that can only be accessed via an algorithm that is physically linked to the PUF in an inseparable way*. In particular, the idea is to force the PUF to be accessed only through cryptographic hashes. For example, a generic CPUF $g(x)$ can be assembled as

$$g(x) = H(f(H(x))) \tag{2.5}$$

Where f(x) is the physical function of the challenge x, and H() is a generic cryptographic hash function.

Computing the hash of the challenge prevents the attacker from choosing arbitrary challenges to pass to the actual physical function. The hash of the response, both prevents the attacker from reading the actual responses and nullifies bit-aliasing, since cryptographic hash functions guarantee a uniformly distributed output.

However, the definition of Controlled PUF is very strong. Building a controlled PUF, as proposed by [18], makes sense only for ASIC based implementations, since any FPGA control logic can be easily circumvented by simply reprogramming the device. Designing a PUF which is resilient even in the abovementioned scenario allows a long term protection against future side channel, internal signal probing and fault injection attacks that may be able to break ASIC based CPUFs and their anti-tampering protections.

It is important to notice that this paragraph is only focused on strong PUFs. Weak PUFs are not considered in this scenario, since they can be cloned without any difficulty by just reading out the entire CRPs set.

2.4.1 Attacks

Modeling attacks

Modeling attacks [17] are attacks in which the adversary tries to mathematically clone the PUF. Their starting point is a subset of CRPs, obtained somehow, from which a mathematical model can be built. The final result is an algorithm that is able to predict with a good reliability the response to any unknown challenge that is presented.

Using diverse machine learning techniques, the authors of [17] successfully broke a ring oscillator PUF and several variations of arbiter PUFs. A crucial role in the

success of every attack has been played by a linear model of the function. Both ring oscillator and arbiter PUFs can indeed be described with good approximation by an additive linear model, where the total delay (and thus the response) is modeled as the sum of the delays of each stage.

The authors of [19], demonstrated that the settling time of a bistable ring depends on non-linear equations, in contrast to the final state, which is linearly dependent on the threshold voltages of the NOR gates. As a result, the bistable ring is a good starting point for building a really strong PUF which is more resilient to modeling attacks.

Side channel attacks

In the case in which the security of the PUF relies on anti-tampering techniques, side channel attacks can be used to leak some pieces of information about the device, including, for example, data stored in memory and digital values of internal pins. Briefly, they involve the measurement of physical quantities such as power consumption, electromagnetic emissions and so on. Side channel attacks will not be considered in this thesis, since our attack model assumes full programming access to the device, which already allows the attacker to access all information side channel attacks are intended to capture. As observed by [17], measuring directly analog quantities and physical parameters, such as delays and threshold voltages, is harder than probing digital signals or reading the value stored in a memory cell. Moreover, side channel attacks require expensive equipment to be carried out. They are usually employed to circumvent protections of weak PUFs, probing signals between actual physical function and control/obfuscation circuitry.

Chapter 3

Development Tools

This chapter describes the development environment used for this thesis work. First, the **SE***Cube*™ platform is presented, used to implement the PUF on the onboard FPGA. Then, there is an overview of the software tools used to develop and implement the design and the test system described in Chapter 4 and Chapter 5 respectively.

3.1 The SE*Cube*™ platform

The hardware platform used for PUF implementation is the **SE***Cube*™ device family, developed and produced by *Blu5 Group*[1]. **SE***Cube*™ Chip (Figure 3.1) is a SiP (System in Package), composed of an STM32 microcontroller, an FPGA provided by Lattice Semiconductor[2] and a EAL5+ *Smartcard*, which has not been used in the context of this work.

The **SE***Cube*™ Chip comes in different form factors, including a development board (Figure 3.2), USB stick (Figure 3.3) and IoT devices.

In this work, the **SE***Cube*™ DevKit has been used, in particular, the internal FPGA has been exploited to build a Physical Unclonable Function, while the microcontroller main task has been the communication with the host system.

In the next lines, the main characteristics of the **SE***Cube*™ components are summarized. For a more detailed description, please refer to the open source project

[1] www.blu5group.com

[2] www.latticesemi.com

Figure 3.1. The **SE***Cube*™

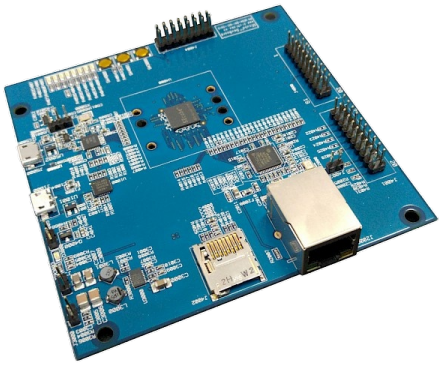

Figure 3.2. **SE***Cube*™ DevKit

Figure 3.3. **SE***Cube*™ USB Stick

wiki[3].

The **microcontroller** embedded in the **SE***Cube*™ is the STM32F429, produced by ST Microelectronics[4] with the following characteristics:

- ARM Cortex M4 32-bits CPU core.
- 256 KB of SRAM.
- Operating frequency of 180 MHz.
- Low power consumption.

[3]https://github.com/SEcube-Project/SEcube-SDK

[4]https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html?querycriteria=productId=SS1577

- Large number of embedded peripherals, such as UART, SPI, USB and SD/MMC.

The **FPGA element** is a Lattice MachXO2-7000 device. Its main characteristics are:

- 7000 LUTs (very small compared with many others available devices in the market)
- 240 Kbits of embedded block RAM
- 256 Kbits of user Flash memory
- Ultra low-power device
- 47 general-purpose I/Os

The FPGA is connected to the microcontroller through a 16-bits internal bus. For security reasons, the JTAG programmer is not exposed externally, but it can only be accessed by the microcontroller.

3.2 Tools

The system in Figure 5.1 is composed of a number of heterogeneous components. Thus, various software tools are needed to program it.

FPGA design - Lattice Diamond

The natural way of programming a MachXO2-7000 FPGA is using the tools directly provided by the Lattice vendor. Lattice Diamond[5] is an FPGA design tool, distributed as freeware software by Lattice Semiconductor for their FPGA devices. It supports both VHDL and Verilog designs, as well as TCL scripting language for synthesis automation.

FPGA *Place and Route* Manipulation - EPIC

Most of the PUF circuits that we've seen in Chapter 2, are not compliant with the requirements of a well-formed hardware design, written either in VHDL or Verilog. In particular, combinational loops are generally forbidden and rejected

[5]https://www.latticesemi.com/latticediamond

by the synthesis tool. Moreover, it has already been mentioned the symmetry demand of bistable-based PUFs, which would be out of control in case of automatic placement and routing. For those reasons, it is not possible to simply describe the wanted design in some hardware description language, but working at LUT and wire level is crucial, since it is required to manually place and route components.

Fortunately, bundled with Lattice Diamond, comes the *EPIC* tool. EPIC is available both as graphical interface and TCL console, and allows a fine-grained manipulation of every programmable structure inside the FPGA device, including LUT logic functions, SLICEs (also called logic elements), wires and switch-boxes configuration.

Microcontroller tools

Since the **SE***Cube*™ is shipped with a very popular STM32 microcontroller, the most straightforward choice for programming it fells into the Eclipse-based STM32 CubeIDE[6]. STM32CubeIDE provides a very easy way for configuring the embedded peripherals, pins, clocks and so on.

Host tools

The host side of the system is mainly composed of Python and Bash scripts that communicate with the SECubes and their ST-Link programmers. In place of proprietary ST-Link programmer software, an open source tool called OpenOCD has been used in order to automate the parallel programming of numerous devices[7].

A MySQL server has been set up in order to store large amount of data coming from PUF responses. With respect to text-based data storage, such as CSV, an SQL server grants better performances and an easier data look-up thanks to the SQL syntax.

The following python external libraries have been used:

- pySerial[8]. Used for UART communication with **SE***Cube*™ devices.
- MySQL Connector[9]

[6]https://www.st.com/en/development-tools/stm32cubeide.html

[7]https://openocd.org/

[8]https://pypi.org/project/pyserial/

[9]https://dev.mysql.com/doc/connector-python/en/

- NumPy[10]. The classical package for python numerical computing, used for the final data analysis.

[10]https://numpy.org/

Chapter 4

Implementation

The goal of this work is to develop a Physical Unclonable Function over an FPGA device. This chapter describes all the steps that have been followed in order to build a **strong** PUF for the **SE***Cube*™ device family. In order for the PUF to be strong, it must own a large challenge-response pairs set. Furthermore, modeling attacks resilience is also an essential condition. Among the PUF architectures have been proposed in literature (Section 2.3), the one that will be selected as a starting point must thus satisfy the abovementioned requirements.

4.1 Chosen baseline: the Bistable Ring

We have already seen in Chapter 2 that the additive delay model is an effective method for breaking any PUF based on delay elements, such as ring oscillator PUF, arbiter PUF and their variants.

Taking that into account, the starting point must thus be chosen among the circuits in the other category, i.e., the ones based on bi-stability.

