Configurable Ring Oscillator PUF

ECE 559 Spring 2016

Secure and Trustworthy Hardware

John Murray, Joey Allen, Patricia Eckhart

Abstract

The operational basis of implementing a ring oscillator as physical unclonable functions (PUFs) relies on the inherent variability between identically designed components as a result of process variations. This study discusses the design and implementation of a configurable ring oscillator on an Artix 7 field programmable gate array(FPGA) and investigates the variance of uniqueness resulting from the implementation of the configurable ring oscillator PUF (CRO-PUF). The average inter-class hamming distance was used to measure the uniqueness of each challenge/response on three different Artix 7 FPGAs as well as the uniqueness of each challenge/response in each clock domain on three different Artix 7 FPGAs.

I. Introduction

The need for secure and trustworthy hardware is becoming paramount in today's society as the reliance upon digital systems to protect sensitive information, intellectual property, communication, and critical system control increases. Historically, hardware has been assumed trustworthy or unaccessible, and security as a design metric has been neglected. As society becomes more tech savvy and the Internet of Things explodes, the vulnerabilities previously thought to be obscure in hardware designs are being exposed and are becoming serious threats to privacy and security. This has lead the research and development community into the study of effective implementation methods of security primitives on physical hardware components. This study focuses primarily on the use of a CRO PUF implementation for private key encryption on an FPGA.

Physical unclonable functions (PUFs) map a set of challenges to a set of responses. In general, a challenge in the form of a known bit sequence is applied to a particular set of physical logic gates. The logic gates to be challenged should be located in a region of the device that is known to respond to stimulus in an unpredictable and uncontrollable manner. In an ideal PUF, the response to the challenge will be unique to that distinct device and reliably repeatable to the known challenge. The uniqueness of the response is the result of faults during fabrication and inherent randomness in electron configuration. The reliability depends on the stability and sensitivity of the actual device. The unique challenge response pairs can be used as private keys to aid in cryptographic encryption and hardware metering. PUF responses are typically dependent on the miniscule difference in propagation delays between identically designed signal paths. RO-PUFs are constructed using N sets of an odd number of identical inverters placed in feedback loops. The feedback loops are challenged. The frequency of the feedback oscillations are compared to determine the response to the challenge.

This study implements a configurable RO PUF ring oscillator containing the pairs three inverters such that each pair of inverters in the feedback loop will be activated or not activated based on the configuration using the select bit of a 2-1 multiplexer. The challenge will select between 16 identical implementations of the CRO. The response from the challenge will result from comparing the frequency of the two CROs. The configurability of the ring oscillators will help to eliminate any environmental interference on the FPGA, increase reliability, and help reduce bias. The uniqueness will be determined after obtaining the frequency of each of the 16 CROs in each clock domain of three different Artix 7 FPGAs.

II. Motivation

Current cryptosystems, such as RSA and AES, require the users to store their secret keys. The user is expected to store the key in a safe and private place. However, if an attacker can infiltrate the key storage, then he can have the ability to decrypt the secret messages. This has created a desire for an innovative method to protect secret encryption keys for modern cryptosystems.

One potential solution was developed by Suh et. al [1]. In their research, they introduced the Ring-Oscillator PUF (RO PUF). The RO PUF leverages the unique delays found within wires and transistors. Since the delays vary across chip-to-chip, they can be used for a method of low-cost authentication.

The RO PUF works in the following way, several ROs are created and placed on a chip. When the RO PUF receives a challenge, it will choose two ROs from the set found on the board. Based on the frequency of the chosen ROs, a response will given back to the user. This creates the challenge-response pair. If the same set of ROs are placed on a different chip, the responses will be different. These unique delays provide the device with a method to identify itself without ever having to store its "secret."

One shortcoming of the RO PUF is reliability. Reliability of a PUF expresses how consistently a response R is reproduced by a PUF. It is not trivial to reproduce the response from a PUF without errors because the process variation is not high enough to safely offset environmental noise [2]. In an RO PUF the reliability is solely dependent on the difference in frequencies of the RO pair used as the challenge. Since even small changes in the secret key can cause an avalanche effects in modern cryptosystems, a reliable key generation scheme is necessary.

