Journal of Engineering Design and Analysis
Volume 4, Issue 2 - 2021, Pg. No. 6-11
Peer Reviewed Journal

Article

Design and Implementation of Novel Design Methodology for Securing the Internet of Things Applications

Swati Kulkarni[1], Vani RM[2], PV Hunagund[3]

[1,2,3]Department of Applied Electronics, Gulbarga University, Gulbarga, Karnataka, India.

DOI: https://doi.org/10.24321/2582.5607.202101

I N F O

Corresponding Author:
Swati Kulkarni, Department of Applied Electronics, Gulbarga University, Gulbarga, Karnataka, India.
E-mail Id:
swatikulkarni494@gmail.com
Orcid id:
https://orcid.org/0000-0002-3282-1685

How to cite this article:
Kulkarni S, Vani RM, Hunagund PV. Design and Implementation of Novel Design Methodology for Securing the Internet of Things Applications. *J Engr Desg Anal* 2021; 4(2): 6-11.

Date of Submission: 2021-09-02
Date of Acceptance: 2021-11-20

A B S T R A C T

Typically, in the Internet of Things (IoT) yields objects are controlled and monitored remotely over a network. A standard IoT system includes various sensors that combine with microprocessors and other custom peripherals to perform their operations. Data collection and processing, computation, and finally communication these operations are involved to carry out IoT applications. Designer and user both demand these operations should be performed without any kind of unauthorized interference. As per the current trend, the way we have relied on technology, the way data exchange has started, it is not possible to rely on existing data security systems. It has become imperative to add a concrete solution to the existing infrastructure of security. Hardware security is attracted attention of researchers because of its importance in IoT applications, IP securities, and controlling counterfeit electronic devices. Physically Unclonable Functions (PUF) is an innovative approach that provides security primitives and also will resist Integrated Circuit (IC) cloning and counterfeiting. PUF works on the principle of process variations present inside the hardware. PUFs carry capricious and event-specific values and can be used to provide hardware security. Nowadays IoT design has been implemented on the SoC platform. Here we have designed PUF on the SoC platform so that it will be helpful to IoT designs. The presented work will help in understanding the ROPUF design with respect to simulation, synthesis, placement and routing and hardware validation.

Keywords: Register Transfer Level (RTL), FPGA-SOC, Simulation, Synthesis, Placement & Routing and Hardware Validation

Introduction

When we are considering information security by concentrating on the goals of cryptography. Cryptology obeys certain guidelines and mathematical calculations to accomplish data security objectives. Normally cryptographic algorithms are used for the security objective with the help of the key. So, the bottom line is in the data security 'key' plays an important role. Normally 'key' has been kept in battery-backed SRAM which is quite expensive and not suitable for small IoT applications as more hardware/ area requirement. One more issue is, what if this 'key' has been stolen? It is possible with the side-channel attack, reverse engineering, or tampering with hardware. Now we need a concrete solution for data security.

PUF is a part of the hardware that produces unpredictable responses over challenges due to their manufacturing variations. Each PUF response is not only a function of the applied challenge but also of the PUFs physical disorder. To apply PUF in the security field, it also needs to have the property of easy to manufacture but hard to duplicate, even under exact the same manufacturing procedures.[20,16,11] PUFs are objects that are unclonable in other words these are altered confirmation and can't be controlled without physically crushing them. Because it attributes randomness and uniqueness, PUFs can be used for random number generation (i.e., secret key generation for cryptographic purposes), authentication (i.e., IC, user, product authentication), and identification, etc. PUFs can be seen as silicon fingerprints which are unique in every piece of hardware, reproducible in the same piece of hardware but not in another. Based on how the randomness is presented, PUFs can be categorized into different types. The PUFs using explicitly introduced randomness's such as optical PUF (Pappu R. S. 2001). that is non silicon PUF and Silicon PUF, using intrinsic randomness's such as delay PUF and memory PUF.[15]

PUF's Input and Output that is challenge-response pairs (CRPs) respectively split the PUF into two types, PUFs can be categorized as weak and strong PUFs.[15]

Weak PUF

The CRPs are the limited set of size, often used for key storage. The weak PUFs usually only has less CRP, and Usually considered that the challenge-response access is secure, not accessible by attacker.[18]