Among them, SRAMs are not suitable mainly for the following reasons:

- Values of FPGA SRAMs are usually forced to a known value at start up time. This aspect introduces difficulties, often insurmountable, to extract actual start-up values taken by SRAM cells just after power supply connection.
- They are intrinsically weak, since the size of any memory is limited. Reading the entire memory may take some time, but it is generally feasible.
- Even if it were possible to disable the reset mechanism, if the memory is also used for other tasks, the PUF could be excited only in the initial moments

that follow the boot phase.

Given the above-mentioned considerations, it is clear that a strong bistable-based PUF for FPGA devices can only be implemented using FPGA logic elements, which provide the desired flexibility to control any aspect of the circuit. The simplest bistable circuit, composed of only two inverting elements, which constitutes the so-called Butterfly PUF, does not contain enough hardware to introduce a sufficiently large challenge. However, its generalization, the Bistable Ring, can be enlarged at will, at least from a theoretical point of view.

The choice has therefore fallen into the *Bistable Ring*, whose scheme is reported again in Figure 4.1. Moreover, as it was remarked in Section 2.4, it has been proven that, if the oscillation time is taken as a response, it has a non-linear dependency with respect to physical parameters (in that case the threshold voltages of inverters). This factor increases, at least theoretically, the difficulty of performing modeling attacks, as shown by the authors of [19].

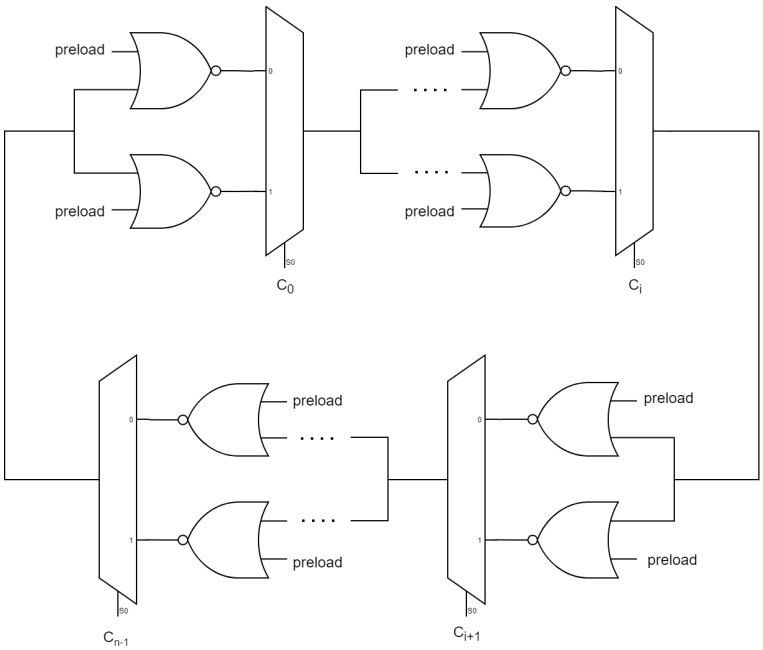

Figure 4.1. Bistable ring PUF

4.2 FPGA adaptation

The circuit in Figure 4.1 is a general description, still not bonded to any specific device. Therefore, it needs to be adapted to **SE***Cube*™ FPGA.

4.2.1 The slice

The Lattice LCMXO2-7000HE FPGA contains 3432 logic elements (called SLICEs in the Lattice terminology), connected each other through the routing matrix. Each logic element can be described as in Figure 4.2.

A slice is composed of two 4-inputs look-up tables (LUT), two programmable flip-flops and some multiplexers. A clock and a reset input are provided to the flip-flops. Some inputs that bypass the LUTs are also available.

The goal is to implement a circuit similar to the one in Figure 4.1 using the slices and the wires provided by the Lattice FPGA, possibly in an area-efficient way.

4.2.2 A programmable delay inverter

The authors of [20] introduced the concept of *programmable delay line* (PDL). A PDL is a combinational circuit in which some inputs act as logic *don't-cares*. However, by acting of them, it is possible to stimulate different physical regions of the circuit, leading to different propagation delays of logic inputs.

It is easy to exploit a LUT to build a programmable delay line, as shown in Figure 4.3. The 3-input LUT behaves as regular inverter of the A1 input. A2 and A3 select the path through which the input is propagated, but do not provide any logic functionality. The programmable delay line has been proposed by the authors of [20] in order to adjust the bias of ring oscillator PUFs. This is done by selecting carefully the *don't-care* inputs, in such a way biased differences among different oscillators are compensated. However, the same idea can be recycled to implement a programmable inverting element. A such inverting element provides the Bistable Ring with a number of inputs that can be used as a challenge.

In order to build a programmable inverting element, with the characteristics required by Bistable Ring and exploiting one of the 4-input LUTs provided by the Lattice FPGA, a preload input must be also attached (Figure 4.1).

A generic i_{th} inverting element, is composed of a LUT configured as in Figure 4.4. When the *preload* signal is high, the In_i output of the LUT is forced to '1'. On the other case it takes the inverted value of the In_{i-1} input. Two *don't care*

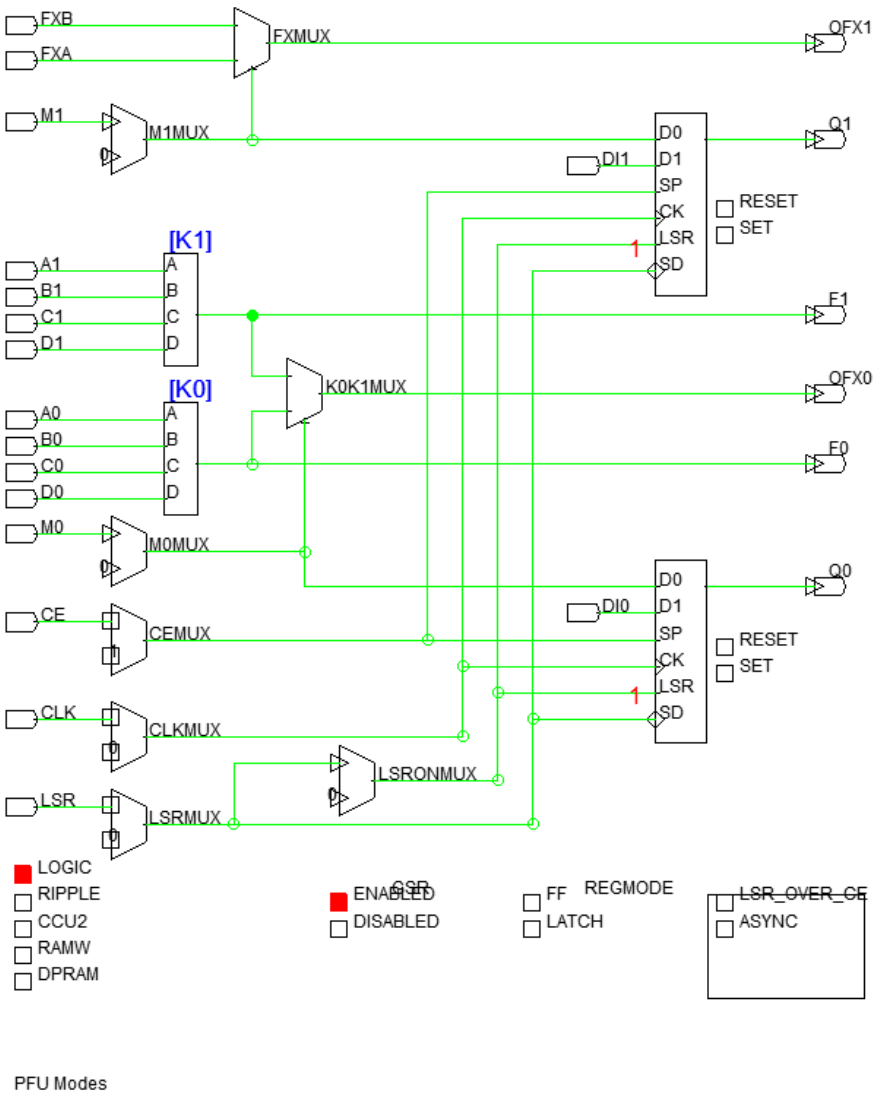

Figure 4.2. Lattice LCMXO2-7000HE Slice

signals, Ca_i and Cb_i, are used as challenge, since the just select the path inside the LUT through which the In_{i-1} input is propagated.

