PUF Evaluation Metrics on 7 Series FPGA: Comparative Analysis of Arbiter, XOR Arbiter, and Double Arbiter PUFs for Uniqueness, Randomness, and Stability

By

Janviben Lunagariya

B.Eng., Gujarat Technological University, 2017

A Report Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

University of Victoria

PUF Evaluation Metrics on 7 Series FPGA: Comparative Analysis of Arbiter, XOR Arbiter, and Double Arbiter PUFs for Uniqueness, Randomness, and Stability

By

Janviben Lunagariya
B.Eng., Gujarat Technological University, 2017

Supervisory Committee

Co-Supervisor

Dr. Mihai Sima,

(Department of Electrical and Computer Engineering)

Co-Supervisor

Dr. Chris Papadopoulos,

(Department of Electrical and Computer Engineering)

ABSTRACT

Hardware security modules play a crucial role in protecting and preserving technologically integrated systems that are used in daily life. They employ cryptographic protocols to secure a system against adversaries. Generally, cryptographic algorithms and security keys are essential for maintaining the security of a system. Cryptography uses a secret key to encipher and decipher the data. The confidential keys are stored in a non-volatile memory, making it easily accessible to potential attackers.The hardware security primitive, Physical Unclonable Function (PUF) is a promising alternative for enhancing the security of interconnected devices. Physical Unclonable Functions are specialized circuit components that exploit the subtle variations inherent in microchip fabrication. These variances enable the creation of unique "fingerprint" output sequences, or responses, in reaction to specific inputs or challenges. The random, device-specific nature of these variations and their replication difficulty - even by the original manufacturer using identical methods, tools, and parameters - make PUFs an excellent choice for cryptographic key generation. Moreover, these characteristics are designed to remain unchanged, reinforcing their suitability for this application.

The Arbiter-based Physically Unclonable Function (PUF) is a type of delay-based PUF that utilizes signal delay-line time differences. However, previous studies indicated that Arbiter PUF, when implemented on Xilinx Virtex-5 FPGAs, produced nearly identical responses by exhibiting low uniqueness. Other variants of Arbiter PUF, such as XOR Arbiter PUF and the Double Arbiter PUF, were introduced to address this issue. This novel technique generates highly unique responses from duplicated Arbiter PUFs on FPGAs at a comparable cost to the 2-XOR Arbiter PUF. The Double Arbiter PUF differs from the 2-XOR version in the mode of operation, particularly regarding wire assignment between the arbiter and the final selector output signals.

This study evaluates these PUFs for uniqueness, randomness, and stability on Xilinx 7-series FPGA Devices and seeks to identify a new Arbiter PUF operation mode that is feasible for FPGA implementation. We propose the 3-1 Double Arbiter PUF, which includes an extra duplicated Arbiter PUF, yielding three Arbiter PUFs that produce a 1-bit response. When compared with the 3-XOR Arbiter PUF, the 3-1 Double Arbiter PUF shows better response uniqueness and randomness estimated at 50%, indicating that the evaluation metrices of the PUF can be improved by using a new Arbiter PUF operation mode. We show that we can improve uniqueness and randomness using the new mode of operation for the Arbiter PUF performance characteristics for 16, 32, and 64-bit selector pairs for 65,536 responses.

Contents

List of Tables

List of Figures

Glossary

FPGA: Field Programmable Gate Array

VLSI: Very large-scale integration

ASIC: Application-specific integrated circuit

PUF: Physical unclonable function

APUF: Arbiter Physical unclonable function

XOR APUF: XOR Arbiter Physical unclonable function

DAPUF: Double Arbiter Physical Unclonable function

MUX: Multiplexer

TPM: Trusted Platform Module

NVM: Non-volatile memory

CRP: Challenge-Response Pair

DUT: Design Under Test

ACKNOWLEDGEMENTS

I am grateful to God; I am thankful to my parents for being my constant cheerleaders. I am also thankful to the people that provided technical advice during this project. I would like to express immense gratitude to my supervisor Dr. Mihai sima and Dr. Chris Papadopoulos for their technical advice, patience, support, and invaluable guidance which undoubtedly helped me to successfully complete this project and my degree. I have deep appreciation for Mr. Harsh Gandhi and Mr. Bhaumik Darji, who were instrumental in starting my journey in this domain from System Level solutions. I would also like to thank Mr. Ernesto from NETINT Technology for providing me an opportunity as a co-op student where I got to learn a lot more. Finally, I would also like to thank all my friends at the University of Victoria for being an integral part of this journey.

CHAPTER 1 INTRODUCTION

The Rapid push to market electronic devices, driven by intense competition, has sometimes led to compromised security and vulnerabilities. The strategy to reduce design cycle durations, involving the re-use of IP cores and automated design methodologies, often exposes systems to potential breaches. Notable examples include the Spectre and Meltdown vulnerabilities, which exploit side-channel timing attacks and out-of-order execution respectively, to access confidential data in modern microprocessors.

In addressing security challenges, the vast range of application domains, from resource limited IoT devices to high-end electronics, adds complexity. Traditional high-end devices often incorporate a Trusted Platform Module (TPM) to enhance security [15]. Meanwhile, TPMs store cryptographic keys safely in non-volatile memories (NVM) such as flash or EEPROM. They manage device authentication and provide a robust security layer, but they come with their own set of drawbacks. TPMs increase the overall device cost due to additional anti-tampering measures and rely heavily on an endorsement key, often compromising user privacy for personal data.

This TPM approach, although suitable for high-performance electronics, becomes economically impractical for resource-constrained devices like many in the IoT spectrum. In contrast, Physical Unclonable Functions (PUFs) emerge as a promising alternative, bypassing the need for memory-based key storage. PUFs address both the vulnerability to physical attacks inherent in TPMs and the associated high manufacturing costs, making them a compelling solution for a broader range of devices [16].

Physically Unclonable Functions (PUFs) are a type of function where a unique input (challenge) leads to aunique output (response) based on physical elements like semiconductor circuits. These PUFs are hard to duplicate due to their dependency on physical variations, making them ideal for device authentication against counterfeiting. A verifying server could store these unique challenge-response pairs for device authentication. PUFs also offer a secure method of storing secret keys, as the keys become unreadable once the device package is opened due to the physical variations [1][2].

PUFs can be implemented on both ASIC (Application-Specific Integrated Circuits) [3] and FPGA (Field Programmable Gate Arrays) [4][5]. FPGAs have the advantage of easily modifiable design and implementation, making them popular in commercial products [6]. FPGAs are inherently reprogrammable. This means you can modify, test, and iterate your PUF design without creating a new chip each time. If you find a vulnerability or wish to upgrade the PUF, it is easier to implement changes on an FPGA.

Evaluation studies have been conducted on FPGA implementations of the Arbiter PUF (APUF), a type of delay-based PUF, as highlighted in references [8] and [11]. Findings from [11] indicate that APUFs, when deployed on Xilinx Virtex-5/Kintex-7 FPGAs, tend to produce responses that lack distinctiveness. A study in [8] suggests that this lack of uniqueness in responses from Virtex-5 FPGA-based APUFs might stem from issues related to SLICEs in the FPGAs. This SLICE problem is a broader issue with FPGAs in general. In the traditional APUF design, a response is derived by contrasting signals across two wires. Ideally, these wires should be of equal length for all challenge inputs in APUFs. However, due to the rigid layout of logic components (like SLICEs) on FPGAs, designers need to have control over wire lengths between these components. Differences caused by wire lengths overshadow the resulting differences in delay times stemming from inherent physical variations. Consequently, responses from distinct APUFs across various devices display minimal variance against numerous challenges, leading to low uniqueness. To produce highly unique responses on Field Programmable Gate Arrays (FPGAs), A proposed technique for generating highly unique responses on Field Programmable Gate Arrays (FPGAs) involves an n-stage XOR

Arbiter and a Double Arbiter PUF (DAPUF). This method involves duplicating another Arbiter PUF(APUF) on adjacent SLICEs where the original APUF resides. The concept assumes that the wire of the duplicated APUF approximates the length of the original one [7].

There is also the n-XOR APUF, where the response is derived by XORing the n-bit (n= 1,2,3…n-1) responses from APUFs on the same FPGA. This approach has the exact circuit costs as DAPUF, i.e., it uses two selector chains. For example, 2-XOR APUF is called 2-1 APUF, and 1-bit output DAPUF with two Arbiter chains is called 2-1 DAPUF.

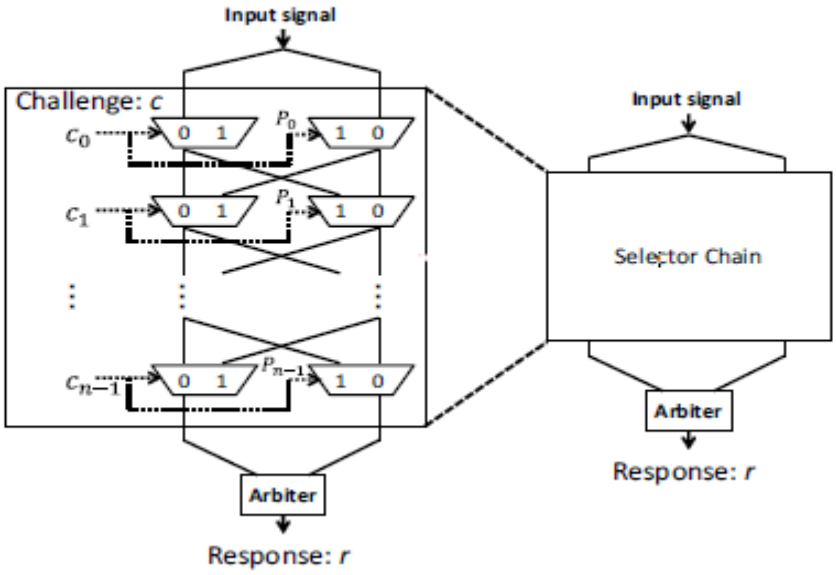

Fig.1.1 Structure of the conventional Arbiter PUF

A. Our Contributions

Figure 1.1 illustrates our comparison of PUF structures, where we define a *selector chain* of conventional APUF as a building block. This chain represents a single Arbiter PUF structure.

We introduced a 3-1 DAPUF using three selector chains, as shown in Figure 2.5. Our evaluation using Artix-7 FPGA indicates that, compared to the traditional 3-XOR APUF as shown in Fig. 2.4, the 3-1 DAPUF notably improves response uniqueness. Our demonstration will show how uniqueness can be enhanced by employing a novel operational mode for APUF and XORing responses from additional duplicated arbiter chains on Artix 7 FPGAs in Xilinx Vivado tool using post implementation timing simulation. By duplicating three APUF chains and introducing three selector chains of APUF for two selector Arbiter chains (2-1 APUF), we increased the uniqueness of APUF responses.

In this study, we introduce the 3-1 DAPUF and illustrate its XOR operation on 6-bit outputs from DAPUFs in Fig. 2.3 (refer to Chapter 6 of this paper for more details).

This contrasts with the 3-XOR APUF, which uses three selector chains but yields only a 1-bit response. This paper labels the 3-XOR APUF as the 3-1 XOR Arbiter PUF. Our experiments indicate that the response uniqueness of the 3-1 APUF is roughly 6%, which remains suboptimal. However, the 3-1 DAPUF demonstrates a uniqueness rate of around 50%, significantly surpassing the 3-1 APUF.

B. Organization of This Paper

The Structure of this paper is as follows: Chapter 2 delves into related work and Introduction of the traditional APUF, 3-1 XOR APUF, and 3-1 DAPUF, and a comparison in the Structure of each PUF. Additionally, it includes a discussion of the evaluation metrics associated with Physical Unclonable Functions.

In Chapters 3,4 and 5, we present Arbiter PUF, 3-1 XOR Arbiter PUF, and 3-1 DAPUF, respectively, and the experimental simulation framework in the Vivado synthesizing tool on Artix 7 FPGA board (xq7a100tcs324-1I) using Verilog (Hardware Description Language).

Chapter 6 presents the findings and discussions about Arbiter PUF, XOR APUF, and Double Arbiter PUF regarding their evaluation metrics. Chapter 7 provides a conclusion to the work presented and future work.

CHAPTER 2 BACKGROUND

This chapter provides details of Physical Unclonable Function (PUF), applications of PUF, threats to PUF, and Evaluation of PUF. A Physical Unclonable Function (PUF) is an object mapped on a physical device and can generate random integers. A PUF output is straightforward to analyze, but predicting one is far more challenging. It leverages the variation of the physical microstructure, and even if we manufacture the devices similarly, they are likely to yield different results [9]. During manufacturing, unpredictable and uncontrollable random physical factors introduce microstructure attributes.

2.1. Physical Unclonable Functions

Physically Unclonable Functions, an input, referred to as a challenge, generates an output, a response. This combination of a specific challenge and its unique response forms a challenge-response pair (CRP). Each CRP is distinct and highly unpredictable due to its dependence on the characteristics of the physical hardware. As a result, PUFs are gaining prominence in hardware-centric security, playing an increasingly significant role in high-security applications.

Every PUF has a distinct behavior because of the minute manufacturing variations and imperfections. When they are produced. Even if two PUFs are manufactured using the same process and design, these uncontrollable variations ensure that each PUF will have a unique response to a given challenge. This inherent unique variation makes it practically impossible to replicate or clone a PUF, even by the original manufacturer. As a result, an adversary cannot create a physically identical copy of a PUF due to these manufacturing variances. PUFs can generate and store cryptographic keys without the need for non-volatile memory. The key is not stored in a traditional sense but is regenerated whenever needed, making it resistant to various attacks that aim to extract the key.

2.2. Delay-Based PUF

A delay-based Arbiter Physical Unclonable Function (PUF) is a specific type of PUF that relies on variations in delay characteristics within an arbiter circuit. This circuit typically comprises two or more paths with varying propagation delays. These paths feed into the arbiter circuit, which, in turn, determines the final response of the PUF. The essential element of the delay-based PUF is delay lines, as They exploit the inherent manufacturing variability, which introduces unpredictable and unique delays in the signal paths.

1. Ring Oscillator PUF

The core of a ring oscillator PUF is a ring oscillator circuit, which consists of an odd number of inverting stages (typically three or more). The output of the last stage is connected back to the input, forming a loop. The propagation delay through each stage of the inverter in the ring oscillator is subject to manufacturing variations and environmental factors. These variations create imbalances in the delay, leading to unique frequency and phase characteristics for each ring oscillator.

2. Arbiter PUF

An Arbiter Physically Unclonable Function (PUF) consists of numerous multiplexers (MUXes) organized into upper and lower stages. We direct the outputs from these stages to a D-type flip-flop. The response generation is contingent on the arrival times of these signals at the flip-flop.

2.2.1. Arbiter PUF

The Arbiter Physically Unclonable Function (APUF) is a type of delay based PUF, that leverages the delay time differences between two signals. Arbiter PUF structure consists of left and right selector pairs connected in series. Each bit of an n-bit challenge corresponds to a selection input to a specific selector pair. The path an input signal takesdepends on the value of the challenge, resulting in a unique 1-bit response determined by which signal reaches the arbiter first.
In an Arbiter PUFs can easily increase the number of challenge bits by using more selector pairs. However, this approach is vulnerable to modeling and simulation that predict responses to challenges, posing a security risk.