Strong PUF

The CRPs are the maximum set of the size of PUF, often used for authentication. For strong PUFs, there can be a large set of CRPs publicly accessible. However, with this large public setting, it is still impossible to predicate an unknown CRP.[18]

Ring Oscillator (RO) PUF Architecture

The two most common silicon PUFs at presence are delay PUF and memory PUF.[17,14] Arbiter PUF[22] and Ring Oscillator PUF[12,5] are the popular subtypes of delay PUF. In this article we will discuss about the FPGA implementation of the delay PUF, Ring Oscillator (RO) PUF. This design was proposed by.[19] A RO PUF has two or more cascaded inverters which are connected in a feedback loop. It is expected that each oscillator has different and unpredicted frequencies from others due to the manufacturing variation resulting in each inverter's different delay. Finally, there are counters (of the same size) for each oscillator whose increment speed depends on the frequencies of the corresponding chains which serve as the clocks. A 1-bit RO PUF is shown in the figure 1.[6]

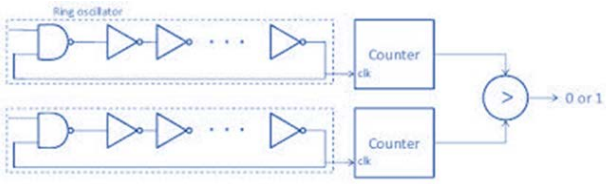


Figure 1.A RO PUF generating 1 bit of output[3]

The above circuit is able to produces 1 bit of PUF output. The challenge will be the input selecting any two ROs from the group of N ROs, and the response is the comparison of the values in the two counters clocked by the outputs of the selected ROs. To generate more response bits with the same challenge input, more RO groups and counters can be added in the design.[10]

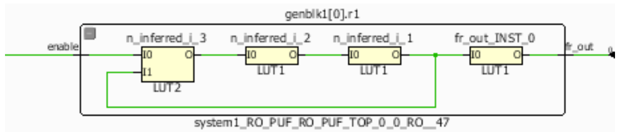


Figure 2(a).The synthesis schematic of RO PUF

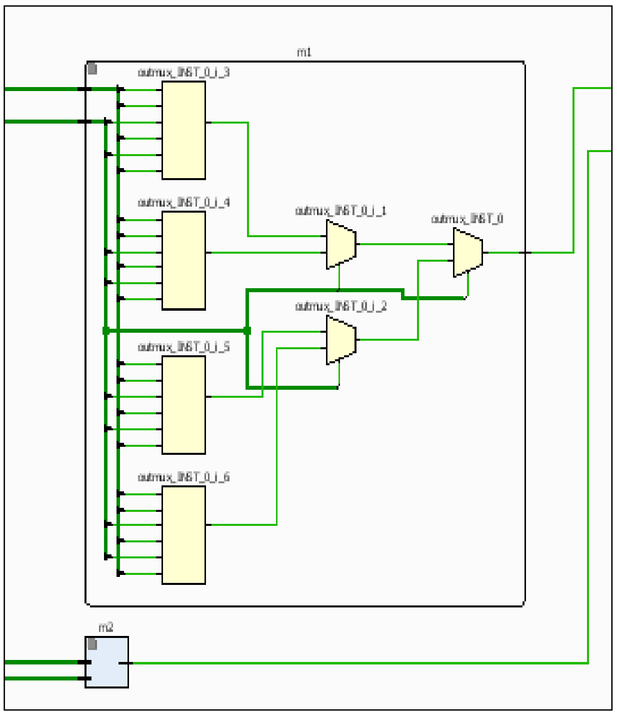


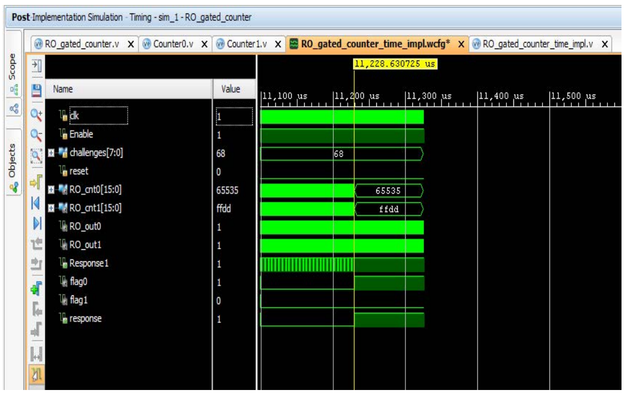
Figure 2(b).The synthesis schematic of MUX 16:1 from RO PUF

