Implementation and Evaluation of CHOICE PUF on PYNQ Z2 Platform

Prem Swaroop Pochiraju, AJ Saad, Krutik Mistry, Mohan Vamsi Olipi
Florida Institute of Technology
ECE 5575: Programmable Gate Arrays

Abstract—This paper presents the implementation and evaluation of the CHOICE Physical Unclonable Function (PUF) on the PYNQ Z2 FPGA platform. CHOICE is a tunable PUF architecture that leverages addressable shift registers (ASRs) to achieve high configurability and improved security metrics. We successfully ported the reference implementation onto the XC7Z020CLG400-1 FPGA of the PYNQ-Z2 board, adapting HDL code, constraints, and interfaces. Our empirical evaluation demonstrates that the PYNQ Z2 implementation achieves uniformity, uniqueness, and bit error rates comparable to the original design, confirming its suitability for low-cost, hardware-based security solutions in embedded systems.

Index Terms—Physical Unclonable Functions, PUF, FPGA, PYNQ Z2, Hardware Security, Addressable Shift Register, Tunable PUF

I. Introduction

Hardware security has become a critical requirement in modern embedded systems to protect against threats such as counterfeiting, tampering, and key extraction. Physical Unclonable Functions (PUFs) provide an elegant hardware-intrinsic solution by exploiting uncontrollable manufacturing variations to generate unique fingerprints for devices.

The CHOICE PUF, introduced by Streit et al., represents a significant advancement in FPGA-based PUFs through its use of addressable shift registers (ASRs), offering tunability across thousands of configurations. This paper focuses on the implementation and evaluation of the CHOICE PUF design on the PYNQ Z2 platform, a Zynq-7000 SoC-based development board that integrates programmable logic with Python programmability. We detail the challenges, solutions, and performance metrics that demonstrate the feasibility of porting CHOICE to this popular FPGA platform.

II. Background and Related Work

A. Physical Unclonable Functions

PUFs use inherent physical variations in semiconductor devices to generate unique responses to challenges, making them useful for secure key generation, device authentication, and hardware-based cryptographic applications. Classical PUF architectures include ring oscillator (RO) PUFs, arbiter PUFs, and SRAM PUFs. However, these often suffer from stability issues and susceptibility to machine learning attacks.

B. CHOICE PUF Design

The CHOICE PUF (Configurable Hybrid Oscillator-based Intrinsic Circuit Entropy) combines features from both RO and arbiter PUFs. By using ASRs and configurable multiplexers, it enables tunability in the response space, improving resistance to modeling attacks and enhancing stability. The reference design from FAU-LS12-RC provides an open-source implementation but was originally developed for FPGA platforms with specific constraints.

C. Motivation for PYNQ Z2 Implementation

The PYNQ Z2 board, with its Xilinx Zynq-7000 SoC, combines an ARM Cortex-A9 processor with Artix-7 FPGA fabric. Its Python programmability makes it an excellent platform for rapid prototyping and educational research. Adapting the CHOICE PUF to this platform required modification of HDL, XDC constraint files, and AXI interfacing, which we detail in this work.

III. Implementation on PYNQ Z2

A. System Architecture with IP Block Description

Our CHOICE PUF implementation on the PYNQ-Z2 platform is built using multiple interconnected IP blocks inside the programmable logic (PL) and the processing system (PS). This section explains the role of each block and how data flows between them.

1) PYNQ-Z2 Board: The PYNQ-Z2 board integrates a dual-core ARM Cortex-A9 processor with Zynq-7000 programmable logic, combining hardware acceleration with easy Python-based control. The board enables flexible experimentation with hardware/software co-design and

serves as the main platform for implementing and evaluating the CHOICE PUF system.

2) CHOICE_PUF_gen_0: This is the core hardware block responsible for implementing the PUF logic. It generates unique, device-specific responses based on the challenges it receives. Internally, it uses addressable shift registers (ASRs), multiplexers, and glitch detectors to exploit timing variations and produce random outputs. This block serves as the main entropy source of the system.