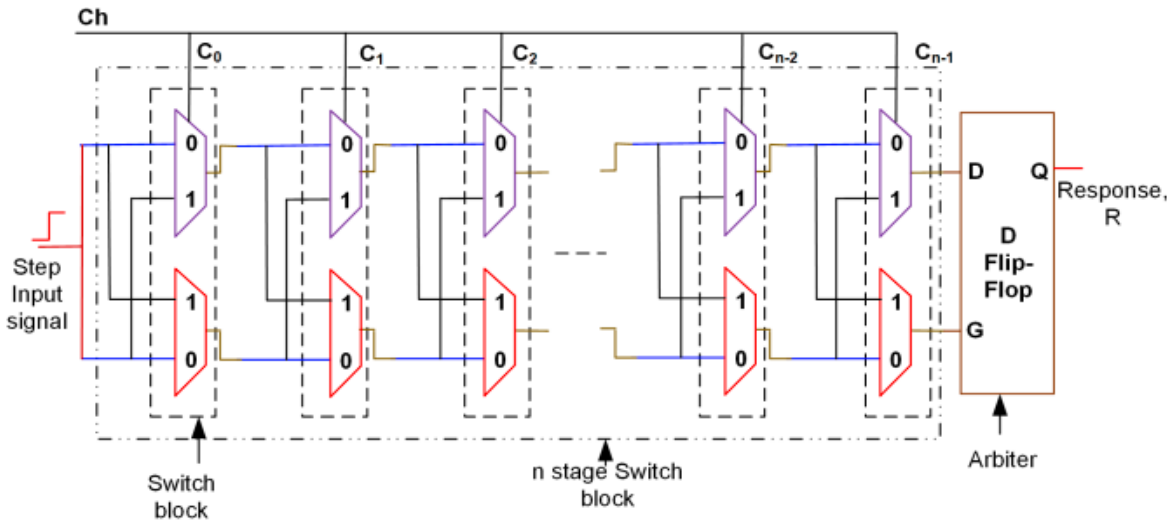

Figure 2.1 Structure of Arbiter PUF

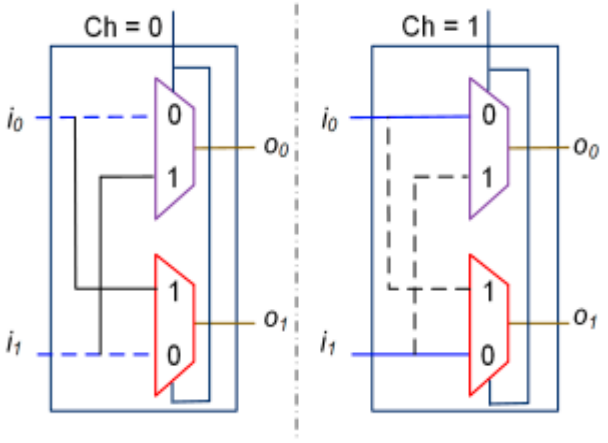

Figure 2.2 Switch block for Ch=0 and Ch=1

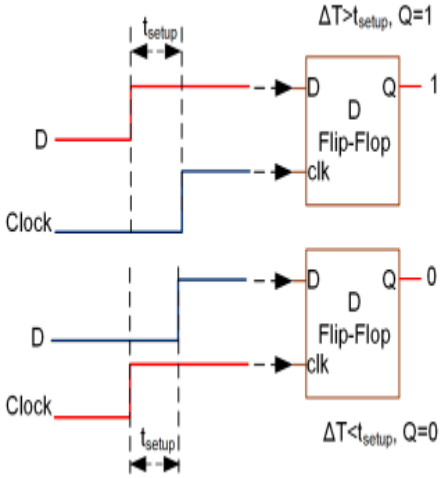

Figure 2.3 Response based on the signal transition in D- FF

Understanding the structure of an arbiter PUF is crucial, as both XOR PUF and DAPUF designs consist of multiple arbiter PUFs, as initially introduced in [15,16].

When applying the input to the APUF, the upper and lower paths convey the input signal with a specific delay based on the challenge. Let us denote the delays created by the upper and lower paths as T1 and T2. As depicted in Fig. 2.2, the delay difference, $\Delta T = T1 - T2$, becomes the input to the arbiter. Fig. 2.3 illustrates that the D-FF samples the response, registering a 1 when the positive edge of input D arrives before the positive edge of input clk by a time value greater than the setup time of the D-FF; otherwise, the output remains at 0. A logical 1 or 0 is achieved when the delay difference is more significant. If the delay difference is considerably less, the arbiter enters a metastable condition, resulting in an unstable response. Thus, designing with more challenge bits, meaning more Multiplexers is advisable. With eight stages of MUXes going to the D flip-flop input, we have 2^8 different paths to choose from, and the final stage response depends on whichever path signal arrives at the arbiter input first.

In Arbiter Physically Unclonable Functions (APUFs), the number of challenge bits can be readily expanded by incorporating additional selector pairs. An APUF with 'n' selector pairs can handle 2^n challenges. APUF's behavior can be predicted and simulated using software models and programs built on the relationship between challenges and responses. However, this simulation capability leaves APUFs vulnerable to attacks, enabling an adversary to predict responses to nearly all challenges [9].

To safeguard against these predictive modeling attacks, N-XOR APUFs have been suggested. In this design, N-bit responses from N different APUFs are combined through an XOR operation to produce a single 1-bit response [8]. In the next section, we will discuss XOR Arbiter PUF in more detail.

2.2.2 XOR Arbiter PUF

The XOR Arbiter PUF is essentially a composition of multiple Arbiter PUFs. For a system with 'N' Arbiter PUFs, the output responses of each of these PUFs are XORed together to generate the final single-bit 7response of the XOR PUF. The XOR operation is inherently nonlinear, adding an extra layer of complexity and unpredictability to the design with evolving security requirements. XOR Arbiter PUF emerged as an enhanced design to address some of the limitations of the original Arbiter PUF.

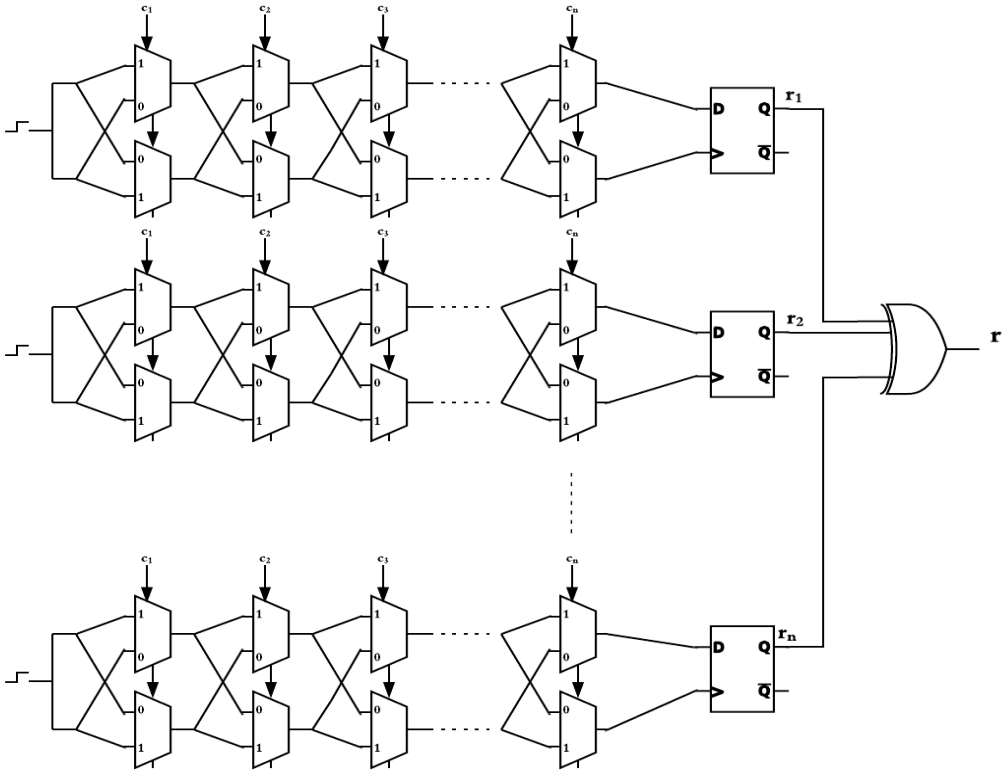

Figure 2.4: Structure of 3-1 XOR Arbiter PUF

Advantages of XOR Arbiter PUF

1.Enhanced Unpredictability: By combining multiple Arbiter PUFs, the XOR Arbiter PUF introduces nonlinearity, making it more challenging to predict or model the final response based solely on individual Arbiter PUF behaviors. For instance, even if an attacker knows the response of 'N-1' Arbiter PUFs, they cannot accurately determine the overall XOR Arbiter PUF response due to the nonlinear XOR operation.

2.Improved Stability: While individual Arbiter PUFs can sometimes generate inconsistent outputs due to environmental variations or noise, the XOR operation can mask some of these inconsistencies. When multiple stable Arbiter PUFs are combined, the overall reliability of the XOR PUF often improves.

Disadvantages of XOR Arbiter PUF:

1. Increased Complexity: More Arbiter PUFs mean more circuitry, leading to a larger design footprint, increased power consumption, and potentially longer response times.

2. Vulnerability to Machine Learning Attacks: Several studies, like [7-9], have shown that XOR Arbiter PUFs can still be susceptible to modeling attacks using advanced machine learning techniques. As a hypothetical example, if someone trains a neural network with enough challenge-response pairs, they can model and predict the XOR PUF's response to new challenges.

3. Reliability Concerns: Although the XOR operation can improve stability, it is a double-edged sword. If even one of the individual Arbiter PUFs has a significantly low reliability, it can adversely impact the overall reliability of the XOR PUF.

While the XOR Arbiter PUF presents a commendable advancement over the traditional Arbiter PUF, especially in terms of unpredictability and, to some extent, reliability, it is not without its flaws. Security designers must weigh its advantages against its vulnerabilities, particularly in environments where machine learning-based attacks are a concern. As with all security tools, understanding its strengths and weaknesses is key to deploying it effectively.

2.2.3 Double Arbiter PUF

The world of Physical Unclonable Functions (PUFs) has witnessed significant evolution and diversification, with the overarching goal of enhancing hardware security. Among these designs, the Arbiter PUF has been fundamental. As security concerns and demands evolved, variants like the XOR Arbiter PUF and the 3-1 Double Arbiter PUF have emerged. Let us explore the 3-1 Double Arbiter PUF and see how it compares to its counterparts.

The 3-1 Double Arbiter PUF is an advancement over the basic Arbiter PUF. In essence, it uses three individual Arbiter PUFs, which then feed into another Arbiter PUF. The first three generate intermediate responses, which are used as challenges for the final (or "double") Arbiter PUF, leading to the ultimate output.

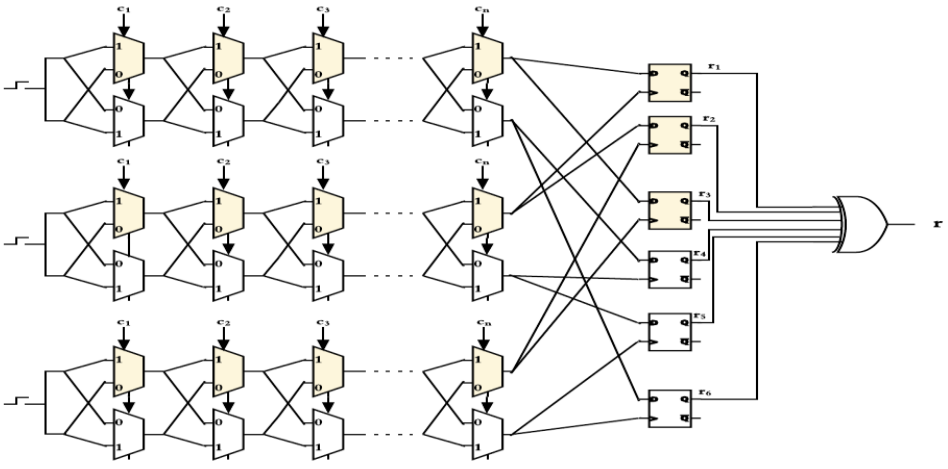

Figure 2.5 Structure of 3-1 Double Arbiter PUF

1. **Complexity and Unpredictability**

The design is inherently more complex than the standard Arbiter PUF. This layered approach aims to boost unpredictability. It is harder for an attacker to discern or reverse-engineer the system since they would have to tackle the unpredictability of multiple interconnected PUFs [8].
Imagine you have two safes you need to crack. The first safe (representing the 3-1 XOR APUF) has a single, intricate lock mechanism. It is complex, but once you figure out this single mechanism, you're in. The second safe (representing the 3-1 Double Arbiter PUF) has three different lock mechanisms layered one behind the other. Even if you crack the first lock, you still have two more to decipher before gaining access.

In this scenario, the 3-1 XOR APUF is like the safe with a singular, complex lock. It is challenging, but ultimately just one system to understand. The 3-1 Double Arbiter PUF, with its multi-layer approach, adds dimensions to the challenge, making it inherently harder to breach since an intruder would have to navigate through multiple unpredictable layers.

2. **Resistance to Modeling**

The multi-tiered nature of the design can potentially offer better resistance to machine learning-based modeling attacks compared to the basic Arbiter PUF. Essentially, the 3-1 Double Arbiter design introduces a higher degree of nonlinearity than a standard and XOR Arbiter PUF.

Comparison with XOR Arbiter PUF and Double Arbiter PUF

1. **Unpredictability**: Both the XOR Arbiter PUF and the 3-1 Double Arbiter PUF aim to improve unpredictability compared to the standard Arbiter PUF. However, the methods differ for each PUF. While XOR Arbiter PUF relies on the XOR operation to combine outputs, the 3-1 Double Arbiter cascades PUFs. The cascading can, theoretically, introduce more intricacies than simple XOR operations.
2. **Reliability:** The multi-stage approach of the 3-1 Double Arbiter PUF might bring in concerns about consistency. Each stage can introduce potential deviations, so there's a cumulative effect to consider. XOR Arbiter PUFs, by comparison, might offer better stability because they do not rely on cascading stages.
3. **Design Complexity:** The 3-1 Double Arbiter PUF is inherently more complex in design than the basic Arbiter PUF. This can be both an advantage (in terms of security) and a disadvantage (in terms of power, area, and implementation concerns). The XOR Arbiter PUF, by virtue of its simpler XOR operation, might be more straightforward to implement than the multi-staged 3-1 design.
4. **Vulnerability to Attacks:** All PUFs, when under scrutiny, can have vulnerabilities. While the 3-1 design is more resistant than the basic Arbiter PUF, it is still crucial to consider the kind of threats and attacks it might face, especially when compared to the XOR Arbiter PUF.

