A Design and Implementation of Ring Oscillator Physically Unclonable Function Using the Xilinx FPGA

Swati Kulkarni, R. M. Vani, and P. V. Hunagund

1 Introduction

What do you consider to be the most significant aspect of security? Banking, the military, data on the internet and networking are all businesses in which security is critical. Embedded security is also used in numerous industries, such as hardware security. Let's start with an explanation of what hardware security entails. Security-related hardware platforms include ASICs, semi-custom, and fully bespoke ICs. FPGAs are getting more popular as a result of advancement and revolution. FPGAs are preferred over full-custom ICs and ASICs because they are reusable and reprogrammable. Researchers are now looking toward SoC-based FPGAs. A built-in high-speed CPU, a large number of FPGA resources, and a shorter time to market distinguish SoC-based FPGAs. The design time is reduced because IPs are pre-defined and pre-verified. To launch a product in a certain amount of time, most designers supply intellectual property (IP). If third-party IP contains a hardware trojan that is damaging to the design, it should be avoided. Due to their increasing demand, FPGAs have become a popular target for piracy [1, 2]. Because of advances in attacking tactics, the secret key used in traditional cryptography is no longer safe. The Cryptographic key is frequently stored in EEPROM or battery backup SRAM. An attacker can utilize side-channel attacks to try to gain the cryptographic key once the memory has been started up. Another disadvantage of keeping the cryptographic key inside the EEPROM or battery-backed SRAM is that it raises the cost by increasing the hardware complexity [3]. That address these issues, we need a promising solution to enable hardware security. The PUF is a blessing in disguise. There is no need to

S. Kulkarni (✉) · P. V. Hunagund
Department of Applied Electronics, Gulbarga University, Gulbarga 585106, India
e-mail: swatikulkarni494@gmail.com

R. M. Vani
University Science Instrumentation Centre, Gulbarga University, Gulbarga 585106, India

S. Mukherjee et al. (eds.), *Intelligent Systems for Social Good*,
Advanced Technologies and Societal Change,
https://doi.org/10.1007/978-981-19-0770-8_13

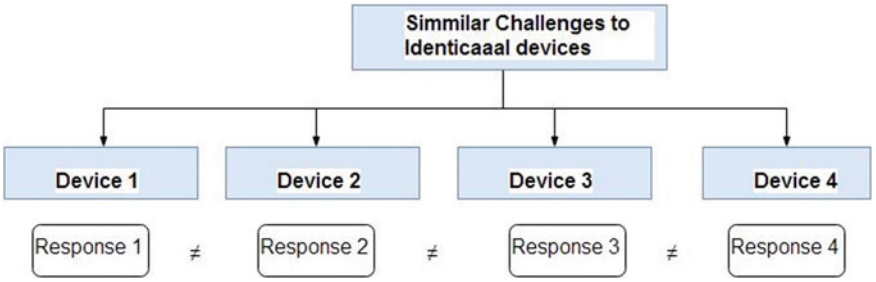

Fig. 1 Uniqueness of PUF (*Source* http://www.ictk.co.kr/servicenproduct/puf)

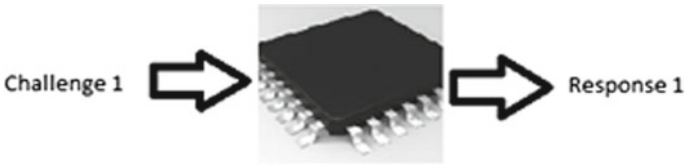

Fig. 2 Reliability of PUF (*Source* https://cryptography.gmu.edu/research/pufs.php)

save the key in PUF-based encryption, and no one can clone or replicate it. PUF focuses on Integrated Circuits' inherent nanoscale process changes (IC). The power, voltage, and temperature changes of an IC are all elements to consider (PVT). Due to manufacturing variance, even two identical ICs or components can have different propagation delays, which can be uncontrollable. PUF accepts input as a challenge and responds with output. Each time PUF is used, and it creates a different and random response [4].

Even if two identical ICs are applied to the same challenge, their responses will differ (see Fig. 1). This one-of-a-kind response may be considered a device signature. This is one of the most distinctive characteristics of PUF [5]. Figure 2 shows another important aspect of PUF which is its reliability. If a particular challenge is applied to the PUF device, it produces a response. If the same challenge is given to the same device after a few years, it should generate the same response as before [6].