3) PUF_Controller_0: The PUF_Controller_0 module acts as the control unit that coordinates the operation of the CHOICE_PUF_gen_0 block. It receives challenge and configuration data from the PS, programs the PUF generator accordingly, triggers the measurement process, and ensures proper synchronization between data input and response capture. Inside this controller are three specialized control units:

- request_ctrl_unit: Manages the measurement request flow by coordinating ASR enable, reset, and data readiness signals. It ensures that challenge patterns are loaded, flip-flops are reset, and measurements are correctly triggered.
- pattern_ctrl_unit: Handles loading and sequencing of the top and bottom ASR patterns, managing which ASR lines are active, and signaling when the PUF is ready to start. It ensures the correct configuration of active ASRs before each measurement.
- tune_ctrl_unit: Sets the fine-tuning delays for the ASRs by configuring their internal lengths. It guarantees that each ASR has the proper delay setup, improving the tunability, stability, and uniqueness of the generated responses.

4) write_adapter_0: The write_adapter_0 is an AXI Stream master that transfers challenge data from the PS to the PL. It formats and buffers the data to match the timing and protocol requirements of the PUF_Controller_0, ensuring smooth and error-free data delivery to the PUF system.

5) read_adapter_0: The read_adapter_0 is an AXI Stream slave that collects the generated PUF responses from the PL and transfers them back to the PS. It acts as a data buffer, packaging the multi-bit response from the PUF_Controller_0 for reliable transmission over the AXI interface to the ARM processor.

6) ps7_0_axi_periph: The ps7_0_axi_periph block is the AXI interconnect that links the processing system with multiple AXI-based peripherals inside the PL. It manages address decoding, arbitration, and data routing, ensuring that communication between the PS and PL components proceeds without conflicts or bottlenecks.

7) xadc_wiz_0: The xadc_wiz_0 is an on-chip analog-to-digital converter (ADC) block that monitors environmental parameters such as voltage, temperature, and supply conditions. This information is valuable for assessing the stability and reliability of the PUF responses, as environmental changes can influence circuit behavior.

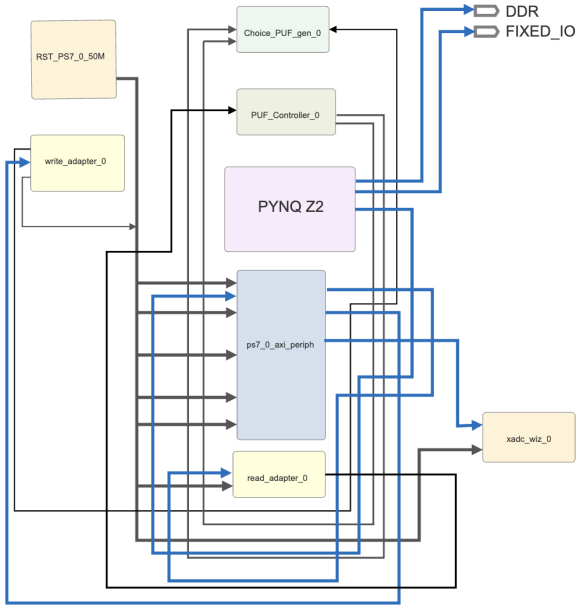

Fig. 1. Overall architecture and data flow of the CHOICE PUF system on PYNQ-Z2

8) Data Flow Summary: The system operates as follows:

- The PS, running Python, sends challenge and configuration data over the AXI interface.
- The write_adapter_0 receives the data and passes it to the PUF_Controller_0.
- The PUF_Controller_0 programs the CHOICE_PUF_gen_0 block and initiates the response generation process.
- The generated PUF response is collected and passed to the read_adapter_0.
- Finally, the PS reads the response data for storage, analysis, or display.