Figure 2, shows the synthesis schematic of RO PUF. In the proposed design the RO PUF has two 16:1 multiplexer. According to the design, we can apply 0 to 256 different challenges to RO PUF. For each challenge, we expect a unique and random response from RO PUF. Comparison is made when the counter overflows and response bit get generated. Even though the basic design was proposed by[19,4] but we have implanted and tested with new approach. Also, we elaborated the entire procedure of the experiment.

Cautious RO Implementation: Difficulties and Solutions

Simulation

To verify the functional behavior of any logic normally simulation is preferred. RO PUF is a delay-based PUF hence internal routing delays of each device is influence the output of PUF. Typical behavior simulation does not consider any routing delays.[9] Vivado has a facility of post-implemented timing simulation. This simulation serves to verify the functionality by considering the net and route delays of course, it does not give the exact output like the hardware but it helps to understand the working of the circuit.


Post Implementation Simulation - Timing - sim_1 - RO_gated_counter
RO_gated_counter.v
Counter0.v
Counter1.v
RO_gated_counter_time_impl.wcfg*
RO_gated_counter_time_impl.v
Scope
Objects
11,228.630725 us
11,100 us
11,200 us
11,300 us
11,400 us
11,500 us
68
65535
ffdd

Name	Value
clk	1
Enable	1
challenges[7:0]	68
reset	0
RO_cnt0[15:0]	65535
RO_cnt1[15:0]	ffdd
RO_out0	1
RO_out1	1
Response1	1
flag0	1
flag1	0
response	1

Figure 3(a).Post implemented timing Simulation

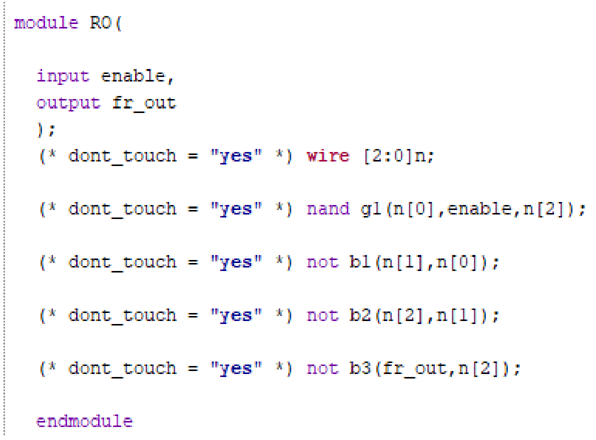
Above Figure 3(a), shows the post implemented timing simulation for 1-bit PUF. 'response' is the final expected output. We can see that the when first counter reaches to the final value before the second counter then only response gets generated.

Synthesis

Although the RO PUF circuit is very simple to understand but while implementing the perfect RO PUF on the Xilinx tool some practical difficulties need to address. The first difficulty is the Xilinx Vivado tool tries to optimize the design while synthesizing the RTL. RO design comprises multiple Ring oscillators with cascades inverters. The synthesis engine will remove the extra inverters and generate the 'empty netlist'. This problem needs to solve using some synthesis attributes provided by the tool. To preserve the inverters in RO from being optimized away, usually the attribute of "KEEP" is used.

In Vivado when a RO schematic is generated, although with the "KEEP" attribute all the inverters can be kept in the RTL schematic view which is pre-optimization, they are not in the technology view which is post-optimization from the synthesized design. The logic trim in the optimization process will still consider any even number of inverters as having no effect on chip outputs, and remove them.