The 3-1 Double Arbiter PUF offers a new perspective in the domain of PUF designs, aiming to strike a balance between unpredictability and implementation feasibility. While it holds promise in boosting resistance to certain attacks, designers must carefully weigh its advantages and challenges, especially when compared against the XOR Arbiter PUF and the standard Arbiter PUF. Like all security designs, its effectiveness is dependent on the context and application. We are going to discuss about evaluation parameter of the PUF.

2.3 Evaluation Metrics for PUF

In a lot of computer science solutions, random numbers are crucial components. To ensure limitations on computational complexity, randomized algorithms need a random source, and sampling techniques frequently depend on randomization to appropriately reflect the data they are collecting [13].

Random numbers are used in cryptographic applications to produce encryption keys, establish starting parameter values, and include random nonces into protocols and padding methods. These random numbers are typically generated by a pseudorandom number generator, a deterministic software function that emulates randomness. Random number assure that no one can predict the next number based on the present one. A vast set of random challenge values ensures that potential attackers can not easily model or clone the PUF. If the challenges were predictable or sequential, attackers might deduce a pattern, reducing the security of the PUF.
With a large set of Challenge response pair, a PUF can be used in various security applications without being easily exhausted and this property makes it more versatile. Also, Random challenge values make modeling attacks considerably harder.

The performance of PUFs can be evaluated with these parameters: uniformity, uniqueness, randomness, unpredictability, stability, and reliability [10][11][12].

2.3.1 Uniformity

The degree to which the proportion of "0" and "1" in a PUF's replies is uniform is referred to as uniformity [12]. The probability of "1"s should ideally equal the probability of "0"s for PUF responses to be unpredictable and random. The fractional Hamming Weight (HW) of the total replies is used to calculate uniformity as given by [13]:

$$Uniformity = \frac{1}{m} \times HW(R) \times 100\% \quad (1)$$

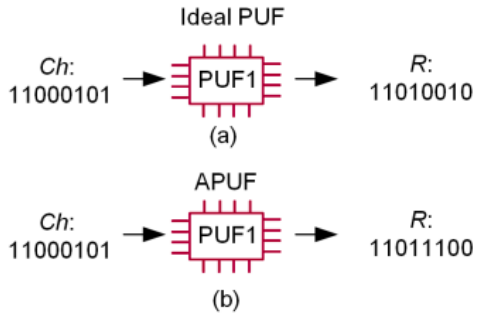

Figure 2.6 Uniformity

Considering an example illustrated in Fig. 2.6a, the uniformity for an ideal PUF is expected to be 50%. If it is equal to either 100% or 0%, all the bits in an 8-bit response are either at logic-1 or logic-0.

As in Fig. 2.6b, the hamming weight for an 8-bit response. (11011100) counts to 5. The uniformity for the given challenge is 62.5% representing the number of 1 is more than the number of 0.

2.3.2 Uniqueness

The ease with which one PUF instance can be identified from another PUF instance is characterized by uniqueness [12]. Since inter-chip differences are measured by uniqueness, every pair of chips should be considered. The average fractional inter-chip Hamming Distance (HD) between responses produced in response to the identical challenges from several chips can be used to evaluate it as given by [13]:

$$Uniqueness = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(R_i, R_j)}{n} \times 100\% \quad (2)$$

Where k is the number of chips, Ri, and Rj represent-bit responses generated on different chips i and j (i ≠ j) respectively. For the uniqueness of the PUF on each device, the value should approach 50% [12]

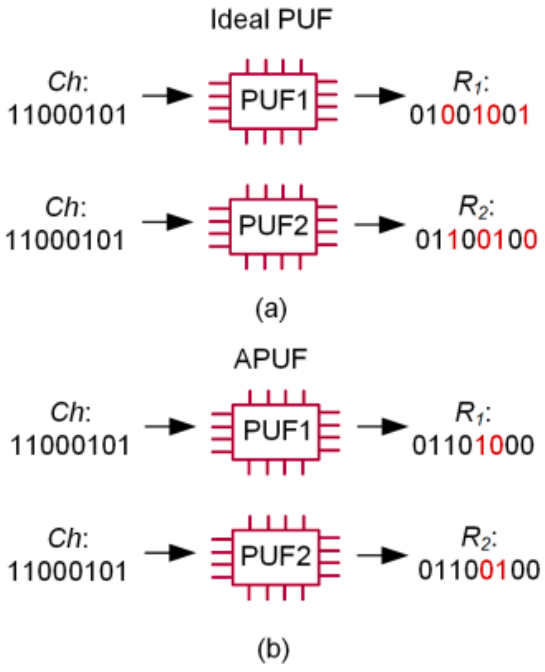

Figure 2.7 Uniqueness

Fig. 2.7 illustrates the uniqueness with an example. Let the responses from PUF1 and PUF2 be 01001001 and 01100100, respectively, for a challenge of 11000101, as shown in Fig 2.7a. The HD between the two

responses for an ideal PUF is four, and the uniqueness equals 50%, illustrating that the responses are unique for PUF1 and PUF2. On the other hand, for the APUF shown in Fig 2.7b, for the same challenge, the HD is two, and the uniqueness is 25%, elucidating that the number of unique responses determined from an APUF is comparatively less.

2.3.3 Reliability

Reliability is a measure to quantify the capability of the PUF to reproduce a reliable response for a given set of challenges irrespective of the changes in environmental conditions such as temperature and supply voltage [12].

Consider an example of a comparison of PUF for the ideal condition with an APUF, as depicted in Fig. 2.8a. It generates a similar output (01001100) for a given input of 11000101 for two different temperatures, thus achieving a reliability of 100%. However, in APUF, a bias exists in the output for varying temperature conditions, as illustrated in Fig.2.8b. For the same challenge of 11000101, the HD for the two responses is two, and the reliability reduces to 75% compared to the ideal PUF, indicating the PUF design is unreliable at varying temperatures.

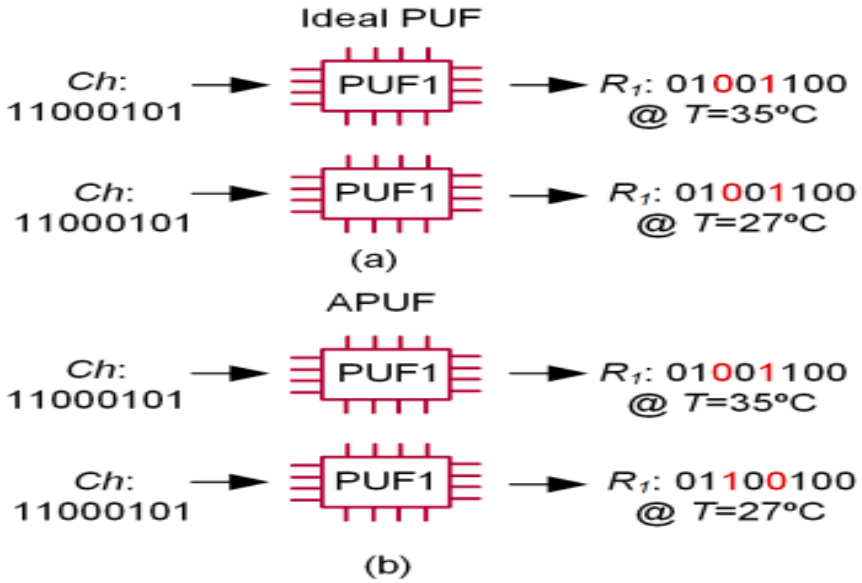

Figure 2.8 Reliability

Reliability is estimated using the average fractional intra-chip HD of replies produced by the same challenges on the same chip as given by [13]:

$$HD_{INTRA} = \frac{1}{k}\sum_{i=1}^{k1}\frac{HD(R_i(n), R_I'(n))}{n} \times 100\% \quad (3)$$

Where x is the number of samples to be collected on chip i in the same environment, Ri is the reference m-

bit responses of chip i in some environmental condition, and R0i; y is the responses of the k sample in a different condition.
The reliability of the PUF on chip *i* is defined below as given by [13]:

$$reliability = 100\% - HD_{INTRA} \quad (4)$$

Ideally, HD_{intra} is expected to be 0 while the reliability is 100%, which means the PUF is totally reliable.

2.3.4 Randomness

A "fair" coin, unbiased and labeled with "0" and "1," can serve as a model for generating a random bit sequence, given that each flip has precisely a 50% chance of yielding either a "0" or a "1." These flips arealso independent events, meaning that the result of any flip has no bearing on the outcome of another.
Consequently, the distribution of "0" and "1" values in the sequence is entirely subject to randomness, making this fair coin an ideal generator for a uniformly distributed [0,1] random bit stream. It is crucial to note that the subsequent element's value remains unpredictable regardless of the number of already produced elements because each element in the sequence is generated independently.

2.3.5 Unpredictability

In the field of cryptography, the unpredictability of generated random and pseudorandom numbers is crucial. For Pseudorandom Number Generators, even with knowledge of previously generated numbers in the sequence, the following output should remain unpredictable if the seed is unknown. This attribute is often referred to as forward unpredictability. Similarly, it should be impossible to discern the seed based on any generated values, which ensures backward unpredictability. Every element in the sequence should seemingly originate from an independent random event with a 50% probability, ensuring no correlation between the seed and any derived value from that seed.

2.3.6 Stability

Stability refers to the ability of the PUF to produce consistent outputs under varying conditions. For example, suppose we have an Arbiter PUF, and we feed it a certain challenge (input). We expect it to produce a particular response. Stability in this context would mean that every time we feed this Arbiter PUF the same challenge, even under varying environmental conditions (temperature, voltage) or across multiple power cycles, it should consistently produce the same response.
The challenge is that physical conditions such as temperature and voltage can alter the delays in the PUF circuit, potentially leading to different responses for the same challenge. Therefore, techniques such as error correction codes and fuzzy extractors can be used to improve the stability of PUF responses.

CHAPTER 3 Arbiter PUF Implementation in Verilog using Xilinx Vivado

In the study, an APUF and other variants are set up on a Xilinx Artix-7 FPGA [16], utilizing the CSG324100T, a standard evaluation board. We used Xilinx Vivado 13.2 for logic synthesis and post-implementation timing simulation and performed floor planning using the same device. We chose a Xilinx Artix-7 FPGA with speed grade -1 to implement this design due to the commercial availability of this FPGA device.

When designing an Arbiter PUF (Physically Unclonable Function) in Verilog, the primary structure revolves around a combination of delay elements (like delay lines) and an arbiter. We can categorize these components in digital logic design as sequential and combinational logic units. Multiplexers to introduce variability in the delay paths are a combinational logic element. The challenge bits set the configuration of these multiplexers, which determines the delay paths. At the end of the delay lines is an arbiter, essentially a type of latch or flip-flop. The arbiter determines which of the two delay paths signal arrives first. If the signal from one path arrives before the other, the output (response) would be '1'; otherwise, it would be '0'. That is the sequential element in the design.

The essential element of the delay-based PUF is delay lines, as they exploit the inherent manufacturing variability, which introduces unpredictable and unique delays in the signal paths.

3.1 Simulation of Arbiter PUF Design Under Test

In this section, we discussed the simulation of arbiter PUF. As mentioned, this project is tested on the Artix-7 CSG324100TFPGA board. It shows the behavior of arbiter PUF.

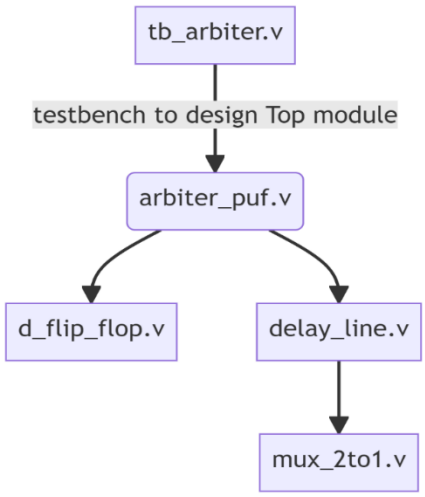

Figure 3.1 Instantiation flow of Arbiter PUF Design

The design comprises five distinct modules, as illustrated in Figure 3.1. The testbench module is **tb_arbiter.v** is pivotal in applying stimulus to the primary design module, **arbiter_puf.v**. As explained in Chapter 2, the Arbiter PUF consists of a dual-stage structure of Delay lines known as the upper and lower stages, and their

output goes to the arbiter.

The top module for the Arbiter PUF design **arbiter_puf. v**, instantiates two key modules: **delay_line. v** and **d_flip_flop.v**. Notably, the **delay_line.v** module incorporates a configurable number of multiplexers (MUXes), allowing adaptability to varying requirements, such as 8, 16, 32, 64, or more MUXes. This flexibility provides the ability to customize the design according to specific needs.

Within the **delay_line.v** module, instances of MUXes form chains representing the upper and lower stages of the Arbiter PUF. The resulting outputs from these delay stages are then directed to the arbiter circuit, implemented as a D flip-flop within the **d_flip_flop.v** module.

This section details the concrete design and implementation of the Arbiter PUF configuration, executed in the Verilog Hardware Description Language. The primary objective is observing consistent behavioral patterns within the FPGA device, enhancing our understanding of the Arbiter PUF design.

Function for generating response of device is given below:

The procedure for obtaining challenge response pair data proceeds as follows:

1. The challenge bits C are chosen which consists of number of stages bits to configure the state of the switching stages. Challenge width variable is configurable and according to out requirement we can change their values suchas 8, 16,32,64 and more etc.

2. One bit pulse `pulse_i` is issued to all the arbiter PUF and feded as an input to initial stage of 2x1 MUX as an input.

3. Based on the input and challenges bits value to all the MUX stages response gets Generated.

Table 3.1 illustrates the response generation function which calculates the device response using upper and lower path delay difference.

```
// Python code that mimics the behaviour of Arbiter
PUF in HDLclass ArbiterPUF:

    def __init_(self,
        challenge_length=32):
        self.challenge_length =
        challenge_length
        self.delay_line = DelayLine()
        self.dff1 = DFlipFlop()

    def execute (self, pulse_i, challenge_i):
        delay_line_mux_out_1_o, delay_line_mux_out_2_o = self. delay_line.process(pulse_i,
        challenge_i)response_o = self. dff1.process(delay_line_mux_out_1_o,
        delay_line_mux_out_2_o)
        return response_o
```

Table 3.1 | Pseudocode for response generation function of Arbiter PUF

Here, the `DelayLine() and DFlipFlop()` are classes that emulates the behavior of their respective hardware counterparts. **`execute`** is a method that corresponds to how Verilog runs continuously. It takes a pulse input and a challenge input, then runs them through the delay line to obtain two delay line outputs. These outputs are then run through the flip flop to obtain the final response output.

