CPE-426 Project Report: Ring Oscillator PUF

An implementation of a physically unclonable function based on silicon manufacturing variations for the AMD Artix-7 FPGA

Dylan Sandall, Forrest Dudley, Asa Grote

10/23/2023

INTRODUCTION

Physically Unclone-able Functions, or PUFs, are a method of providing unique device signatures by utilizing the innate randomness of silicon. This randomness is found at the nanoscale structure of the device that cannot be cloned, even by the manufacturer. PUFs can take many different forms, and vary device by device. In this paper, we show how to implement a Ring Oscillator (RO) PUF into a Basys FPGA.

DESIGN

Modeled after the PUF from *A Configurable Ring-Oscillator-Based PUF for Xilinx FPGAs*, our PUF is both configurable, lightweight, and fast. The core of the design is the configurable ring oscillator (RO), which takes a 6 bit challenge and produces a variable frequency. This frequency is counted, and each of the 9 ROs are run for a set time, determined by the system clock. After this set time, the ROs are compared, with these comparisons creating an 8 bit response. The challenge and response are fed to the SHA256 module, and produce a final 128 bit response. The key advantage that our design has over the inspiration is a parallelized process for faster response time.

The biggest hurdle in the design process was the Vivado synthesis process, which simplifies seemingly unnecessary circuitry to reduce hardware footprint and power consumption. Through the use of the attributes “dont_touch”, “keep”, and “s” (save), we were able to instruct Vivado to trust the process, and retain core functionality. The control circuitry and interface logic were designed for the Basys 3 board, and utilized switches and the 7 segment display for both challenge input and PUF response.

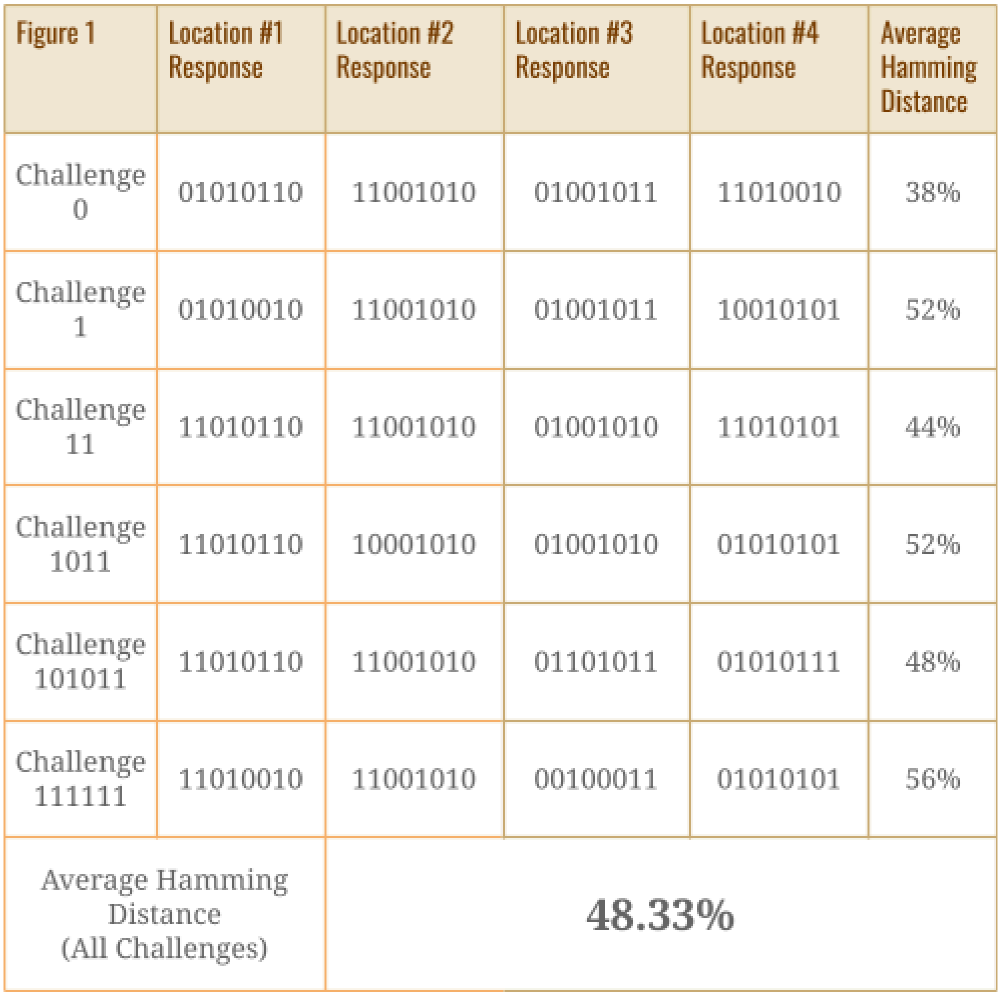

Figure 1	Location #1 Response	Location #2 Response	Location #3 Response	Location #4 Response	Average Hamming Distance
Challenge 0	01010110	11001010	01001011	11010010	38%
Challenge 1	01010010	11001010	01001011	10010101	52%
Challenge 11	11010110	11001010	01001010	11010101	44%
Challenge 1011	11010110	10001010	01001010	01010101	52%
Challenge 101011	11010110	11001010	01101011	01010111	48%
Challenge 111111	11010010	11001010	00100011	01010101	56%
Average Hamming Distance (All Challenges)		48.33%			

RESULTS

An ideal Hamming distance for a random or pseudorandom function is 50%, and for a small sample size, we have achieved 48.33%. Tests were performed at 4 different locations on a single Artix-7 FPGA die. All tests were performed at the same temperature, and although no variation in response for a given challenge was observed, there may be variation leading to inconsistent responses. Inconsistent responses could lead to falsely failing challenge requests, and flagging verified hardware as untrustworthy. Further testing would need to be performed, at varying temperatures and power supply voltages, in order to see what range of operating conditions a given PUF response pair is stable at.

ANALYSIS

1. What did you change about the provided configurable ring-oscillator?
    a. The provided configurable ring oscillator featured an 8 bit challenge, rather than 6 - this meant that one of their four blocks could be removed from our design. We also made some changes in the overall layout, moving the enable mux into it's own slice, and giving each configurable RO it's own counter, to parallelize the frequency measurement process.
2. How many ROs are necessary?
    a. To minimize the effect of the design, and maximize the effect of variation in silicon, 9 ring oscillators were necessary, so that each pairing is "more fair". While 8 discrete comparisons could be made between a minimum of 5 ring oscillators, using 9 was a better approach.
3. Why will a 50 MHz clock work for this design?
    a. As long as the clock speed is stable, and somewhat comparable (within an order of magnitude or so) to the ring oscillator frequencies, it should work. The main concern is that the primary clock variation doesn't overshadow the variation between different ring oscillators.
4. What did you decide to use for the max value of std_counter?
    a. The max value of the primary counter needed to be large enough that the ring oscillators had enough time to complete a sufficient amount of cycles. If the sampled cycles were too low, ring oscillators with similar frequencies would not respond to the challenge consistently, with comparisons being essentially random. Another factor was the initial clock divider we added to the counter for each ring oscillator. The purpose of this clock divider is to remove any slight inconsistency for a given ring oscillator in a given configuration - this made verification simpler.
5. How does this value impact the PUF?
    a. The primary counter length affects the time each RO will take to run, and in the initial design, each ring oscillator will need to be run sequentially. In our design, the ring oscillators are parallelized, and all ring oscillators can be run in the time it takes a single RO to complete.
6. Why would we want to know if the challenge response is complete?
    a. Sampling the challenge response too early could produce inconsistent results, and a signal that indicates completion is valuable for controlling modules such as the display and SHA128 hash function.

7. What is the purpose of the SHA256 module (why might we want to hash the challenge concatenated with the response)?
    a. Hashing the challenge and the response makes the final hash more difficult to crack, as it is based on a longer and more varied seed. Hashing at all is critical for obscuring the challenge and response, and obfuscating the PUF further from prying eyes and potential attackers.
8. Are the challenge-response pairs the same for each location on a given chip? Should they be?
    a. No, this means that the silicon variation is effecting the PUF as intended. Each single location on a single chip should however return the same challenge-response pairs.
9. What are the intra-board and inter-board hamming distances? What are ideal?
    a. For a single board, across different locations, we have achieved 48.33%. Testing across different boards returned a similar number, hovering around 50%, plus or minus 5%.
    b. An ideal Hamming distance for a random or pseudorandom function is 50%.
10. Discuss anything of note about your particular implementation or results.
    a. The key advantage that our design has over the inspiration is a parallelized process for faster response time. Our results are likely not effected by this change, but our results are quite good, considering a target hamming distance of 50%.
11. How difficult would it be to expand the number of challenge bits and the bits of the response? Describe the process and any considerations needed to ensure the PUF functions correctly.
    a. It would not be difficult to do either. Increasing challenge bits would just need more muxes and delay paths within a CRO, and increasing response bits would just require more CROs within the PUF.
12. How might variations in temperature and voltage of the chip effect PUF? Think both raw entropy and the result of the hash. What are some methods which could mitigate the effect?
    a. Temperature and voltage fluctuations would decrease the stability of the PUF, meaning one given chip might not always respond the same. This is bad for reliability and security verification. Increasing the initial clock divider (or increasing the timer counter max) would take a better average of the frequency, mitigating this to some extent. Perhaps a temperature measurement could be taken on the chip, and rather than a

challenge-response pair, a challenge+temperature/response tuplet could be used, further varying the input options. Each chip might respond to temperature differently, giving another source of variation for the response to depend on.

13. You might have noticed in the paper and in our implementation the hamming distance between adjacent challenges is fairly minimal (15% in Xin et. Al). What might cause this? Can you think of a way of changing how we use the counter output to increase the hamming distance? Hint: This might mean reducing the number of usable response bits in the counter.
    a. Perhaps the lower bits of each CRO's count could be discarded, and the remaining higher bits could be scrambled. This would mean that slight variations in frequency would have a higher effect on the RO comparison, and large variations in frequency would have a smaller effect. This could however reduce stability, and would need to be fine-tuned. Another option would be to make each delay path on the CRO more distinct, hopefully increasing the effect of silicon variation on the CRO's frequency for a given challenge.
14. How could this implementation be modified on a board so that it could be verified from an external source and ensure freshness? (Don't think about the BASYS 3 board, instead consider the fundamental structure of the device, including generic inputs and outputs)
    a. One possible implementation is to send a timestamp along with the challenge when a PUF request is sent, and this timestamp would be fed into the SHA256 module along with the PUF response. Then, an attacker could observe the PUF module for as long as they want, and the challenge/response pairs that they observe would not be valid after the timestamp expires.