To improve reliability Mahiti et. al introduced the Configurable Ring Oscillator (CRO)[2]. The CRO PUF enables the designer to create multiple instantiations of ROs inside a single configurable logic block (CLB). This allows the designer to configure the ROs chosen by the

challenge to have a maximum frequency difference, which improves the overall reliability of the PUF.

III. Implementation for Configurable Ring-Oscillator-Based PUF

Our implementation was inspired by the work found in [2]. We broke down the implementation into two sections. First we developed the configurable ring oscillator. We then worked on creating an implementation of the CRO PUF. Our final design created a CRO PUF that had 16 CROs placed inside one CLB and 120 challenge-response pairs.

a) Implementation of the Configurable Ring Oscillator:

A schematic of our configurable ring oscillator can be found in Figure 1(a). The design consists of six inverters, three multiplexers, and an enable AND. The challenge bits, C_1, C_2, C_3. are used to configure the configurable ring oscillators. Different configurations create different paths for the ring oscillator to travel. These different paths create different frequencies. Our design was implemented in VHDL and the post-synthesis design can be seen in Figure 1(b). When comparing figure 1(a) to 1(b) it can be seen that our design was correctly inferred by VHDL.

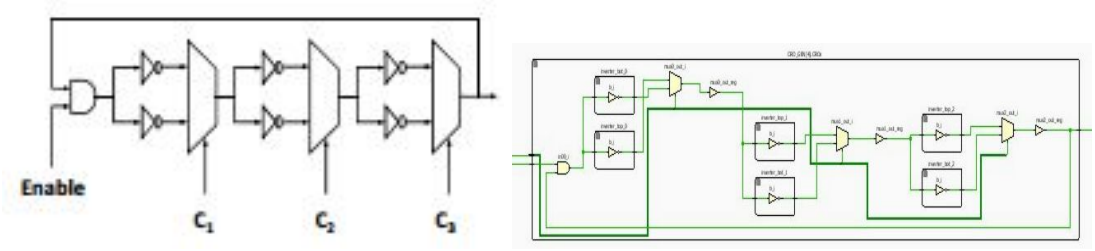
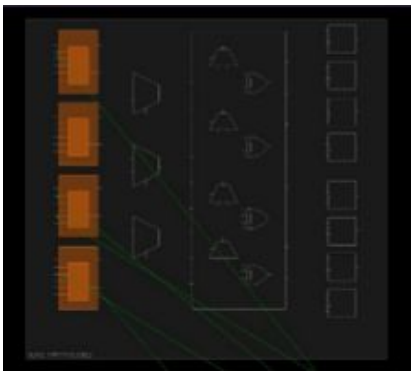


Figure 1: A comparison of the CRO schematic (b) Our design post synthesis.

The final challenge of implementing the CRO was placement. In order to implement a CRO PUF correctly, there must be minimal bias between different CROs and replication of a CRO must create almost identical results. Following [2] we placed each CRO in its own separate CLB on a Xilinx FPGA. Figure 3 shows our CRO place inside one CLB on a Artix-7 FPGA.



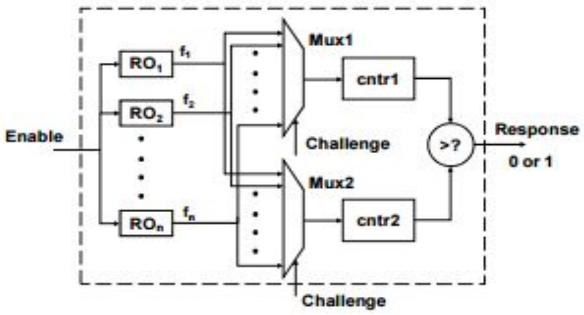


Figure 3: A CRO place inside one CLB

Figure 4: A Schematic of the CRO PUF

b) Implementation of the CRO PUF:

After successfully creating the CRO and placing it inside one CLB, our next design goal was to implement the CRO PUF, and a schematic of our design can be found in Figure 4. A CRO PUF can be implemented with *n* CROs. *n* should be based on your design goals and should take into consideration the area available and the amount of challenge-responses pairs needed to fulfill the desired application. Since our design goal was simply evaluation, we set *n* to 16, which provided us with 120 challenge-response pairs.