The applied input determines the output of any mathematical or software function; however, the output of PUF is determined not just by the applied input, but also by the device's internal variation. This feature of PUF makes it different than other mathematical functions or software functions. Please keep in mind that the PUF response is not a cryptographic key. The response bits must undergo error repair and detection.

1.1 Related Work

The two types of PUF are silicon-based PUF and non-silicon-based PUF. The first PUF, an optical PUF or non-silicon PUF, was created by authors [7]. Silicon-based PUF comes in two types: delay-based and memory-based. The most prevalent delay-based PUF is an Arbiter PUF (APUF), which is also known as a Strong PUF because it contains the most challenge-response pairings. Weak PUF was the name given to RO-PUF when it was originally introduced to a small number of challenge-response pairs by [8]. Author [9] described a PUF based on a modified Ring oscillator (RO-PUF). Delay PUF assesses devices' propagation delays and develops unique replies by posing challenges to them. The Arbiter PUF is powerful because it has a large number of challenges and response pairings (CRPs). Because it contains a low number of CRPs, RO-PUF is a weak PUF. Routing symmetry is required for APUF, which is challenging to implement on an FPGA [10]. On Arbiter PUF, modeling attacks are more widespread. RO-PUF does not require symmetric routing and maybe developed quite quickly on an FPGA board [11]. We implemented and tested the PUF on FPGAs in this study. ASICs are less flexible than FPGAs. Because FPGAs are preconfigured, FPGA design requires less time to market. In comparison to ASIC fabrication, FPGAs are a low-cost option. These are the main reasons why we chose FPGA technology over ASIC implementation.

In 2012, the Xilinx came with seven series unified architecture FPGA with a new tool, Vivado. The Xilinx Launched 28 nm technology, System on chip (SoC) boards. SoC is a combination of a hard-wired Processing System (PS) and Programmable Logic (PL) that consists of many Configurable Logic Blocks (CLBs) and a complex routing system. Figure 3 shows the device view on the Xilinx Zed Board with PS and PL. PS refers to the black box in the left corner, while PL refers to the rest of

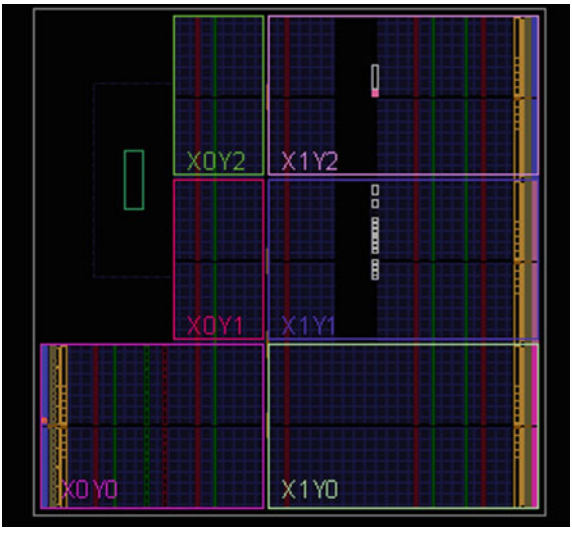

Fig. 3 Device view with PS and PL [12]

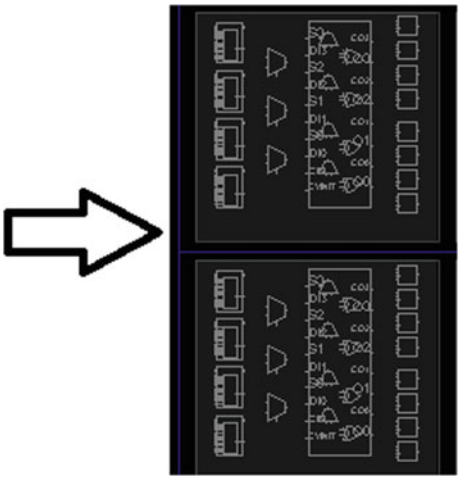

Fig. 4 Single CLB structure

the colorful marking zones. PS and PL are both available on a single chip, which is referred to as an SoC. On the PS side, the Xilinx Zed Board has a dual-core ARM A9 dedicated CPU, and on the PL side, the Artix 7. The CLBs consists of a switch matrix that connects the slices and the outside world. Slice L and slice M are the two slices that make up a CLB shown in Fig. 4. LUTs are used to create logic functions, whereas FFs are used to store data in a single slice. Each CLB on the Xilinx Zynq chip used in this study contains two slices. Four LUTs and eight FFs make up each slice. The SoC-based design can fulfill these requirements. In this chapter, we have implemented RO-PUF and tested it on the Xilinx Zed Board (SoC) [13]. The main focus of this paper is on an area-optimized RO PUF implementation with good reliability. The design was tested for its area usage, speed, uniqueness, uniformity, and reliability. With a brief introduction to PUF, we discussed the Ring oscillator PUF with mathematical delay model in the second section, the implementation of RO PUF on FPGA in the third section, results in section four, and a conclusion in the fourth section.

2 Ring Oscillator

Figure 5 shows a schematic of the Ring oscillator formed using one NAND Gate and three inverters. RO will generate a square waveform. The frequency of the square wave depends on the net delays and propagation delay of each gate [14].

Figure 6 shows the basic block diagram of the 1-bit RO PUF. Ring oscillator PUF design comprises 32 ROs. The first sixteen ROS connected to one of the 16:1 multiplexer another 16 ROs connected to another 16:1 multiplexer. Select lines of multiplexers will decide which RO to be selected. Both multiplexers will use select lines, i.e., ‘challenge’ input, to choose any two ROs at the same time. Multiplexers

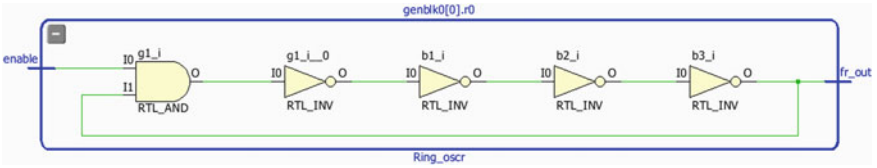

Fig. 5 Ring oscillator [15]

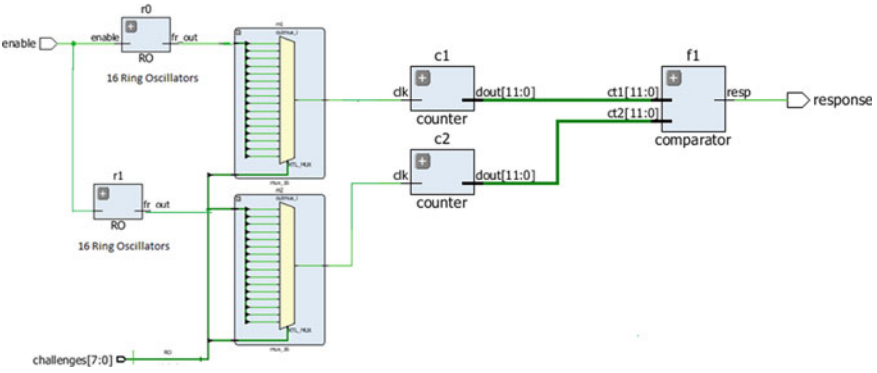

Fig. 6 One-bit RO PUF schematic

connected clock input to two distinct 12-bit counters via the output of multiplexers. The counting will be carried out on every affirmative edge of ROs. The comparator compares the counter output. As previously said, each RO generates frequencies with distinct periods. Even if both counters are linked to the multiplexers, counting may start at the respective incoming positive edge. It causes any one of the counters may get overflows first. If the upper counter gets overflowed first than the lower counter then comparator output will be '1' otherwise '0'. The output of the comparator is a 'response' bit. ROs on two different devices have different frequencies. This difference allows the RO PUF to characterize devices to authenticate them [1, 16].

This is an implementation of the RO-PUF in 8 bits. Figures 7 and 8 demonstrate

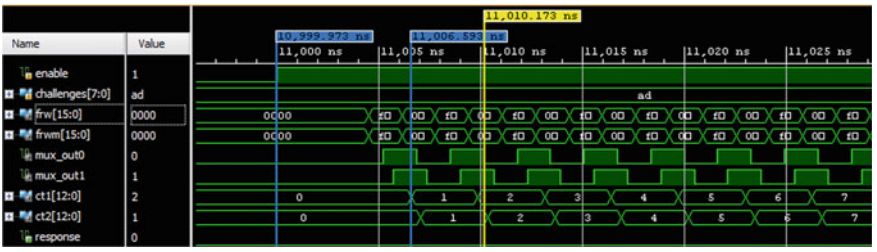

Fig. 7 Post implemented timing simulation (**a**)

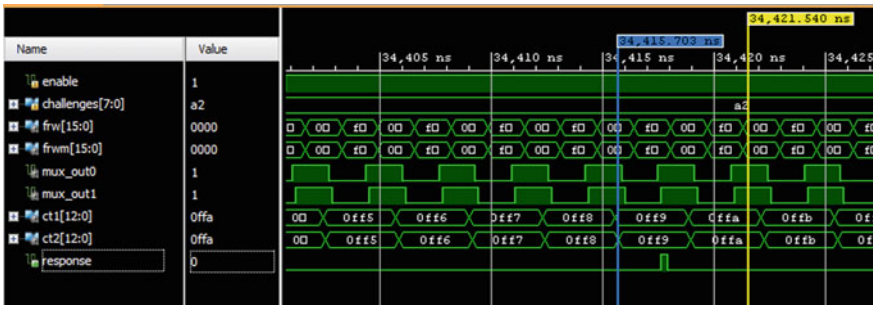

Fig. 8 Post implemented timing simulation (**b**)

the time simulation of a single bit RO PUF after it has been implemented (b). The inputs are Enable and Challenges. The output of each module can be seen here. ROs will produce square waves, as previously indicated. The outputs of Ring Oscillators are frw and frwm. Mux out1 and mux out2 are the mux outputs. The mux outputs are mux out1 and mux out2, and two multiplexers can simultaneously select any two ROs. Two counters use the clock input from these mux outputs as well. Every positive edge of the incoming clock will be counted as well. The outputs of the two counters are referred to as Ct1 and Ct2. Because no counters are overflowing, the response bit remains '0'. If Ct1 reaches its maximum value before Ct2, the response bit will be set to '1'.

In this Fig. 8, we can observe the 'response' bit. As soon as the first counter gets overflows response bit became '1'.

2.1 Mathematical Function of Delay

To understand the RO PUF's operation, we must first know the delay module. According to [5], every path consists of two delay elements, a static delay element and a random delay element. A Static delay element is the delay of a circuit calculated by adding the individual gate and net delays for each path a Random delay element is present due to process variations. Ideally, the PUF output should only be dependent on its process variation. Hence, out of the two components, for a PUF, the random delay component should be the significant factor. As RO-PUF is delay-based PUF, we defined Delay D of a net.

$$D = DS + D\dot{R} \tag{1}$$

where DS is a static delay and DR is a random delay. The Static delay can be determined using the timing tool available in Vivado by applying timing constraints. It will remain the same for the same RTL and Board. But the random delay is delay

generated due to internal process variation of devices hence it will be different for each board. Mainly PUF functionality depends on Random delay. The conclusions of the Arbiter, Butterfly and Ring Oscillator PUFs are based on the premise that the symmetrical pathways' static delays cancel out [17].

Let us consider two nets N1 and N2 from the same design which need to compare,

$$D(N1) = Ds1 + DR1 \tag{2}$$

$$D(N2) = Ds2 + DR2 \tag{3}$$

Equations 2 and 3 can be used to express the delay D differences between the two networks. If two nets $N1$ and $N2$ are identical, $Ds1 = Ds2$ and the delay skew D between them can be calculated.

$$\Delta D = D1 - D2 = DR1 - DR2 = \Delta DR \tag{4}$$

The static delay differences DS should tend to zero in an ideal case, and the delay comparison between the two net delays should be a function of the random delay component. However, if $DS1DS2$ (that is, $N1$ and $N2$) are not similar, the delay skew can be determined as follows:

$$\Delta D = Ds1 - Ds2 + DR1 - DR2 = \Delta DS + \Delta DR \tag{5}$$

This equation suggests that the delay difference between two nets is primarily the sum of the difference of the individual components. Even a slight contribution by the static delay component can result in a biased PUF output. If $DS > DR$, then the effect of random variation on the output will be insignificant and the output of PUF depends on static delay rather than random delay. In that case we can say that the output of PUF will be biased [4].

3 Implementation of RO PUF on FPGA

While implementing the RO PUFs on an FPGA, we need to consider the following things. The synthesis tool will always try to optimize a design concerning speed and area. The ROs with unpredictable behavior would either throw a latch warning or be completely optimized out as it serves no purpose from a tool point of view. Additionally, every pair of ROs that is being compared is to be implemented the same. ROs design needs to instantiate many times but the optimization point of view tool will remove the multiple instances. To avoid this, the placement and routing had to be thoroughly constrained. The usage of slices, logic, pins, and routing needs to be done manually to prevent any optimization by the synthesis tool. Ring Oscillator PUFs have the requirement of symmetric routing. We must construct it extremely

carefully to avoid biassing of delays and to maintain all ROs in the PUF identical. RO PUF necessitates the placement of ROs by hand. Each slice of the 7series FPGA consists of four LUTs, as we explained previously. Because each RO contains four logic gates, we assigned each LUT to one of them. One RO was implemented per slice. In FPGA, hard macros are advised for manual installation [18].

Hard macros are required to avoid any additional delays caused by the various instantiations required for the implementation. Instead of the entire design being symmetrically placed and routed, the RO PUF simply requires each Ring Oscillator to be similar. Furthermore, utilizing hard macros on Xilinx FPGAs, similar ring oscillator routing is simple [19]. Formerly to perform manual placement and routing, hard macros were employed. In the earlier version of the Xilinx tool, the ISE suite supports an FPGA editor to crate Hard macro. Now the Xilinx Vivado does not support the FPGA editor instead it has a TCL shell. The Xilinx Design Constraint (XDC) constraints are written in TCL scripting. The Tcl script helps to place all the resources at the desired location. Figure 9 shows the device view after the manual placement of the component [15].

Figure 9 shows the TCL script written in the TCL console of the Vivado. The Tool will automatically convert the TCL script into the Xilinx Design Constraints (XDC) shown in Fig. 10. The constraint "**set_property**" is responsible to place the instance into the cell. BEL specifies a specific placement within a Slice for a register or LUT. BEL is generally used with an associated LOC property to specify the exact placement of a register or LUT. A primitive component's LOC indicates where it should be placed (Fig. 11).

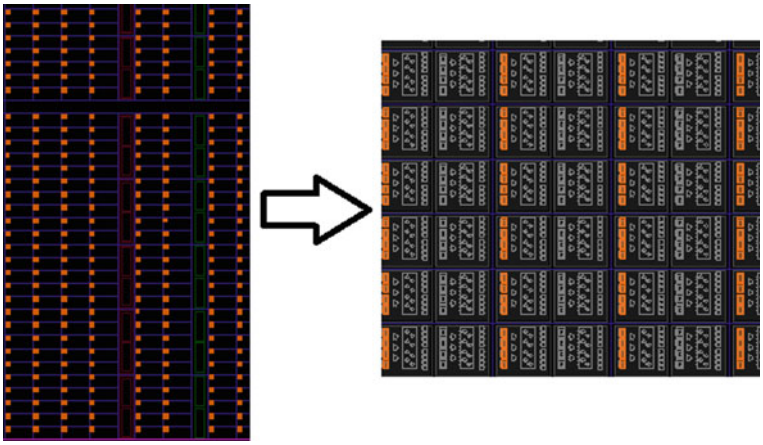

Fig. 9 Device view after manual placement of the components

```
for {set q 0}  {$q <= 15}  {incr q}  {
set current_pos_x [expr $start_pos_x+0]
set current_pos_y  [expr $start_pos_y + $q]

startgroup
place_cell "generate_block_identifier[0].t0/genblk0[$q].r0/fr_out_INST_0" SLICE_X${current_pos_x}Y${current_pos_y}/D6LUT
endgroup
startgroup
place_cell "generate_block_identifier[0].t0/genblk0[$q].r0/n_inferred_i_1" SLICE_X${current_pos_x}Y${current_pos_y}/C6LUT
endgroup
startgroup
place_cell "generate_block_identifier[0].t0/genblk0[$q].r0/n_inferred_i_2" SLICE_X${current_pos_x}Y${current_pos_y}/B6LUT
endgroup
startgroup
place_cell "generate_block_identifier[0].t0/genblk0[$q].r0/n_inferred_i_3" SLICE_X${current_pos_x}Y${current_pos_y}/A6LUT
endgroup
```

Fig. 10 The TCL script

```
set_property BEL D6LUT [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/fr_out_INST_0}]
set_property LOC SLICE_X0Y0 [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/fr_out_INST_0}]
set_property BEL C6LUT [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/n_inferred_i_1}]
set_property LOC SLICE_X0Y0 [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/n_inferred_i_1}]
set_property BEL B6LUT [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/n_inferred_i_2}]
set_property LOC SLICE_X0Y0 [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/n_inferred_i_2}]
set_property BEL A6LUT [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/n_inferred_i_3}]
set_property LOC SLICE_X0Y0 [get_cells {generate_block_identifier[0].t0/genblk0[0].r0/n_inferred_i_3}]
```

Fig. 11 The XDC location constraints

4 Experimental Setup and Results

We converted our design to IP for hardware testing. Figure 12 shows the connection diagram. The Xilinx VIO and ILA IPs are interfaced with PUF IP. ILA and VIO are used for Hardware Debugging. The LogiCORE IP Virtual Input/Output (VIO) is a programmable IP core for real-time monitoring of the design's internal signals.

ILA is used to monitor internal programmable logic signals and ports for post-analysis. Using VIO and ILA we are applying the input to our PUF design and observing the outputs. Figures 13, 14, 15, and 16 show the design's real-time output in terms of ILA output waveform for different combinations of CRP.

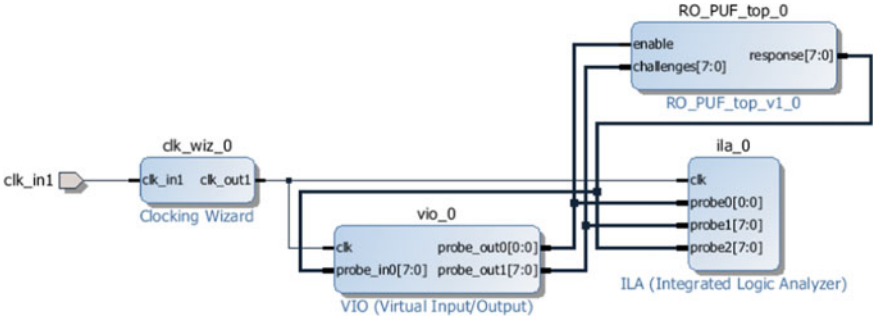

Fig. 12 Block diagram for Hardware debugging

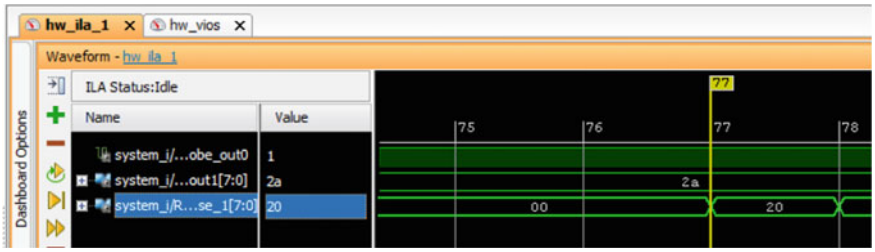

Fig. 13 ILA output waveform (**a**)

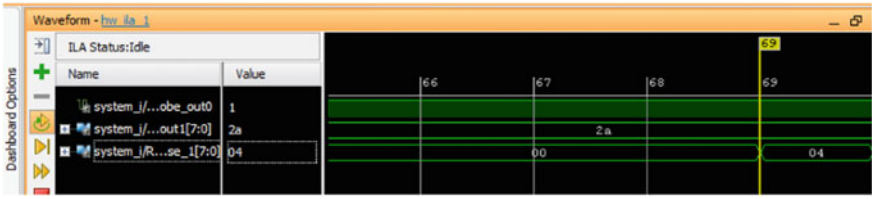

Fig. 14 ILA output waveform (**b**)

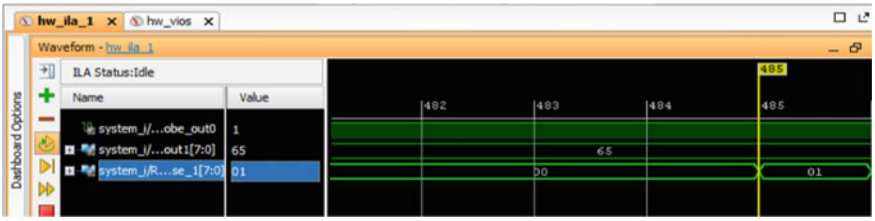

Fig. 15 ILA output waveform (**c**)

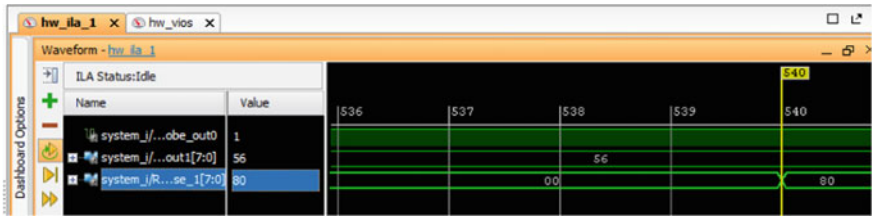

Fig. 16 ILA output waveform (**d**)

We have tested the same RO-PUF design on two different Zed Board FPGA. Here we can observe that even for two similar challenges we are getting different response bits on two different boards. This shows the reliability of the PUF design. Similarly, we got various answer bits when we applied the different challenges to the same

Table 1 We have verified the output for the following cases

Sr. nos.	Criteria	Observations
Case I	Same design on two different boards (inter) with same challenges	We observed unique and random response bits. It proves the uniqueness of PUF
Case II	Same design, same board but different clock region (intra) with same challenges	We observed unique and random response bits
Case III	Same design, Same board, same region but different day and time with same challenges	We observed same response bits again and again. It proves the reliability of PUF
Case IV	Same design, same board, same region but same day and time with multiple challenges	We observed unique and random response bits

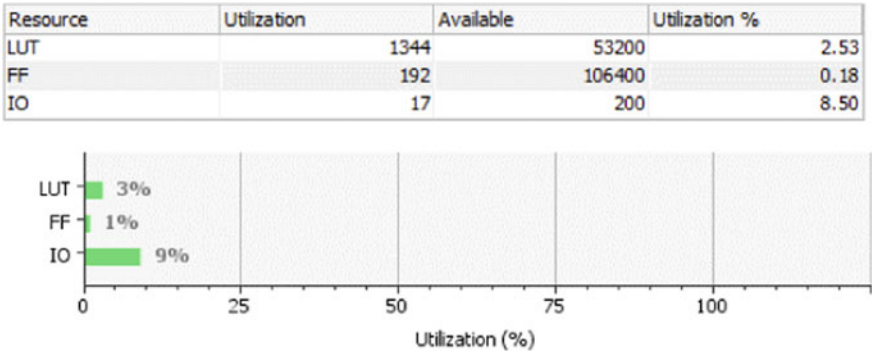

Fig. 17 Utilization summary

design and board. We have presented two challenges at random. We have confirmed that we are getting the unique answer bits for all of the tasks.

Figure 17 shows the utilization summary of the resource. The design uses less than 3% of the total FPGA Logic resources and less than 10% of IO resources.

The design's power analysis is shown in Fig. 18. The total onchip power, junction temperature, and thermal margin are all defined via power analysis. The implemented system logic consumes 61% of the total power available.

5 Conclusion

Hardware-oriented security is an upcoming field in the electronics industry. Today hardware designers are paying more focus on hardware security. Many different PUF designs are getting designed by the researchers. The RO PUF architecture is straightforward, but the FPGA implementation proved difficult. We did so till the RO

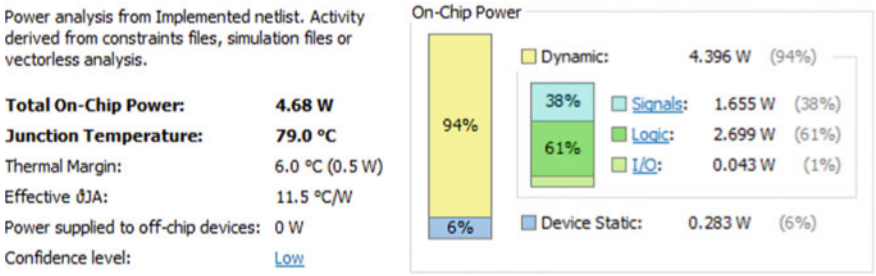

Fig. 18 Power report

PUF on FPGA was successfully implemented. We verified the uniqueness property and reliability property of PUF. However, PUF output is noisy, the generated response needs to filter out using an error correction algorithm. After that, the FPGA board will connect to the computer for more Hamming distance and Entropy research.

References

1. Kumar, M.A., Bhakthavatchalu, R.: FPGA based delay PUF implementation for security applications. In: IEEE International Conference on Technological Advancements in Power and Energy (TAP Energy). Kollam, India (2017)
2. Yin, C., Qu, G., Zhou, Q.: Design and implementation of a Group-based RO PUF. In: Automation & Test in Europe Conference & Exhibition. Grenoble, France (2013)
3. Lim, D., Lee, J.W., Gassend, B., Suh, G.E., van Dijk, M., Devadas, S.: Extracting secret keys from integrated circuits. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **13**, 1200–1205 (2005)
4. Bernard, F., Fischer, V., Costea, C., Fouquet, R.: Implementation of ring-oscillators-based physical unclonable functions with independent bits in the response. Int. J. Reconfig. Comput, **2012**, 13 (2012)
5. Suh, G.E., Devadas, S.: Physical unclonable functions for device authentication and secret key generation. In: 44th ACM/IEEE Design Automation Conference. San Diego, CA (2007)
6. Handschuh, H., Schrijen, G.-J., Tuyls, P.: Hardware intrinsic security from physically unclonable functions. In: Sadeghi, A.-R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security, Information Security and Cryptography, pp. 39–53. Springer, Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-14452-3_2
7. Wallrabenstein, J.R.: Practical and secure IoT device authentication using physical unclonable functions. In: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 99–106. Vienna (2016)
8. Kodýtek F., Lórencz, R.: A design of ring oscillator based PUF on FPGA. In: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, pp. 37–42. IEEE (2015)
9. Bhargava, M., Mai K.: An efficient reliable PUF-based cryptographic key generator in 65nm CMOS design. In: Automation & Test: in Europe Conference & Exhibition. Dresden (2014)
10. Alkatheiri, M.S., Zhuang, Y., Korobkov, M., Sangi, A.R.: An experimental study of the state-of-the-art PUFs implemented on FPGAs. In: IEEE Conference on Dependable and Secure Computing. Taipei, Taiwan (2017)

11. Kokila, J., Ramasubramanian, N.: Enhanced authentication using hybrid PUF with FSM for protecting IPs of SoC FPGAs. J. Electron. Test. **35**, 543–558. (2019)
12. The Xilinx Xilinx Vivado Design Suite User Guide Synthesis UG901 (v2019.2) - January 27, 2020
13. Pappu, R., Recht, S.B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Sci. J. 5589. **297**, 2026–2030
14. Morozov, S., Maiti, A., Schaumont, P.: A comparative analysis of delay based PUF implementations on FPGA: IACR. Cryptol. ePrint Arch. 629 (2009)
15. Kulkarni, S., Vani, R.M., Hunagund, P.V.: FPGA based hardware security for edge devices in internet of things. In: 5th International Conference on Communication and Electronics Systems (ICCES), pp. 1133–1138. IEEE, Coimbatore, India (2020)
16. Maiti, A., Schaumont, P.: Improved ring oscillator PUF: an FPGA-friendly secure primitive. J. Cryptogr. **24**, 375–397 (2011)
17. Tajik, S., Dietz, E., Frohmann, S., Seifert, J.-P., Nedospasov, D., Helfmeier, C., Boit, C., Dittrich, H.: Physical characterization of arbiter PUFs. Springer, Berlin Heidelberg (2014)
18. Gehrer, S., Leger, S., Sigl, G.: Aging effects on ring-oscillator-based physical unclonable functions on FPGAs. In: 2015 International Conference on ReConfigurable Computing and FPGAs (ReConFigure). Mexico City (2015)
19. Xin, X., Kaps, J.-P., Gaj, K.: A configurable ring-oscillator-based PUF for Xilinx FPGAs. In: 4th Euromicro Conference on Digital System Design—Oulu, pp. 651–657 (2011)
20. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525), pp. 176–179. IEEE, Honolulu, HI, USA (2004)
21. Eiroa, S., Baturone, I.: An analysis of ring oscillator PUF behavior on FPGAs. In: International Conference on Field-Programmable Technology, New Delhi, India, pp. 1–4 (2011)