Effects of Runtime Reconfiguration on PUFs Implemented as FPGA-Based Accelerators

Hassan Nassar, Lars Bauer, and Jörg Henkel, *Fellow, IEEE*

***Abstract*—Physical unclonable functions (PUFs) are a handy security primitive for resource-constrained devices. They offer an alternative to the resource-intensive classical hash algorithms. Using the IC differences resulting from the fabrication process, PUFs give device-specific outputs (responses) when given the same inputs (challenges). Hence, without using a device-specific key, PUFs can generate device-specific responses. FPGAs are one of the platforms that are heavily studied as a candidate for PUF implementation. The idea is that a PUF that is designed as an HDL code can be used as part of the static design or as a dynamic accelerator. Previous works studied PUF implementation as part of the static design. In contrast to the state-of-the-art, this letter studies PUFs when used as runtime reconfigurable accelerators. In this letter, we find that not all regions of an FPGA are equally suitable for implementing different PUF types. Regions, where clock routing resources exist, are the worst suited for PUF implementation. Moreover, we find out that for certain PUF types, the property of dynamic partial reconfiguration can lead to performance degradation if not applied carefully. When static routing passing through the region increases, the PUF performance degrades significantly.**

***Index Terms*—Hardware security, reconfigurable hardware, security primitives.**

I. Introduction

FPGAS are an established technology used in academia and industry that allows using custom accelerators based on user specifications. Dynamic partial reconfiguration (DPR) is a very important feature of FPGAs. While the system is running, the accelerators can be changed and loaded to partially reconfigurable regions (PRRs). Each PRR is constrained to a developer-defined area on the FPGA and has a static interface to the rest of the FPGA [1]. When DPR is used, the same PRR can be used to load different accelerators with one of the accelerators used as a *primary* design, and all the others are used as *secondary* designs. The main difference between primary and secondary designs is that the static routing crossing the PRR is optimized for the primary one, while the secondary has to fit into the preoptimized PRR.

DPR applications benefit standalone custom accelerators [2] as well ass accelerators incorporated into processors [3]. The accelerators are loaded to PRRs only when needed instead of implementing them statically and risking that they are potentially idle most of the time (depending on the workload, etc.). This allows designers to have a wide variety of accelerators in a constrained area by dynamically swapping them at runtime.

Physical unclonable functions (PUFs) are security functions that depend on the differences in the fabrication process. That means, despite using the same circuit, their behavior differs when implemented on different ICs. They are used in a challenge-response protocol. When given a certain challenge, a PUF gives an IC-specific response. Each challenge and its corresponding response is called a challenge-response-pair (CRP) [4], [5], [6], [7], [8], [9].

PUFs are usually idle as they are only used when the device ID has to be verified [10]. Hence, they can be implemented as runtime reconfigurable accelerators on FPGAs. This is important for resource constraint devices where the reconfigurable fabric could be tiny and switching the accelerator at runtime would allow for using the freed-up PRR to accelerate another application. For example, for attestation of FPGAs, PUFs, and DPR can be used as explained by SACHa [11]. In such a case, the PUF can be loaded to a PRR to confirm its identity (which PRR on which FPGA). Once the identity is confirmed, the accelerator, usually containing sensitive IP, can be loaded to the authenticated PRR on the authenticated FPGA. Another example is secure boot [12] where PUFs are used within the security protocol of the startup and then are idle for the actual runtime, DPR would help to benefit from the PRR for other accelerators at runtime. Finally, when used for attestation in general [10], the PUF design can be loaded to the PRR in parallel to executing the software calculation of the hash digest. By the time the hash digest is calculated, the PUF would be ready to use and calculate the hardware hash-like response to the challenge. After such calculation, the PRR can load other accelerators which would not have been possible without DPR.

Previous works studied the implementation of PUFs on FPGAs such as [4], [5], [6], [7], [8], and [9]. However, no previous work primarily focused on studying in-depth the effects of DPR on the performance of PUFs. In this letter, we evaluate the feasibility of using PUFs as runtime reconfigurable accelerators. We implement PUFs over all regions on FPGA and characterize their performance metrics. Moreover, we study if DPR would degrade the PUF performance. In comparison to the state of the art, our solution offers the following novel contributions.