4.2.3 Assembling the Bistable Ring

As a final step, we have to place and connect sequentially an even number of above-described inverting delay lines, forming a bistable ring loop. Figure 4.5 shows a

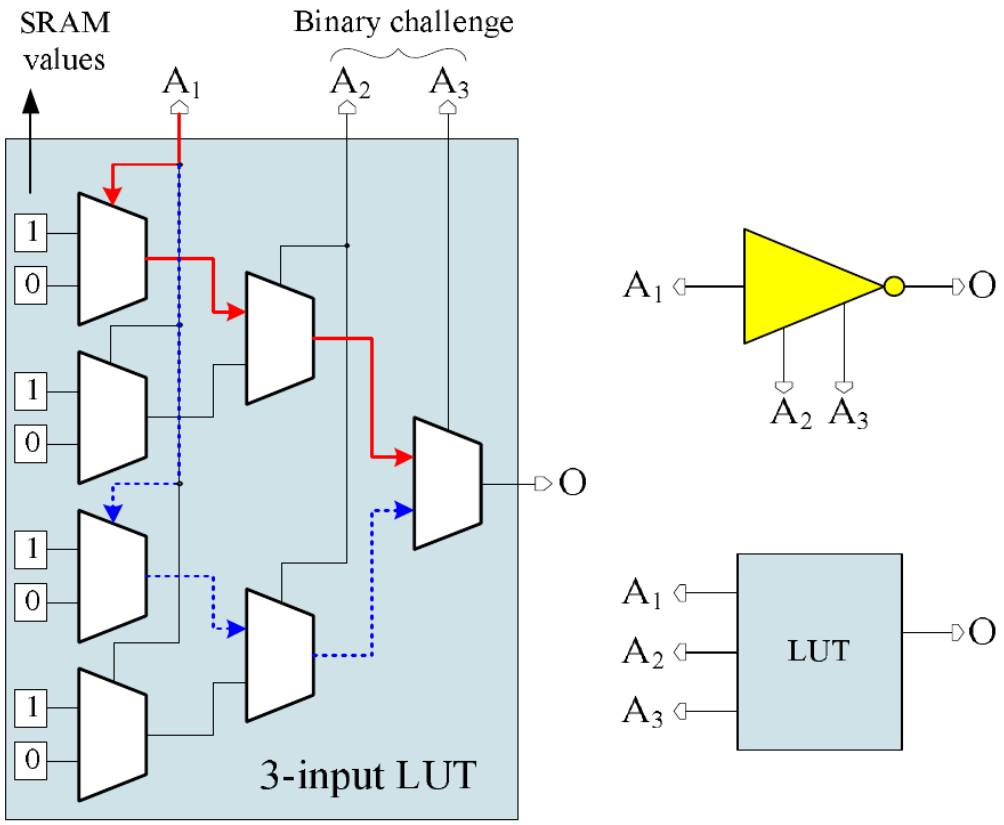

Figure 4.3. Programmable delay line [20]

Bistable Ring composed of 4 inverting elements and an 8-bits challenge. Each inverting element is a Programmable Delay Line like the one described above.

This step must be done very carefully, since any asymmetry, such as a net routed in a much longer wire with respect to others, would bring to a PUF with very bad performances. In particular, attention must be placed on the nets connecting each inverter to the following one (In_i signals). The *challenge* and *preload* signals have less strict requirements and can be routed freely.

Embedded Lattice algorithms are meant to explore the area/performance design space. There is not any intention in them to target design symmetry. Because of that, placement and routing must be done manually, using the EPIC tool. Fortunately, EPIC also provides a TCL console that can be used to automate the process.

Figure 4.6 shows the arrangement that has been chosen in order to minimize

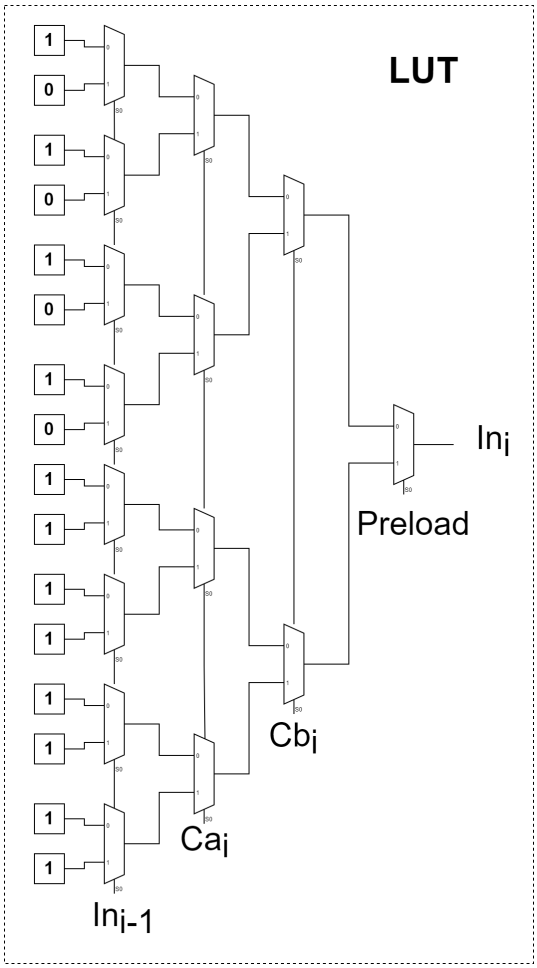

Figure 4.4. PDL with preload

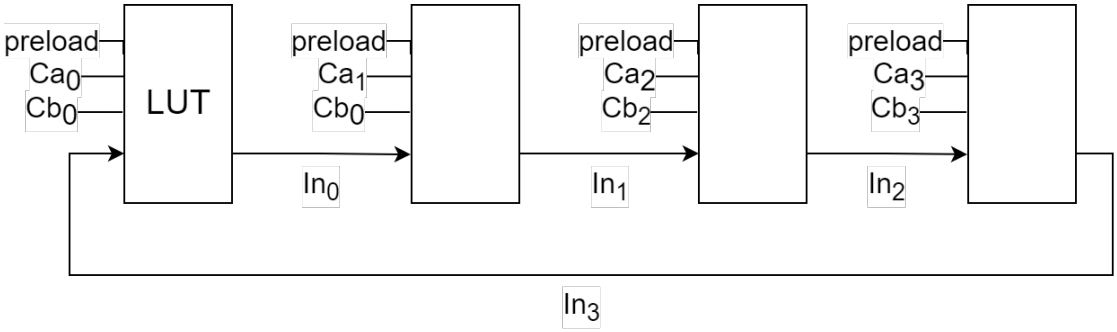

Figure 4.5. 4-Inverters Bistable Ring

differences among In_i wires. In order to keep the image as clean as possible, routes of *preload* and *challenge* nets are not shown, since they are not critical. Moreover, only relevant connections are shown (in particular, switch-boxes are actually much bigger and contain a lot of possible routes). The circuit is the same as the one in Figure 4.5, but LUT components are placed and nets are routed. Since there is not any particular reason to have full control of *preload* and *challenge* routes, they are routed resorting to automatic Lattice algorithms.

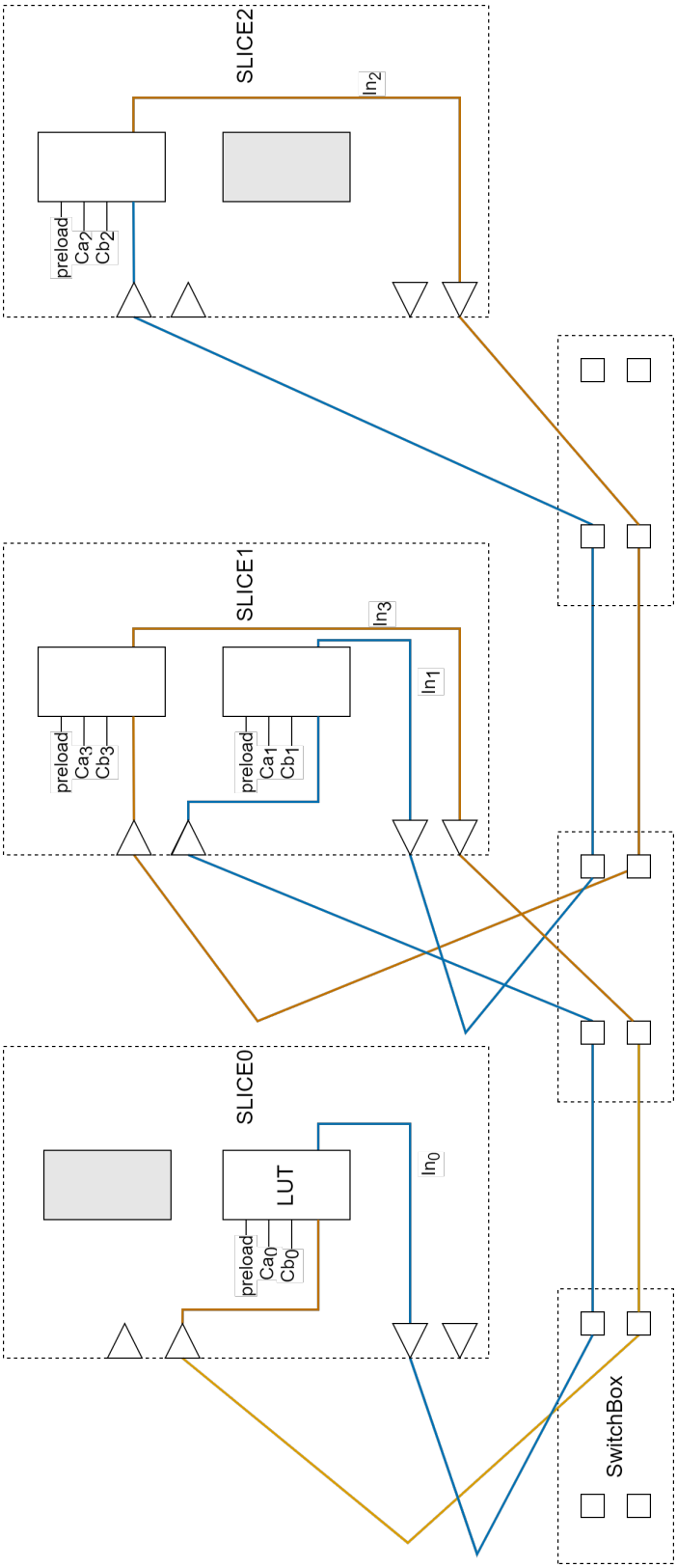

Figure 4.6. Routed 4-Inverters Bistable Ring

4.3 Control logic

Having a bistable ring alone is not of course enough. Initially, it is necessary to decide what to take as response and how. It has been decided to count the number of oscillation, starting from the instant in which the loop is kept freed to oscillate up to its stabilization. Counting oscillations is an easier task with respect to count settling time. It is sufficient to connect any net of the ring to the clock input of a counter. Besides this, a finite state machine that manages a challenge-response protocol must be designed. In particular, the finite state diagram in Figure 4.7 has been implemented.

