POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

TERO PUF Implementation on Xilinx Artix-7 FPGA

Project Report for Embedded Systems Course

Aspesi Andrea, 962608
Putortì Alessandro, 953258

Professors:
Prof. Fornaciari William
Prof. Zoni Davide
Prof. Galimberti Andrea

Academic year:
2021-2022

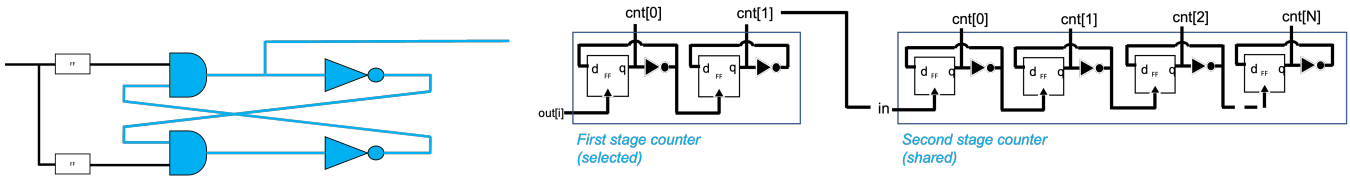
Abstract: Physical Unclonable Functions (PUFs) are becoming important in recent years for various purposes such as Device Identification, True Random Number Generators, Cryptographic Key Generators, IP Protection. Different ways of realizing PUFs have been proposed in literature, exploiting process variations occurring inside chips. Among them, some can be efficiently implemented in FPGAs: in our project, after analyzing the state of the art, we chose to implement a Transient Effect Ring Oscillator (TERO) PUF. TERO PUF offers high resistance to surrounding logic influence and temperature changes, providing at the same time high uniqueness and reliability. As target, we chose Basys3 boards from Digilent, shipped with Xilinx Artix-7 XC7-A35T.

1. Introduction

A wide variety of Physical Unclonable Functions have been proposed in literature, exploiting some random and unpredictable properties of electronic chips to generate unique and reliable responses required by specific application like chip identification, cryptographic key generation, logic obfuscation and IP protection. Among the ones which are suitable to be implemented on FPGA, Transient Effect Ring Oscillator (TERO) PUFs have been proved to have very good properties in terms of uniqueness, reliability and resistance to environmental changes.

In [4] by A.Wild et al., an in-depth analysis of RO, Loop and TERO PUFs has been performed comparing the responses coming from 100 Basys3 boards on which 1280 PUFs were implemented. Our project consists in a further analysis of the data acquired by the authors of the paper, followed by a SystemVerilog implementation of the **TERO PUF for device identification purposes**, targeting both Basys3 and CmodA7-35t boards (both mounting XC7-A35T FPGA), tested at 100Mhz.

In order to design and test the PUF modules Xilinx Vivado 2020.2 was used. The analysis of the PUF responses and the generation of the IDs has been performed through some dedicated Python scripts and jupyter notebooks.



(a) TERO loop structure (b) 2-stages counter

Figure 1: Structure of TERO loop and 2-stages counter

2. Design

2.1. Architecture

The TERO PUF is composed of a variable number of TERO loops. Each of them is based on two AND gates and two inverters (see *Figure 1a*).
The two inverters are cross-coupled and brought into an unstable state via AND gates. While the loop tries to resolve the unstable state, it oscillates for a short time. The number of oscillation of each loop is counted using a two-stages counter (see *Figure 1b*). Combining the outcomes of the loops with a specific algorithm (see *Section 3*), we can obtain the overall response of the TERO PUF (ID of the FPGA).
In an Artix-7 Slice we can put two TERO loops (see *Figure 2*), differing for the internal routing. These TERO slices are arranged in four adjacent columns, placed in two different positions in the FPGA (see *Figure 3*).

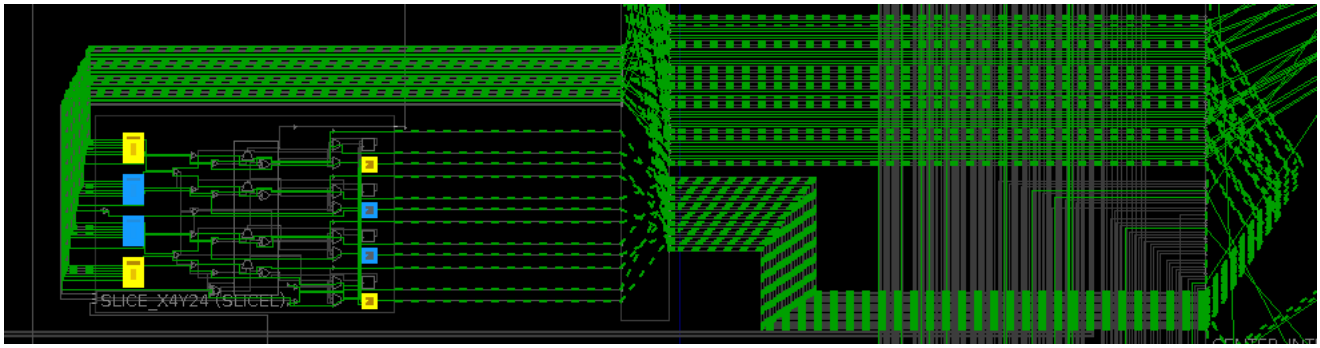


Figure 2: Connection of the two TERO loops inside one Slice

In this way, we obtain eight different types of loops (two for the differences at slice-instance, times four due to the column arrangement). Each TERO loop has an enable signal that starts the oscillation when asserted. These signals are controlled by an FSM during an evaluation cycle, where each loop is activated and its oscillation measured and averaged. The evaluation time and the number of measurement used for the average are design parameters (see *Subsection 5.1*).

2.2. Communication

The communication with a PC is handled by another FSM, which controls the UART module, the FIFO storing the loop responses and the challenge memory.
The protocol is the following:

1. The PC sends a single byte request to start the interaction.
2. The PUF replies with a single byte known identifier and waits for a challenge byte.
3. The PC sends a challenge number encoded in a single byte.
4. The PUF starts the evaluation of the TERO loops, as specified in details in *Section 5*. Each intermediate result is stored in a FIFO queue.
5. When the evaluation is completed, the FIFO data is sent to the PC all-in-once.
6. The communication can start again from 1.

To handle the UART communication we exploited open-source Verilog designs found in the net[1], after verifying them.

[1] can be found at https://github.com/ben-marshall/uart

2.3. Challenges

The challenge number sent to the PUF **could** be used to avoid the evaluation of all the loops. The needed frequencies [2] depend on the algorithm used to generate the final response, during the post-processing on the PC. At the moment, we decided to expose all the frequencies in order to be able to experiment with various algorithms.

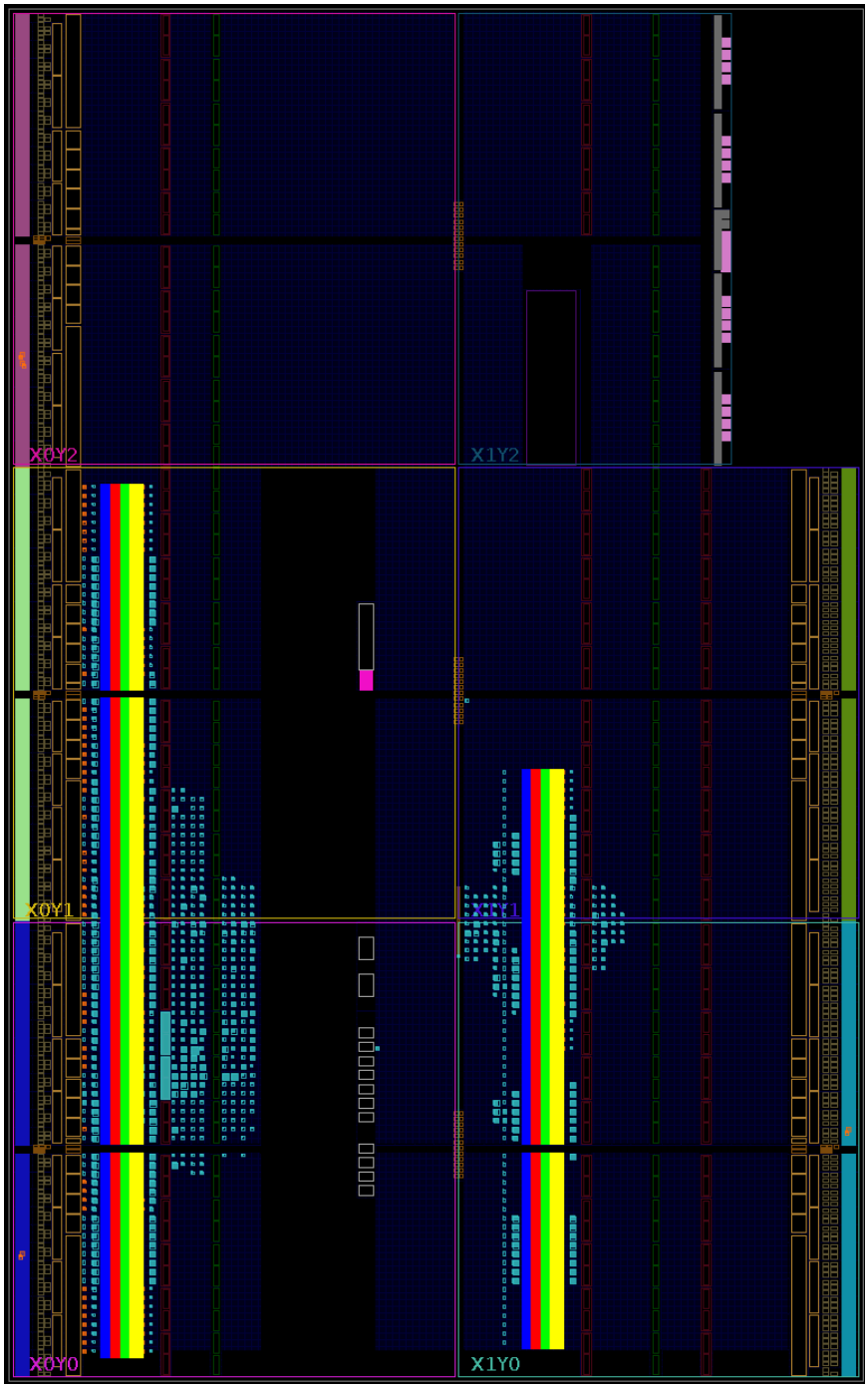


Figure 3: The TERO loops in different columns have different position in relation to the switch matrix, leading to different loop characteristics.

[2] In this report we use the word **frequency** as synonym of number of oscillations performed by a TERO loop during an evaluation.

3. ID Generation

As anticipated in *Section 2*, the generation of the ID from the PUF responses can be made in different ways. This choice, naturally, influences the way the frequencies must be exposed in output (which responses to expose and in which order, given the input challenge). In order to compare different algorithms we decided to output all the 1280 frequencies, regardless of the challenge. Before computing the actual IDs, we split the responses in 80 batches of 16 frequencies each (as in the reference paper [4]). Each batch contains frequencies coming from loops of the same type. In the following, some further details about the main algorithms.

3.1. 2Compare

2Compare algorithm takes two frequencies f_i and f_j from each batch, and returns 1 if $f_i > f_j$, else 0. Thus, the resulting ID has 80 bits, one for each batch, for a total of 120 unique responses.

Below the pseudo-algorithm:

Algorithm 1 2Compare Algorithm

Require: $i, j \in [1, 16]$, $frequencies(batchIndex, freq)$
1: **for** $batchIndex = 1$ *to* 80 **do**
2: $\quad ID[batchIndex] = frequencies(batchIndex, i) > frequencies(batchIndex, j)$
3: **end for**
4: **return** ID

After our analysis we stuck to this algorithm since it's easy to be implemented and the resulting uniqueness and reliability are quite good.

3.2. 2Comp Neighbor

2Compare neighbor is a variation of the previous one. Instead of taking two frequencies from each batch, it compares the "neighbor" frequencies of the same batch. We decided not to use this algorithm since, with 16 frequencies batches, the number of bits for each ID is 8, way too low to identify a sufficient number of devices. Increasing the number of frequencies per batch partially solves this problem, since (for the same number of TERO instances) the number of batches (and thus of challenges) is consequently reduced.

3.3. Lehmer-Gray

The response generation through Lehmer-Gray encoding is the best in terms of extracted entropy, allowing to extract 49 bits from a 16 bits batch. It was previously used in [3], based on the method proposed by Yin and Qu in [5] to extract maximum entropy.
The algorithm consists in computing the Lehmer coefficients s_i for each frequency of a batch, with the following formula:

$$s_i = |\{j > i : freq(j) < freq(i)\}| \tag{1}$$

After that, the obtained coefficients are Gray encoded, and some of the bits of the encoded coefficient are taken as a response.
We did not choose this algorithm since the choice of the bits to take is not trivial, and depends on the particular implementation.

4. Differences with Reference implementation

Though the composition and placement of the TERO loops are the same compared to the reference paper [4], other design choices differ.

4.1. Averaging of TERO loop oscillations

We implemented our version of the averaging step in the evaluation cycle, since it was missing in the original implementation code. Multiple measurements of each loop's oscillations are summed using the same shared counter (to optimize resources), with a number of bits extended for this purpose. At the end, a shift is performed to get the averaged measurement.

4.2. Finite State Machines

During our revision of the reference implementation, we found zero to little comments and lots of unused code. We decided to write from scratch the two FSMs (both present also in the original implementation), and changed completely the one dedicated to the communication.

4.3. Routing

Most of the routes were conserved. We rewrote the script in a more readable way, and fixed some warnings about combinatorial loops that prevented the generation of the Bitstream in newer versions of Vivado.

4.4. Memorizing oscillations

We used a Xilinx FIFO generator IP, in First-Word-Falls-Through configuration to generate the FIFO that memorizes the final measurements of each TERO loop. It also converts the width of each measurement in single byte words, ready to be sent via UART without further transformations. The FIFO's *empty* signal was used instead of a dedicated counter to detect the end of the data transfer, in order to avoid unrequired additional logic. Some internal registers used to buffer signals were removed because not needed.

5. Results and Conclusion

After realizing the design, we transferred it to our two Basys3 boards.
With the help of a utility script, we collected frequencies in various configurations. In particular, we verified the best number of repetitions and evaluation time, so as to have a high reliability, but waiting a reasonable amount of time for the PUF response.

- TERO loops are expected to oscillate for a short period of time, then settle in a stable state. Using a too short evaluation time truncates the oscillation measurement: although the resulting read frequency values display a unique pattern, we can better capture the uniqueness if we wait for most of the loops to stop oscillating. At the same time, a high evaluation time can significantly slow down the process, and make the PUF unusable.
- Repeating the measurement several times (and averaging the values) is necessary as the oscillation frequency of a TERO loop depends, apart from the instance-specific process variations, on environmental noise (e.g., due to temperature or supply voltage differences) and random temporal noise (e.g., temporal fluctuations in the power supply). The latter type is not constant and can be averaged out. This approach has been used several times, as it has been proved to increase the reliability (as we verified via experiments).

To compute reliability and uniqueness we used the formulas reported in [1], also used in the original TERO design [2].

5.1. Evaluation time and Repetitions

Using the data we collected, we got to the same conclusions of the paper [4]: using more than 2^5 clock cycles for evaluation does not produce a distinguishable improvement in reliability nor uniqueness (as this latter is little influenced by evaluation time). Instead, increasing the number of repetition from 1 to 2^{12} adds an extra 5% to the reliability indicator (but slows the response time of the PUF).
Using an evaluation time of 2^5 clock cycles and 2^{12} repetitions is the best compromise between reliability and response time of the PUF.

5.2. Reliability

To calculate the reliability of our implementation, we used 100 measurements for each different configuration, comparing the responses to the same challenge with the following formula:

$$Reliability_i = 100\% * (1 - \frac{1}{s}\sum_{t=1}^{s}\frac{HD(Xr_i, X_{i,t})}{n}) \tag{2}$$

were i is the i-th challenge (among the 120 available), HD is the Hamming Distance, Xr_i is the first response measured for that challenge, $X_{i,t}$ represents each of the 100 responses generated for the i-th challenge, $s = 100$ are the measurements taken and $n = 80$ is the number of bits of each response.

Using the chosen configuration (evaluation = 2^5, repetitions = 2^{12}), we got a mean reliability of 97.4% for the first PUF ($TERO\_AA$), 96.9% for the second PUF ($TERO\_AP$).
The ideal value for the reliability is 100% (otherwise stated: for each evaluation of the same PUF, fixed the challenge, we would like the same exact response bitwise)

5.3. Uniqueness

With the same 100 measurements for the two PUFs of before, we tried to estimate the uniqueness of the response for each given challenge. Of course to get to a more accurate result, lots of PUFs are required.
The formula used for the uniqueness is the following:

$$Uniqueness_i = 100\% * \frac{k}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\frac{HD(X_i, X_j)}{n} \tag{3}$$

were i is the i-th challenge (among the 120 available), $k = 2$ is the number of PUFs used, HD is the Hamming Distance, X_i is the response of the challenge for the i-th PUF, X_j is the response of the challenge for the j-th

PUF and $n = 80$ is the number of bit of each response.

Using the chosen configuration (evaluation = 2^5, repetitions = 2^{12}), we got a mean uniqueness of 50.25%. The ideal value is 50% (otherwise stated: the probability of a bit flip between two responses of different PUFs, fixed the challenge, is 50% - the toss of a coin).

5.4. Conclusion

Seen such results, we consider the implementation of the TERO PUF successful, and in line with the results of the paper.
Further work can be done to use the challenge directly in the FPGA to send only the needed frequency measurements, and in this way avoid exposing the whole CRP (i.e. Challenge Response) space. Little effort is required to achieve this with our implementation, since a dedicated block has already been designed for this purpose.

References

[1] N. Nalla Anandakumar, Mohammad S. Hashmi, and Mark Tehranipoor. Fpga-based physical unclonable functions: A comprehensive overview of theory and architectures. *Integration*, 81:175–194, 2021.

[2] Lilian Bossuet, Xuan Thuy Ngo, Zouha Cherif, and Viktor Fischer. A puf based on a transient effect ring oscillator and insensitive to locking phenomenon. *IEEE Transactions on Emerging Topics in Computing*, 2(1):30–36, 2014.

[3] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 302–319, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[4] Alexander Wild, Georg T. Becker, and Tim Güneysu. A fair and comprehensive large-scale analysis of oscillation-based pufs for fpgas. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7, 2017.

[5] Chi-En Daniel Yin and Gang Qu. Lisa: Maximizing ro puf's secret extraction. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 100–105, 2010.