1) We study the effect of clock routing on the reliability of PUFs.
2) We are the first to study in depth the effects of DPR on PUF performance.
3) We are the first to show that static routing can significantly affect PUF reliability when used as a runtime reconfigurable accelerator.

Manuscript received 11 July 2023; accepted 22 July 2023. This work was supported in part by the "Helmholtz Pilot Program for Core Informatics (kikit)" at Karlsruhe Institute of Technology. This article appears as part of the ESWEEK-ESL special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis, (CODES+ISSS), 2023. This manuscript was recommended for publication by P. R. Panda. *(Corresponding author: Hassan Nassar.)*

The authors are with the Chair for Embedded Systems, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: hassan.nassar@kit.edu; lars.bauer@kit.edu; henkel@kit.edu).

Digital Object Identifier 10.1109/LES.2023.3299214

The remainder of this letter is structured as follows: Section II gives the needed background. Section III shows the framework we use to perform our characterization. Section IV shows our results. Finally, we conclude this letter in Section V.

II. Background

A. FPGA Internal Structure

Xilinx FPGAs are widely used. The basic logic cell of Xilinx FPGAs is a configurable logic block (CLB). Every CLB consists of two SLICEs, each containing four look up tables (LUTs) with six inputs. The two slices are stacked on each other, one being top, the other being bottom. The slices of the same CLB and LUTs of the same slice have no direct connections. But rather they connect to each other and to the outside of the CLB through switching boxes [13].

There are two types of slices: 1) SLICEL (logic slice) and 2) SLICEM (memory slice). SLICEM can implement combinatorial logic like SLICEL, in addition to distributed RAM, or shift registers which SLICEL does not support. Each CLB consists of one SLICEL and one SLICEM or two SLICELs.

Slices are arranged in columns and each column consists of the same type of slices, i.e., either SLICEM or SLICEL. Moreover, the two slices of the same CLB do not belong to the same column but to two adjacent columns.

B. Physical Unclonable Functions

PUFs can be classified into weak and strong categories. Weak PUFs have a linear number of CRPs relative to the challenge bitwidth, while strong PUFs have an exponential number of CRPs. Three key metrics used to evaluate PUFs are reliability, uniformity, and uniqueness. Reliability aims for a 100% match between responses obtained under different conditions and a precollected golden response. Uniformity measures the distribution of "1" bit in the response, ideally aiming for 50% to avoid biases. Uniqueness assesses the similarity of PUF responses on different integrated circuits (ICs). Ideally, the outputs of different PUFs should appear random when compared, resulting in a uniqueness metric of 50% [4], [5], [6], [7], [8], [9].

PUFs are known to be broken by modeling attacks. In order to mitigate the attacks, the same PUFs can be used as a building block for a larger ML-resilient PUFs as shown in [6]. Our goal in this letter is to show how the basic main quality metrics for PUFs get affected by the use of runtime reconfiguration. The degradation of these metrics, especially the reliability would make attacks easier. Hence, the first step for building ML-resilient PUFs is to ensure that all the main metrics are in the acceptable range.

III. Characterization Framework

To perform our characterization, we implement a framework consisting of a static design and PRRs. The PRRs are used to dynamically load the PUFs at runtime and are distributed over the whole chip area. Moreover, each PUF is implemented once as the primary design and another time as the secondary design. This ensures that 1) any effect of DPR is discovered and 2) any anomalies in the chip or deviation between the different areas are discovered. This is of high importance as knowing these effects upfront helps in making design decisions as we discuss in Section IV-D.

As for the static design, it has a PUF control unit, RAM, and UART. The PUF control unit is used to feed the PUFs with challenges, enable or disable them, and collect the responses based on the challenges. The collected responses are stored in RAM until they are forwarded to a PC using UART. All these components are not device specific. Hence, our framework can be easily adapted to any FPGA.