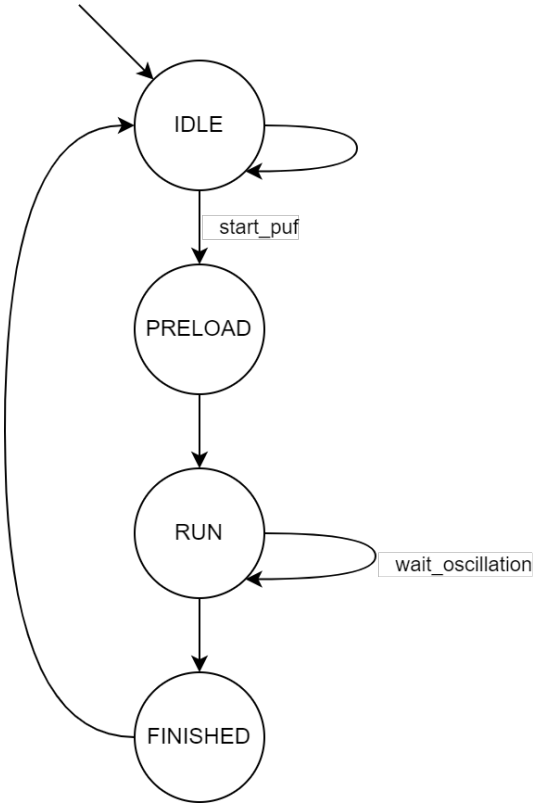

Figure 4.7. PUF control finite state machine

- At power up, the FSM starts waiting in the **IDLE** state.
- When a new PUF request comes from the microcontroller (*start_puf* signal), it goes to the **PRELOAD** state. In the *PRELOAD* state, the challenge is applied, and the *preload* signal is raised, forcing the output of every element

of the ring to a high value. In addition, the content of the counter is reset to 0.

- Just a clock cycle later, the **RUN** state releases the *preload* signal. The loop behaves now as a bistable circuit, but, at the beginning, finds it-self in an unstable state. Then, an oscillatory phase begins. The FSM remains in the *RUN* state as long as oscillations are ongoing, while the counter is counting them.
- When the oscillatory phase finishes, the **FINISHED** state informs the microcontroller that it can read the number of oscillations from the counter. Then, the FSM goes back to the *IDLE* state, waiting for a new request.

In order to detect the termination of the oscillatory phase, there are two options available:

- Continuously monitor the state of the bistable and detect somehow it is no longer oscillating.
- Wait for a reasonable fixed amount of time.

The latter may seem a rough solution, but with its simplicity is also very effective. It is sufficient to determine experimentally a number of clock cycles, and thus an amount of time, after which the oscillatory phase is terminated with high probability. In some cases it may take a relatively long time, or in any case longer than the defined timeout, but if that happens with a low rate, then it is acceptable. Hence, in order to wait for a predefined amount of time, an additional counter has been introduced, that simply counts a fixed number of clock cycles the FSM must remain in the RUN state.

In order to optimize the area occupation even more, the oscillations counter has been also described in terms of LUT elements. A very simple, easy to describe and compact asynchronous counter is sufficient for this task. In this way, a Bistable Ring with a 64-bits challenge can fit in just 33 slices, divided in 17 slices for the actual ring (Figure 4.6) and 16 for the asynchronous counter (one slice for each bit).

The FPGA design is organized as shown in Figure 4.8:

- A low level design, described using the TCL commands provided by EPIC. It is described by manually placing and routing components inside the FPGA array. It includes only the physical PUF circuit and a counter. Preload signal is treated as an input, while the output consists in the value of the counter.

- A VHDL design, whose inputs and outputs include the signals coming from and going to the low level design. It includes the logics for executing the PUF, and the bus interface that allows the communication with the microcontroller. It must provide as output the preload signal, in order to interact with the Bistable Ring. The value of the counter, which corresponds to the PUF response, is considered an input of the VHDL design.

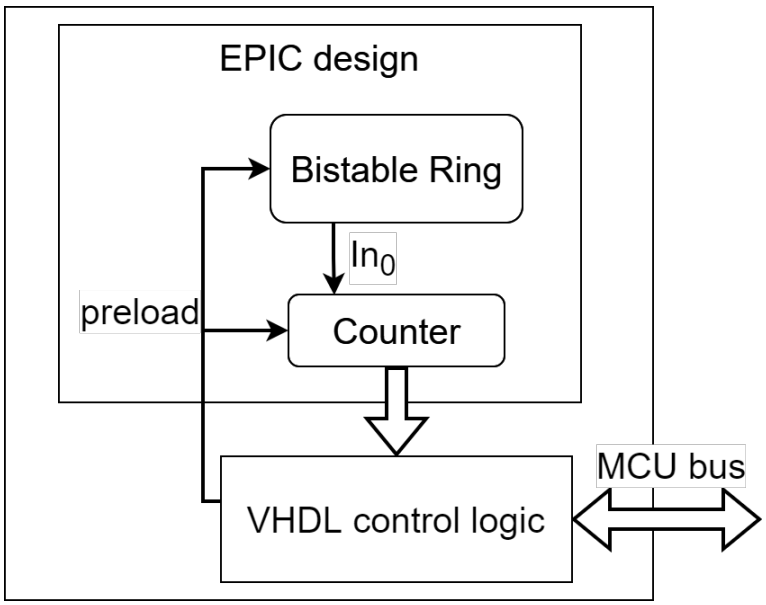

Figure 4.8. FPGA overview

4.4 Compilation Process

A TCL script, then, is used to perform the chain of operations needed to produce the final bit-stream for the FPGA. The procedure can be summarized as follows:

- **Synthesis**: The VHDL design is synthesized with Lattice Synthesis Engine, embedded in Lattice Diamond. The purpose of synthesis is to generate a gate level netlist starting from a VHDL description of the circuit. In this stage, and in the following one, only control logic is involved.
- **Mapping**: The Map process translates the output of synthesis, which is composed of generic logic gates, into device specific logic elements (SLICEs). The

output is an *ncd* file containing circuit description, which is used as input for *Place and Route*. Lattice Diamond is the reference tool for this stage as well.

- **Place and Route**: This process is accomplished using the EPIC tool, and can be divided in the following steps:
  - The *ncd* file containing the mapped design is loaded. *Ncd* file contains a set of unplaced components and unrouted nets, results of VHDL design compilation (synthesis and mapping).
  - Components of Bistable Ring and counter are instantiated and placed. A component is identified by a name and defined by a configuration string for the target SLICE. Configuration includes LUTs logical functions, MUXs, Flip-Flops and so on. In general, any element present in Figure 4.2 can be configured. Placing a component means associating it exclusively to one of the available SLICEs in the FPGA.
  - Nets connecting Bistable Ring and counter components are instantiated and routed. A net is identified by a name and described by the set of pins to which is connected. Routing a net means defining a set of exclusive wires, among the ones available in the FPGA, which is able to connect all the pins related to it.
  - Finally, the remaining (i.e., the ones coming from the VHDL design) components and nets are placed and routed resorting to automatic algorithms provided by Lattice.
  - A new *ncd* file is generated. This time, the *ncd* file contains a list of fully placed components and routed nets.
- **Bit-stream generation**. This step is carried out again with Lattice Diamond. The *ncd* file produced by EPIC in the previous step is taken as input. Starting from the circuit description, the sequence of bit (bit-stream) to be provided to the FPGA in the programming phase is produced.
- **Bit-stream C file**. The bitstream produced in the previous step is converted into a C language array and placed in a C source file template. This step is needed because the microcontroller is the only entity that can program the FPGA, because FPGA programming pins are not exposed to the external world. During the programming phase, the microcontroller communicates with FPGA using the JTAG protocol to upload the bitstream.

Chapter 5

Test and Data analysis

Once the FPGA design containing the PUF circuit is implemented, a system that collects data in a systematic way must be developed. Alongside the Lattice FPGA, **SE***Cube*™ is provided with an STM32 microcontroller, which is the most straightforward way to read raw data from the FPGA. However, it is obvious that data cannot be stored in a low power and resource limited microcontroller. For this reason, a Linux machine is used to collect data from a certain number of Secubes. Then, data can be analyzed offline, resorting to the many available Python libraries.

5.1 Test system

The general architecture of the system is depicted in Figure 5.1.

In order to acquire more data from a single device, 8 bistable rings for each **SE***Cube*™ are instantiated.