To preserve the complete RO structure, the following



```
module RO(

  input enable,
  output fr_out
  );
  (* dont_touch = "yes" *) wire [2:0]n;

  (* dont_touch = "yes" *) nand g1(n[0],enable,n[2]);

  (* dont_touch = "yes" *) not b1(n[1],n[0]);

  (* dont_touch = "yes" *) not b2(n[2],n[1]);

  (* dont_touch = "yes" *) not b3(fr_out,n[2]);

  endmodule
```

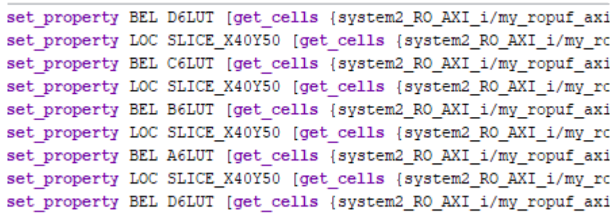
Figure 4.RTL of Ring Oscillator

attribute should be used before each inverter and wire, as well as the RO module instantiation (* DONT_TOUCH = "TRUE" *).[1]

Xilinx tool does not support the combinational loop. This issue is also overcome by using another attribute. (* ALLOW_COMBINATORIAL_LOOPS = "TRUE" *).[23] The problem is not over only after adding this attribute the third important issue need to resolve is about the gated clock. In the RO PUF, since the counter is clocked by a combinational logic, this scenario called "gated-clock.[23]

Implementation

After adding all the necessary attributes in the synthesis stage, the RTL generates the required netlist. The implementation engine of the tool will take care of the Placing and routing of the gate-level netlist generated after synthesis. The tool will always try to place optimum resources with minimum routing delays. Properly functional PUF design should generate different responses on different FPGA boards with the same HDL and configuration.



```
set_property BEL D6LUT [get_cells {system2_RO_AXI_i/my_ropuf_axi
set_property LOC SLICE_X40Y50 [get_cells {system2_RO_AXI_i/my_rc
set_property BEL C6LUT [get_cells {system2_RO_AXI_i/my_ropuf_axi
set_property LOC SLICE_X40Y50 [get_cells {system2_RO_AXI_i/my_rc
set_property BEL B6LUT [get_cells {system2_RO_AXI_i/my_ropuf_axi
set_property LOC SLICE_X40Y50 [get_cells {system2_RO_AXI_i/my_rc
set_property BEL A6LUT [get_cells {system2_RO_AXI_i/my_ropuf_axi
set_property LOC SLICE_X40Y50 [get_cells {system2_RO_AXI_i/my_rc
set_property BEL D6LUT [get_cells {system2_RO_AXI_i/my_ropuf_axi
```

Figure 5.Tcl script for manual placement

However, if the routing and placement are automatically carried out by the design tools, then the timing caused by this automated process will dominate the system timing. This will result in the same CRPs on different FPGA boards Unlike an Arbiter PUF, RO PUF does not require symmetric routing but All RO must be placed properly inside the FPGA layout. To accomplish this necessity of the design, we need to arrange the manual placement of components using the Tcl script. Figure 5, shows the Xilinx design constrained for the location of the components.[24,2]

We targeted the Xilinx ZedBoard all programable soc FPGA. Zedboard has two parts Processing System (PS) consist hard cored ARM cortex A9 processor and Programmable Logic (PL) has Artix 7 FPGA fabric.[8] We intend to use the hard IP of ARM cortex A9 i.e. ZYNQ IP along with RO PUF shown in figure 6 (a).