The next step in the CRO implementation process was creating the top and bottom paths. These paths need to be identical to reduce bias. In our design we had one 16x4 mux on each path. The inputs to these muxes were the CROs and the selection bits were the challenge. When a challenge was sent to the CRO PUF, two unique CROs would be selected. The frequency was calculated using two 32 bit counters, one for the top path and one for the bottom path. Setting the bit vector to 32 provided us with a large enough count to ensure reliability. The synthesis of our final CRO PUF design can be found in Figure 5.

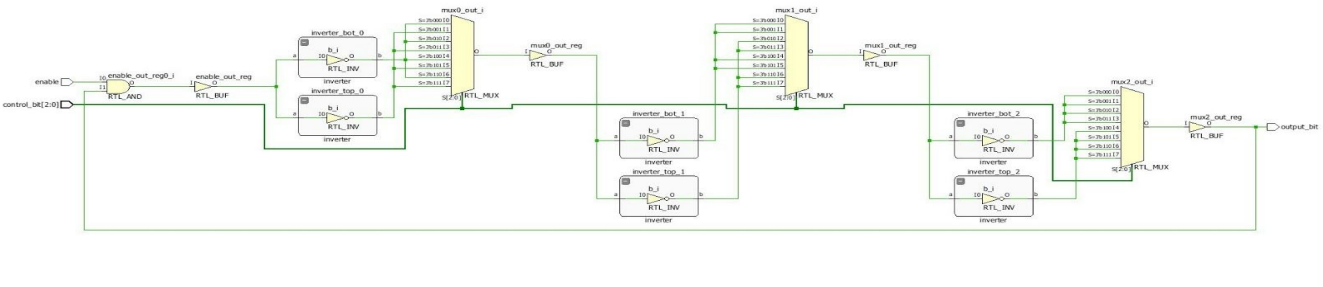


Figure 5: The synthesis of our CRO PUF.

IV. Testing

A four bit challenge was applied to the CROs located in each of the eight clock domains of the Artix 7 FPGA. The four bit challenge acts as the select bits of two 16-4 multiplexers. When the challenge is applied a counter is enabled. One multiplexer chooses the ring oscillator located at the value of the challenge and the second multiplexer inverts the challenge and choose the ring oscillator at that inverted challenge value. The multiplexers allow the signals from the two ring oscillators to enter a counter. The counter is set to rollover when it reaches 32. The first of the

two challenged CROs to cause the counter to rollover set the response bit. The first method used to display the ring oscillator data took the four most significant bits of the thirty-two bit counters and displayed their binary digits on the seven-segment displays. The values from the counter corresponding to the output of the top mux was displayed on the left four displays while the values output from the bottom mux were displayed on the right set of four displays. Observing these incrementing values showed a noticeable difference in the incrementation interval depending upon the challenges sent to the CROs. However, this interval needed to be found so that analysis could be performed on the resulting challenge response pairs.

In the initial project design, a Microblaze MCS IP core would be used to collect and analyze the data, as well as perform an automated enrollment operation for the CROs in the varying clock domains. Through the Vivado SDK, the procedures to execute these operations with Microblaze core could be written in C/C++. But, the significant overhead and previous inexperience of this approach caused the design to change in the final version. Thus, the frequency of each of the two CROs is calculated.