The communication between the FPGA and the microcontroller is done through a memory-like logic, which is accessed by the microcontroller via a 16-bits internal BUS provided by **SE***Cube*™. The response of each bistable ring is mapped to its own address that can be accessed by the microcontroller.

The microcontroller communicates with the host through a UART interface, which runs at 1 Mbit/s.

The host computer runs a python backend, that stores results into a MySQL database, for a later analysis. The host, which is basically a Linux machine, supports of course multiple SEcubes connected at the same time, so that we can also perform a uniqueness analysis in a reasonable time.

The python script runs a random sequence generator, which feeds the device

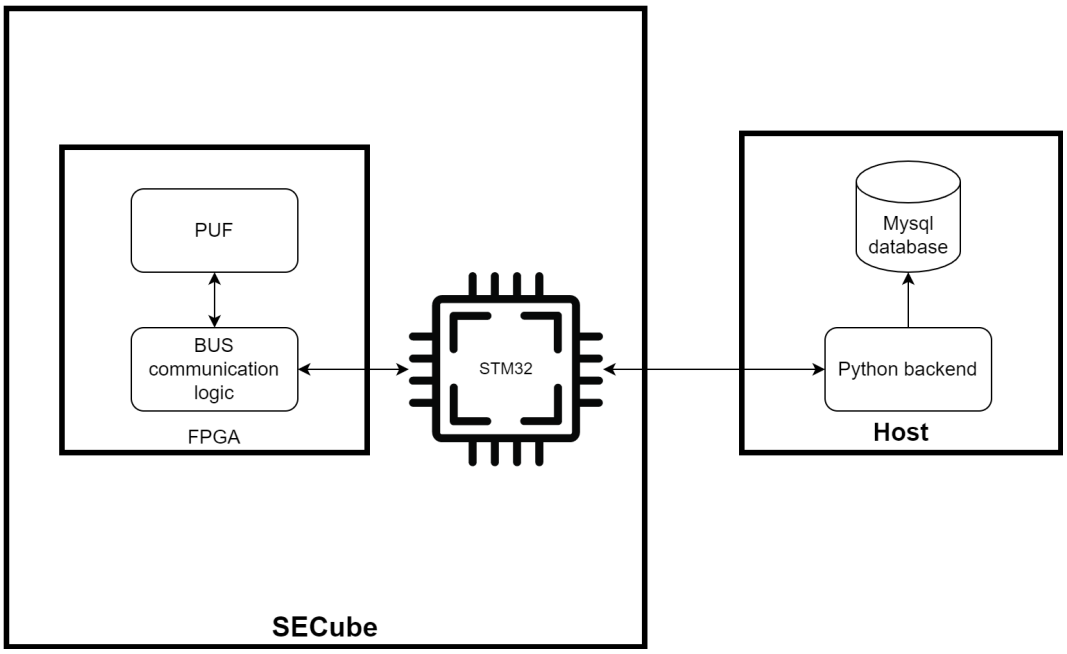

Figure 5.1. System overview

with a large number of pseudo-random challenges.

The purpose of this scheme is to extrapolate the distribution of the number of oscillation, given a device and a challenge. It has been observed that at least 10 thousands runs are needed to obtain a satisfactory approximation of the distribution. In order to not overload the UART link, some computations are carried out directly by the microcontroller, rather than transferring raw data to the host.

The working scheme can be described as follows:

1. A challenge and a number of runs are sent to the device and received by the microcontroller.

2. Once the microcontroller has received the challenge, it writes it into the FPGA memory. From now, any PUF run will be executed with that challenge.

3. The microcontroller runs the PUF for the requested number of times. For each run, the following operations are done:

   (a) The microcontroller writes a '1' at a specific FPGA memory address in order to signal the start of a new PUF run. This basically set the `start_puf` signal (Section 4.3).

(b) The internal FSM, which is listening to the `start_puf` signal, starts the evaluation of the 8 Bistable Ring PUFs.

(c) While the Bistable Rings are oscillating, the microcontroller polls the same address it has written in the first step, waiting for it to be reset.

(d) When the PUF evaluation is completed, the FPGA signals it back to the microcontroller (by resetting the abovementioned address). The microcontroller can read the number of oscillations from the FPGA.

(e) The number of oscillations is used as an index to access an array value which is incremented. In this way the distribution is assembled. The size of the array is determined experimentally. It has been observed that a maximum of 400 oscillations for each Bistable Ring is sufficient, since a higher number is uncommon.

4. The microcontroller sends back, via UART, the extrapolated distribution to the host.

5. Finally, the python script stores the result into the database with all the necessary information. Stored information includes: device serial number, Bistable Ring identification number (from 0 to 7), challenge, timestamp.

Firmware

The procedure described in the previous section is implemented with a bare metal firmware which runs on the microcontroller. The communication between the host and the microcontroller is handled by an ad-hoc protocol. Host can send *commands* to the microcontroller, that sends back a response. The size of the response may vary between different commands. Therefore, the host must know it in advance. The size of a command is, on the contrary, fixed. 4 bytes are used to define mnemonic ASCII commands.

The following commands are available:

- `helo`: The only purpose of this command is to test the communication functionality. Its behaviour is to send back the "Hello World!" string.

- `fpga`: When the microcontroller receives this command, it begins programming the FPGA, or rather, loading the bitstream to it. When the programming is completed, it signals it back to the microcontroller with the "FPGA programmed" message. This command must be called at least one time for each device. There is no need to reprogram the FPGA at every reset.

- `chal`: Loads a challenge and a requested number of runs for the following PUF running sessions. After receiving this command, the microcontroller also expect 8 bytes for the challenge and 2 bytes for the number of runs. This command must be called before the `puff` command, since it defines its configuration.
- `puff`: Begins the PUF evaluation with the configuration specified by the `chal` command. The host receives, as-is via UART, the array containing the number of oscillations distributions of the 8 Bistable Rings (400 2-bytes values for each BR).

5.2 Qualitative results

In Section 2.1 it has been presented a simple PUF-based authentication protocol; it has been said that the response given by the device we want to authenticate must match with the one stored in the database. However, it is unlikely that a PUF will respond in the same exact way every time it is excited. The causes of this behavior are various. They range from purely random noise, to systematic variations, such as room temperature and supply voltage. Despite this, in practice there is no need to get an exact match in order to decide that a device is authentic [21]. What matters is the possibility to define a distance threshold above which the device is not considered as authentic. The definition of distance depends on nature of the response. For example, if the response is composed of a sequence of bits, Hamming distance is the trivial choice.

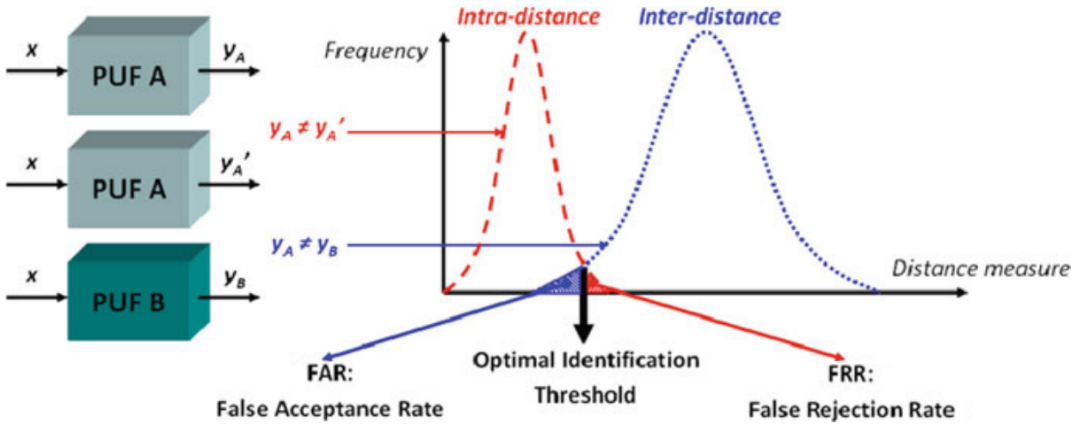

Figure 5.2. FAR and FRR with respect to distance threshold [21]

In order to define an optimal threshold, it is convenient to plot both the distributions of intra- and inter- distances (Figure 5.2). A possible optimum threshold is the crossing point between the two distributions. However, other trade-offs can be applied depending on the application. The authors of [21] also introduced the concept of False Acceptance Rate (FAR) and False Rejection Rate (FRR). The blue area, on the left of the threshold, represents the percentage of unauthentic devices that are recognized as legitimate (False Acceptance Rate). Vice-versa, the red area represents the percentage of legitimate devices that, due to a not perfect PUF reliability, do not pass the authentication process. It is important to notice that the authors of [21] refers to inter-distance, represented as a blue line in Figure 5.2, as what can be called inter-device distance, since it corresponds to the distance among responses from different devices. However, the same reasoning can be applied to inter-challenge distance within a unique device. In the ideal case, it must be difficult to distinguish the case in which two responses are generated by two distinct devices, from the one in which the same device is stimulated with different challenges. On the other hand, it must be easy to distinguish between a legitimate response and a wrong one (either generated by another device, or by the same one starting from a different challenge). Despite this, more studies are needed to explore the effectiveness of modeling attacks on this kind of PUF circuits, and on the possibilities to craft a fake response that passes the authentication process.