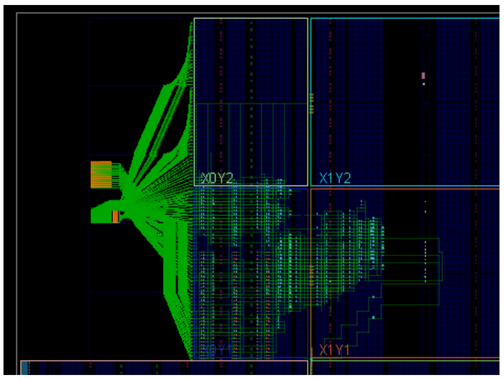


Figure 6.(a) the placement of RO PUF with ZYNQ

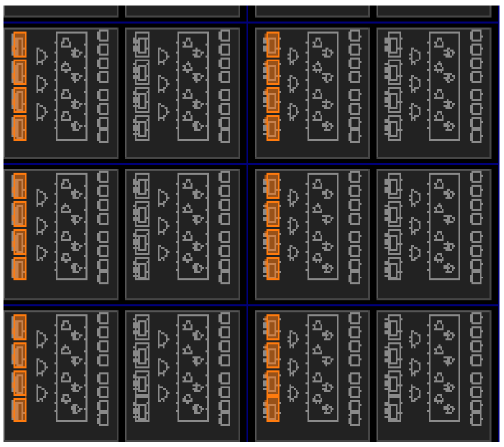


Figure 6.(b) the placement of RO PUF with ZYNQ

In the figure 6 (b), the orange color shows the manual Placement using the Tcl commands. To overcome the setup and hold violation we must place the design near to PS side. The ROs placed vertically from the logic slice X26Y50 onwards. This placement eliminates the timing difference caused by place and route, all the ROs need to be manually placed and routed with identical placement and routing, so that the LUTs' intrinsic delays will be the only factor making a difference in the counters. The relative placement can be set through the set_property LOC and BEL keyword in the. xdc constraints file.[23,21,13]

System Overviews of the RO PUF

Figure 7, shows the block diagram of proposed system. RTL of RO PUF is converted into AXI compatible IP. In the IP integrator of the Xilinx Vivado, Zynq processing unit is connected with IP block of proposed RO PUF.

The PS part can communicate with PL with the AXI interface. To validate the design on the hardware RO PUF is configured

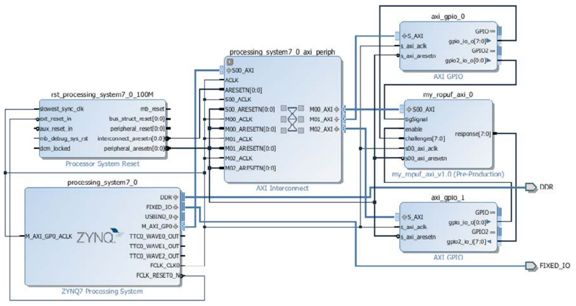


Figure 7.Block diagram of proposed system

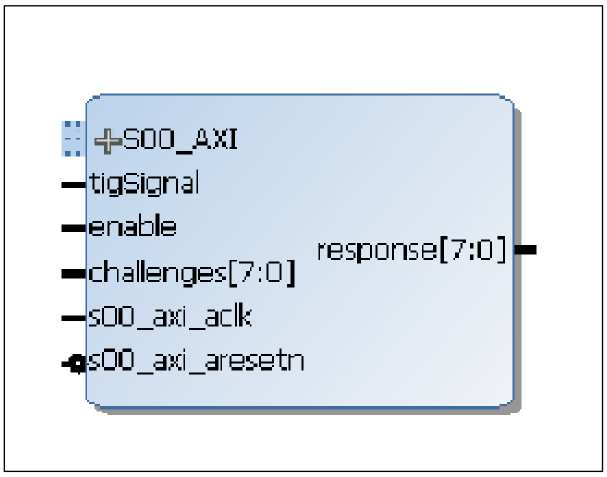


Figure 8.(a) AXI_RO PUF

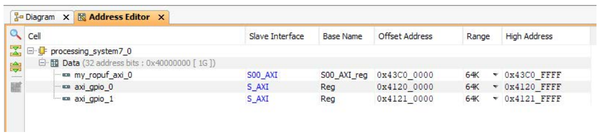


Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
my_ropuf_axi_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_gpio_1	S_AXI	Reg	0x4121_0000	64K	0x4121_FFFF

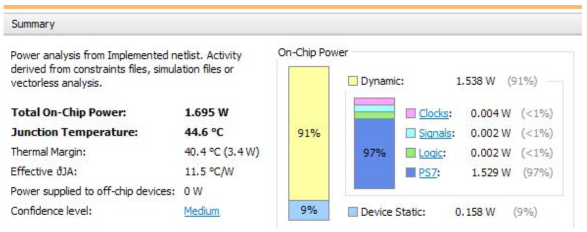
Figure 8.(b)Address Editor contains address of the RO PUF IP

with the inbuilt processor present on the Zedboard. We have converted RTL into the AXI compatible IP shown in the figure 8 (a) and (b).