In order to determine the of the frequencies of the two challenged CROs, on a 1 KHz clock the current values of the thirty-two bit counters are read. At the next rising edge of this clock, the current values are recorded again and the difference between the newly read and previously obtained values is calculated. The difference is multiplied by 1000 to obtain an approximation of the frequency in hertz. These values were in the MHz range, so to visually compare the frequencies of the top and bottom mux outputs side-by-side, the data shown on the seven-segment display needed to be limited to four digits each. Thus, even though the values calculated took into account all thirty-two bits of the counters, the frequencies were truncated to the MHz range. However, the response value displayed on the LED, indicating which of the paths is the "winner" of the challenge, utilized the more precise, pre-truncated data, so there would always be a clear response result, even if the frequencies were in the same 1 MHz range.

The frequencies were recorded. Implementing the challenge bits in this manner restricted the selection of the two ring oscillators to be challenged, but this implementation exposed the frequencies of each of the 16 ring oscillators after sequencing through the first eight challenges. Once all the frequencies from each of the 16 ring oscillators was obtained it was possible to deduce the responses that would result from all combinations of challenged CROs. A second set of data was generated from the 16 frequencies, simulating the response bits that would be produced when each CRO challenged every other CRO. This produced a total of 120 responses from each clock domain of three different Artix 7 FPGAs to analyze.

V. Experimental Results

- **RQ1: What is the uniqueness of our implementation?**
- **RQ2: Is the uniqueness of our implementation comparable to the results in [2].**
- **RQ3: Can our design be extended to identify the internal clock domains of an FPGA?**

a) Uniqueness: Identifying each FPGA

The uniqueness measures how distinctly a PUF can identify the device it is located in. The uniqueness is obtained by calculating the inter-class Hamming distance using the following formula:

$$U = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \frac{HD(R_i, R_j)}{n} \times 100$$

To answer RQ1 we evaluated our CRO PUF's uniqueness across three FPGAs. Each board contained 8 CRO PUFs and each CRO PUF was placed in a different clock domain. The uniqueness was calculated for each clock domain. That is, we compared the CRO PUF challenge-response pairs on boards one, two, and three for each specific clock domain. Our results are shown in the Table 1.

Clock Domain	0	1	2	3	4	5	6	7	Avg
Uniqueness	34%	23%	3.2%	9.5%	6%	23%	17%	6.9%	15.5%

Table 1. Uniqueness results in each clock domain.

Our results show that the uniqueness is significantly different depending on the clock domain the CRO-PUF was placed on. Clock domain zero provided the best results, with a 34% uniqueness and clock domain two performed the worst with a uniqueness of only 3.2%.

One method for improving our uniqueness would be to increase the amounts of CROs. In [2] they tested their results on three different input sizes; 64, 128, 256. Their results show that as the number CROs is increased, the uniqueness score increases. Finally, to answer RQ2 we compare our results from Clock domain 0 to the results of the 64 CRO PUF in [2]. Using controlled placement of the CROs they received a uniqueness value of 34.92 which is comparable to the results of our CRO PUF found in clock domain 2. Due to the vast difference in the amount of CROs used, it is difficult to fully answer RQ2. However, our results suggests we are going in the correct direction.

1) Uniqueness: Identifying each clock domain.

Next, we were interested in verifying if a CRO PUF placed could be used to identify the different clock domains within an FPGA. There results are shown in Table 2.

FPGA	1	2	3	Avg
Uniqueness	22%	19%	20.2%	20.5%

Table 2. Uniqueness results for identifying clock domains within an FPGA

The results in Table 2 provide us with the answer to RQ3. These results suggest that our current implementation would need more CROs and stricter control placement to successfully identify internal clock domains. We leave these modifications for future work.

b) Reliability

To test for reliability it is required to alter the FPGAs surrounding environment. Two methods for doing this is to increase the amount of voltage received by the FPGA or altering the surrounding temperature. The goal of altering these environmental variables is to place the FPGA in an unstable state. Placing FPGAs in these harsh environments can lead to damage to the board. Since the FPGAs we used to test our experiments were property of the University of Tennessee and not our own, we did not feel comfortable in taking the risk of damaging the boards permanently. We leave testing the reliability of our implementation to future work when permission to potentially damage the boards is given.