Port	Width	Direction	Description
pulse_i	1 bit	Input	It activates the first stage of the MUXes
challenge_i	Variable	Input	Based on the width it indicates number of MUXes .
response_o	1 bit	Output	1 bit output
C_LENGTH	Variable	Input	Number of MUXes

Table 3.2 Arbiter PUF Port Descriptions

As shown in the Table 3.2, In the Verilog coding context, the notation '_i' is used to denote the input port, while '_o' is used for the output port. `C_LENGTH` is a global parameter to set the number of Multiplexer in a chain for this design.

```
// This is the TOP module of the Arbiter PUF.
module arbiter_puf
#(

  parameter C_LENGTH = 32 //the length of the chain of the multiplexer / number of challenge bits
 )
 (
  input              pulse_i,
  input[C_LENGTH-1:0] challenge_i,
  output             response_o
    );

    wire delay_line_mux_out_1_o;
    wire delay_line_mux_out_2_o;


    delay_line inst_delay_line (
    .pulse_i     (pulse_i),
    .challenge_i (challenge_i ),
    .up_mux_o    (delay_line_mux_out_1_o),
    .do_mux_o    (delay_line_mux_out_2_o)
    );

   d_flip_flop inst_dff1(
    .id          (delay_line_mux_out_1_o),
    .iclk        (delay_line_mux_out_2_o),
    .oq          (response_o)
    );

endmodule
```

Table 3.3 | Arbiter PUF Design Top module

1. Define the module `arbiter_puf` with parameters C_LENGTH = 32, which determines the length of the challenge and, consequently, the multiplexer chain. It is configurable so according to the requirement. One may change it from the TOP module.

2. Set up inputs for the `pulse_i` and `challenge_i`[C_LENGTH-1:0] Set up output for `response_o`.

3. Inside the module, define two wires `delay_line_mux_out_1_o`, and `delay_line_mux_out_2_o`, which will carry the output from the delay line.

4. Instantiate a delay line module named `inst_delay_line` with inputs as `pulse_i` and `challenge_i` and outputs as `delay_line_mux_out_1_o` and `delay_line_mux_out_2_o`.

5. Instantiate a `d_flip_flop` module named inst_dff1 with the output from the delay line as the inputs. Theoutput of inst_dff1 is `response_o`.

6. Output response_o is generated.

```
module tb_arbiter;
reg         pulse_i;
reg [ 7:0] challenge_i;
wire        response_o;

// Instance of Arbiter_PUF top module DUT
arbiter_puf dut (pulse_i,challenge_i ,response_o);

// Provide stimulus to the input ports of the DUT
initial begin
  pulse_i=1'b0;
    forever #40 pulse_i = ~pulse_i; end
repeat(100)

begin
challenge_i  =$random;
#32 $display ("challenge_i=%d",challenge_i );
   end

endmodule
```

Table 3.4 Arbiter PUF Testbench

Table 3.3 and 3.4 describes the testbench for the design module `arbiter_puf`. This module, `tb_arbiter`, appears to be a testbench for the `arbiter_puf` module. A testbench is typically used in hardware description languages like Verilog to provide stimuli for the design under test (DUT) and observe its behavior.

Here's what this Verilog code is doing:

1. It first defines a module `tb_arbiter`.

2. Then, it declares `pulse_i` as a register and `response_o` as a wire, `pulse_i` and `challenge_i` will be used to provide input stimulus to the Design Under Test.

3. An instance of the `arbiter_puf` module (DUT), named **dut**, is created with `pulse_i`, `challenge_i`, and `response_o` as its inputs and outputs.

4. Inside an **initial** block, the testbench:

5. Sets `pulse_i` to 0.

6. Then it starts a forever loop with a delay of 40 time units. In each loop iteration of , `pulse_i` is inverted

(**~**`pulse_i`). This effectively creates a clock signal for `pulse_i` with a period of 80 time units.

7. It then begins another loop that iterates 100 times. In each loop iteration, `challenge_i` is set to a random 8-bit value (using the **$random** system function) after a delay of 32-time units.

3.2 Simulation of Arbiter PUF in Verilog using Xilinx Vivado

Simulation means writing a testbench to test the behavior of the Design. A testbench is an HDL module used to test another module, called the device under test (DUT). The testbench contains statements to apply inputs to the DUT and, ideally, to verify that the system produces the correct outputs. The input and desired output patterns are called test vectors in the case of arbiter PUF.

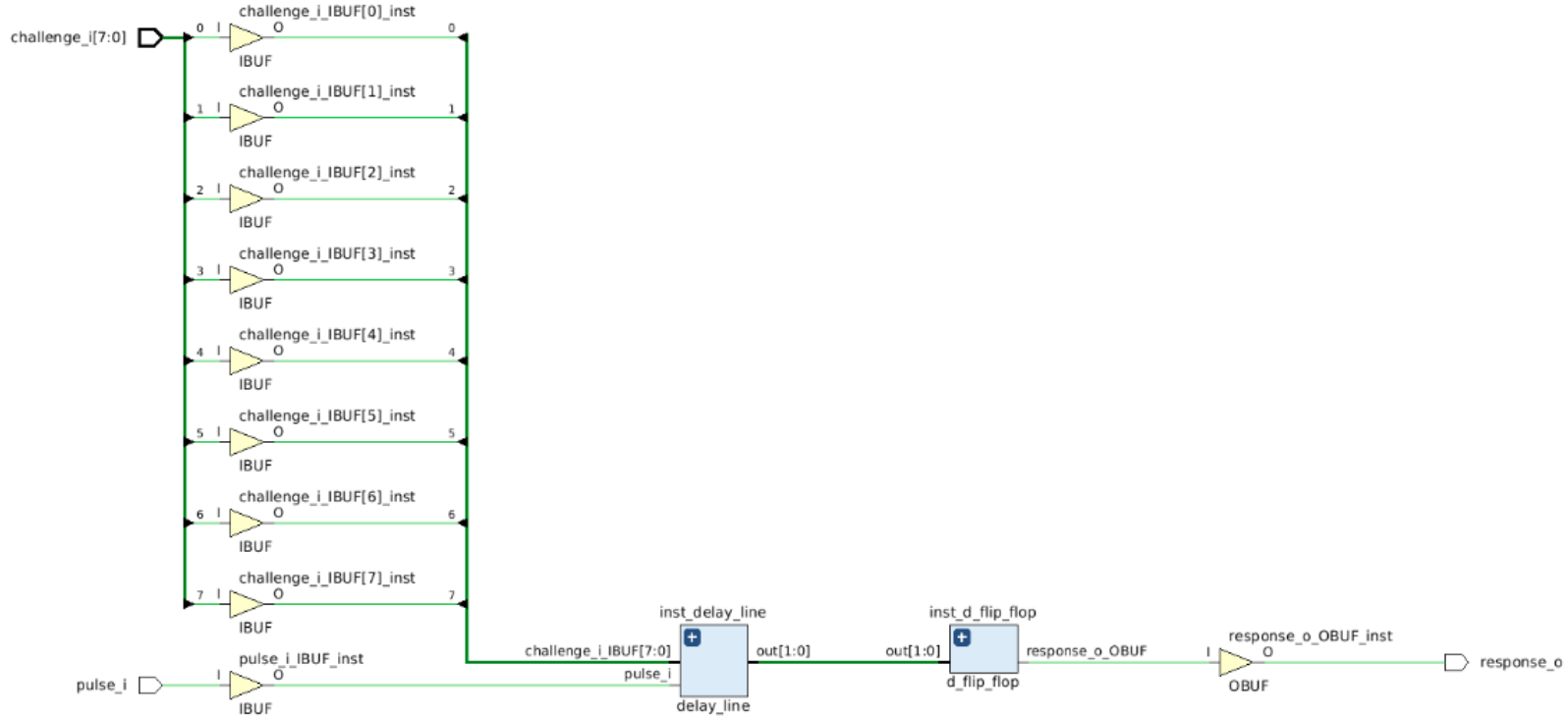

Figure 3.2 Schematic of Arbiter PUF

As depicted in Figure 3.2, we have chosen 8 Multiplexers in the Arbiter PUF Design for this design. `challenge_i` represents the number of multiplexers. `pulse_i` represents an initial input to the first MUX and goes to the other MUXes based on the `challenge_i` value. `inst_delay_line` had 8 MUXes, and the output of the last MUX in the chain out [1:0] goes to the `inst_d_flip_flop`.

D flip flop Delay instance named `inst_d_flip_flop`, as shown in Figure 3.2, out [0] to the data input (D or d_in) and out [1] to the clock input

(C or clock). The upper and lower MUXs derive the outputs out[0] and out[1]. Owing to distinct propagation delays in each MUX, these outputs do not reach the D Flip Flop(`inst_d_flip_flop`) at the same time. Depending on which input (D_in or clock) reaches the D Flip Flop first, the output (`response_o`) toggles between 0 and 1.

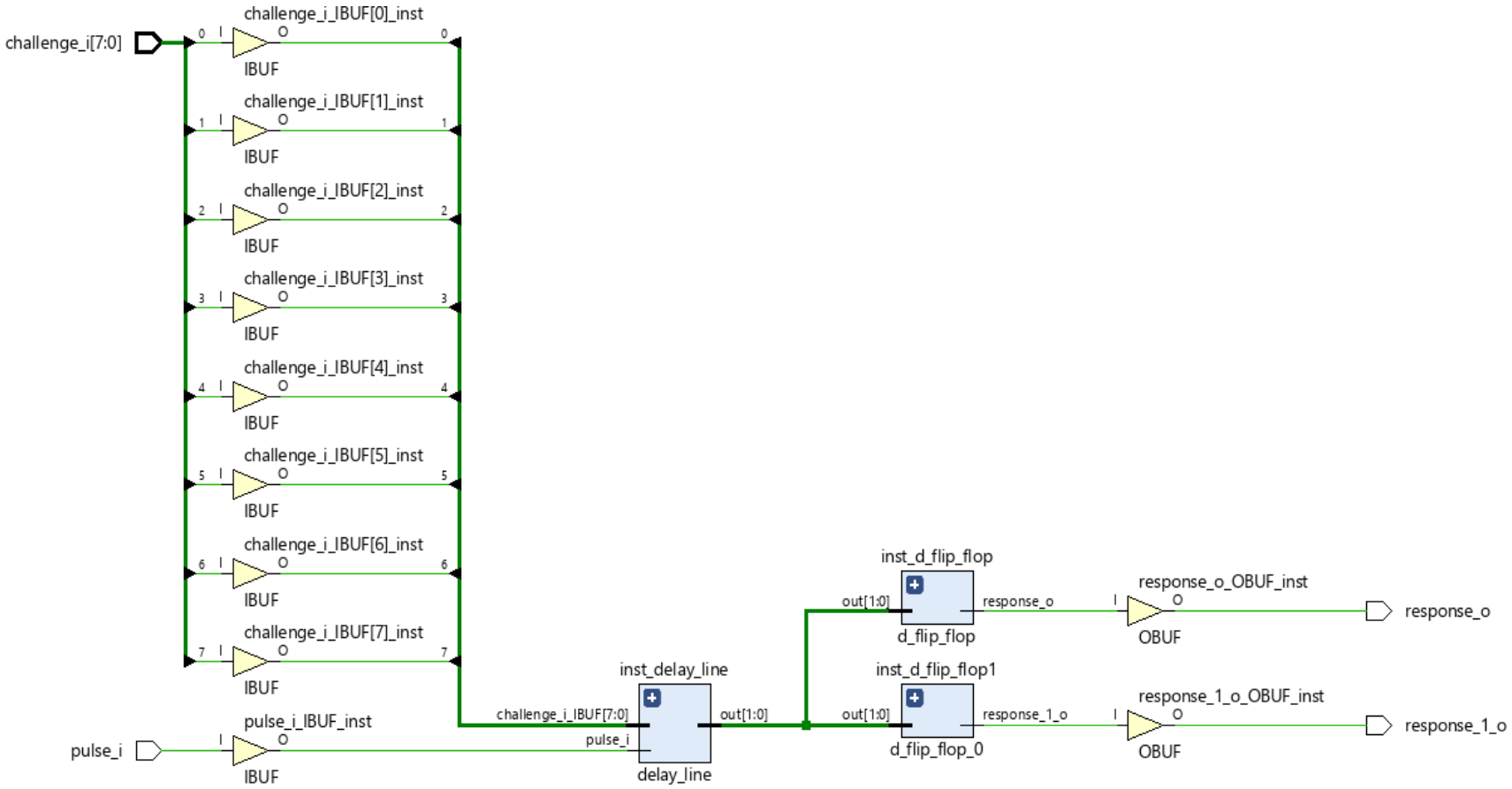

Figure 3.3 Schematic of Arbiter PUF: two D -FF for debugging

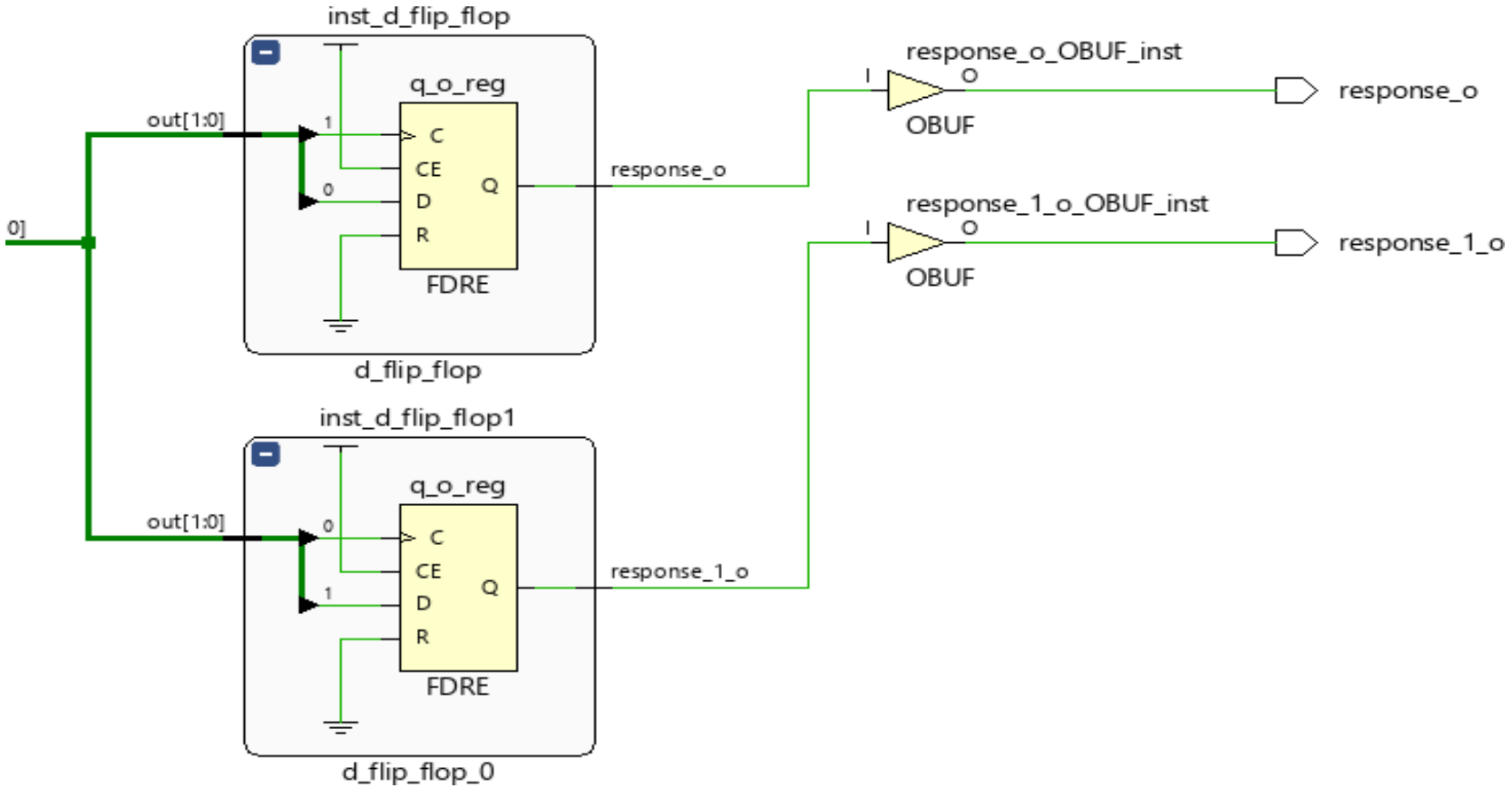

Figure 3.4 Schematic of Arbiter PUF: two D -FF cross connection closer look

As depicted in Figure 3.4, We created two instances of D flip-flops. For the first one, named inst_d_flip_flop, we have assigned out [0] to the data input (D or d_in) and out [1] to the clock input (C or clock). The upper and lower MUXs derive the outputs out[0] and out[1]. Due to distinct propagation delays in each MUX, these outputs do not reach the D Flip Flop simultaneously. Depending on which input (D_in or clock) reaches the D Flip Flop inst_d_flip_flop first, the output (response_o) toggles between 0 and 1.