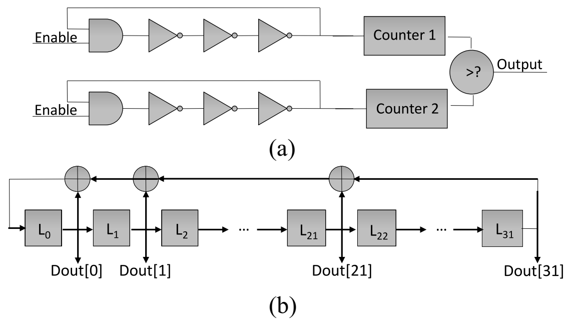

Fig. 1. PUFs studied in this letter (a) weak ROPUF and (b) strong PLPUF.

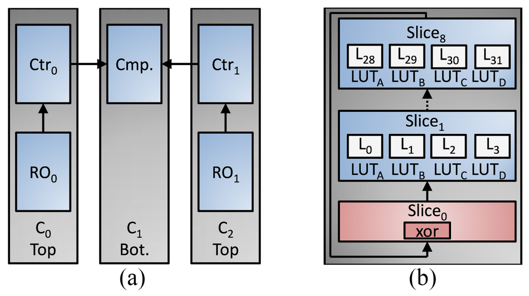

Fig. 2. Placement of PUFs through constraints for (a) ROPUF and (b) PLPUF.

Fig. 1 shows the PUFs studied in this letter. We focus on one strong PUF and one weak PUF. We choose two PUFs that are easy to implement on FPGAs with minimal overhead. For the weak PUF, we choose the ring oscillator PUF (ROPUF) [7]. To produce one PUF bit using ROPUF on FPGA, two ring oscillators are implemented using LUTs. Based on which of the two has a higher frequency, the bit is either "0" or "1." For the strong PUF we use the pseudo linear feedback shift register PUF (PLPUF) [5]. It is implemented on FPGAs with the same structure as a linear feedback shift register (LFSR) but using combinatorial logic instead of sequential (hence the pseudo part of the name).

The VHDL codes for designing both ROPUF and PLPUF are simple. The main effort for designing PUFs with good quality metrics lies in the placement constraints. Thus, ensuring that the performance is mainly influenced by the manufacturing deviations and not by the place and route done by the synthesis tool.

ROPUF is implemented using three columns as shown in Fig. 2(a). Two top columns (C_0 and C_2) are used to implement the ROs and their respective counters. The middle bottom column (C_1) has only the comparator. Each RO is built using 4 LUTs and hence fits within one slice. This placement ensures that the frequency of each RO is only governed by the manufacturing deviations. If we use a mix of top and bottom columns to implement the ROs the routing to the switch box would be different. In general, top columns have longer paths and hence will have lower frequency when used as ROs. Moreover, we only include SLICEL type and ignore SLICEM type, as each of them has a different internal structure, which leads to differences in the achieved frequency.

PLPUF is implemented on a single slice column as shown in Fig. 2(b). It consists of nine slices in a chain. $Slice_0$

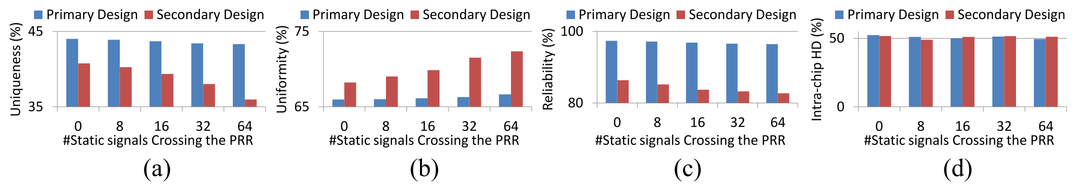

Fig. 3. Performance of PLPUF degrades significantly when used as a secondary design, especially with the increase of static routing crossing the PRR. (a) Uniqueness ideally 50%. (b) Uniformity ideally 50%. (c) Reliability ideally 100%. (d) Intra-chip HD ideally 50%.