VI. Challenges

The Vivado tools used to take high level hardware designs down to the register and then physical level include built in strategies and optimizations that remove and absorb any logic or elements that may be redundant or unnecessary. This is usually a useful feature, but here it posed a challenge to the implementation of the CRO PUF. During the synthesis process the software tools absorbed all the inverters in the ring oscillator into one because the chain of inverters was logically equivalent to just one. The software documentation provided information on how to remedy this issue. Attributes can be assigned to the signals in the high level design to tell the synthesis tools how to handle the signals. The “KEEP” attribute instructs the synthesis tool to keep the signal it was placed on, and that signal is then placed in the netlist.[3] By assigning the “KEEP” attribute to each of the inverter signals, the inverters remained in the design after the synthesis process was complete. However, during implementation the inverter signals were then again absorbed away. It was then discovered that the “KEEP” attribute does not force the place and route process to keep the signals. A second attribute of “DON’T TOUCH” was necessary to ensure that the signals remained throughout the place and route process.

Another challenge of implementing the hardware design using the Vivado software was specifying the physical location of each of the ring oscillators of the design. One of the intents of this study was to determine if the uniqueness and reliability metrics produced different results when identical CROs were placed in various locations across an FPGA and then also to compare how the CRO's uniqueness and reliability metrics compared with other FPGAs. To do this the placement of the CRO needed to be controlled. To meet this challenge a basic element (BEL) constraint was applied to every signal to be used in each of the 16 inverters of each of the eight clock domain of all the FPGAs to be tested. BEL is an advanced placement constraint which locks a logical symbol to a particular BEL site in a slice.[3] The BEL constraint allowed the place and route tools to be directed to place each of the inverters in the desired LUTs of the specific slices of the chosen CLBs in each clock domain. Without this constraint and setting the CRO's would have been placed in different random locations each time the design was implemented making it impossible to determine if the location of the CRO actually affected the uniqueness and reliability measures. The Figure a shows where the 16 CROs were placed in each of the 8 clock domains. Figure b shows how the inverters fit into the LUTs of the two slices of one CLB on the Artix 7.