Address editor contains the address of each peripheral device connected with Zynq processing unit.

Results

Figure a show the power consumption on the proposed RO PUF with ZYNQ IP. Figure 9 (a) and (b) is about the resource utilization of the proposed system. The implemented design uses the optimum power consumption and resource utilization.



Summary

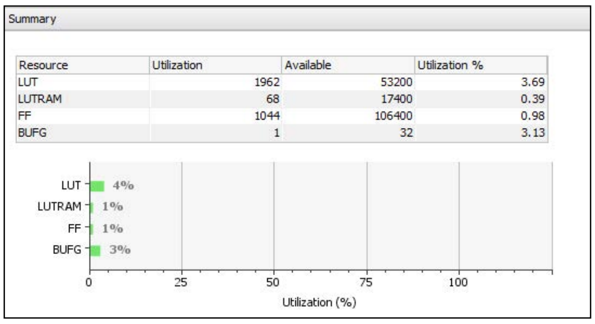
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.695 W**
Junction Temperature:	**44.6 °C**
Thermal Margin:	40.4 °C (3.4 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

On-Chip Power

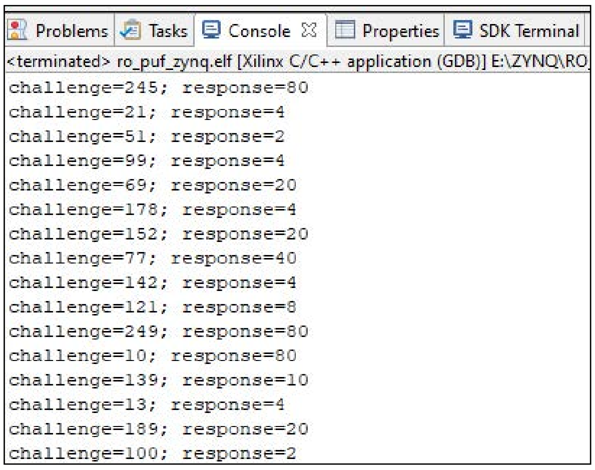
Dynamic:	1.538 W	(91%)
Clocks:	0.004 W	(<1%)
Signals:	0.002 W	(<1%)
Logic:	0.002 W	(<1%)
PS7:	1.529 W	(97%)
Device Static:	0.158 W	(9%)

Figure 9.(a)Power Utilization


Summary

Resource	Utilization	Available	Utilization %
LUT	1962	53200	3.69
LUTRAM	68	17400	0.39
FF	1044	106400	0.98
BUFG	1	32	3.13

Figure 9.(b) Resource Utilization



```
<terminated> ro_puf_zynq.elf [Xilinx C/C++ application (GDB)] E:\ZYNQ\RO
challenge=245; response=80
challenge=21; response=4
challenge=51; response=2
challenge=99; response=4
challenge=69; response=20
challenge=178; response=4
challenge=152; response=20
challenge=77; response=40
challenge=142; response=4
challenge=121; response=8
challenge=249; response=80
challenge=10; response=80
challenge=139; response=10
challenge=13; response=4
challenge=189; response=20
challenge=100; response=2
```

Figure 10.SDK console Contains the final Input and Output of RO PUF

ZYNQ will allocate the address for each connected peripheral. UART of the Zedboard is configured and connected to the laptop. Challenges are applied through the Xilinx Software Development Kit. XSDK allows controlling of the processor to serve that purpose developed the 'C' language program. Randomly 256 challenges are applied 10000 times and observed the output on the SDK Console by setting the baud rate to 115200 bps. We got the responses from RO PUF design figure 10.

Conclusion

We successfully designed and implemented a basic 8:1 PUF on SOC-FPGA. We defined the challenges and solution to implement RO PUF on the SOC-FPGA hard Zedboard FPGA evaluation board ware. The entire experiment is carried out at room temperature. Verified Post-Route simulation to understand the PUF operating as it produces the output bit. The responses produced by the PUF on SDK were performed considerably perfectly. Our future research is to perform a Static timing analysis of this design and find out the valid CRPs of the design.