5.2.1 Response extraction

The next step is to decide what to take as PUF response. The initial idea was to plot the oscillations number distribution in order to understand if the number of oscillations (or only a subset of its bits, as proposed by [16]) was sufficiently reliable to be used directly as a response. Figure 5.3 shows the oscillation number distribution computed with 10000 PUF runs of a single Bistable Ring with the same challenge. Figure 5.4 shows the same distribution after filtering out "noise".

It appears immediately clear that the number of oscillations can vary a lot, and it is very far from being suitable to be used as PUF response. The same apply for almost any challenge.

Figure 5.5 shows that distributions from distinct challenges are overlapped with each other for large intervals. The same happens if the distributions of number of oscillations from different devices are plotted (Figure 5.6).

This prevents a device from being authenticated reading the response of a single PUF run.

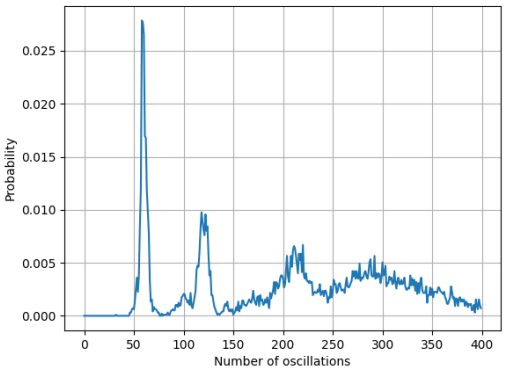

Figure 5.3. Oscillations number distribution

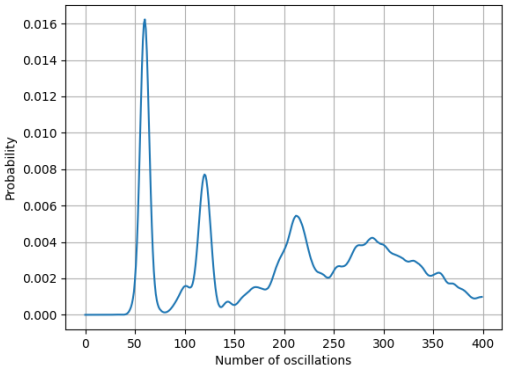

Figure 5.4. Oscillations number distribution (filtered)

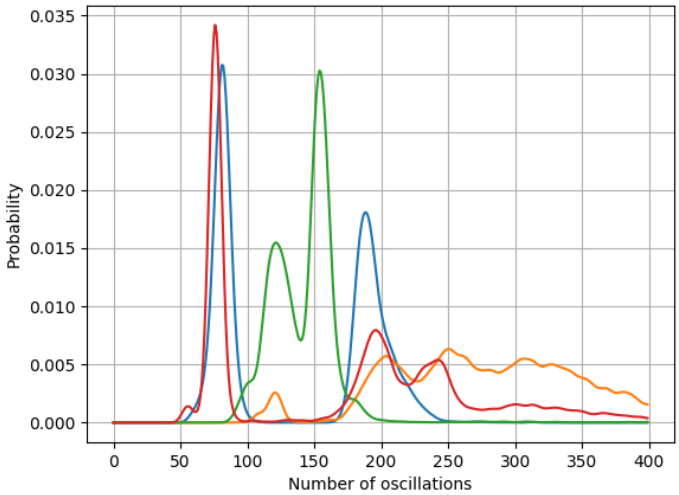

Figure 5.5. Oscillations number distributions with 4 different challenges

However, it has been observed that, even if the number of oscillations can vary a lot, its distribution is much more stable.

Figure 5.7 shows 4 realizations of the number of oscillations distribution applying the same challenge to the same device. The 4 distributions have been obtained at days distance among each other. Despite that, they are almost overlapped.

Given the above observations, a new idea comes in mind. Rather than taking the oscillations number, its distribution can be used as response. This has the

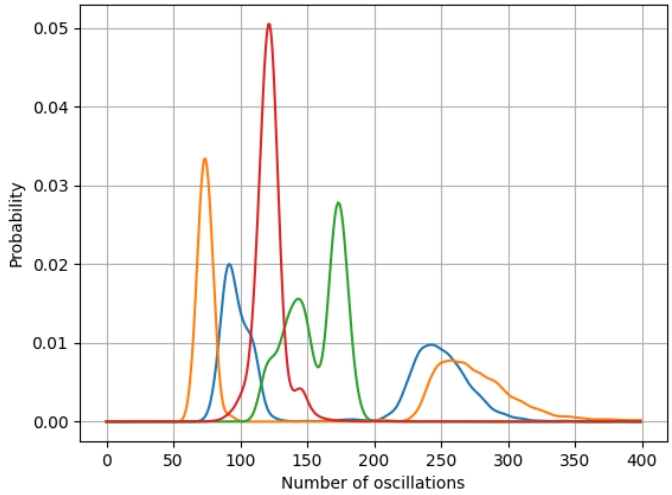

Figure 5.6. Oscillations number distributions with 4 different devices

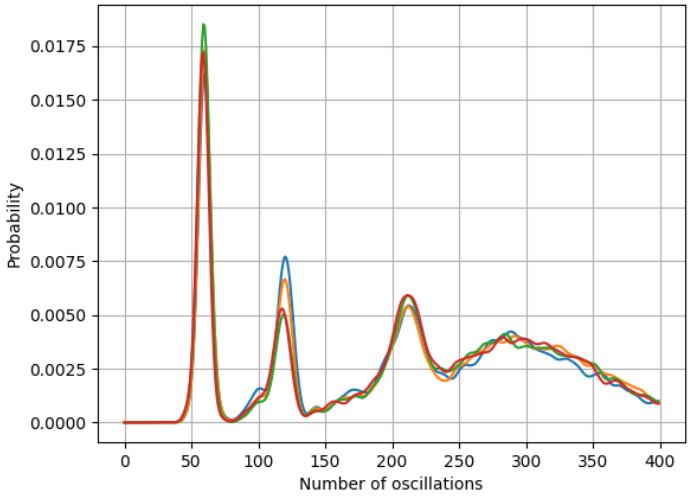

Figure 5.7. Oscillations number distributions, 4 times the same challenge

following consequences:

- In order to authenticate a device, a significant number of consecutive PUF runs with the same challenge are needed. In this case 10000 seems a reasonable number.

- Given the previous point, authentication needs a longer time. With our design, computing a response distribution takes at least 2 seconds. However, this is not necessarily a drawback. Having a very slow PUF prevents an attacker from leaking numerous CRPs, if it has access to the device for a limited amount of time. On the other hand, the enrollment phase, which consist in the construction of a CRPs authentication database, will be more expensive in terms of time. Nevertheless, only small attempts have been carried out to improve PUF evaluation time.
- If the distribution is taken as response, a different distance metric, with respect to Hamming distance, must be chosen or defined.
- If a bit array response is required (for example for performances and database storage optimization), an algorithm that converts the response into a bit string is needed. The conversion must preserve the distance relationship among responses. In other words, if two responses are close together according to the defined distance metric, the Hamming distance between the respective bit strings must be small as well.

Starting from this point, the word *response* will be used to identify the number of oscillations distribution, obtained, in this work, resorting to 10000 consecutive PUFs runs.

5.3 Quantitative analysis

In order to obtain a good amount of data for statistical evaluations, 12 **SE***Cube*™ devices have been connected to the host. As already mentioned, each device is provided with 8 Bistable Rings, in order to increase the size of the response.

Each device has been asked to compute for 100 times the responses to 2000 challenges. Each response needs 10000 PUF runs to be computed, as explained before. The test took around 15 days to be completed. Around 20 GBs of data have been collected.

5.3.1 Response interpretation and distance metric

One of the most straightforward distance metric that can be used is the Euclidean distance. The distribution in Figure 5.3 can be interpreted as a point in 400-dimensional space, where each number of oscillations represents a coordinate, and

its probability is the value of that coordinate for that point. Therefore, the Euclidean distance between two distributions can be computed as:

$$D(y_A, y_B) = \sqrt{\sum_{k=1}^{n} y_A(k)} \tag{5.1}$$

Where, in case of a single Bistable Ring $n = 400$, which is the maximum number of oscillations that has been set. If more than one Bistable Ring are present for each device, then the dimension of the space will increase proportionally.

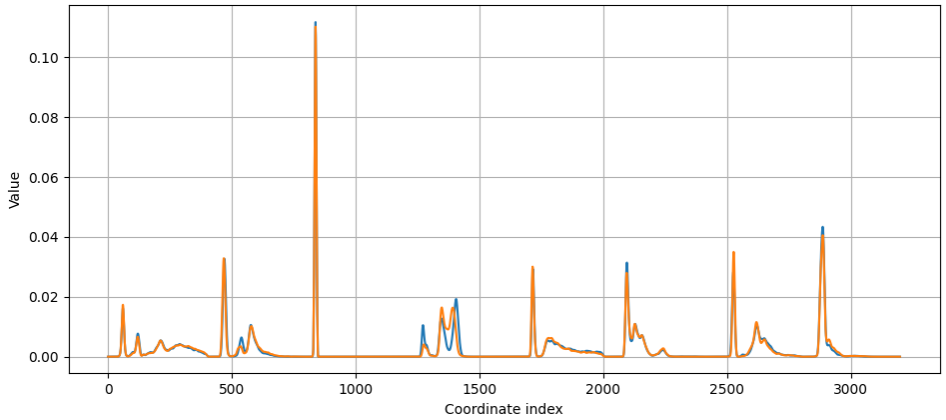

Figure 5.8. Response from 8 Bistable Rings, 2 times the same challenge

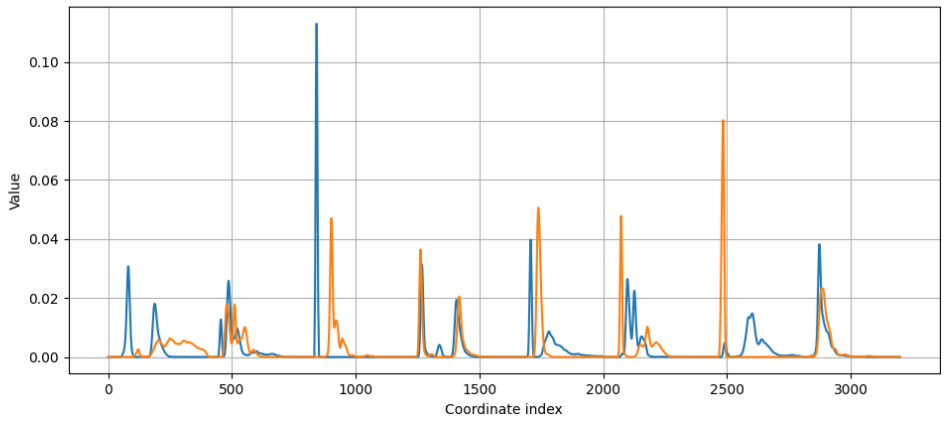

Figure 5.9. Response from 8 Bistable Rings, 2 different challenges

For example, Figure 5.8 shows two responses to the same challenge obtained from a device provided with 8 Bistable Rings. On the contrary, Figure 5.9 shows the response to two distinct challenges. Beyond qualitative observations already performed in Section 5.2, it is possible to compute, for both cases, the Euclidean distance between the two responses. In this particular case, a distance of 0.007 exists between the two responses obtained from the same challenge (Figure 5.8), whereas the distance between the responses to distinct challenges (Figure 5.9) is one order of magnitude larger (0.82). From this perspective, it is precipitous to state about an easy distinction between an authentic device and a fake one. Therefore, a clever analysis, exploiting all available data, is performed.

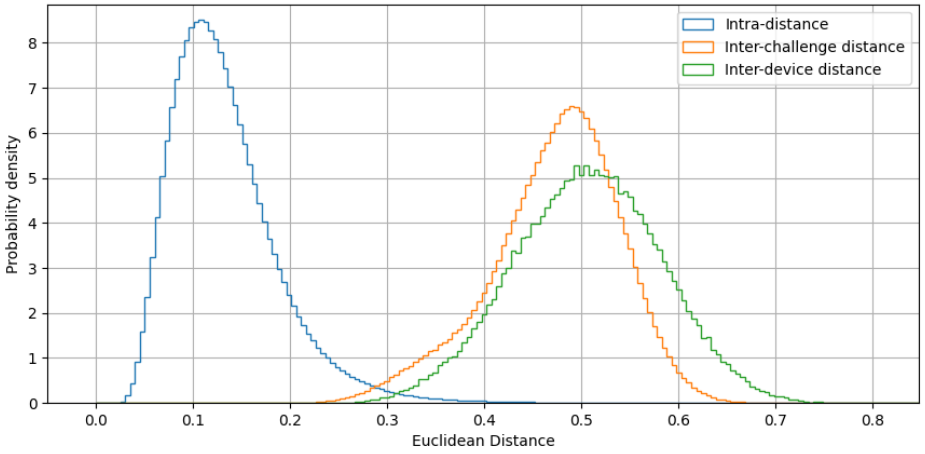

Figure 5.10. Distance distributions - 8 Bistable Rings

Figure 5.10 shows the distributions of intra-distance, inter-challenge distance and inter-device distance. Starting from them and given an authentication threshold, both FRR and FAR can be computed. It is important to notice that if the definition of FRR is unambiguous, FAR can be computed either based on inter-device or inter-challenge distance. The former represents the probability that an attacker with an equivalent (but distinct)device will respond correctly to an authentication challenge. The latter is the probability an attacker, who eavesdropped a response to a given challenge from the legitimate device, will be able to authenticate when asked with a different challenge. Since, looking at Figure 5.10, inter-challenge distance gives more pessimistic results, it will be used for FAR computation.

By moving the threshold it is possible to compute various FAR and FRR values,

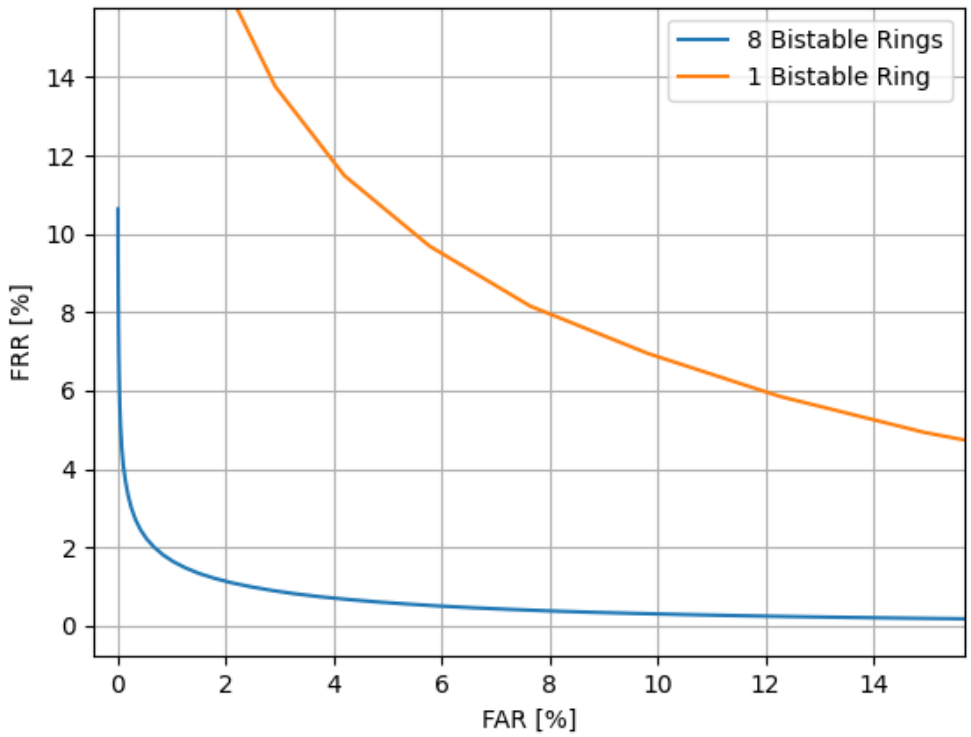

Figure 5.11. FAR vs FRR Pareto curve

in order to allow choosing the most appropriate trade-off for the application. Figure 5.11 shows Pareto curves of FAR vs FRR for a combined response from 8 Bistable Rings (which corresponds to the distributions in Figure 5.10) and from a single Bistable Ring. It is very clear that having multiple rings improves a lot the PUF performances. In particular, it is possible to set a threshold that gives both FAR and FRR lower than 2%. In any case, even more Bistable Rings can be used, or either combined differently in order to define a more complex authentication protocol (e.g., authentication is acknowledged if at least 6 over 8 Bistable Rings pass the challenge).

5.4 Obtaining a binary string response

Response as defined in the previous section is very unmanageable. It occupies a lot of storage space in the database and Euclidean distance computation takes a

relatively long time. Moreover, it is not possible to compute on a response with a such form the metrics as defined in Section 2.2, which assume a bit string response and make use of Hamming distance.

Fuzzy extractors [22] are cryptographic primitives that can be used to extract a bit string from a noisy response. Fuzzy extractor were originally developed with the purpose of using biometric data as input for standard cryptographic algorithms. They assure tolerance on raw data noise, as well as a uniformly distributed output. In PUFs domain, they are employed in cryptographic keys generation. However, computing the PUF metrics on a such generated output, would give an almost perfect result. In fact, Fuzzy extractors are intended to tolerate the unreliability of raw data and to mask its weaknesses, such as biased responses. They shall be part of the final application, but it would be unfair to use Fuzzy extractor output to evaluate PUF quality. This work is focused on the development of the underlying physical unclonable function, and not on the post-processing techniques that can be used to improve it a posteriori.

Another way for generating a bit string response must be found. The authors of [23], proposed a method, called Spectral Hashing, for building a binary code that maps similar items to similar code-words. Items are defined similar in terms of Euclidean distance. Therefore, it is possible to map a set of items to a set of code-words whose Hamming distance correlates with the Euclidean Distance of original items. Spectral Hashing has been developed with the focus on image recognition. However, it also fits well with the response of the PUF designed in this work. In fact, a way to map PUF responses in a Euclidean space has been already defined in Section 5.3. It is possible thus to apply Spectral Hashing to responses of a device and obtain bit strings whose Hamming distances reflect the Euclidean distances among the original responses. This allows a fair evaluation of the Bistable Ring PUF using the metrics defined in Section 2.2. In particular, it is important to compute reliability, uniqueness and diffuseness.

