Ring-Oscillator PUF on FPGA

Joshua Green
Rutgers University (Electrical and Computer Engineering)

***Abstract*—Physically Unclonable Functions (PUFs) are functions that generate unique and repeatable outputs to challenges inputs. This project focuses on implementing a Ring-Oscillator PUF on an Field-Programmable Gate Array (FPGA). Ring-Oscillators are instantiated with VHDL, and the frequency of different Ring-Oscillators is used to generate unpredictable sequences of numbers.**

***Index Terms*—FPGA, PUF, Ring oscillator**

I. Introduction

Physically Unclonable Functions (PUFs) are hardware-based functions that when given an input, will generate a unique, random, and repeatable output. PUFs leverage process variation that occurs during manufacturing to generate a corresponding output to an input. Because of this fact, each device will generate a different output to the same input. As a result, PUFs can be used for verification since no two devices will have the same response to a challenge [1].

There are three main methods of PUF implementation: Arbiter PUFs, SRAM PUFs, and Ring-Oscillator PUFs. Arbiter PUFs utilize race conditions caused by cascaded multiplexers to generate unique outputs to a challenge. SRAM PUFs read uninitialized SRAM to generate a starting seed that can be used for the PUF [2]. Ring oscillator PUFs compare ring oscillator frequencies to generate PUF outputs. Although the method for each of these PUFs is different, they all utilize some inherent random variance at the hardware level for the PUF [3].

The goal of this project is to design, simulate, and implement a ring oscillator PUF on a Field-Programmable Gate Array (FPGA). A Ring-Oscillator PUF was chosen as the PUF to be implemented because of VHDL synthesis behavior. When synthesizing, race conditions are synthesized and Block RAM is automatically pulled low on startup. However, combinational loops can be more easily utilized without being synthesized out.

II. Technical Background

A. Ring-Oscillator PUF Behavior

The Ring-Oscillator PUF compares the frequency of two ring oscillators to generate a corresponding challenge-response pair. A ring oscillator can be created using an odd number of cascaded inverters to generate a periodic output signal that will oscillate. This periodic signal will oscillate at a certain frequency, which can be used for PUF generation.

In practice, many different ring oscillators will be implemented. The more ring oscillators that are included in the design, the higher the variability of the PUF will be. A challenge to the PUF will pick two different ring oscillators, at which point the frequency of the two selected ring oscillators can be compared. Because each ring oscillator will oscillate at different frequencies due to process variation, each pair of selected ring oscillators will generate a different random result, regardless of the output from the previous challenge [4].

Fig. 1. Ring Oscillator Schematic

B. Field-Programmable Gate Arrays

Field-Programmable Gate Arrays are programmable devices that can be used to implement custom digital circuits, such as a ring oscillator PUF. FPGAs utilize Combinational Logic Blocks (CLBs) to implement digital logic. The CLB consists of a Look-Up Table for combinational logic, a flip-flop for sequential logic, and a mux to choose whether a CLB should be used for combinational or sequential logic.

III. System Setup and Implementation

A. Ring-Oscillator PUF Design

This project utilizes 32 ring oscillators divided into two groups of 16. *Figure 1* shows the ring oscillator consisting of 9 cascaded inverters. Two 16-input multiplexers are used to select one ring oscillator from each bank of 16. An arbiter is then connected to the outputs of the two multiplexers. Because each multiplexer uses a 4-bit select signal, the challenge is 8 bits long. This allows for 256 unique challenges. An arbiter is connected to the two outputs from the multiplexers. The arbiter compares the two selected ring-oscillator frequencies to generate a response to a challenge. The arbiter times how long it takes each selected ring oscillator to oscillate 65,535 times, and then gets the difference between the two times.

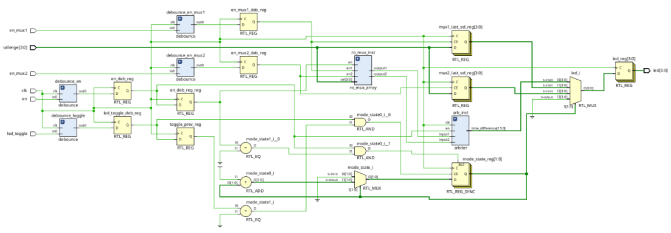

Fig. 2. PUF Top Level Schematic

That difference can then be used for the PUF output. *Figure 2* shows the schematic for the top level PUF implementation

B. Implementation

The PUF design is implemented in VHDL. A hierarchal design is used to simplify any troubleshooting. Simulations were run to validate the functionality of the ring oscillator, multiplexers, and arbiter, and a top level was created for synthesis. All design, testing, and implementation was done with the Xilinx Vivado Design Suite.

The target was the Zybo Z7. This FPGA development board from Digilent is equipped with a Zynq 7000 FPGA, four buttons, four LEDs, and four switches. One of the buttons is an enable for the arbiter, and another button serves as a toggle for the information the LEDs display. The other two buttons are used to load a select signal to each multiplexer. A select signal value can be chosen by using the four switches, and pressing one of the last two buttons will set a multiplexer to have that as a select signal. To give a challenge input, a user would set the four switches, press one of the buttons to change a multiplexer select signal, set the switches again, and press the other button to set the other multiplexer. The LEDs show the output of the PUF. The output of the PUF is the four least significant bits of the difference between the two ring oscillators. A constraint file was modified to ignore combinational loops to prevent the ring oscillators from being optimized out by the synthesis tool, and then a bitstream was generated.

C. Testing

The implementation was tested with simulations and by physically running the design on the Zybo Z7. For the physical testing, the FPGA was through JTAG, at which point it could be observed that different challenge inputs were generating unique outputs. For simulations where timing delay was necessary, a small, random delays were directly added into the entity. This delay is ignored by the synthesis tool, but the delay will be shown by the synthesis tool, allowing for better simulations of the ring oscillator specifically.

IV. Evaluation

The Ring Oscillator PUF was tested with simulations and physically on the Zybo Z7.

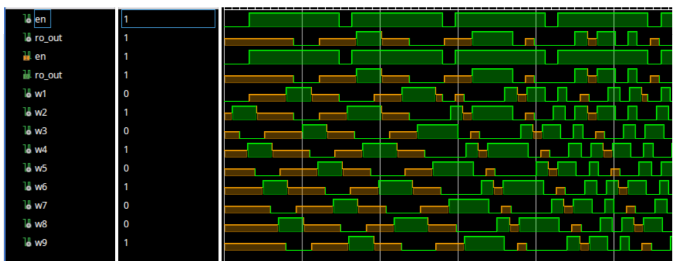

Fig. 3. Simulation of Ring Oscillator with predefined timing delay

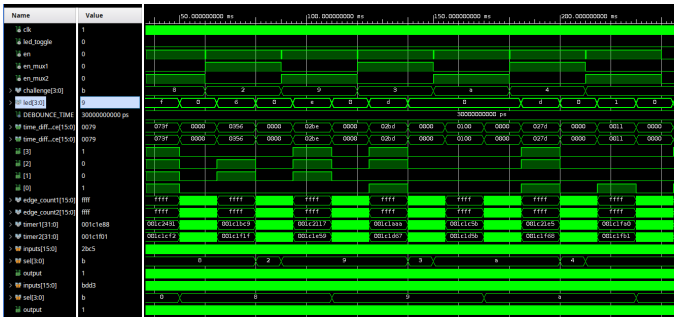

Fig. 4. Simulation of Top Level PUF

A. Simulation

For the ring-oscillators, random delays were added to mimic propagation delays caused by process variation. This was necessary because the simulation tool assumes zero delay by default. In other words, a simulation without the delay would always show zero as a response because there wouldn't be any difference in ring oscillator frequency. *Figure 3* demonstrates the ring oscillator with timing delays implemented. For more advanced simulations, these delays were added with a for loop that would increment a delay to be used for the simulation. When specifying explicit timings in VHDL, the synthesis tool will ignore them.

The simulation of the top level PUF is then run for an extended period of time to demonstrate its effectiveness. *Figure 4* shows the simulation for the Top Level PUF implementation. From the simulation is observed to be unique for different challenges.

B. Physical Implementation

After simulating the design and validating its functionality, the FPGA is then programmed and physically tested to confirm the results of the simulation. There is some slight variation observed in the challenge response between the simulation and physical implementation because of other process variation delays that weren't taken into account during simulation. Despite this, the challenge response pairs are still unique.

V. Source Code Description

The source code for the project is hosted in a Github Repository found at https://github.com/jgreen124/HWSecFinalProjectPUF.

If in possession of a Zybo Z7-10, the bitstream can be used to program the FPGA right away with no modifications necessary. When using a Zybo Z7-10, the left-most button is used as the select for the first multiplexer, the second button from the left is used as the select for the right multiplexer, and the right-most button activates the PUF. The last button toggles the LEDs to switch between the PUF output and the two multiplexer select signals. Pressing the enable button activates the arbiter, allowing for a unique and random output to be displayed. To test the PUF on a different FPGA, the constraints file will need to be modified accordingly, and "puf.vhd" will need to be synthesized again.

There are two sets of files provided. One set of files is used for synthesis and the other set is used for simulation. The files ending in ”test.vhd” are used for simulation, and the other files are used for synthesis. The files for testing include functions for generating delays, which make them unusable for synthesis. It is sufficient to set the desired simulation as the top and run it to see its outputs. The simulations don't need to be modified at all in order for them to run.

REFERENCES

[1] S. S. Kumar, M. Majzoobi and F. Koushanfar, “Lightweight secure PUFs: Possibilities and challenges,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2145–2158, Nov. 2018, doi: 10.1109/TCAD.2018.2846658.

[2] R. Maes, P. Tuyls and I. Verbauwhede, “A soft decision helper data algorithm for SRAM PUFs,” in *IEEE International Symposium on Information Theory*, 2009, pp. 2101–2105, doi: 10.1109/ISIT.2009.5205855.

[3] S. Kataria, K. Basu and S. Ghosh, “Hardware-based device authentication using PUFs: A survey,” in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 4, pp. 1637–1650, April 2023, doi: 10.1109/TCSI.2023.3238988.

[4] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk and S. Devadas, “A technique to build a secret key in integrated circuits for identification and authentication applications,” in *Proceedings of the IEEE Symposium on VLSI Circuits*, 2004, pp. 176–179, doi: 10.1145/1278480.1278484.