References

1. Alkatheiri MS, Zhuang Y, Korobkov M et al. An experimental study of the state-of-the-art PUFs implemented on FPGAs. in 2017 IEEE Conference on Dependable and Secure Computing, 2017: 174-180.
2. Choudhury M, Pundir N, Niamat et al. Analysis of a novel stage configurable ROPUF design. in 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017: 942-945.
3. Deng D, Hou S, Wang Z et al. Configurable Ring Oscillator PUF Using Hybrid Logic Gates. *IEEE Access* 2020; 8:161427-161437.
4. Halak B, Hu Y, Mispan MS. Area efficient configurable physical unclonable functions for FPGAs identification. 2015 *IEEE International Symposium on Circuits and Systems* (ISCAS), 2015; 946-949.
5. Herder C, Yu M-D, Koushanfar F et al. Physical unclonable functions and applications: A tutorial. *Proc. IEEE* 2014; 102(8): 1126-1141.
6. Herder C, Yu M-D, Koushanfar F et al. Physical unclonable functions and applications: A tutorial. *Proc. IEEE* 2014; 102(8): 1126-1141.
7. Kodytek F, Lórencz R. A design of ring oscillator based PUF on FPGA. 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits \& Systems, 2015: 37-42.
8. Kokila J, Ramasubramanian N. Enhanced Authentication Using Hybrid PUF with FSM for Protecting IPs of SoC FPGAs. *J Electron Test* 2019; 35(4): 543-558.
9. Kulkarni S, Vani RM, Hunagund PV. FPGA based Hardware Security for Edge Devices in Internet of Things. in 2020 5th International Conference on Communication and Electronics Systems (ICCES), 2020: 1133-1138, doi: 10.1109/ICCES48766.2020.9138013.
10. Mahalat MH, Ugale N, Shahare R et al. Design of latch based configurable ring oscillator puf targeting secure fpga. in 2018 IFIP/ IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2018; 261-266.
11. Maiti A, Casarona J, McHale L et al. A large scale characterization of RO-PUF. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2010: 94-99.
12. Maiti A, Schaumont P. Improved ring oscillator PUF: An FPGA-friendly secure primitive. *J Cryptol* 2011; 24(2): 375-397.
13. M. Masoumi and A. Dehghan. Design and implementation of a ring oscillator-based physically unclonable function on field programmable gate array to enhance electronic security. *Int J Electron Secur Digit Forensics* 2020; 12(3): 243-261.
14. Ning H, Farha F, Ullah A et al. Physical unclonable function: architectures, applications and challenges for dependable security. *IET Circuits, Devices & Syst*, 2020; 14(4): 407-424.
15. Patterson M, Zambreno J, Sabotta C et al. Ring oscillator PUF design and results 2011.
16. Schaub A, Danger J-L, Guilley S et al. An improved

analysis of reliability and entropy for delay PUFs. 2018 21st Euromicro Conference on Digital System Design (DSD), 2018: 553-560.

17. Shamsoshoara A, Korenda A, Afghah F. A survey on physical unclonable function (PUF)-based security solutions for Internet of Things. *Comput. Networks* 2020; 183: 107593.
18. Sklavos N. Securing communication devices via physical unclonable functions (PUFs),” in ISSE 2013 Securing Electronic Business Processes, Springer, 2013: 253-261.
19. Suh GE, Devadas S. Physical unclonable functions for device authentication and secret key generation. in 2007 44th ACM/IEEE Design Automation Conference, 2007: 9-14.
20. Usmani MA, Keshavarz S, Matthews E et al. Efficient PUF-based key generation in FPGAs using per-device configuration. *IEEE Trans very large scale Integr Syst* 2018; 27(2): 364-375.
21. Yan W, Chandy J. Phase Calibrated Ring Oscillator PUF Design and Application. *Computers* 2018; 7(3): 40.
22. Zhang J-L. Techniques for design and implementation of an FPGA-specific physical unclonable function. *J Comput Sci Technol* 2016; 31(1): 124-136.
23. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug835-vivado-tcl-commands.pdf
24. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug901-vivado-synthesis.pdf