Spectral Hashing application

Spectral Hashing algorithm needs a training phase, from witch a set of coefficients is generated. The training phase takes as input the required number of bits, and some PUF responses. Then, that set of coefficients can be used to generate a string of bits from unknown data.

In this case, 30k out of 200k responses from various devices and challenges have been used to train the algorithm. Then, bit strings are generated from the

remaining ones. The process is repeated with different number of bits of the output string. The resulting metrics are presented in Table 5.1.

Table 5.1. Quality metrics

Number of Bits	Reliability [%]	Diffuseness [%]	Uniqueness [%]
4	97.5	23.0	32
8	97.0	24.8	37.3
16	96.1	34.1	39.3
32	94.9	37.5	40.9

It is clear that a trade-off between reliability and diffuseness/uniqueness must be found. By training the algorithm to produce different number of bits, various trade-off points can be selected.

Chapter 6

Conclusions and future work

Physical Unclonable Functions are promising primitives for security and cryptographic applications. In particular, they can be used for strong multifactor authentication and for cryptographic keys generation. However, designing a really strong PUF revealed to be a challenging task. Modeling attacks arose as the main threat for PUF resiliency. Moreover, many issues emerge during implementation, due to the non-conventionality of this kind of circuits, that makes impossible to describe them in standard hardware description languages. Finally, reliability is also a difficult target, especially if the device is supposed to work in a relatively large environmental conditions range.

In this work, an attempt to design a strong Physical Unclonable Function has been carried out. On the basis of previous work, the Bistable Ring has been selected as the most promising architecture for a strong PUF. The design has been implemented for the **SE***Cube*™ device family. Specifically, the **SE***Cube*™ FPGA has been exploited to implement a Bistable Ring PUF.

Statistical analysis showed comparable results with respect to other designs and implementations that can be found in literature. However, theoretical assumptions seem to grant a better resiliency to modeling attacks.

Finally, one method to extrapolate a response in the form of a string of bits has been explored, even if there are for sure better techniques that are, however, out of the competences of the author of this work.

6.1 Future work

A further analysis of PUF strength is needed. In particular, modeling attack resistance must be corroborated by targeting the PUF with all known machine learning attacks. This is a crucial point, since if any attack exists which is able to predict PUF responses, the designed PUF could not be considered strong (and secure) anymore.

Speaking about reliability, a deep analysis on environmental conditions effects must be done. Specifically, power supply voltage and room and package temperatures are the most common environmental variations a chip is subjected to during its lifetime. A quantitative measure of how much the variation of these parameters influences the PUF must be done. Moreover, aging of the chip may modify its physical parameter, and thus its PUF responses. The possibility of applying burn-in, i.e., quickly aging the device by stressing it with high voltages and temperatures, should be explored. Intuitively, according to the classical bathtub curve, most of the physical parameters modifications may happen in the early phases of the device lifetime. Burn-in may help in the sense that all these modifications are already happened when the device reaches the operational phase.

In this work, the designed PUF has been implemented for only a single device (the Lattice MachXO2-7000 mounted in **SE***Cube*™). It would be great to demonstrate the generality of the design by trying to implement it over other platforms, such as Xilinx and Altera FPGAs.

Finally, in order to demonstrate its functionality, the PUF should be placed in an actual example application. A possibility is trying to implement a reliable authentication protocol or a secure cryptographic key generation mechanism. In both cases, the main problem to solve would be taking care of the unreliability of the PUF, since so far it is evident that a perfect reliability is impossible, even under very stable environmental conditions.

Bibliography

[1] Sergei Skorobogatov. «Semi-invasive attacks-A new approach to hardware security analysis». In: (Jan. 2005).

[2] Oliver Kömmerling and Markus G. Kuhn. «Design Principles for Tamper-Resistant Smartcard Processors». In: *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*. WOST'99. Chicago, Illinois: USENIX Association, 1999, p. 2.

[3] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. «Physical One-Way Functions». In: *Science* 297.5589 (2002), pp. 2026–2030. DOI: 10.1126/science.1074376. eprint: https://www.science.org/doi/pdf/10.1126/science.1074376. URL: https://www.science.org/doi/abs/10.1126/science.1074376.

[4] G. Edward Suh and Srinivas Devadas. «Physical Unclonable Functions for Device Authentication and Secret Key Generation». In: *2007 44th ACM/IEEE Design Automation Conference*. 2007, pp. 9–14.

[5] K. Lofstrom, W.R. Daasch, and D. Taylor. «IC identification circuit using device mismatch». In: *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*. 2000, pp. 372–373. DOI: 10.1109/ISSCC.2000.839821.

[6] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. «Silicon Physical Random Functions». In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. CCS '02. Washington, DC, USA: Association for Computing Machinery, 2002, 148–160. ISBN: 1581136129. DOI: 10.1145/586110.586132. URL: https://doi.org/10.1145/586110.586132.

[7] Roel Maes and Ingrid Verbauwhede. «Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions». In: Oct. 2010, pp. 3–37. ISBN: 978-3-642-14451-6. DOI: 10.1007/978-3-642-14452-3_1.

[8] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. «FPGA Intrinsic PUFs and Their Use for IP Protection». In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 63–80. ISBN: 978-3-540-74735-2.

[9] G. Edward Suh and Srinivas Devadas. «Physical Unclonable Functions for Device Authentication and Secret Key Generation». In: *2007 44th ACM/IEEE Design Automation Conference*. 2007, pp. 9–14.

[10] Abhranil Maiti and Patrick Schaumont. «Improved ring oscillator PUF: An FPGA-friendly secure primitive». In: *J. Cryptology* 24 (Apr. 2011), pp. 375–397. DOI: 10.1007/s00145-010-9088-4.

[11] Yohei Hori, Takahiro Yoshida, Toshihiro Katashita, and Akashi Satoh. «Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs». In: *2010 International Conference on Reconfigurable Computing and FPGAs*. 2010, pp. 298–303. DOI: 10.1109/ReConFig.2010.24.

[12] Abhranil Maiti, Jeff Casarona, Luke McHale, and Patrick Schaumont. «A large scale characterization of RO-PUF». In: *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2010, pp. 94–99. DOI: 10.1109/HST.2010.5513108.

[13] Daihyun Lim, J.W. Lee, B. Gassend, G.E. Suh, M. van Dijk, and S. Devadas. «Extracting secret keys from integrated circuits». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.10 (2005), pp. 1200–1205. DOI: 10.1109/TVLSI.2005.859470.

[14] Jorge Guajardo, Sandeep S. Kumar, Geert-Jan Schrijen, and Pim Tuyls. «FPGA Intrinsic PUFs and Their Use for IP Protection». In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 63–80. ISBN: 978-3-540-74735-2.

[15] Sandeep S. Kumar, Jorge Guajardo, Roel Maes, Geert-Jan Schrijen, and Pim Tuyls. «Extended abstract: The butterfly PUF protecting IP on every FPGA». In: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. 2008, pp. 67–70. DOI: 10.1109/HST.2008.4559053.

[16] Qingqing Chen, György Csaba, Paolo Lugli, Ulf Schlichtmann, and Ulrich Rührmair. «The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions». In: *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. 2011, pp. 134–141. DOI: 10.1109/HST.2011.5955011.

[17] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. «Modeling Attacks on Physical Unclonable Functions». In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. Chicago, Illinois, USA: Association for Computing Machinery, 2010, 237–249. ISBN: 9781450302456. DOI: 10.1145/1866307.1866335. URL: https://doi.org/10.1145/1866307.1866335.

[18] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. «Controlled physical random functions». In: *18th Annual Computer Security Applications Conference, 2002. Proceedings.* 2002, pp. 149–160. DOI: 10.1109/CSAC.2002.1176287.

[19] Yuki Tanaka, Song Bian, Masayuki Hiromoto, and Takashi Sato. «Coin Flipping PUF: A Novel PUF With Improved Resistance Against Machine Learning Attacks». In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 65.5 (2018), pp. 602–606. DOI: 10.1109/TCSII.2018.2821267.

[20] Mehrdad Majzoobi, Farinaz Koushanfar, and Srinivas Devadas. «FPGA PUF using programmable delay lines». In: *2010 IEEE International Workshop on Information Forensics and Security*. 2010, pp. 1–6. DOI: 10.1109/WIFS.2010.5711471.

[21] Roel Maes and Ingrid Verbauwhede. «Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions». In: *Towards Hardware-Intrinsic Security: Foundations and Practice*. Ed. by Ahmad-Reza Sadeghi and David Naccache. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 3–37. ISBN: 978-3-642-14452-3. DOI: 10.1007/978-3-642-14452-3_1. URL: https://doi.org/10.1007/978-3-642-14452-3_1.

[22] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. «Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data». In: *Advances in Cryptology - EUROCRYPT 2004*. Ed. by Christian Cachin and Jan L. Camenisch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540. ISBN: 978-3-540-24676-3.

[23] Yair Weiss, Antonio Torralba, and Rob Fergus. «Spectral Hashing». In: *Advances in Neural Information Processing Systems*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Vol. 21. Curran Associates, Inc., 2008.