implements the XOR function while the rest of the slices implement the pseudo shift register. Slice$_1$ till Slice$_8$ implement the 32 stages L_i of the LFSR in increasing order. Slice$_1$ implements L_0 till L_3 and it continues till Slice$_8$, which implements L_{28} till L_{31}. Within each slice, the L_i with the smallest i is assigned to LUT$_A$. $L_{i+1}, \ldots, L_{i+3}$ are then assigned to LUT$_B$, $\ldots$, LUT$_D$. This highly regular chain structure is used to normalize the delays between the LUTs. If the LUTs would be randomly placed, the delay will not be governed by the process variations but rather by the routing between the LUTs. Hence, it would significantly degrade the uniqueness. Moreover, the flip-flops to store the results are placed in the same column to keep the delay in the PUF at the same level.

IV. Evaluation

The designed framework is then uploaded to four different VC707 boards containing Xilinx Virtex-7 FPGAs. The floorplan of the FPGA is partitioned into 14 PRR areas, arranged as a 2-D grid X_iY_j, $i \in [0,1]$, $j \in [0,6]$. The PUFs are implemented as primary and secondary designs and are loaded into the different PRR areas, and the results are collected by the PC to calculate the three PUF metrics. We collect 256 CRPs from each PUF implemented on each PRR and we repeat the CRP generation process 100 times to evaluate uniformity, uniqueness, intrachip Hamming distance, and reliability.

A. Performance of Strong PLPUF

For PLPUF, the different PRR areas on the FPGA did not affect the PUF performance. The performance using the different PRRs over the different FPGAs was always near the ideal values for uniqueness, uniformity, and reliability. However, we noticed significantly worse reliability and slightly worse uniqueness and uniformity when PLPUF is used as a secondary design instead of a primary design.

We investigated this further, by making each PRR include static routing from the neighboring PRRs. We increase static routing by making a communication channel between two PRRs that are separated by another PRR. For example, PRR X_0Y_0 communicates with PRR X_0Y_2 which PRR X_0Y_1 separates them. The communication would probably be routed at least partially through the PRR separating them. We run several trials where the communication channel bitwidth changes from 0, i.e., no communication is included till 64 bits of data being transmitted.

Fig. 3 shows the results. For both primary and secondary designs when the signals crossing the PRR increase the PUF performance degrades. However, for the primary design, the degradation of all cases stays within the limit of 1%. For the secondary design, the performance is always inferior to their primary design counterpart. Additionally, the performance from one case to another degrades significantly up to 5% which is more than the 1% limit of the primary one. Only the intrachip Hamming distance does not show any degradation. However, this is misleading as with the increase of the static routing, the response is governed by the routing and not the manufacturing differences. Each region has its distinct routing leading to each PUF having distinct responses. Hence, the intrachip Hamming distance stays the same.

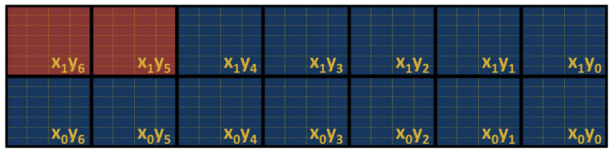

Fig. 4. Plot of the Floorplan for the Virtex-7 FPGA from VC707; highlighted in red is the area where placing ROPUF results in the worst performance.

B. Performance of Weak ROPUF

As for ROPUF, there was no difference between primary and secondary design regarding the behavior. However, different areas of the FPGA resulted in different performances. For the two regions, X_1Y_5 and X_1Y_6 of the FPGA (highlighted in red in Fig. 4) the performance of the PUF extremely degrades on both FPGAs used to test the PUFs. We note that these two regions are the regions where all the BUFG primitives are located. BUFG is a high fanout buffer used to drive the clock. We investigate further the effect of the increase in usage of BUFG on the PUF performance.