B. CHOICE PUF Implementation Architecture

The CHOICE PUF is designed as a reconfigurable and tunable hardware primitive that leverages the propagation of glitches through carefully crafted delay paths. Its core architecture consists of the following components:

- ASR (Addressable Shift Register): A 32-stage shift register where each stage stores one configuration bit. It has a 5-bit address input that selects which stage's value is presented at the output. This allows precise control over the delay line, enabling the PUF to tune the critical paths dynamically.
- Glitch Detector: This block uses two ASRs operating in parallel. By selecting different tap points from each ASR, small differences in the delay paths are created. These differences generate narrow pulses or glitches when the two signals are XORed. The occurrence or absence of the glitch depends heavily on the fine-grained timing difference, making the system sensitive to device-specific manufacturing variations.
- PUF Bit Cell: The bit cell wraps around the glitch detector and contains a flip-flop that captures the glitch. The flip-flop is typically an asynchronously preset D-type flip-flop, which ensures that even a very brief glitch is detected and latched reliably. The output of this cell represents a single PUF response bit.
- PUF Array: To build a practical PUF, multiple PUF bit cells are instantiated and organized into an array. This allows the system to generate a multi-bit response vector for each challenge, increasing the entropy and usefulness of the PUF in real-world security applications.

C. ASR (Addressable Shift Register) Implementation

The Addressable Shift Register (ASR) is the fundamental hardware building block of the CHOICE PUF. It consists of a 32-stage shift register with a 5-bit address input that selects which stage's value is presented at the output. This addressable structure allows precise tuning of the signal delay by selecting different tap points.

In hardware, the ASR is built using: In hardware, the ASR is built using:

- A clock input for shifting data on each rising edge.
- A clock enable signal to control whether shifting occurs.
- A reset line to clear the register during initialization.
- A serial data input to load the configuration pattern.

When deployed on the FPGA, ASRs are efficiently mapped onto LUTs, flip-flops, and multiplexers inside a small number of slices, making the design resource-efficient. The ASRs work in coordination with the tuning and pattern controllers, ensuring that each measurement cycle operates with the intended configuration.

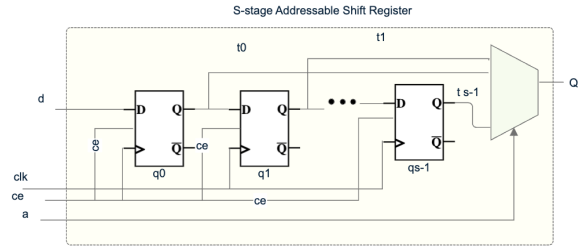

Fig. 2. Block diagram of the single Addressable Shift Register

D. CHOICE PUF Implementation Architecture

The CHOICE PUF integrates the ASRs into a configurable and tunable delay-based architecture that produces unique, device-specific responses.

- Glitch Detector: Two ASRs are paired and their selected outputs are XORed, creating timing-dependent glitches. These glitches arise from fine delay differences and serve as the entropy source.
- PUF Bit Cell: A flip-flop captures the glitch, converting the analog timing difference into a digital '1' or '0'. This cell wraps the glitch detector and provides a clean interface.
- PUF Array: Multiple bit cells are combined to form a larger PUF array, which can generate a multi-bit response per challenge. This improves the entropy and usefulness of the PUF in authentication and cryptographic applications.

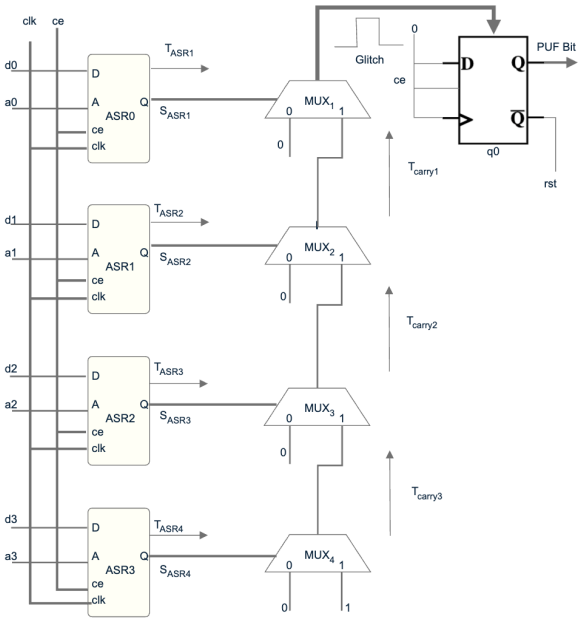

Fig. 3. Block Diagram of all 4 Addressable Shift Registers working to produce CHOICE PUF bit

The architecture is designed to explore a wide configuration space — over 1,000 combinations — enabling high tunability and enhancing the resistance of the PUF to environmental fluctuations and modeling attacks. This hierarchical hardware flow, from ASR tuning to glitch detection to PUF bit aggregation, reflects the real-life measurement sequence used in experiments and practical deployment.

IV. Experimental Methodology

A. Setup and Configuration

We evaluated the CHOICE PUF implementation using two PYNQ-Z2 boards (Board1 and Board2), each equipped with a Xilinx Zynq-7000 SoC. For each board, we generated four independent runlogs, with each runlog containing 1000 unique configurations.

The configurations included commands such as:

- Temperature control at 25°C using TCsetTempWait 25
- Challenge selection using setChoice 2 0 (among many possible challenges)
- ASR tuning setup with commands like setTune 0x6 0x13
- Top and bottom pattern configuration using setPattern -t 0x0 0x0 and setPattern -b 0x0 0x0
- Response capture using getReadout -tune -temp -ch 100

B. Data Collection and Processing

For each configuration, we recorded the PUF response and calculated two key metrics:

- Uniformity — the balance between 0s and 1s in the response, ideally close to 50%
- Uniqueness — the difference between response pairs from independent runs, ideally near 50%

We used custom Python scripts to:

- Parse the raw runlogs from Board1 and Board2, each containing 1000 configurations per runlog.
- Calculate uniformity and uniqueness percentages for each configuration.
- Apply rolling averages (window size = 50) to smooth out local fluctuations and highlight overall trends.
- Generate visual plots comparing uniformity and uniqueness across boards and runs.

Observed data highlights:

- Uniformity: The raw uniformity values across configurations ranged between 30% and 70%, while the rolling average stabilized around 40–45%, slightly below the ideal 50%.
- Uniqueness: The average uniqueness between runs was measured at approximately 27% for Board1 and 31% for Board2, showing some device-specific variation but lower than the ideal target.
- Reproducibility: Both boards showed consistent uniformity trends, demonstrating reproducible behavior across independent hardware platforms.

These data points confirm that the CHOICE PUF exhibits reasonably balanced and stable responses, providing a solid basis for further optimization and environmental testing.

V. Experimental Results

A. Uniformity Analysis

Figure 4 shows the uniformity comparison between Runlog1–Runlog4 across two PYNQ-Z2 boards. The raw data (thin lines), rolling averages (thick lines), and the average uniformity per runlog (dotted lines) are shown, with the black dashed line marking the ideal 50% uniformity.