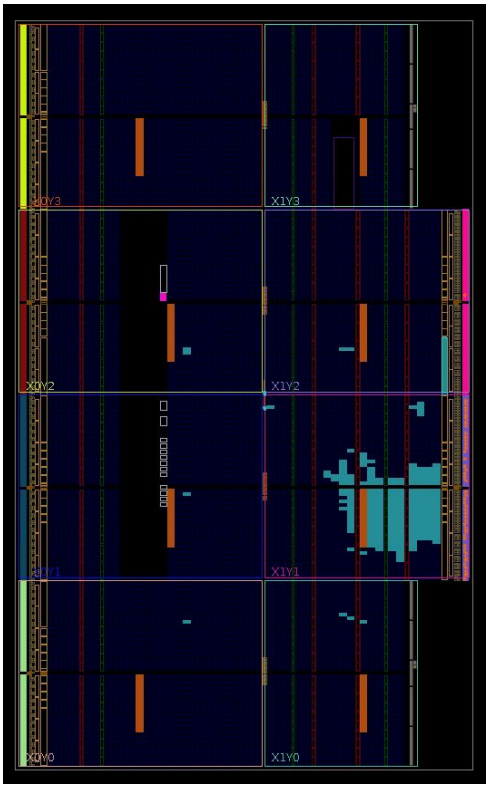


Figure a

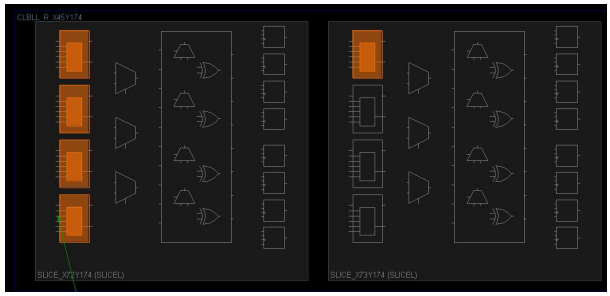
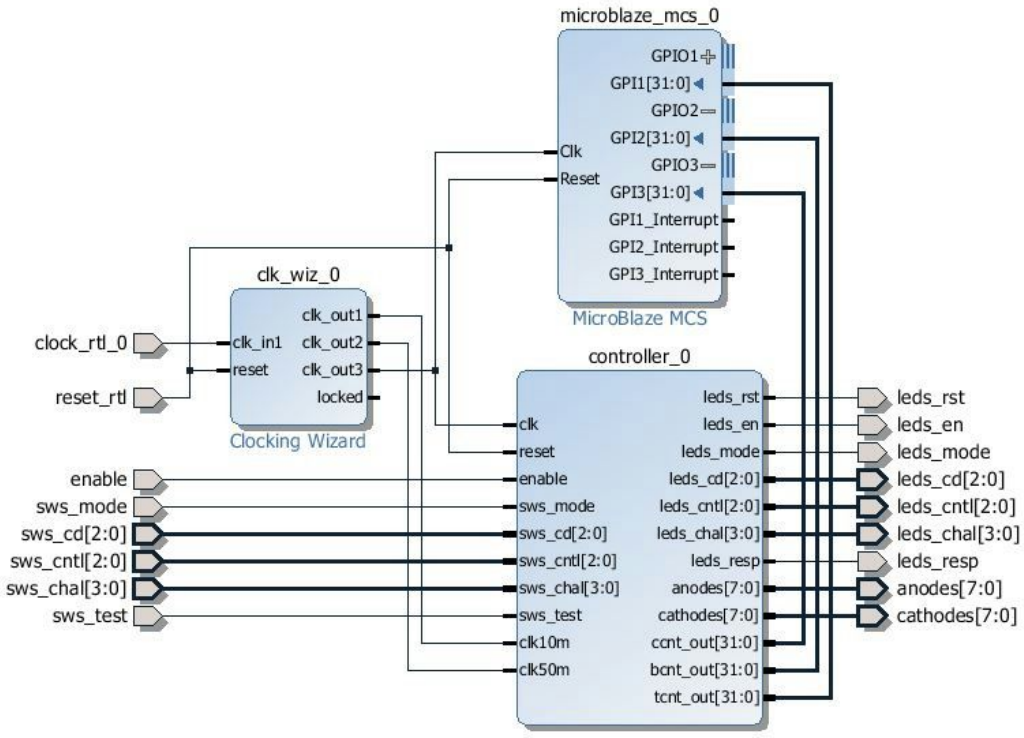


Figure b

In addition to the aforementioned challenges, utilizing the Microblaze soft processor IP in collaboration with the Vivado SDK proved to have its own set of difficulties. Originally, the

project design planned to include an automated enrollment processing capability implemented with the Microblaze, programmed in C/C++ with the SDK. However, the significant overhead needed to set up this portion of the design flow, particularly for a HDL design that already included the Microblaze and other IP cores led this approach to be abandoned. The project would require a complete restructuring. This meant tearing out the IP blocks and altering all the existing control code and components, especially the input and output signals between these components, to accommodate for this, then creating a Block Design in Vivado which emulated the already functional project. The HDL components also needed to be packaged as IP blocks so that they could be used in the Block Design. From there, Vivado supposedly could export the Hardware Definition and Board Specifications to the SDK, allowing for an application project to be created. This restructuring was performed, but Vivado continually failed to export all of the necessary information for the SDK, preventing any application project to function.



An attempted Block Design implementation

References:

[1] Suh, G. Edward, and Srinivas Devadas. "Physical unclonable functions for device authentication and secret key generation." *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007.

[2] Maiti, Abhranil, and Patrick Schaumont. "Improved ring oscillator PUF: an FPGA-friendly secure primitive." *Journal of cryptology* 24.2 (2011): 375-397.

[3] *Xilinx Constraints Guide* [Vivado]. 11.4. San Jose, California: Xilinx; 2009.

Appendix: