Lab 01: PUF Evaluation

Jasmin Kuar and Alec Braynen
University of South Florida
Tampa, FL 33620

I. Introduction

Integrated circuits (ICs), despite the standardization in their fabrication, are not 100% physically identical. These physical variations lead to differences in the electrical and transistor properties of the IC. In this work, we perform Physically Un-clonable Function (PUF) evaluations on two CMOD S7 FPGAs to capture their physical differences in some quantitative ways. More specifically, we implement an Arbiter-PUF (APUF), which is a circuit with two paths, created by a series of multiplexors that has an output of 0 or 1 which corresponds to the path that is fastest. We represent this physical variance in the CMOD S7 FPGAs, by giving each the same input vector, called a challenge, and calculating the Hamming Distance (HD) of their output. This gives us the Inter-chip HD which represents quantitatively, physical differences in the different FPGA devices in the same FPGA family. Additionally, we calculate the HD within each FPGA as well by feeding as input, the challenge multiple times, and calculating the HD of the outputs of the multiple challenges. This gives us the Intra-chip HD.

II. Reading Check

Question 1: What is the purpose of the arbiter in the APUF?

Answer: The purpose of the arbiter is to arbitrate. The purpose of an ”arbiter” in any system would probably be to arbitrate. Therefore in the PUF the purpose is for it to decide which path won the race condition (This is implemented via a DFF).

Question 2: Why might the PUF response bits change when different challenges are applied to the same circuit on the same FPGA?

Answer: The response bits change since the FPGA is being given different challenges.

Question 3: Why might the PUF response bits change when the same challenges are applied to the same circuit on different FPGAs?

Answer: The response bits may change due to 1) variance in the physical condition of the IC when each challenge was ran (i.e., temperature or the minute amount of current or voltage being applied, 2) physical degradation of the chip over time etc.

III. Methods

To implement this experiment, we use two CMOD S7 FPGAs, a virtual machine running Ubuntu OS, Dilgent Adept 2 software, Jupyter Notebooks, the programming language Python, and the libraries, NumPy, Matplotlib and SciPy. Broadly, our experiment is done by implementing the APUF design on the FPGAs, sending a 64 bit challenge vector to each, recording the instructions with variables in Python and then using NumPy, MatPlotLib to calculate the Hamming Distance in the responses and to plot the data.

A. Hardware Setup

The hardware setup consists of two CMOD S7 FPGA boards connected via a micro-USB to USB cable to an Ubuntu OS running inside a virtual machine on a Windows computer. This allows us to communicate with the FPGA via the USB to UART conversion chip on the development board which interfaces with the FPGA itself. For this lab, two PUF implementations are used - ”APUF_CmodS7_01.bit” and ”APUF_CmodS7_02.bit”, and the responses are recorded for 50,000 challenges. The PySerial python library is used to configure the FPGAs to a particular APUF BITFILE, send the challenge vectors to the FPGAs, and to read the respective responses of the FPGAs for the given challenges. Refer to Figure 1 for the image of the hardware,

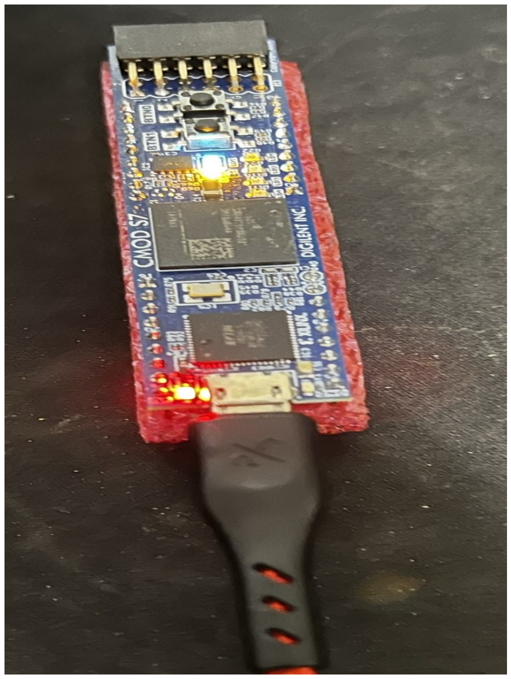

Fig. 1. Image of FPGA and USB cable

B. Software Setup

The software setup for the experiment consists of a virtual machine running Ubuntu OS on virtualbox. The Python code is implemented via Jupyter notebooks on this virtual machine. Additionally, the software interfacing with the FPGA is the Diligent Adept 2 software.

The directory structure of the files in the Jupyter notebook is structured as follows:

```
PHS-Lab-01
├─ Challenges
│  ├─ challenges_05k.npz
│  ├─ challenges_10k.npz
│  └─ challenges_50k.npz
├─ Code
│  ├─ Lab-01-Acquisition.ipynb
│  └─ Lab-01-Analysis.ipynb
├─ Figures
│  ├─ avg_hd_bitfile1.pdf
│  ├─ avg_hd_bitfile1.png
│  ├─ avg_hd_bitfile2.pdf
│  ├─ avg_hd_bitfile2.png
│  ├─ std_bitfile1.pdf
│  ├─ std_bitfile1.png
│  ├─ std_bitfile2.pdf
│  └─ std_bitfile2.png
├─ Hardware
│  ├─ APUF_CmodS7_01.bit
│  └─ APUF_CmodS7_02.bit
└─ Responses
   ├─ fpga0_resp1_bitfile1_alec_50k.npz
   ├─ fpga0_resp1_bitfile1_jasmin_50k.npz
   ├─ fpga0_resp1_bitfile2_alec_50k.npz
   ├─ fpga0_resp1_bitfile2_jasmin_50k.npz
   ├─ fpga0_resp2_bitfile1_alec_50k.npz
   ├─ fpga0_resp2_bitfile1_jasmin_50k.npz
   ├─ fpga0_resp2_bitfile2_alec_50k.npz
   ├─ fpga0_resp2_bitfile2_jasmin_50k.npz
   ├─ fpga0_resp3_bitfile1_alec_50k.npz
   ├─ fpga0_resp3_bitfile1_jasmin_50k.npz
   ├─ fpga0_resp3_bitfile2_alec_50k.npz
   └─ fpga0_resp3_bitfile2_jasmin_50k.npz
```

1) The "Challenges" directory holds the challenge vectors used as inputs
2) The "Code" directory hold the jupyter notebook files where the python code is executed
3) The "Figures" directory holds the generated figures of the analyzed data
4) The "Hardware" directory holds the two APUF bitfiles used to program the FPGA circuit
5) The "Response" directory holds the responses received from the FPGAs as output

C. Experimental Procedure

Part A: Set up FPGA and the Challenge

Our procedure for Part A will take 3 steps:

Step 1: Import relevant Python libraries (NumPy, PySerial, matplotlib).

Step 2: Program the FPGA with the BITFILE containing an APUF configuration.

Step 3: Load the challenge vector data (50k individual challenges).

Part B: Send Challenge and Collect Responses

Our procedure for Part B will take 3 steps:

Step 1: Send the 50k challenges to the FPGA (3 times each, for each FPGA).

Step 2: Collect the 50k responses.

Step 3: Save the responses to the Jupyter notebook.

Part C: Gather the results

Our procedure for Part C will take 4 steps:

Step 1: Load the relevant Python libraries (SciPy, NumPy, Matplotlib).

Step 2: Load the relevant response data (i.e. 3 response files for one FPGA to calculate the Intra-chip HD for 2 response files, one for each FPGA to calculate the Inter-chip HD).

Step 3: Calculate the Hamming Distances using SciPy python library.

Step 4: Record/Plot the Data using Matplotlib python library.

IV. Results

The results for the experiment (as described in Section III-C) are given in Tables I and II. For responses of each FPGA configured with each APUF the minimum, the maximum, the average, and the standard deviation (SD) of the HD are calculated. Additionally, Figures 2 and 3 show the average HD for the FPGAs with the two respective APUFs, and Figures 4 and 5 show the standard deviation of the HD for the respective APUFs.

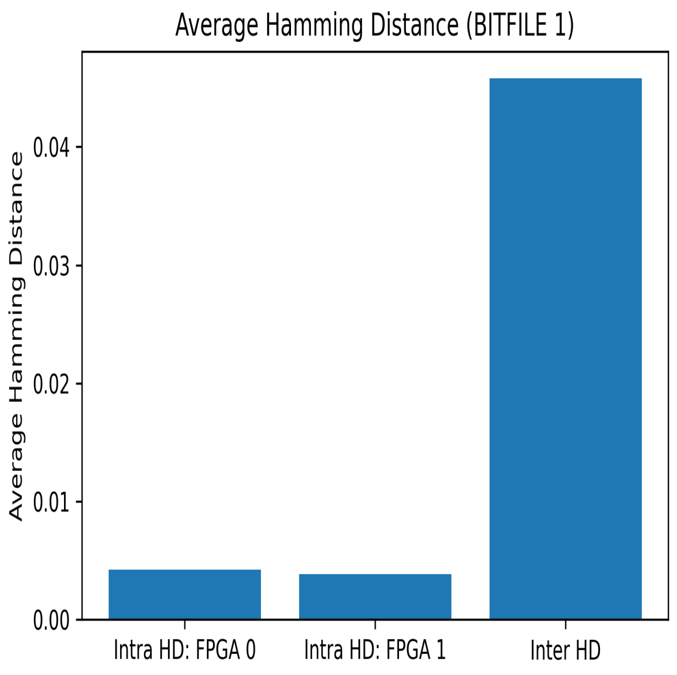

Fig. 2. Average Hamming Distances of FPGAs using APUF 1

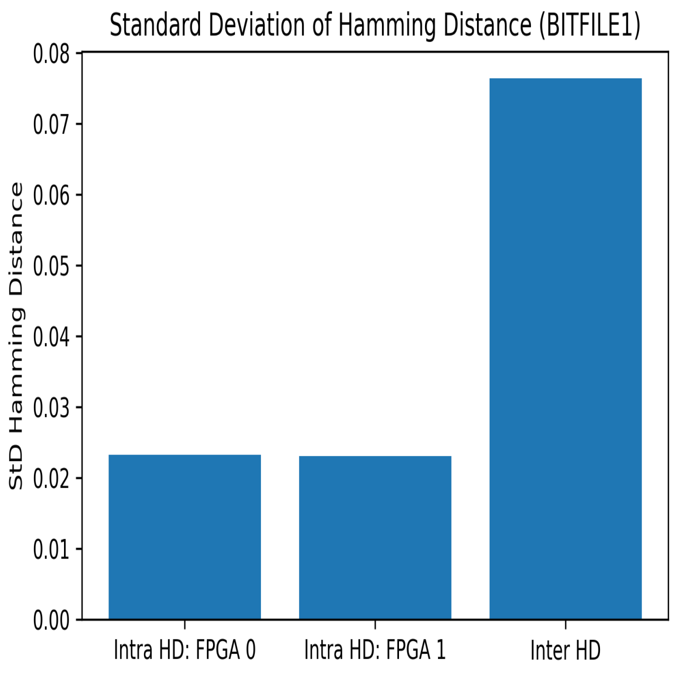

Fig. 4. Standard Deviation of HDs of FPGAs using APUF 1

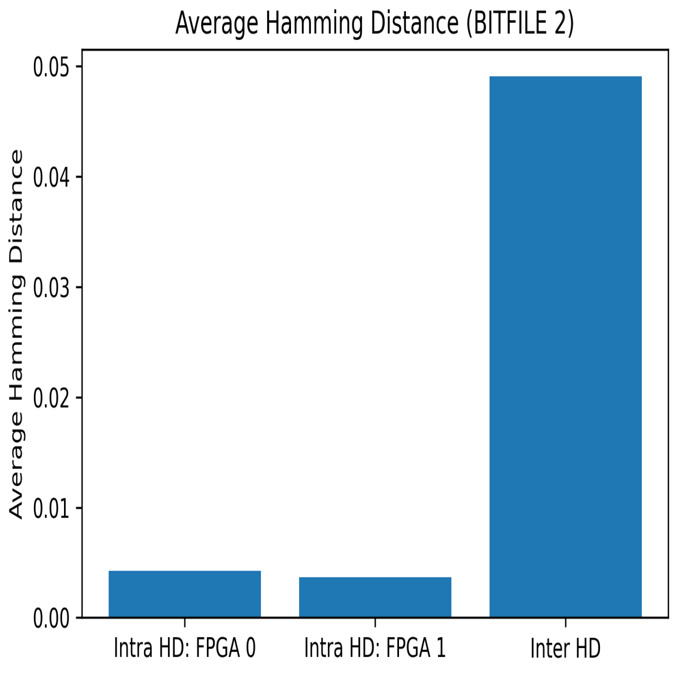

Fig. 3. Average Hamming Distances of FPGAs using APUF 2

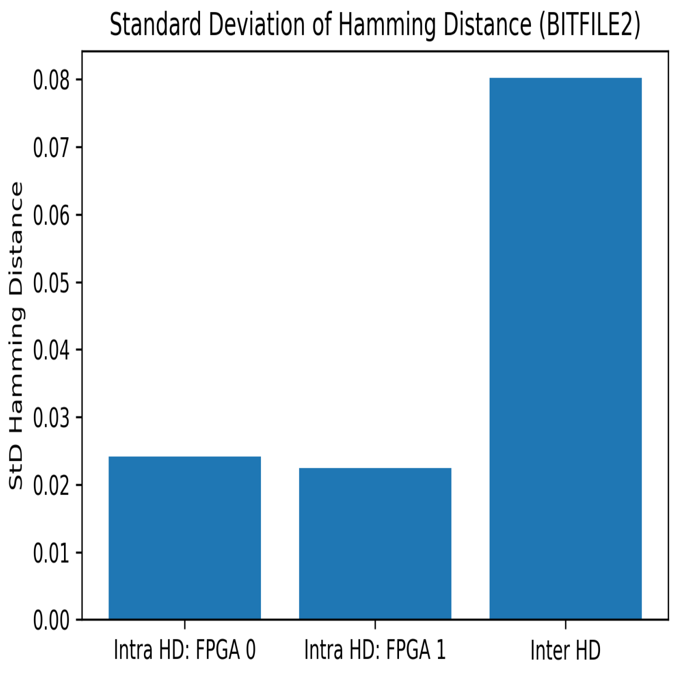

Fig. 5. Standard Deviation of HDs of FPGAs using APUF 2

V. Discussion

The quality of a PUF can be evaluated in two main ways: its *reproducibility*, and its *uniqueness*. A PUF with good *reproducibility* will output the same value (0 or 1) for the same challenge consistently while a PUF with good uniqueness will output a 0 or 1 with close to 50% probability for the same PUF design on different chip, thus ensuring that the output on the second chip is in no way be related to the output on the original chip.

From the results presented in Tables I and II, we notice that the intra-chip HD for both the FPGAs, configured with BITFILE1, is quite low (averaging at 0.423% for FPGA 1 and 0.384% FPGA 2) as expected. The standard deviations for the same implementation is 0.0233 for FPGA 1 and 0.0231 for

TABLE I
RESULTS FOR FPGA 1 AND FPGA 2 USING APUF 1

Intra-chip HD	**Min.**	**Max.**	**Avg.**	**SD**
FPGA 1	0.0	0.5625	0.00423	0.0233
FPGA 2	0.0	0.5	0.00384	0.0231
Inter-chip HD	0.0	0.6875	0.0458	0.07638

TABLE II
RESULTS FOR FPGA 1 AND FPGA 2 USING APUF 2

Intra-chip HD	**Min.**	**Max.**	**Avg.**	**SD**
FPGA 1	0.0	0.6250	0.00427	0.0242
FPGA 2	0.0	0.5	0.00369	0.02245
Inter-chip HD	0.0	0.6875	0.0458	0.07638

FPGA 2. This shows that each FPGA has good *reproducibility*. However, the inter-chip HD between both the FPGAs is also low, averaging at 4.58%. Therefore, for the given set of challenges for the first PUF configuration (BITFILE 1), there was not much difference between the responses produced by both the FPGAs. This means that the FPGAs lack *uniqueness*. Similar results for minimum, maximum, average, and SD for intra/inter-chip HD are obtained for both the FPGAs configured using BITFILE 2.

In regards to the low Interchip Hamming Distance, we consulted the literature to find out if there were any other occurences of this phenomena; and, there were. In [1] Machida et al., note that APUF designs have a low interchip uniqueness which makes it easy for Machine Learning Algorithms to attack the design. Additionally, other works such as [2] and [3] also note that basic APUF implementations have low interchip uniqueness. This allowed us to deduce that perhaps our low Interchip hamming distances are due to the properties of the APUF implementation on the FPGAs.

VI. CONCLUSION

In this lab, two PUFs were implemented and evaluated on two Spartan-7 FPGAs. The aquisition of responses for 50,000 challenges and analysis of the Hamming distance of these responses was performed using Python in Jupyter notebooks. The calculation and analysis of the inter-chip and intra-chip Hamming distance of the two FPGAs for each PUF configuration was done using numpy, scipy, and matplotlib python libraries. Analyzing the HD values of the two FPGAs, it was shown that each FPGA had very low intra-chip HD as expected for *reproducability*, however the inter-chip HD was also low showing low *uniqueness* between the two FPGAs.

REFERENCES

[1] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "Implementation of double arbiter puf and its performance evaluation on fpga," in *The 20th Asia and South Pacific Design Automation Conference*, 2015, pp. 6–7.

[2] N. Pundir, F. Amsaad, M. Choudhury, and M. Niamat, "Novel technique to improve strength of weak arbiter puf," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2017, pp. 1532–1535.

[3] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A new mode of operation for arbiter puf to improve uniqueness on fpga," in *2014 Federated Conference on Computer Science and Information Systems*, 2014, pp. 871–878.