Fig. 5 shows the results of the performance metrics, once for the X_1Y_5/X_1Y_6 regions as both had similar bad performance and once for the average case of all the other regions. We have five different runs with different BUFG utilization from 12.5% to 62.5%. For the two regions where the BUFG resources are located, the uniqueness and uniformity are extremely bad. Moreover, they degrade with the increase of the BUFG utilization which shows that the counters get heavily influenced by the noise from the BUFG primitives. The reliability is very high, however, this is misleading. The high value comes from the fact that most of the bits are stuck at zero not that the output is random and stable. For the other regions, the change in the value of the three metrics fluctuates in a very small range which cannot be seen of any statistical relevance.

C. Comparison to Related Work

Related works focusing on characterizing the implementation of PUFs on FPGAs exist; Table I shows the comparison with them. All are limited to studying only one type either weak PUFs or strong PUFs. Moreover, they do not study the effect of the clock routing, i.e., usage of BUFG on the PUF performance. Additionally, [4] does not use DPR, and while [7] uses DPR, they do not study the effect of using PUFs as a secondary design on the performance of the PUF.

In general, each of the related works targets something different. Sahoo et al. [4] tried to find the best implementation method for Arbiter PUF on FPGA, Herkle et al. [7] tried to find the best slice type for RO-PUF implementation, Herkle et al. [8] tried to find the best method to gather

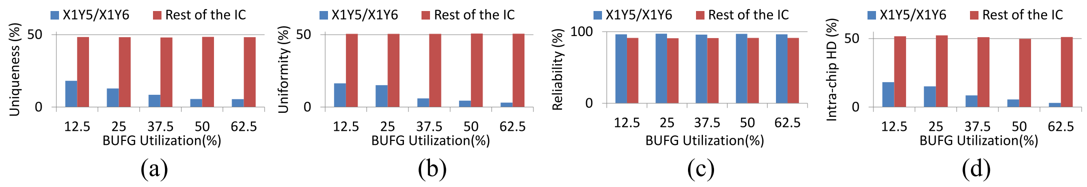

Fig. 5. ROPUF sensitivity to area on chip: areas X_1Y_5/X_1Y_6 suffer from degradation with the increase of the usage of the clock BUFG primitive. (a) Uniqueness ideally 50%. (b) Uniformity ideally 50%. (c) Reliability ideally 100%. (d) Intrachip HD ideally 50%.

TABLE I
Comparison to Related Work

	Weak PUFs	Strong PUFs	Using DPR	Clock routing effects	Secondary design
Our work	✓	✓	✓	✓	✓
Ref. [4]	✗	✓	✗	✗	✗
Ref. [7]	✓	✗	✓	✗	✗
Ref. [8]	✓	✗	✗	✗	✗
Ref. [9]	✓	✗	✗	✗	✗

the CRPs with minimal overhead from FPGAs, finally, Hesselbarth et al. [9] studied the temperature effects on PUFs implemented on FPGAs. Our work complements all of them by studying both the effect of DPR and the effect of clock routing.

D. Discussion

Both ROPUF and PLPUF suffered from performance degradation. However, both PUFs were affected by different reasons. For PLPUF, using it as a secondary design causes performance degradation as the routing within the PRRs is optimized for the primary design, not for the secondary design. As mentioned in Section III, PLPUF is implemented as a chain. This chain is highly optimized to have a structure as regular as possible. However, when the routing is not optimized for this regular structure, nonoptimum routing introduces more noise to the delay paths of the PUF. Consequently, the reliability gets severely affected as seen in Fig. 3. Moreover, the uniqueness also gets affected, as the response is mainly governed by the routing on the FPGA than by the process variations of the FPGA chips. Finally, when more static routing has to pass within the PRR, the PLPUF performance degrades. This degradation is very low for the case of primary design, while it is more pronounced for the case of secondary design.

In contrast, ROPUF is not affected by DPR as its output relies mainly on the LUT latency when operating as an RO. It is, however, extremely sensitive to the area where it is implemented. Even when implementing ROs of ROPUFs exclusively using top SLICEL columns, to equalize any bias from using asymmetric combinations, we saw a degradation of Uniqueness and Uniformity for a certain area of the FPGAs containing the regions X_1Y_5/X_1Y_6. The degradation seems to be caused by the noise from the BUFG primitives contained in these two regions, as with increasing the utilization of the primitives the degradation increased. Mainly the counter, which is the final stage of the RO-PUF, is affected by the noise and causes the performance degradation. For the rest of the FPGA, no degradation occurred, as they are far enough from the resources.

No significant changes were observed when modifying the areas and BUFG utilization for PLPUF, or when adjusting static routing for ROPUF used as a secondary design. These results were excluded for brevity. Limited information on FPGA manufacturers' routing, noise, and technologies restricts our explanation.

V. Conclusion

In this letter, we tackle the characterization of PUFs when used as reconfigurable accelerators on FPGAs. Our results show that for ROPUF, it is very important to choose the area on-chip where the PRR is used for implementing ROPUF to keep good performance. It is of utmost importance to keep the ROPUFs implemented away from the area where the clock resources are located, otherwise, the performance degrades. As for PLPUF, it should be implemented as a primary design in order not to degrade the quality metrics. As when it was implemented as a secondary design, the reliability degrades significantly. The degradation of the performance increases in relation to the amount of static routing passing within the PRR. In comparison to the state-of-the-art, we are the first to study the effects of runtime reconfiguration on PUF performance.

Acknowledgment

The authors thank Zhiyao Wang for helping to implement the PUFs for the VC707 board.

References

[1] Xilinx. *Partial Reconfiguration UG (v2018.1)*. (2018). [Online]. Available: https://docs.xilinx.com/v/u/ZI5fjL~KOubo_96KHVIxdg
[2] A. Hassan, H. Mostafa, H. A. H. Fahmy, and Y. Ismail, "Exploiting the dynamic partial reconfiguration on NoC-based FPGA," in *Proc. NGCAS*, 2017, pp. 277–280.
[3] J. Henkel, J. Becker, and L. Bauer, "Adaptive application-specific invasive micro-architectures," in *Invasive Computing*. Boca Raton, FL, USA: FAU Univ. Press, 2022.
[4] D. P. Sahoo, R. S. Chakraborty, and D. Mukhopadhyay, "Towards ideal arbiter PUF design on Xilinx FPGA: A practitioner's perspective," in *Proc. DSD*, 2015, pp. 559–562.
[5] Y. Hori, H. Kang, T. Katashita, and A. Satoh, "Pseudo-LFSR PUF: A compact, efficient and reliable physical unclonable function," in *Proc. ReConFig*, 2011, pp. 223–228.
[6] H. Nassar, L. Bauer, and J. Henkel, "CaPUF: Cascaded PUF structure for machine learning resiliency," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4349–4360, Nov. 2022.
[7] A. Herkle, H. Mandry, J. Becker, and M. Ortmanns, "In-depth analysis and enhancements of RO-PUFs with a partial reconfiguration framework on Xilinx Zynq-7000 SoC FPGAs," in *Proc. HOST*, 2019, pp. 238–247.
[8] A. Herkle, P. Rossak, H. Mandry, J. Becker, and M. Ortmanns, "Comparison of measurement and readout strategies for RO-PUFS on Xilinx Zynq-7000 SoC FPGAs," in *Proc. ISCAS*, 2020, pp. 1–5.
[9] R. Hesselbarth, F. Wilde, C. Gu, and N. Hanley, "Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs," in *Proc. HOST*, 2018, pp. 126–133.
[10] J. Kong, F. Koushanfar, P. K. Pendyala, A.-R. Sadeghi, and C. Wachsmann, "PUFatt: Embedded platform attestation based on novel processor-based PUFs," in *Proc. DAC*, 2014, pp. 1–6.
[11] J. Vliegen, M. M. Rabbani, M. Conti, and N. Mentens, "SACHa: Self-attestation of configurable hardware," in *Proc. DATE*, 2019, pp. 746–751.
[12] K.-U. Müller, R. Ulrich, A. Stanitzki, and R. Kokozinski, "Enabling secure boot functionality by using physical unclonable functions," in *Proc. PRIME*, 2018, pp. 81–84.
[13] Xilinx. *7 Series FPGAs Configurable Logic Block UG (v1.8)*. (2017). [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB