Design Flow to Evaluate the Performance of Ring Oscillator PUFs on FPGAs

Macarena C. Martínez-Rodríguez, Eros Camacho-Ruiz, Santiago Sánchez-Solano and Piedad Brox
Instituto de Microelectrónica de Sevilla (CSIC / University of Seville)
Email: {macarena, camacho, santiago, brox}@imse-cnm.csic.es

Abstract

This work presents a unified framework to design, implement and evaluate the performance of Ring Oscillator Physical Unclonable Functions (RO PUFs) on FPGAs. The design flow uses a Digital Signal Processing (DSP) tool integrated into the Matlab environment. The use of this tool eases the evaluation of the PUF performance. The DSP tool provides an environment to apply the challenges to the RO PUF, acquire the responses by using hardware (HW) co-simulation, and compute a set of metrics to quantify the stability, probability and entropy of the PUF response. Additionally, the robustness of the PUF response is proved in the generation of secret keys. The design flow was applied to evaluate the performance of RO PUFs implemented on 17 Basys 3 Artix-7 FPGA Boards.

I. Introduction

PUFs have become of increasing interest in the field of hardware security [1]. A PUF performs a functional operation that maps an input (challenge) to an output (response) [2]. Each input challenge with its corresponding measured response receive the name of Challenge-Response Pair (CRP). The implementation of a PUF must rely on an intrinsic physical feature so that each instance of it creates a unique challenge-response mapping and, thus, cannot be cloned.

There are three essential properties a PUF must meet: uniqueness (i.e., different PUF instances should return different responses when the same challenge is applied), reliability (i.e., a PUF instance should return a response as unchanged as possible when the same challenge is applied), and unpredictablility (there is no way to predict an output PUF response, even the PUF designer can not guess it). These properties make that PUF responses can be used to derive a unique digital identity inherent to the electronic device in which the PUF circuitry is inserted. In this sense, there is a direct analogy with human biometric identifiers (e.g. fingerprints) that are used to digitally identify a person.

The security of electronic systems is built around binary keys. To increase the resilience of these systems, the keys are usually stored in a Non-Volatile Memory (NVM) and protected against invasive attacks with expensive tamper-sensing circuitry [3]. However, this solution is not affordable in certain scenarios where the extra cost is not acceptable, such as Internet of Things (IoT) devices. PUFs have emerged as alternative with some remarkable security advantages, since the PUF response can be used to retrieve on-line cryptographic keys as many times as necessary, avoiding their storage in a memory.

Robustness of the PUFs to avoid unstable bits at the output, i.e. small differences at the PUF output obtained with the same challenge from run to run, is a critical feature in certain applications such as key generation [4]. To mitigate the effects of noisy outputs, post-processing based on Error Correction Codes (ECC) techniques are required [5]. In order to reduce the complexity of ECC techniques, the majority of PUF structures introduce modifications to make it resilient against noise [6].

Silicon PUFs exploit the manufacturing variability of CMOS manufacturing process to implement unclonable functions. A rough classification of silicon PUFs divides them into two categories: memory-based and delay-based PUFs. Among memory-based ones, SRAM PUFs are based on the unpredictable start-up values of cells to obtain a PUF response [7]. Many efforts have been focussed on selection techniques to classify cells into stable or unstable in order to improve the reliability of the PUF response. SRAM PUFs can be implemented using a dedicated memory or re-using the memory available on a chip for further purposes [8]. Unlike SRAM PUFs implemented in ASICs, which have been quite popular in academic and industrial sectors during last years [9], their implementation using the Block RAMs (BRAMs) available on some FPGAs is practically unfeasible since the start-up values are usually forced to a certain value in this type of memory.

Delay-based PUFs are based on the relative time delay differences between two theoretically identical circuits. Some examples are arbiter PUF and Ring Oscillators (RO) PUF. An arbiter PUF is based on the relative delay difference between two paths with the same layout length. It is built using a set of concatenated multiplexers and one arbiter. The challenges (control bits of the multiplexers) determine the delay paths and the PUF output depends on which path is faster [10]. RO PUFs are based on closed delay chains (delay loops) whose oscillation frequencies are compared to obtain the PUF output [11]. These kinds of delay-based PUFs can be implemented both in ASICs and FPGAs. However, the design of arbiter PUFs on FPGAs is complicated since the designer has to ensure the same length of paths and this is difficult to achieve with the default options taken by the synthesis and implementation tools. Although it is possible to place and route circuits manually, it is difficult to ensure the same delay paths due to the structure of the FPGAs. Comparing both delay-based PUFs, RO PUFs usually present a better performance in terms of reliability and entropy than arbiter PUFs [12].