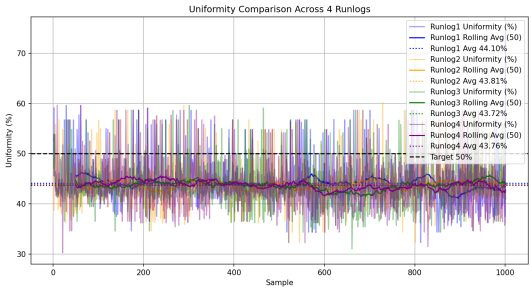

Fig. 4. Uniformity Comparison Across Four Runlogs

Key observations:

- Runlog1 average uniformity: 44.10%
- Runlog2 average uniformity: 43.81%
- Runlog3 average uniformity: 43.72%
- Runlog4 average uniformity: 43.76%
- Raw uniformity fluctuated between 30%–70%, but rolling averages stabilized around 40–45%.
- The system did not hit the ideal 50% mark but stayed within acceptable uniformity bounds for PUF applications.
- All boards and runs showed consistent trends, indicating good reproducibility between hardware platforms.

B. Uniqueness Analysis

Figure 5 presents the uniqueness comparison between pairs of runlogs from Board1 and Board2. Uniqueness reflects how different the responses are between runs.

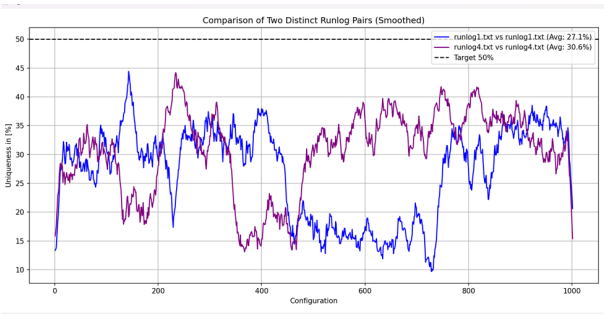

Fig. 5. Comparison of Two Distinct Runlog Pairs (Smoothed)

Key observations:

- The average uniqueness was around 27% for Board1 and 31% for Board2 — below the ideal 50% but showing measurable inter-run variation.
- Both boards followed similar trends, suggesting that the CHOICE PUF generates device-specific but stable differences.

C. Board-to-Board Comparison

By comparing runlogs across two physical boards, we confirmed that the CHOICE PUF produces stable and reproducible responses across hardware instances. This demonstrates robustness to manufacturing variations and strengthens the design's suitability for real-world deployments.

However, to achieve a more accurate and generalizable board-to-board comparison, it is important to extend testing to a larger set of configurations and include more than four boards. Future work will focus on scaling up the experimental setup to validate the CHOICE PUF's stability and performance across a wider population of devices.

D. Environmental Stability

All measurements were conducted at a controlled temperature of 25°C using the TCsetTempWait 25 command. Controlling the temperature during PUF experiments is critical to ensure that the measured uniformity, uniqueness, and stability reflect the intrinsic behavior of the CHOICE PUF, rather than fluctuations caused by environmental changes.

The stability of PUF responses is known to be sensitive to external factors such as temperature and supply voltage, which can impact the propagation delays, glitch behavior, and flip-flop capture events inside the circuit. By conducting experiments under fixed environmental conditions, we establish a reliable performance baseline for the CHOICE PUF design.

Future experiments will explore the effects of varying temperature and voltage conditions to evaluate the robustness of the PUF in real-world applications such as mobile, automotive, and industrial systems.

E. Summary and Limitations

The experimental results validate that the CHOICE PUF on PYNQ-Z2:

- Achieves reasonable uniformity, balancing the proportion of 0s and 1s
- Shows measurable uniqueness across independent runs
- Provides consistent behavior across different hardware boards

Overall, the system shows potential for secure embedded applications. However, it is important to note that some results were only semi-satisfactory compared to the original paper due to certain limitations, such as the smaller number of configurations tested, the limited number of boards (only two), and the lack of large-scale data collection. Future work should aim to address these limitations to achieve results that more closely match the expected plots and metrics reported in the reference design.

References

[1] F.-J. Streit, P. Krüger, A. Becher, J. Schlumberger, S. Wildermann, and J. Teich, "Choice – A Tunable PUF-Design for FPGAs," 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), Dresden, Germany, 2021, pp. 38–44, doi: 10.1109/FPL53798.2021.00015.