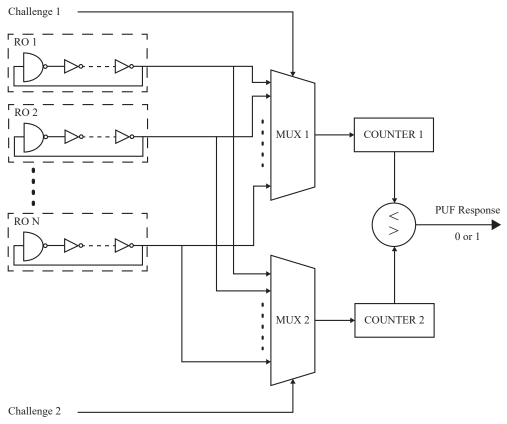


Fig. 1: Conventional RO PUF

This paper presents a design flow to evaluate the performance of RO PUFs implemented on FPGAs. This paper focusses on a RO PUF whose architecture is described in Section II. The proposed design flow, built around a DSP tool for FPGAs, is described in Section III. The use of this tool facilitates the verification of the RO PUF using HW co-simulation, while its full integration with Matlab & Simulink environment accelerates the evaluation of the PUF performance, since both challenges and responses of the PUF can be generated and processed, respectively, using Matlab scripts that can be easily adapted according to the desired strategy. In Section IV, the strategy to select the most appropriate bits of the PUF output is presented. The selection is based on a well-known suite of metrics that evaluate the PUF reliability and entropy. The PUF reliability is further corroborated with the obfuscation of secret keys providing a wide set of experimental results on several FPGA development boards. The experimental results are included in Section V. Finally, the conclusions of this work are presented.

II. Ring Oscillator PUF

The conventional implementation of a RO PUF uses N identical ring oscillators. Each one of them comprises a chain of an odd number of inverters to form a combinational loop. The frequency of oscillation of the Nth RO is roughly calculated by:

$$f_N = \frac{1}{2 \cdot n_{stages} \cdot \tau_{INV}} \tag{1}$$

where n_{stages} is known as the number of stages of the RO and it is equal to the number of inverters, and τ_{INV} is the nominal time delay per inverter. The total delay in the RO loop, which must also take into account the delays in the connecting lines, is subject to variations due to physical effects and noise [13]. Usually, one inverter in the chain is replaced by a NAND gate to enable the oscillation.

The architecture of a RO PUF was originally proposed in [11] (see Figure 1). It consists of N Ring Oscillators named as RO1, RO2, and so on until RON. Each one of these identical ROs oscillate with unique frequency because of the inherent variability of the CMOS manufacturing process. The control signals of two multiplexers that are the inputs called $challenge1$ and $challenge2$ select two ROs which are compared together (pair). The two counters compute the number of oscillations of each of the two ROs in a fixed time interval known as comparison time. After finishing this interval, the outputs of the two counters are compared. Depending on which of the two counters has the highest value, the output bit can be set to 0, if *Counter 1* is higher than *Counter 2*, and can be set to 1, otherwise. Since applications require the generation of bit streams of a certain length, a large number of identical ROs has to be implemented in the RO PUF.

If the frequencies of both RO selected by the challenge are very close to each other, the output of the PUF may be different from one run to another one. This makes difficult to ensure that the RO PUF provides the same output with the same challenge over time. The technique in [11] reduces the number of comparisons to selected RO pairs with the largest differences in terms of frequencies to solve this issue. Other options to increase the reproducibility of the RO PUF output are the use of larger counters or longer comparison intervals [12]. Another reported technique to increase the reliability of the PUF is to order ROs into pairs that group ROs whose frequencies are adequately apart from each other [14]. This strategy increases the robustness of the RO PUF against environmental variations and noise.

A RO PUF can be modified to produce more than one output bit per comparison by increasing the number of multiplexers, counters and comparators. The idea is to compare several pairs of ring oscillators at the same time at expense of an increase of resources. As alternative, the architecture proposed in [15] achieves more bits for the PUF output using same ROs. The approach uses two arrays with the same number of identical ROs as illustrated in the schematic of Fig. 2. A unique challenge controls both multiplexers, which select ROs with identical labels in each one of the RO banks. The oscillations of the selected RO pair are simultaneously computed with the two counters. Unlike the conventional proposal, in this case the time interval is fixed by the overflow of one of the counters. Thus, the measurement is stopped as soon as one of the counters ends and the value of the counter that did not overflow is used for further processing. The analysis of PUF performance in a particular device will determine which bits of the counter are used to generate the PUF output. Usually two or three bits are selected per comparison instead of the unique bit obtained with the conventional RO PUF. The total bit number that is required to the PUF response depends on the application. For the sake of illustration, a key generation of 256-bits will require 128 CRPs if two bits are extracted from one measurement. Thus, the proposal in [15] halves the resources and time to retrieve a key since the conventional approach only provides one bit per comparison.

III. Design flow of RO PUFs using SysGen

FPGA vendors offer different design tools to facilitate the implementation of DSP algorithms in FPGA. Among them, Xilinx provides the System Generator toolbox (SysGen) [16] integrated into the Matlab environment. SysGen is based on the Simulink tool, which allows to model, analyze, and simulate dynamic systems in Matlab. It encompasses a Simulink library that contains basic blocks to build digital systems, and other components to link the design described in SysGen with the synthesis and implementation tools within Vivado.

The use of SysGen eases the simulation, implementation, and also the hardware (HW) co-simulation of a DSP design. The full integration into Matlab & Simulink eases the generation of input/output signals to verify the functionality of the DSP system. Additionally, SysGen enables the access to the synthesis and implementation tools within Vivado. Thus, the FPGA programming file can be obtained in a direct way from Simulink. Moreover, all the different synthesis and implementation strategies included in Vivado are also available in SysGen. And finally, it includes appropiate interfaces to perform HW co-simulation, in which the design implemented on the FPGA development board interacts with the rest of the components modelled using the versatility of Matlab & Simulink.

Traditionally, SysGen has been used for the design and implementation of systems devoted to signal processing and arithmetic operations. However, to the best of our knowledge, SysGen has never been used to verify the performance of PUFs and its use introduces some advantages. Firstly, the HW co-simulation enables that the PUF is working on the FPGA while the inputs (challenges) and the outputs (responses) are, respectively, generated and processed with Matlab. This utility is crucial for the analysis of a silicon PUF since the variations of CMOS manufacturing process are exploited only when the PUF is implemented. An analysis of the PUF behavior based on simulations would be useless. Secondly, a PUF structure should fulfill the properties described in Section I. To ensure this, it is necessary the analysis of a huge number of PUF responses. This task can be facilitated with the use of Matlab scripts.

SysGen has been used along this work to design, implement and evaluate the performance of the RO PUF described in Section II. To model and verify the behaviour of the RO PUF, a component of the SysGen library called blackbox is used. It allows the instantiation of VHDL or Verilog files with the description of a HW design. In this case, a Verilog description of the structure in Fig. 2 is instantiated within the blackbox block. PUF inputs (challenges) are introduced in the model using source blocks provided by Simulink, which select two ROs with the same label from each one of the arrays. Similarly, RO PUF outputs (responses) are captured or visualized by means of conventional Simulink drain blocks. Once verified the design functionality, different versions of the PUF can be built using different synthesis and implementation strategies in the Vivado tools. The implemented design is then included in the Simulink model and HW co-simulation is used in order to verify the RO PUF performance. Figure 3 illustrates the different stages of this design flow, while the procedure used to conform and evaluate the RO PUF is detailed in Section IV.

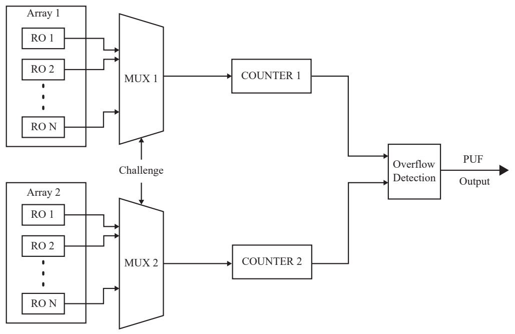


Fig. 2: RO PUF proposed in [14]

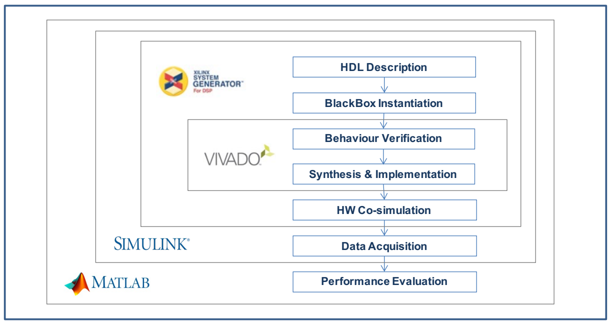


Fig. 3: Design flow of RO PUFs using SysGen

IV. Performance evaluation of RO PUFs

The evaluation of any PUF consists of corroborating if the PUF response fulfills the properties (uniqueness, reliability, unpredictability) described in Section I. The RO PUF selected for this example requires a previous analysis to decide which bits of the output counter are used as the PUF output. This first stage of the evaluation process is described in subsection IV-A. In a second evaluation stage, the PUF properties are evaluated for the complete PUF response (resulting from the concatenation of the selected bits of the different responses after applying different challenges), as detailed in subsection IV-B. Finally, the RO PUF performance is evaluated when it is used for the obfuscation and recovery of a secret key in subsection IV-C.

A. Challenge-response RO PUF bit performance

For each of the bits provided by the overflow detection block, the reliability is measured by the average bit stability, the more stable, the more reliable. The unpredictability is measured by the average bit probability, its ideal value being 50%. Entropy will determine the uniqueness (ideally both values, Hintra and Hinter, must be one). To calculate all those metrics, a significant amount of responses for all the possible input challenges must be acquired using different devices over time. For this, a set of Matlab scripts was developed in order to facilitate the configuration and execution of HW co-simulations by repeatedly applying a sequence of challenges and capturing their responses for different PUF versions on multiple identical FPGA development boards. Matlab functions were also developed to calculate, from the captured responses, the average bit stability and probability and the entropy values (Hinter, Hintra) following the procedure described in [15].

The conclusions extracted from this first evaluation stage will determine which bits are selected to generate the RO PUF response.

B. RO PUF performance

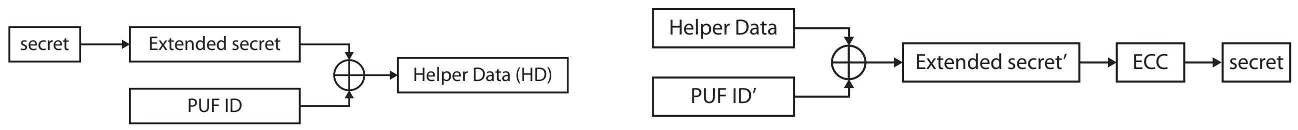
In this second stage, the goal is to evaluate the performance of the bitstream conformed by the concatenation of the selected bit of the RO PUF responses to the applied set of challenges. On the one hand, the intra-Hamming distance (HDintra) determines the similarity between responses. Its ideal value is 0, which means that the RO PUF response is always the same when the same sequence of challenges is applied to the same device. On the other hand, the inter-Hamming distance (HDinter) determines the uniqueness of the responses generated among different devices, being its ideal value 50%. Matlab functions were also used to calculate HDintra and HDinter.

C. Evaluation of the use of the RO PUF to obfuscate and recover a secret key

The procedure for obfuscating and recovering a secret key requires the use of an ECC capable of mitigating the differences that RO PUF bitstreams can present in successive invocations. The RO PUF bitstream resulting after concatenating the selected bit responses is identified as PUF ID in Fig. 4. When used to obfuscate the secret key, a Helper Data (HD) structure is generated using the schema shown in Fig. 4a. HD will be needed to recover the secret key, but a counterfeit device other than the owner of the secret cannot recuperate the secret key, even having available the HD. The secret key can be recovered by the PUF owner from a new (and slightly different) RO PUF bitstream (PUF ID') and using the ECC, according to the schema shown in Fig. 4b.

Once again, Matlab scripts were used to automate the obfuscation phase of a secret key using a repetition ECC with codeword length r. First, the key is extended by repeating each bit value r times. Then, the RO PUF response is generated by HW co-simulation using as many challenges as needed to compound the PUF ID. Finally, an XOR operation is carried out among the extended secret and the PUF ID generating the HD as shown in Fig. 4a.

Recovery of the secret at a later point in time requires obtaining a new PUF ID that may slightly differ from the one used to generate the HD. This new PUF ID is XOR-ed with the HD obtaining a new extended secret key. The new extended key is decoded by using a majority vote algorithm. The extended secret is divided into pieces of r bits. Each bit of the secret is recovered by each piece, and its value is determined for the one with the larger occurrence. Fig. 4b illustrates the procedure.



(a) obfuscation (b) recovery

Fig. 4: Secret obfuscation and recovery using PUF response

A script was developed by which, first, the PUF is used to obfuscate a secret key and generate the corresponding HD. Then, an attempt is made repeatedly to recover the key both on the same and on different devices. False negative rates (FNR) and false positive rates (FPR) are finally calculated. Ideal values are 0% for FNR, which means that the secret owner can always recover it, and 0% for FPR, which means that any other (counterfeit) device can never recover others' keys using its corresponding HD.

V. Experimental results

We have evaluated the RO PUF performance in several Basys 3 Artix-7 FPGA Boards for a sake of demonstration. The design and evaluation of the RO PUF was carried out using Matlab R2018a and the version of Systen Generator included in Xilinx Vivado 2018.2.

RO PUF design in Fig. 2 was described in Verilog. It includes two banks of 150 ROs each. Each RO has 5 stages composed by 4 inverters and 1 NAND gate. The size of the output counters is 16 bits. Due to the asynchronous nature of ROs, two directives must be included in the HDL description:

- (*ALLOW_COMBINATORIAL_LOOPS = "true", KEEP = "true" *). It is used to avoid the error "[DRC LUTLP-1] Combinatorial Loop Alert" during the bitstream generation.
- (DONT_TOUCH = "yes") wire <wire_name >; It is used to avoid that some wires were suppressed during the elaboration phase. This declaration must be added for each wire between inverters.

A blackbox component instantiates the Verilog files in the Simulink model. Concretely, the RO PUF performance has been evaluated using 17 different boards, and 1000 RO PUF responses have been captured for each challenge using HW co-simulation.

The first of the tests carried out compares different variants of the PUF obtained by using different strategies in the synthesis and implementation stages. Table I shows the possible different synthesis and implementation strategies. For one of the boards, 1000 responses are captured for all the possible challenges and for different combinations of synthesis and implementation strategies. The mean and the standard deviation of the responses is analyzed for all the combinations of strategies. Means values generate a fingerprint of the PUF for each combination of strategies, as shown in Fig. 5. After analyzing the fingerprints, we

TABLE I: Synthesis and implementation strategies

Code	Synthesis strategy	Implementation strategy
A	Default	Default
B	Flow_AreaOptimized_high	Area_Explore
C	Flow_AlternateRoutability	Performance_ExplorePostRoutePhysOpt
D	Flow_PerfOptimized_high	Performance Explore
E	Flow_RuntimeOptimized	Flow_RuntimeOptimized
G		Congestion_SpreadLogic_high

TABLE II: Average stability and probability, Hintra and Hinter

	Average stability					**Average probability**					**Hintra**					**Hinter**				
	AA	**BB**	**EE**	**CG**	**All**	**AA**	**BB**	**EE**	**CG**	**All**	**AA**	**BB**	**EE**	**CG**	**All**	**AA**	**BB**	**EE**	**CG**	**All**
1	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
2	1,000	0,999	0,999	1,000	1,000	0,761	0,760	0,758	0,753	0,757	0,7208	0,8300	0,7454	0,8756	0,7930	0,0508	0,0404	0,0266	0,0528	0,5862
3	0,999	0,999	0,999	0,999	0,999	0,662	0,652	0,658	0,658	0,657	0,8794	0,9314	0,9487	0,9388	0,9246	0,0871	0,0675	0,0632	0,0800	0,7375
4	0,999	0,999	0,998	0,998	0,998	0,567	0,574	0,574	0,584	0,574	0,9984	0,9807	0,9714	0,9678	0,9796	0,1975	0,1546	0,1482	0,1858	0,7828
5	0,997	0,997	0,996	0,996	0,997	0,529	0,537	0,542	0,538	0,536	0,9987	0,9951	0,9853	0,9941	0,9933	0,3854	0,4262	0,3817	0,4254	0,8902
6	**0,993**	**0,993**	**0,993**	**0,993**	**0,993**	**0,567**	**0,578**	**0,570**	**0,561**	**0,568**	**0,9886**	**0,9876**	**0,9845**	**0,9628**	**0,9809**	**0,7324**	**0,8062**	**0,7571**	**0,7332**	**0,9374**
7	**0,985**	**0,986**	**0,984**	**0,986**	**0,985**	**0,521**	**0,512**	**0,518**	**0,523**	**0,517**	**0,9937**	**0,9949**	**0,9982**	**0,9904**	**0,9943**	**0,9496**	**0,9471**	**0,9417**	**0,9280**	**0,9825**
8	**0,967**	**0,970**	**0,968**	**0,967**	**0,968**	**0,510**	**0,508**	**0,518**	**0,513**	**0,511**	**0,9975**	**0,9934**	**0,9954**	**0,9935**	**0,9950**	**0,9553**	**0,9525**	**0,9576**	**0,9528**	**0,9888**
9	0,934	0,941	0,936	0,937	0,937	0,509	0,493	0,520	0,501	0,505	0,9951	0,9968	0,9940	0,9938	0,9949	0,9550	0,9570	0,9515	0,9496	0,9891
10	0,869	0,876	0,871	0,871	0,872	0,499	0,491	0,493	0,504	0,496	0,9955	0,9935	0,9942	0,9981	0,9953	0,9540	0,9512	0,9603	0,9557	0,9898
11	0,739	0,741	0,741	0,742	0,741	0,504	0,506	0,507	0,494	0,502	0,9952	0,9959	0,9967	0,9961	0,9960	0,9580	0,9640	0,9530	0,9539	0,9884
12	0,562	0,561	0,563	0,563	0,562	0,500	0,502	0,503	0,500	0,501	0,9973	0,9958	0,9950	0,9958	0,9960	0,9530	0,9618	0,9542	0,9516	0,9897
13	0,513	0,513	0,513	0,513	0,513	0,500	0,500	0,501	0,500	0,500	0,9933	0,9982	0,9939	0,9942	0,9949	0,9601	0,9587	0,9510	0,9654	0,9882
14	0,513	0,513	0,513	0,513	0,513	0,499	0,500	0,501	0,500	0,500	0,9966	0,9930	0,9945	0,9968	0,9952	0,9532	0,9621	0,9515	0,9585	0,9889
15	0,513	0,513	0,513	0,513	0,513	0,500	0,500	0,501	0,500	0,500	0,9959	0,9948	0,9924	0,9939	0,9942	0,9662	0,9608	0,9535	0,9535	0,9886
16	0,513	0,513	0,513	0,513	0,513	0,500	0,500	0,500	0,499	0,500	0,9947	0,9976	0,9913	0,9966	0,9950	0,9603	0,9551	0,9544	0,9548	0,9908

TABLE III: HDinter, HDintra results for the PUF using bits 6, 7, and 8, using bits 6 and 7, and using bits 7 and 8

	PUF 6 - 8				PUF 6 - 7				PUF 7 - 8			
	HDInter	**HDIntra**	**HDIntra_min**	**HDIntra_max**	**HDInter**	**HDIntra**	**HDIntra_min**	**HDIntra_max**	**HDInter**	**HDIntra**	**HDIntra_min**	**HDIntra_max**
AA	45,1209	1,8522	0,8873	2,6367	**42,7500**	**1,0979**	**0,4580**	**1,5963**	49,7255	2,4211	1,1880	3,3220
BB	46,4804	1,7317	1,0176	2,2249	**44,8235**	**1,0926**	**0,6447**	**1,7967**	49,5882	2,2157	1,3583	2,9950
EE	45,5980	1,6682	1,0816	2,1707	**43,3578**	**0,9951**	**0,5473**	**1,5030**	49,5588	2,1963	1,4753	2,9507
CG	44,8268	1,8614	1,2282	2,9862	**42,3529**	**1,1313**	**0,4113**	**2,3470**	48,9559	2,4438	1,7233	3,6150
All	48,6667	1,7784	0,8873	2,9862	**48,0187**	**1,0792**	**0,4113**	**2,3470**	49,7464	2,3192	1,1880	3,6150

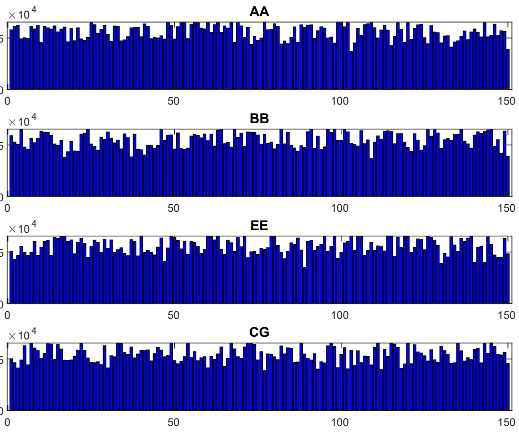


Fig. 5: RO PUF fingerprint: Mean values of the responses of the 150 challenges for all the strategies

conclude that only four combinations generate different fingerprints, and therefore we only calculate the RO PUF bit performance and the complete RO PUF performance for those that are unique. Therefore, the rest of the tests have been only evaluated for the following strategies: AA, BB, EE and CG, where the first letter indicate the synthesis strategy and the second letter indicates the implementation strategy provided by Table I.

Furthermore, after the previous analysis, we can conclude that if the RO PUF responses using AA, BB, EE, and CG strategies are different, we can consider them as responses of different boards, and therefore the metrics can be extracted for a larger amount of data, i.e. 17 boards $\times 4$ strategies $= 68$ different RO PUFs.

A. Challenge-response RO PUF bit performance

Once selected the strategies, we must select the bits of the RO PUF response that satisfies the properties of a PUF. For that purpose, test scripts for HW co-simulation were run to acquire responses using all the strategies (AA, BB, EE and CG), and all the 17 different boards.

Table II shows the obtained values of average stability and probability, Hintra and Hinter per bit.

For all the strategies, we can conclude that the average stability is over the 95% for the bits from 1 to 8. If we evaluate the Hintra, we conclude that bits from 6 to 16 show it over the 75% for each strategy, and if we consider all the 68 RO PUFs we obtain a Hinter over 93%. Therefore, we could consider that the response of bits 6, 7, and 8 are candidate for the PUF. The average probability is around 50% and Hinter is above the 95% for these bits, therefore, bits 6, 7, and 8 satisfy all the properties of a PUF. Among the different strategies, BB is the one that provides better results of these three bits.

B. RO PUF performance

Using the same data acquired in the previous section, the RO PUF performance was also evaluated. The complete RO PUF response was composed in three different ways.

- RO PUF 6-8: the concatenation of bits 6, 7, and 8 of all the responses using all the possible challenges. The size of this RO PUF response is $3 \times 150 = 450$ bits.
- RO PUF 6-7: the concatenation of bits 6 and 7 of all the responses using all the possible challenges. Its size is $2\times150 = 300$ bits.
- RO PUF 7-8: the concatenation of bits 7 and 8 of all the responses using all the possible challenges. Its size is $2\times150 = 300$ bits.

The results of HDinter, and mean, max and min values of HDintra are shown in Table III. PUF 6-7 obtain the best results in terms of HDintra, with an acceptable HDinter around 50%. BB strategy presents the best trade-off among HDinter and HDintra.

TABLE IV: FNR and FPR after recovering a 32-secret key using PUF 6-7 responses and different-length repetition codes

	AA		BB		EE		CG	
r	**FNR**	**FPR**	**FNR**	**FPR**	**FNR**	**FPR**	**FNR**	**FPR**
3	0,641	0,000	0,000	0,000	0,392	0,000	0,711	0,000
5	0,000	0,000	0,034	0,000	0,230	0,000	0,216	0,000
7	0,000	0,000	0,001	0,000	0,006	0,000	0,216	0,000
9	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

TABLE V: FNR and FPR after recovering a 32-bit secret key using r=9 for all boards and strategies

	AA	**BB**	**EE**	**CG**	**all**
FNR	0,0001	0,000	0,000	0,0144	-
FPR	0,000	0,000	0,000	0,000	0,000

C. Evaluation of the use of the RO PUF to obfuscate and recover a secret key

Finally, we have evaluated the performance of the application test for the obfuscation and recovery of a secret key. Due to the limitation of the bit number of the PUF output (300 bits), we have calculated the FNR and FPR when trying to recover a secret key of 32 bits using different values of the repetition code, $r = \{3, 5, 7, 9\}$. The PUF ID is extracted for the RO PUF 6-7. Table IV shows the FNR and FPR after recoveries the key 1000 times in the device that obfuscated it and in the rest of the 16 (counterfeit) devices. Results show that the secret key was always recovered using r=9 for all strategies and no counterfeit device could recover the secret key.

For $r = 9$, the previous test was extended for all the devices (17 key obfuscations and 17 recovering for each obfuscation, for each strategy), and also considering each strategy as a different device. Therefore, the key was obfuscated 64 times and recovered 1000 times for each obfuscation at each of the 64 different considered devices. Table V shows the obtained FNR and FPR, where the maximum FNR is 1.44%, being the FPR always 0.

VI. Conclusions

This work presents a design flow to evaluate the performance of RO PUFs on FPGAs. It is based on a DSP tool for FPGAs that provides HW co-simulation. This utility is crucial, since variations of CMOS manufacturing become apparent when PUFs are implemented. Several scripts and functions were developed to evaluate the properties of the PUFs. After extensive experimental results, 2-bits per each CRP can be used to generate the PUF response obtaining a good performance in terms of entropy, probability and stability. The reliability of the PUF in the obfuscation and recovery of a secret key is corroborated with experimental results. No counterfeit device is able to retrieve a secret in any of the studied scenarios.

Acknowledgment

This work has received funding by the SPIRS Project with Grant Agreement No. 952622 under the EU H2020 research and innovation programme, Grant PID2020-116664RB-I00 funded by MCIN/AEI/10.13039/501100011033, and LINKA20216 from CSIC. M.C.M-R holds a Post-doc grant supported by the Andalusian government under EU PO. FSE.

References

[1] C. Herder, M-D. Yu, F. Koushanfar and S. Devadas,"Physical Unclonable Functions and Applications: A Tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126 - 1141, 2014.
[2] R. Pappu, B. Recht, J. Taylor and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026 - 2030, 2002.
[3] S. P. Skorobogatov, "Semi-Invasive Attacks - A New Approach to Hardware Security Analysis," in *PhD. Dissertation*, University of Cambridge, 2005.
[4] P. Koeberl, J. Li, A. Rajan and W. Wu, "Entropy Loss in PUF-based Key Generation Schemes: The Repetition Code Pitfall," *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014.
[5] M. Hiller, L. Kurzinger and G. Sigl, "Review of error correction for PUFs and evaluation on state-of-the-art FPGAs," *Journal of Cryptographic Engineering*, vol. 10, pp. 229 - 247, 2020.
[6] Y. Gao, S. F. Al-Sarawi and D. Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, pp. 81 - 91, 2020.
[7] D. E. Holcomb, W. P. Burleson and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proceedings of the Conference on RFID Security*, 2007.
[8] M. C. Martínez-Rodríguez, M. A. Prada-Delgado, P. Brox and I. Baturone, "VLSI Design of Trusted Virtual Sensors," *Sensors*, vol. 18, no. 2, article 347, 2018.
[9] SRAM PUF: The Secure Silicon Fingerprint, "https://www.intrinsic-id.com/resources/white-papers/"
[10] J.W. Lee, D. Lim, B. Gassend, G.E. Suh, M. van Dijk and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *Proceedings of Symposium on VLSI Circuits. Digest of Technical Papers*, 2004.
[11] G. Edward Suh and S. Devadas. "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of Design Automation Conference*, pp. 9 - 14, 2007.
[12] A. Maiti and P. Schaumont, "Improved ring oscillator PUF: An FPGA-friendly secure primitive," *Journal of Cryptology*, vol. 24, no. 2, pp. 375 - 397, 2011.
[13] G. Komurcu, A. E. Pusane and G. Dundar, "Analysis of Ring Oscillator structures to develop a design methodology for RO-PUF circuits," in *Proceedings IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, 2013.
[14] G. Komurcu, A. E. Pusane and G. Dundar, "Enhanced challenge-response set and secure usage scenarios for ordering based RO-PUFs," *Devices, and Systems (IET-CDS)*, vol. 9, no. 2, pp. 87 - 95, 2014.
[15] F. Kodytek and R. Lorencz, "A design of ring oscillator based PUF on FPGA," in *Proceedings IEEE 18th International Symposium on Design and Diagnostics of Electronics Circuits and Systems*, pp. 37 - 42, 2015.
[16] "UG640 - System Generator for DSP User Guide," ver. 14.3, Xilinx, Inc., 2012.