For the second instance, inst_d_flip_flop1, we have reversed the roles of the inputs, without [1] serving as the data input and out[0] as the clock input. That is for debugging purposes.

3.2.1 Arbiter PUF behavioral simulation

Behavioral simulation of an Arbiter Physical Unclonable Function (PUF) involves assessing its high-level design functionality without considering timing, delays, and physical attributes. This initial testing phase focuses on verifying logical correctness by inputting various challenges and confirming expected responses. The goal is to ensure the PUF's appropriate and reliable operation before delving into detailed simulations and the physical implementation stage. This preliminary evaluation helps identify and address potential issues early in the design process, laying the foundation for subsequent, more intricate simulations that consider timing and physical characteristics.

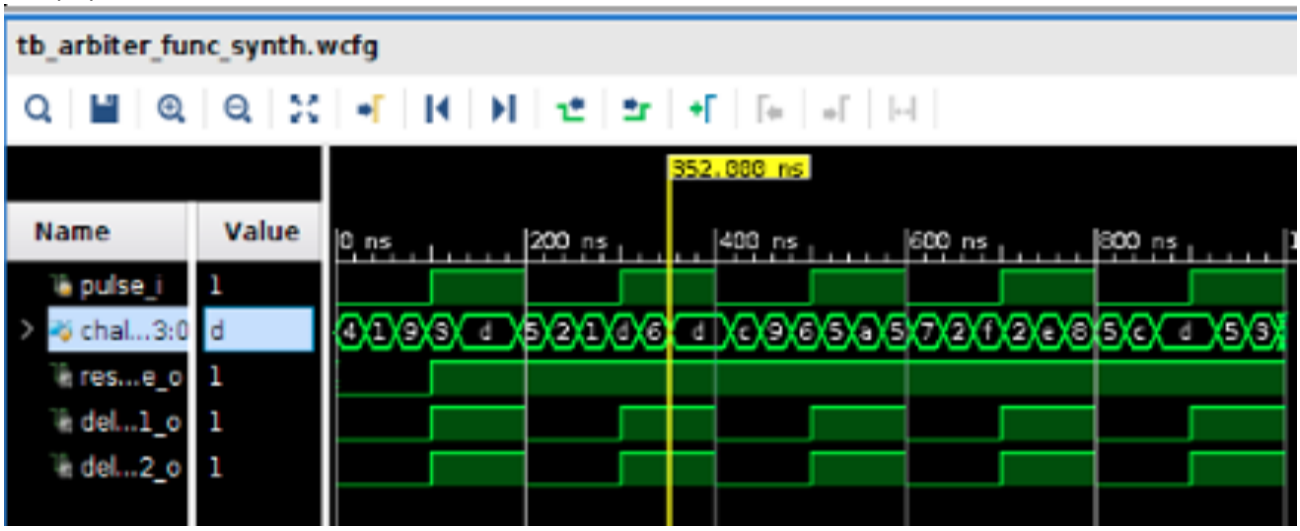

Figure 3.5 Arbiter PUF Behavioral simulation

As shown in Figure 3.5, When the input `pulse_i` is set to 0, all the 2x1 multiplexers receive an input 1. However, regardless of the stateof the selection lines, a 0 is propagated through to the final multiplexer stage due to the 0 value of `pulse_i`. As a result, the output, `response_o`, is observed as "0".
When the input `pulse_i` switches to 1, the behavior of the system changes accordingly. As we are not consideringdelay in this context, the D flip-flop (D-FF) retains its previous state, which in this instance is 1. Therefore, the output does not revert to 0, ensuring the system response reflects the updated `pulse_i` value.`delay_line_mux_out_1_o` (Upper stage MUX output) and `delay_line_mux_out_2_o` (Lower stage MUX out) is coming simultaneously since no delay element is being considered.

3.2.2 Arbiter PUF post implementation timing simulation

Timing simulation in the context of an Arbiter Physical Unclonable Function (PUF) involves a comprehensive analysis of the design by considering the timing characteristics inherent in the circuit. This simulation type is particularly crucial for Arbiter PUFs due to their reliance on delay differences in generating responses.

In more detail, timing simulation involves examining the temporal aspects associated with the signal propagation throughout the entire circuit. It includes considering the time it takes for signals to traverse various logic gates within the Arbiter PUF structure. As signals progress through the logic gates, they encounter propagation delays, the temporal gaps between the input and output transitions at these gates.

In Arbiter PUFs, the responses intricately depend on the delay differences within the circuit. Therefore, timing simulation becomes imperative for accurately assessing the behavior of the PUF. By accounting for propagation delays, the simulation captures the temporal intricacies that influence the responses generated by the Arbiter PUF.

To conclude, timing simulation provides a dynamic view of the Arbiter PUF's operation, considering the temporal characteristics introduced by the propagation delays. This meticulous analysis ensures a more realistic representation of the PUF's behavior, enhancing our understanding of its performance and aiding in identifying potential issues related to timing constraints within the circuit.

1) When D input is faster than the clock

In a rising edge D (Data) flip-flop, the state of the D input is transferred to the output (`response_o`) at the moment of a rising edge of the clock(clk) signal. The flip-flop essentially "captures" the state of the D input atthat moment and holds it until the next rising edge of the clock.

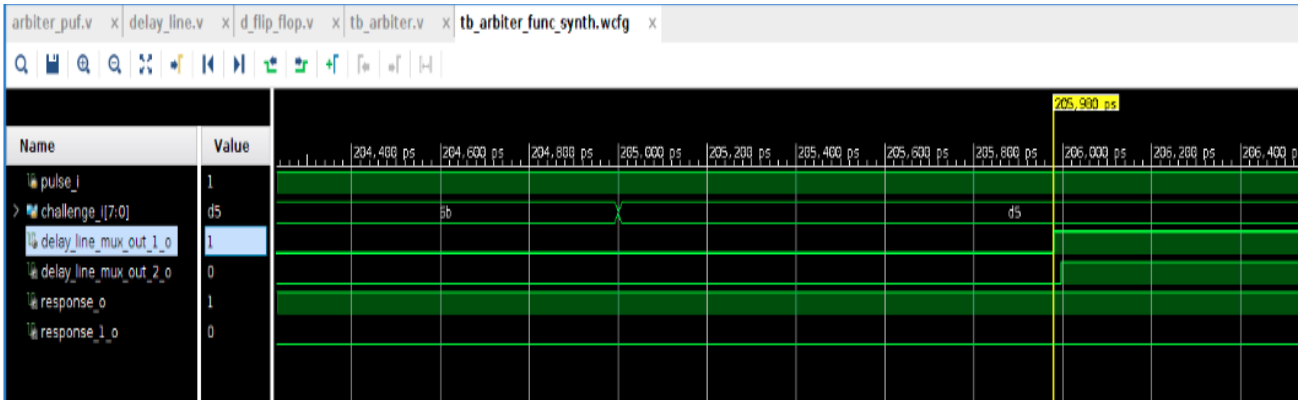

Figure 3.6 Arbiter PUF post implementation timing simulation d_in is faster than clk

In this situation from Figure 3.6 , the data input (d_in) turns high before the arrival of the clock signal's rising edge. Consequently, d_in is stable when the clock rises, and this state is captured as the output. Therefore, the output of the first D flip-flop, denoted as `response_o` , becomes 1.

2) When D input is slower than the clock

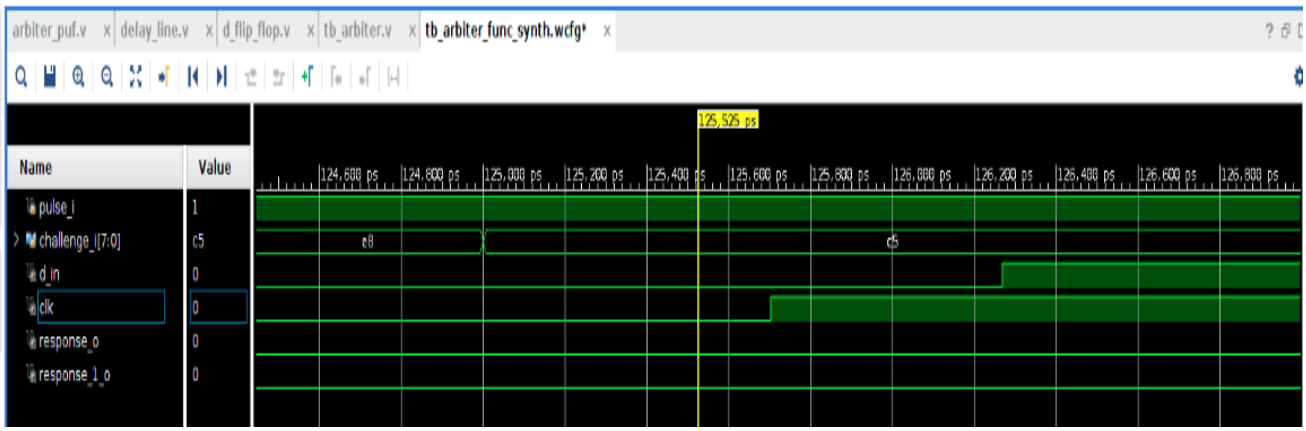

Figure 3.7 Arbiter PUF post implementation timing simulation d_in is slower than clk

Here, data input (`d_in`) remains low (or becomes low) before the arrival of the clock signal's rising edge, then `d_in` will be stable at a low state when the clock rises. Consequently, this state is captured by the D flip-flop. This leads to the output of the first D flip-flop, referred to as `response_o`, remaining at 0.

CHAPTER 4 XOR Arbiter PUF implementation in Verilog using Xilinx Vivado

Introducing the XOR (exclusive OR) operation in an Arbiter Physical Unclonable Function (PUF) is a strategic enhancement to boost its security, particularly against modeling attacks. This augmentation contributes to an overall improvement in the unpredictability of the PUF's response, reinforcing its effectiveness as a security measure.

In the context of the proposed 3-1 XOR Arbiter PUF, a notable adaptation involves the incorporation of three distinct Arbiter chains. Each of these chains generates an individual response. The innovative aspect lies in the subsequent application of the XOR operation to these individual responses, forming the final 1-bit response for the PUF.

Using three Arbiter chains and the XOR operation introduces extended non-linearity and diffusion in the PUF's response generation mechanism. Non-linearity is crucial for preventing attempts at modeling the PUF, as the relationship between the challenges and responses becomes less predictable. The XOR operation, in this case, combines the outputs of the three Arbiter chains in a way that ensures the final response is not a straightforward linear combination of the individual responses, and the XOR operation contributes to diffusion, spreading the influence of each Arbiter chain's response across the final 1-bit response. This diffusion aspect is advantageous for dispersing the impact of any localized variations or perturbations, adding another layer of complexity to the PUF's behavior.

4.1 Simulation of 3-1 XOR Arbiter PUF

In this section, we discussed the simulation of 3-1 XOR arbiter PUF. This project is tested on the Artix-7 CSG324100T FPGA board. This shows the behavior of arbiter PUF in Vivado synthesizer.

There are 5 Module in the design and their instantiation flow diagram is shown in the Figure 4.1 below.

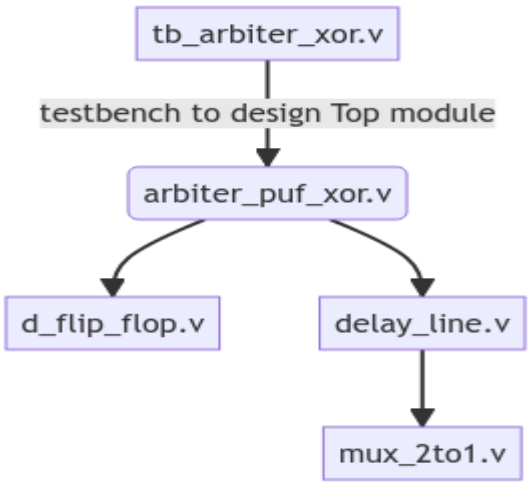

Figure 4.1 Instantiation flow of 3-1 XOR Arbiter PUF Design

Port	Width	Direction	Description
pulse_1_i	1 bit	Input	It activates the first stage of the MUXes for 1st chain of arbiter
pulse_2_i	1 bit	Input	It activates the first stage of the MUXes for 2nd chain of arbiter
pulse_3_i	1 bit	Input	It activates the first stage of the MUXes for 3rd chain of arbiter
challenge_1_i,	Variable	Input	Based on the the width it indicates number of MUXes for 1 st chain.
challenge_2_i,	Variable	Input	Based on the the width it indicates number of MUXes for 2nd chain.
challenge_3_i,	Variable	Input	Based on the the width it indicates number of MUXes for 2nd chain.
response_1_o, response_2_o, response_3_o	1 bit	Output	Outputs from each arbiter chains
response_o	1 bit	Output	Final Xor'ed response of 3 arbiter chian
C_LENGTH	Variable	Input	Number of MUXes

Table 4.1 3-1 XOR Arbiter PUF Port Descriptions

In the Verilog coding context, the notation '_i' is used to denote the input port, while '_o' is used for the output port. C_LENGTH is a global parameter for this design to set the number of Multiplexer in a chain.

1.**Challenge Bits Configuration**: The XOR arbiter PUF comprises multiple parallel arbiter chains. Each chain requires a set of challenge bits *c*(*i*) to configure the switching stages. The number of bits needed for each challenge depends on the number of stages in each arbiter chain. This challenge length is flexible and can be varied as needed, allowing for challenge lengths of 8, 16, 32, and 64 bits. Depending on the application and design requirements.

2.**Pulse Generation**: A single-bit pulse is generated and provided as input to all the arbiter PUF chains. This pulse travels through each chain's configured multiplexer (MUX) stages. For our 3-1 XOR arbiter PUF, we would have three pulses (`pulse_1_i, pulse_2_i, pulse_3_i`) - one for each arbiter chain.

3.**Response Generation**: As the pulse travels through each MUX stage of an arbiter chain, it determines its path based on the challenge bits c(i). Depending on the configuration set by these challenge bits, the pulse may take longer in one path than another. The final arbiter at the end of each chain, essentially a D flip-flop,

captures the race condition and produces a single-bit response for that chain. For a 3-1 XOR arbiter PUF, these () single-bit responses from the three chains are then XOR' ed together to produce the final single-bit PUF response.

As a result, incorporating three parallel arbiter chains and XOR'ing their outputs in the 3-1 XOR arbiter PUF enhances unpredictability. This configuration provides better resistance against certain types of attacks and increases the unpredictability of the responses compared to a single-chain arbiter PUF.

For example, if an attacker manages to approximate a model of a single arbiter PUF through machine learning or some other method. This means the attacker can input a challenge, and the model will predict the PUF's response with relatively high accuracy. Once this model is constructed, the real PUF becomes vulnerable since the attacker can predict its responses without the requirement of the actual FPGA device.

In the case of the 3-1 Arbiter PUF, we have three parallel arbiter chains, and their responses are XOR 'ed together to produce the final PUF response. Even if an attacker successfully models one of the chains, they need to model all three chains accurately to predict the final XOR 'ed response.

An attacker has individually modeled all three chains with an 80% accuracy. The probability that the attacker correctly predicts the XOR 'ed result (meaning all three predictions are correct) is 0.8×0.8×0.8=0.512 = 0.512 or 51.2%. That is a significant reduction in confidence for the attacker compared to the 80% accuracy for each chain individually.

4.2 Simulation of 3-1 XOR Arbiter PUF in Xilinx Vivado

In Figure 4.2, we illustrate the design of a system that incorporates three distinct instances of the Arbiter PUF. Each of these instances is triggered by its own unique single-bit pulse, denoted as `pulse_1_i`, `pulse_2_i`, and `pulse_3_i`. Importantly, In this design, every arbiter chain possesses an equal count of challenge bits, represented as `challenge_1_i`, `challenge_2_i`, and `challenge_3_i`. These challenge bits directly correspond to the number of multiplexers (MUXes) within each arbiter chain, ensuring uniformity across all three chains.

The three MUX chains, `inst_delay_line_1`, `inst_delay_line_2`, and `inst_delay_line_3`, each contain two parallel MUX sequences. These sequences produce outputs termed as `up_mux_o` and `do_mux_o`. Owing to varying propagation delays in each MUX, based on their corresponding challenge values, the propagation delays in each MUX vary, leading to the outputs not reaching the D flip-flops same simultaneously. The challenge value further influences this.

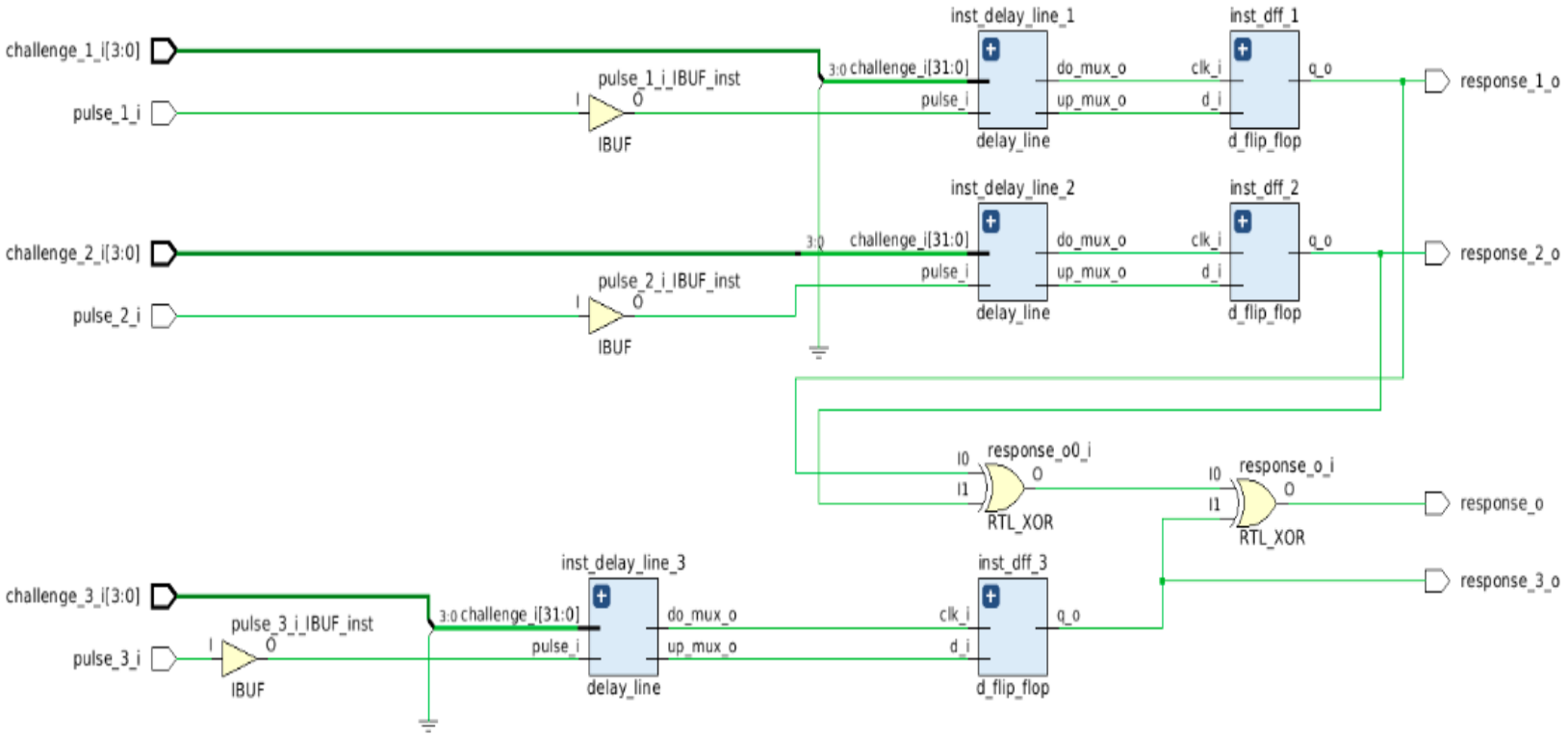

Figure 4.2 Schematic of 3-1 XOR Arbiter PUF

The three MUX chains, **inst_delay_line_1**, **inst_delay_line_2**, and **inst_delay_line_3**, contain two parallel MUX sequences. These sequences produce outputs termed as **up_mux_o** and **do_mux_o**. Due to varying propagation delays in each MUX, based on their corresponding challenge values, the propagation delays in each MUX vary, leading to the outputs not reaching the D flip-flops simultaneously. The challenge value further influences the response of the physical unclonable function.
The operation of each Arbiter PUF is consistent with the detailed explanation provided in Chapter 4. After processing through their respective chains, the three resulting single-bit outputs — `response_1_o`, `response_2_o`, and `response_3_o` — are aggregated using an XOR operation. At last, this combination produces the final PUF output, labeled as `response_o`.

4.2.1 3-1 XOR Arbiter PUF behavioral simulation

Behavioral simulation of an Arbiter PUF implies testing its high-level design functionality, excluding factors like timing, delays, and physical attributes. It typically involves inputting varied challenges and confirming the expected responses. The goal is to ensure the logical correctness of the design before proceeding to detailed simulations and, ultimately, physical implementation.

When the input pulses (`pulse_1_i, pulse_2_i, pulse_3_i`) is set to 0, all the 2x1 multiplexers receive an input of 0. However, regardless of the state of the selection lines, a 0 is propagated through to the final multiplexer stage due to the 0 value of input in each arbiter PUF. As a result, the output `response_o`, is observed as "0". As XOR of three 0 would be resulted as 0.

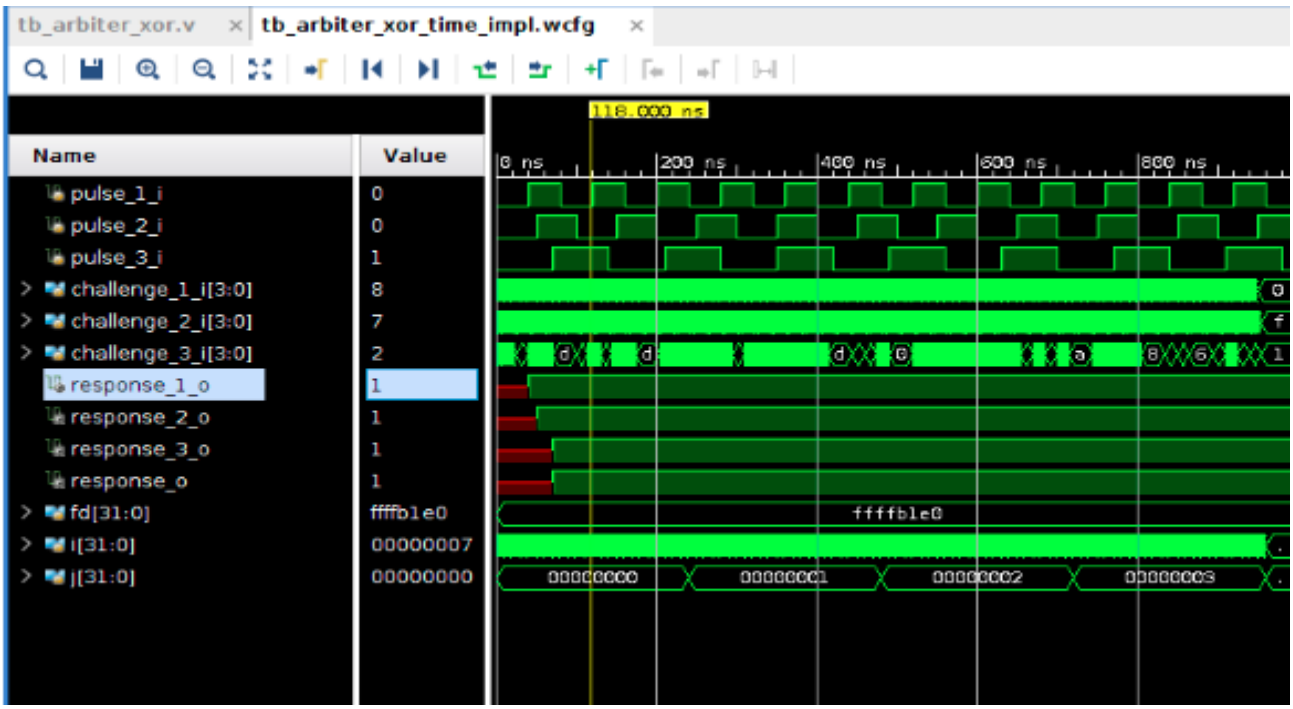

Figure 4.3 3-1 XOR Arbiter PUF Behavioral simulation

When the input pulses (`pulse_1_i, pulse_2_i, pulse_3_i`) switch to 1, that means all 3 MUX gets

inputs and now according to the challenge provided the behavior of the system changes accordingly. As this is a behavioral simulation, it does not consider the propagation delay of the circuit, so the D flip-flop (D-FF) retains its previous state, which in this instance is 1. Therefore, the output does not revert to 0, ensuring the system response reflects the updated initial input value. In this case, where D – FF behaves like a 1-bit memory element it stays in the same state.

4.2.2 3-1 XOR Arbiter PUF post implementation timing simulation

Timing simulation of an XOR Arbiter PUF refers to the process of simulating the design while considering the timing characteristics of each arbiter PUF in the design. This simulation type considers the propagation delays that occur when signals pass through the logic gates, which is particularly important in Arbiter PUFs as their responses are based on delay differences.

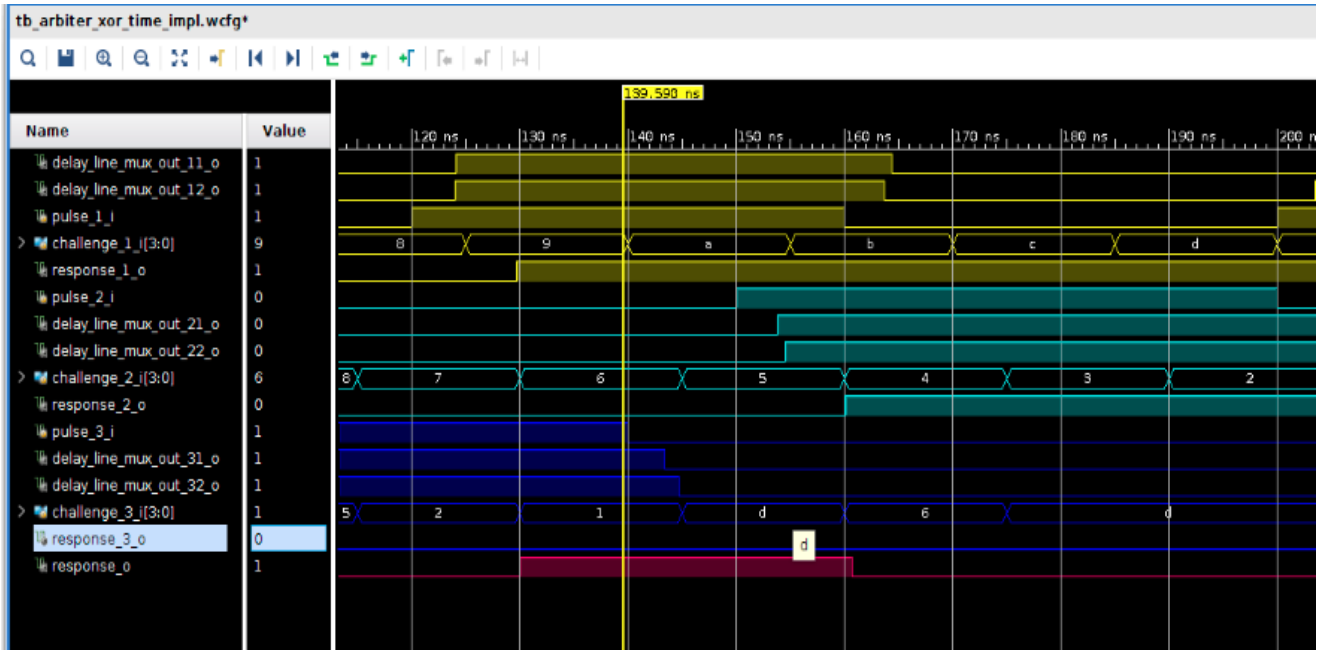

Figure 4.4 3-1 XOR Arbiter PUF post implementation timing simulation

As illustrated in the referenced Figure 4.4, the outputs `delay_line_mux_out_21_o` and `delay_line_mux_out_22_o` originates from the last multiplexer's upper and lower stages of Arbiter PUF 2. Due to the varying values of `challenge_2_i`, the two MUX outputs do not arrive at the D flip flop simultaneously. Consequently, without knowledge of `challenge_2_i`, the `response_2_o` value becomes unpredictable with each iteration. Same thing happens with `response_1_o` and `response_3_o`.

In the post-implementation timing simulation of the 3-1 XOR Arbiter PUF, individual responses are generated from each of the three Arbiter PUFs, denoted as `response_1_o`, `response_2_o`, and `response_3_o`. The final output, `response_o`, is derived by XOR-ing these three individual responses together.

CHAPTER 5 Double Arbiter PUF implementation in Xilinx Vivado

This variant aims to mitigate certain limitations inherent in the XOR Arbiter PUF while preserving or enhancing its resilience against modeling attacks.

The XOR Arbiter PUF represents a well-established advancement over the conventional Arbiter PUF. It achieves this by employing XOR operations on the outputs of multiple parallel Arbiter PUFs, thereby intensifying the system's complexity and rendering it more resistant to machine learning attacks. The justification behind this approach lies in the exponential increase in complexity as the number of XOR stages (parallel Arbiter PUFs) rises. This complexity is a deterrent against effective modeling, making it significantly more challenging for machine learning algorithms to replicate the PUF's behavior accurately.

However, a potential drawback of escalating the number of XOR stages is a potential decrease in the PUF's reliability. This is attributed to heightened sensitivity to environmental conditions, such as temperature and voltage. The more stages involved, the more intricate and susceptible the system becomes to variations in these conditions.

In response to this trade-off, the Double Arbiter PUF is introduced, offering a nuanced solution that optimizes the delicate balance between reliability, security, and resource efficiency. The Double Arbiter achieves this by generating a 6-bit response through the utilization of three arbiter chains, effectively doubling the response length compared to the XOR Arbiter PUF. This design choice allows for a more intricate structure within the constraints of the same resource allocation as the XOR Arbiter PUF. By doing so, the Double Arbiter seeks to provide an optimal compromise that enhances security while maintaining practical resource efficiency and reliability.

5.1 Simulation of 3-1 Double Arbiter PUF

In this section, we have discussed the simulation of Double arbiter PUF. This project is tested on the Artix7CSG324100T FPGA board. It demonstrates the behaviour of the Double Arbiter PUF in actual FPGA.

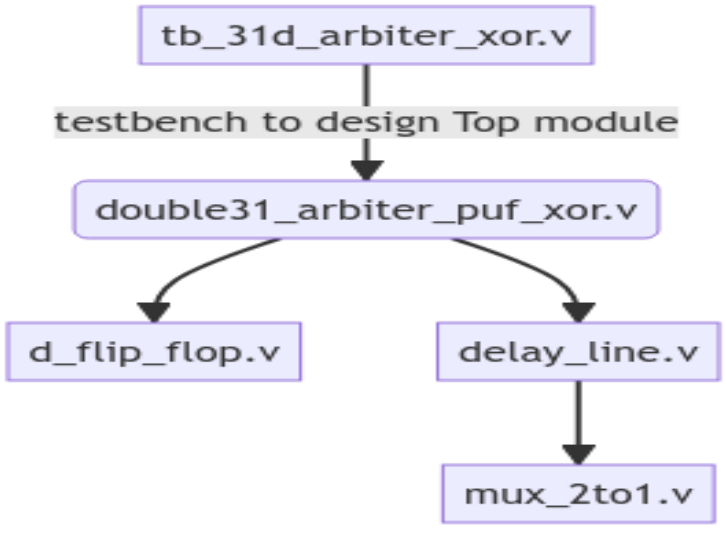

Figure 5.1 Instantiation flow of 3-1 Double Arbiter PUF Design

There are 5 Module in the design and their instantiation flow diagram is shown in the Figure 5.1 below.

Port	Width	Direction	Description
pulse_1_i	1 bit	Input	It activates the first stage of the MUXes for 1st chain of arbiter
pulse_2_i	1 bit	Input	It activates the first stage of the MUXes for 2nd chain of arbiter
pulse_3_i	1 bit	Input	It activates the first stage of the MUXes for 3rd chain of arbiter
challenge_1_i,	Variable	Input	Based on the the width it indicates number of MUXes for 1 st chain.
challenge_2_i,	Variable	Input	Based on the the width it indicates number of MUXes for 2nd chain.
challenge_3_i,	Variable	Input	Based on the the width it indicates number of MUXes for 2nd chain.
response_11_o, response_12_o	1 bit	Output	Outputs from first arbiter chains upper and lower D flip flop
response_21_o, response_22_o	1 bit	Output	Outputs from second arbiter chains upper and lower D flip flop
response_31_o, response_32_o	1 bit	Output	Outputs from third arbiter chains upper and lower D flip flop
response_o	1 bit	Output	Final Xor'ed response of 6 DFF from 3 arbiter chain
C_LENGTH (parameter)	Variable	Input	Number of MUXes

Table 5.1 3-1 Double Arbiter PUF Port Descriptions

In the Verilog coding context, from Table 5.1, for all the ports, the notation '_i' is used to denote the input port, while '_o' is employed for the output port. C_LENGTH is a global parameter for this design to set the number of multiplexers in a chain.

1. **Challenge Bits Configuration**: The 3-1 Double arbiter PUF comprises multiple parallel arbiter chains. Each chain requires a set of challenge bits *c*(*i*) to configure the switching stages. The total number of bits needed for each challenge depends on the number of stages in each individual arbiter chain. This challenge length is flexible for each Arbiter Stages, allowing for challenge lengths of 8, 16, 32, 64 bits but it must be same in all three stages.

2. **Pulse Generation**: A single-bit pulse is generated and provided as an input to all the arbiter PUF chains. This pulse travels through each chain's configured multiplexer (MUX) stages. For our 3-1 Double arbiter PUF, we would have three pulses (`pulse_1_i, pulse_2_i, pulse_3_i`) - one for each arbiter chain.

3. **Response Generation**: Within each arbiter chain, the pulse navigates through multiple MUX stages, with its route being defined by the challenge bits, denoted as c(i). The configuration of these challenge bits can cause variations in the pulse's traversal time. Specifically, in a configuration with three parallel arbiter PUF MUXes (comprising three upper-stage MUXes and three lower-stage MUXes), there are six distinct pathways to determine the input and the clock for the corresponding 6 D flip-flops. The outputs of these flip-flops are then combined using an XOR operation, culminating in the final response, termed as response.

5.2 Simulation of 3-1 Double Arbiter PUF in Xilinx Vivado

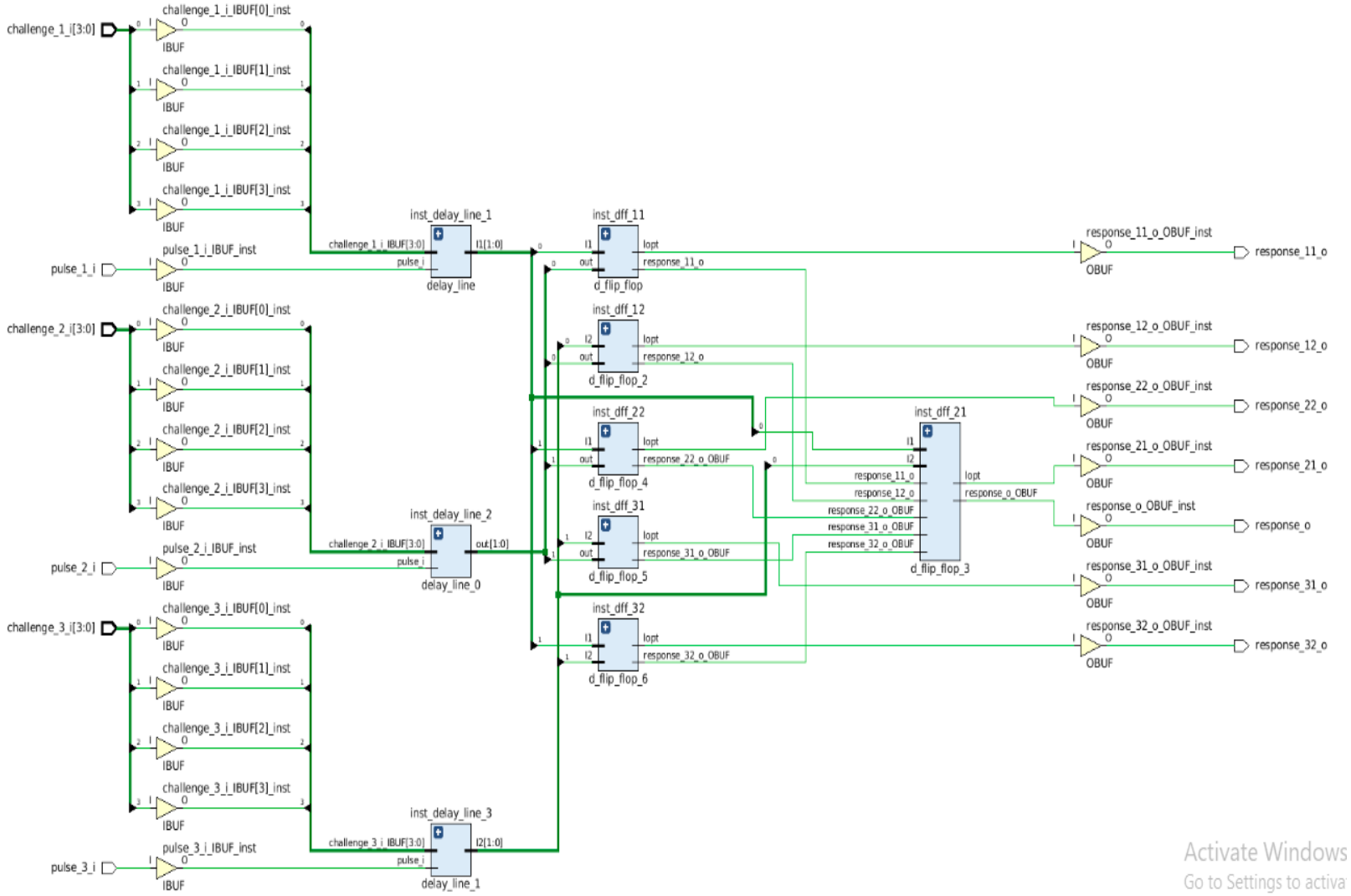

Figure 5.2 Schematic of 3-1 Double Arbiter PUF

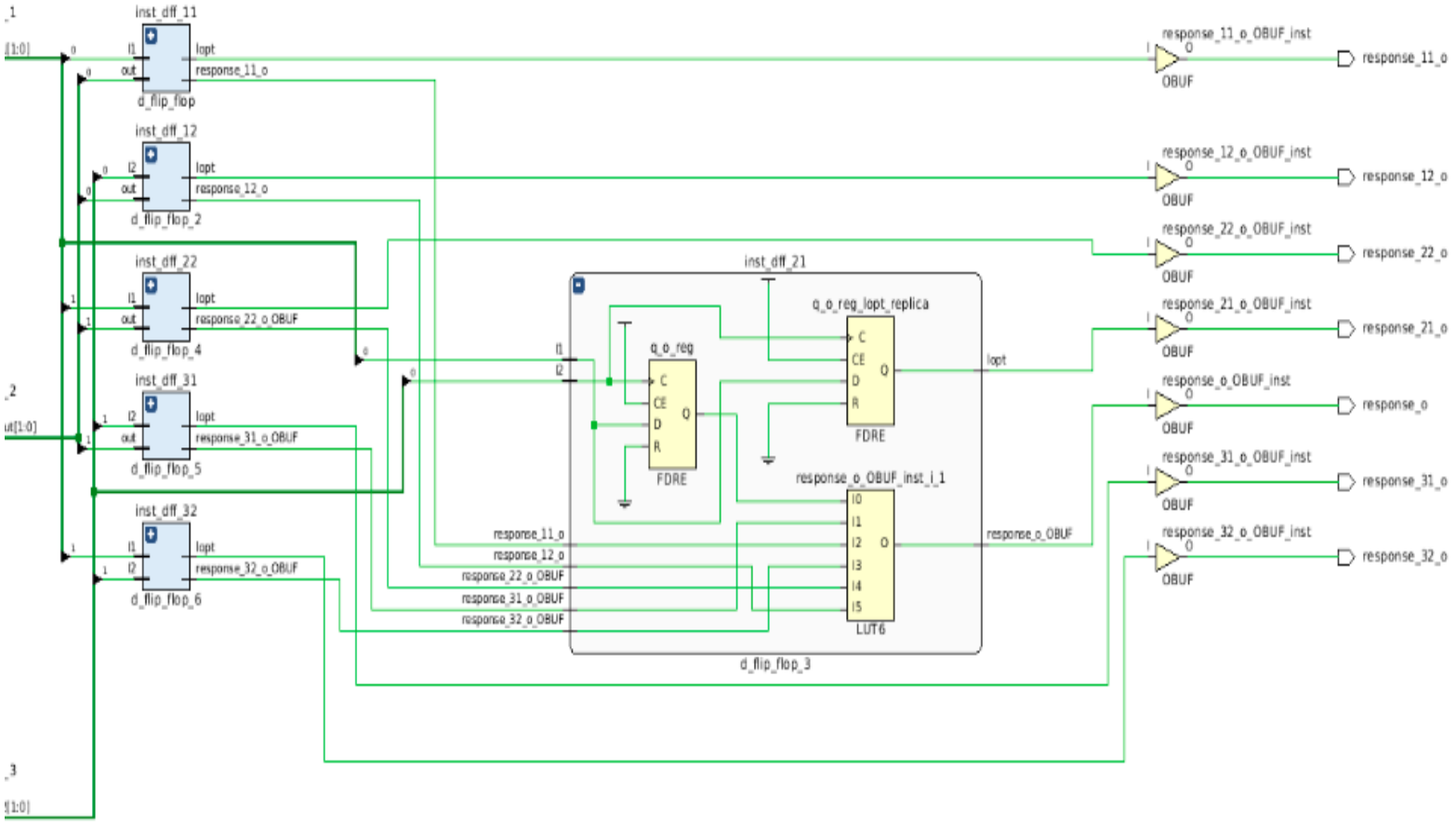

Figure 5.3 Schematic of 3-1 Double Arbiter PUF: closer look of D -FF configuration

The operation of each individual Arbiter PUF is consistent with the detailed explanation provided in Chapter 3. After processing through their respective chains, we will get 6 port as MUX out (`delay_line_mux_out_11_o;delay_line_mux_out_12_o;delay_line_mux_out_21_o, delay_line_mux_out_22_o,delay_line_mux_out_31_o, delay_line_mux_out_32_o`) which is being feded to the Input to the six D flip flop (D_in and clock) in such combinations such that it results single-bit outputs —`response_11_o,response_12_o,response_21_o, response_22_o,response_31_o,response_32_o`— are aggregated using an XOR operation. At last, this combination produces the final PUF output, labeled `response_o`.

5.2.1 Double Arbiter PUF Behavioral simulation

As mentioned in Chapter 3 and 4 Behavioral simulation of an Arbiter PUF implies testing its high-level design functionality, excluding factors like timing, delays, and physical attributes. It typically involves providing various challenges and confirming the expected responses. The goal is to ensure the logical correctness of the design before proceeding to detailed simulations and, ultimately, physical implementation.

In the case of 3-1 Double arbiter PUF, we have three arbiter chains for 3-1 Double arbiter PUF, so When the input pulses (`pulse_1_i, pulse_2_i, pulse_3_i`) are set to 0, all the 2x1 multiplexers receive an input of 0. However, regardless of the state of the selection lines, a 0 is propagated through to the final multiplexer stage due to the 0 value of the input in each arbiter PUF chain. As a result, the output, response_o, is observed as "0". An XOR of three zeroes would result in 0.

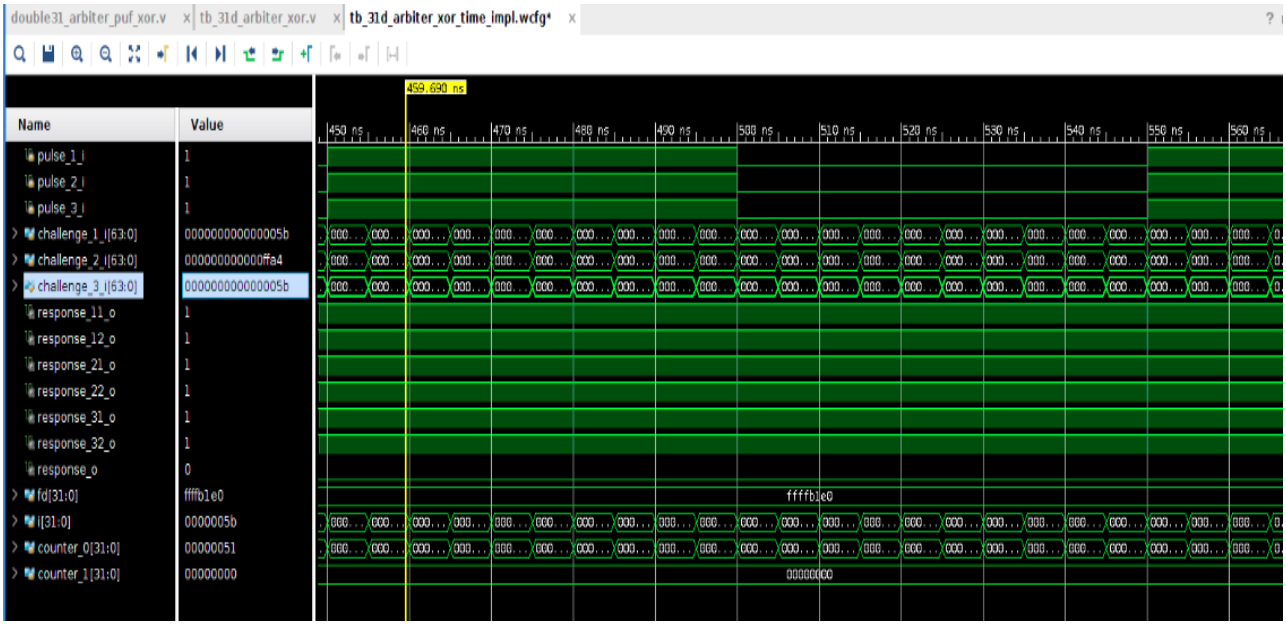

Figure 5.4 3-1 Double Arbiter PUF behavioral simulation

When the input pulses (`pulse_1_i, pulse_2_i, pulse_3_i`) switch to 1, that means all 3-arbiter chain

gets inputs. According to the challenge value provided, the system's behavior changes accordingly. As this is a behavioral simulation, it does not consider the propagation delay of the circuit and other path delays. Hence, the D flip-flop (D-FF) retains its previous state, which in this instance is 1. Therefore, the output does not revert to 0, ensuring the system response reflects the updated initial input value. This is the case where D – FF behaves like a 1-bit memory element and stays in the same state.

5.2.2 Double Arbiter PUF post implementation timing simulation

Timing simulation of a double-arbiter PUF refers to the process of simulating the design while considering the timing characteristics of each arbiter PUF in the design. This simulation type considers the propagation delays when the signal passes through the logic gates. It is essential in PUFs as their responses rely on delay differences. 3-1 Double arbiter PUF has their MUXout (6 ports) going to the input to the 6-D flip flop in variations and generates a 1-bit final response by doing XOR of the 6-D- flip flop outputs.

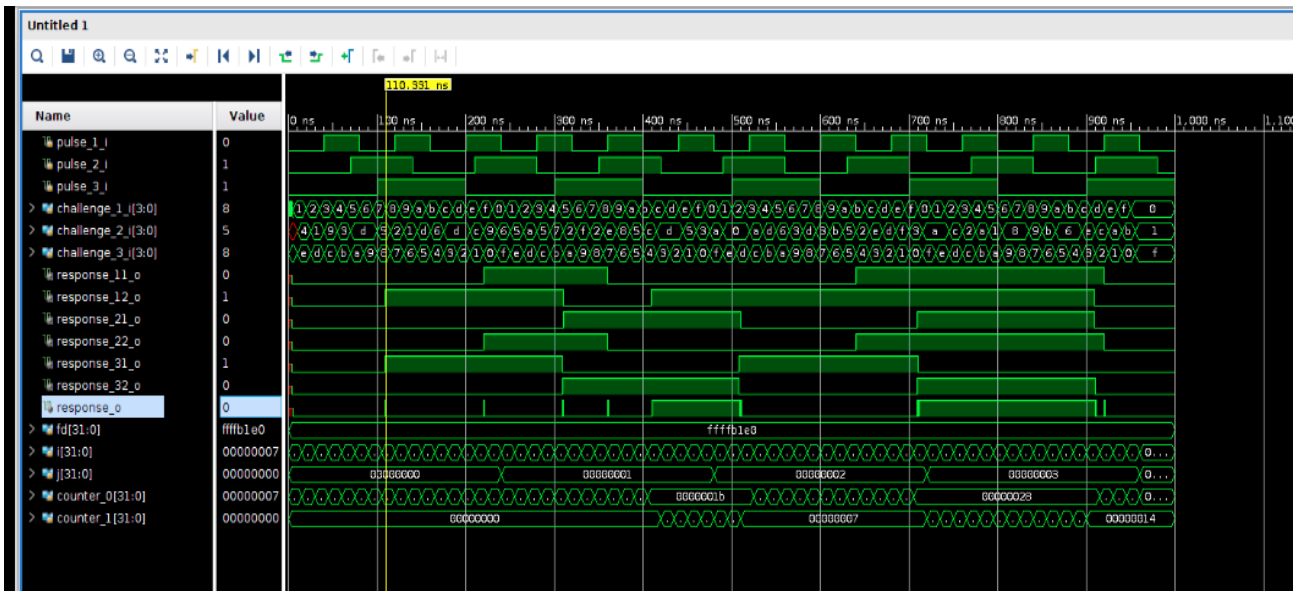

Figure 5.5 3-1 Double Arbiter PUF post implementation timing simulation

As depicted in Figure 5.5 above, the 3-1 Double Arbiter PUF exhibits a configuration with three arbiter chains, the same as the 3-1 XOR Arbiter PUF. However, in the case of the 3-1 Double Arbiter PUF, a more intricate structure with approximate resource utilization has been introduced. This design aims to enhance unpredictability and to get resistance against machine learning attacks.

The input ports are denoted as `pulse_1_i` and `challenge_1_i` in the first arbiter chain. The two-multiplexer stages yield `response_11_o` and `response_12_o`, corresponding to the upper and lower stages, respectively. The second arbiter chain features input ports `pulse_2_i` and `challenge_2_i`, producing `response_21_o` and `response_22_o` from its upper and lower multiplexer stages. Similarly, the third arbiter chain employs input ports `pulse_3_i` and `challenge_3_i`, generating `response_31_o` and `response_32_o` from its upper and lower multiplexer stages. All stages of the Multiplexer come at different times because of the propagation delay of the MUXes, which affects All three

chains' output from the Multiplexer. As a result, the D-flip flop output also gets affected, and this unknown propagation delay of each FPGA Device plays a significant role in all physical unclonable functions.

All six outputs from the multiplexer stages across the three arbiter chains feed into a 6-D flip-flop input in the subsequent stage. The final output is generated by XOR'ing the outputs of the six D flip flops, resulting in a single-bit response. This configuration is designed to enhance the complexity of the structure, aiming to increase unpredictability and resilience against machine learning attacks.

CHAPTER 6 Results and Discussion

In this chapter, we will analyze Arbiter PUF Implementation using Xilinx VIVADO on 7-series FPGAs and compare the behavior of all three PUF variants regarding the evaluation metrics of the PUF randomness, uniqueness, and stability. As discussed in Chapter 2, Randomness and uniqueness should be 50 percent for the ideal PUF, and stability should be 100 percent. Let us discuss the results obtained from the analysis.

6.1 Randomness Results

Randomness refers to the unpredictability and lack of discernible patterns in the responses generated by the PUF for various challenges. A PUF's response should ideally be random, meaning that given a challenge, it is computationally infeasible to predict the corresponding response based solely on previous challenge-response pairs or any inherent patterns. This randomness ensures the security and uniqueness of each PUF instance.

In terms of zeros and ones in the total responses, randomness refers to the equal and unpredictable distribution of '0's and '1's in the output. That means that over a large set of responses, approximately 50% of the bits should be '0' and 50% should be '1'. No discernible pattern or bias towards '0' or '1' should exist. In an ideal random sequence, the probability of any given bit being a '0' or '1' is equal. Previous bits in the sequence do not provide any information about subsequent bits, making the system secure and unpredictable. A 50 percent distribution makes it more uncertain for the attacker as it gives maximum entropy.

In this section, we will compare the randomness of the 3-1 XOR APUF to that of the 3-1 DAPUF.

```
Procedure CalculateRandomness(filename):
    // Read the file content into a data variable
    Open the file with name 'filename' in read mode as 'file'
        Read the entire content of 'file' into a variable called 'data'
    Close the file

    // Split the data into lines
    Split 'data' into a list of strings called 'lines', using newline as a separator

    Initialize an empty list called 'responses'

    // Extract the responses (0's and 1's) from the lines
    For each 'line' in 'lines':
        If 'line' is not empty:
            Split 'line' using ':' as separator and store the last element as 'response'
            Trim spaces from 'response'
            If 'response' is "0" or "1":
                Append 'response' to 'responses' list

    // Count the number of 0's and 1's in the 'responses' list
    Set 'count_0' to the count of "0" in 'responses'
    Set 'count_1' to the count of "1" in 'responses'

    // Calculate the percentage of randomness for 0's and 1's
    If 'responses' list is not empty:
        Set 'randomness_0' to (count_0 divided by length of 'responses') multiplied by 100
    Else:
        Set 'randomness_0' to 0
    Set 'randomness_1' to 100 minus 'randomness_0'
```

Table 6.1 Pseudocode for randomness calculations

6.1.1 Randomness comparison 8-bit input

Exploring various variations in challenges input in a limited set of around 60 iterations, an analysis is conducted to understand the dynamic behavior of the 3-1 Double Arbiter PUF and 3-1 XOR Arbiter PUF, explicitly focusing on the challenges posed by all three arbiter chains (`challenge_1_i`, `challenge_2_i`, `challenge_3_i`).

Listed below are some of the cases with 8-bit inputs:

3-1 XOR Arbiter PUF:

challenge_1_i	challenge_2_i	challenge_3_i	Number of 0's	Number r of 1's
DEC	DEC	DEC	21	39
INC	INC	INC	30	30
Random	Random	Random	26	34
INC	DEC	Random	35	25
15	Random	DEC	21	39
INC	Random	DEC	30	30

Table 6.2 3-1 XOR Arbiter PUF randomness results

3-1 Double Arbiter PUF:

challenge_1_i	challenge_2_i	challenge_3_i	Number of 0's	Number of 1's
DEC	DEC	DEC	50	10
INC	INC	INC	40	20
Random	Random	Random	31	29
INC	DEC	Random	50	10
INC	Random	DEC	41	19

Table 6.3 3-1 Double Arbiter PUF randomness results

To Calculate the randomness:

- Calculate the proportion of 1's and 0's in the sequence.
- A perfectly random sequence will have 50% of each.

Given a sequence, you can compute this ratio:

Ratio of 1's=Number of 1's / Total number of bits

Ratio of 0's=Number of 0's/ Total number of bits

For a 60-response sequence:

If you have *n* '1's, then the randomness or the proportion of '1's is n /60*n*.

Similarly, the proportion of '0's is (60−*n) / 60* .

For example: for the case from Table 6.3 where all 3 challenges are random.

randomness or the proportion of 0's : 26 /60 = 43.33 = 43,
randomness or the proportion of '1's : 34 /60 = 56.66 = 57

The results are based on a limited dataset of only 60 iterations involving the inputs pulse_m_i and challenge_n_i (where m,n = 1,2,3). Due to this constrained sample size, the randomness metric might not reflect the actual behavior of the PUF. This should provide a more accurate representation, ideally bringing the randomness closer to an ideal 50 percent.

If we want to conclude from the table 6.3 and 6.4, 3-1 XOR arbiter PUF performance for the randomness is superior compared to 3-1 Double Arbiter as in the previous one, we have an odd number of XOR gates that help to make an a number of zeros and a number of one's count nearly equal. Moreover, this is also being observed in the reference paper [14].
Randomness evaluation of 3-1 XOR PUF for 200 responses:

3-1 XOR Arbiter PUF:

challenge_1_i	challenge_2_i	challenge_3_i	Number of 0's	Number of 1's
DEC	DEC	DEC	159	42
INC	INC	INC	110	90
Random	Random	Random	97	103
Random	Random	Random	92	108
INC	DEC	Random	103	97
457	Random	DEC	135	65
1023	Random	DEC	100	100
INC	Random	DEC	128	72

Table 6.4 | 3-1 XOR Arbiter PUF randomness results for 200 responses

From the Table 6.4, it is evident that in certain instances, the randomness measures above 45% and approaches 49%. Thus, for 200-bit responses, the outcomes can be deemed satisfactory.

3-1 Double Arbiter PUF:

challenge_1_i	challenge_2_i	challenge_3_i	Number of 0's	Numbe r of 1's
Random	Random	Random	117	83
DEC	INC	Random	118	82
1023	Random	DEC	86	114

Table 6.5 | 3-1 Double Arbiter PUF randomness results for 200 responses

If we compare the results of 60 and 200 responses from Tables 6.4 and 6.5, then when all Arbiter chains have Random chains to all 10 MUXes,
(C_LENGTH= 10 bit = 10 MUXes)
for the case where all three challenges are random :

Randomness or the proportion of 0's: 97 /200 = 43.33 = 0.485
Randomness or the proportion of 1's: 103 /200 = 56.66 = 0.515

From the above observation of 200 iterations, a more significant number of responses gives a more accurate result. We encountered issues recording responses for extended bit lengths due to simulation time constraints. We have adjusted the simulation time from 10ns to 400us to address this. As a result, we can store more Challenge response pairs in the text file for accurate performance evaluation.

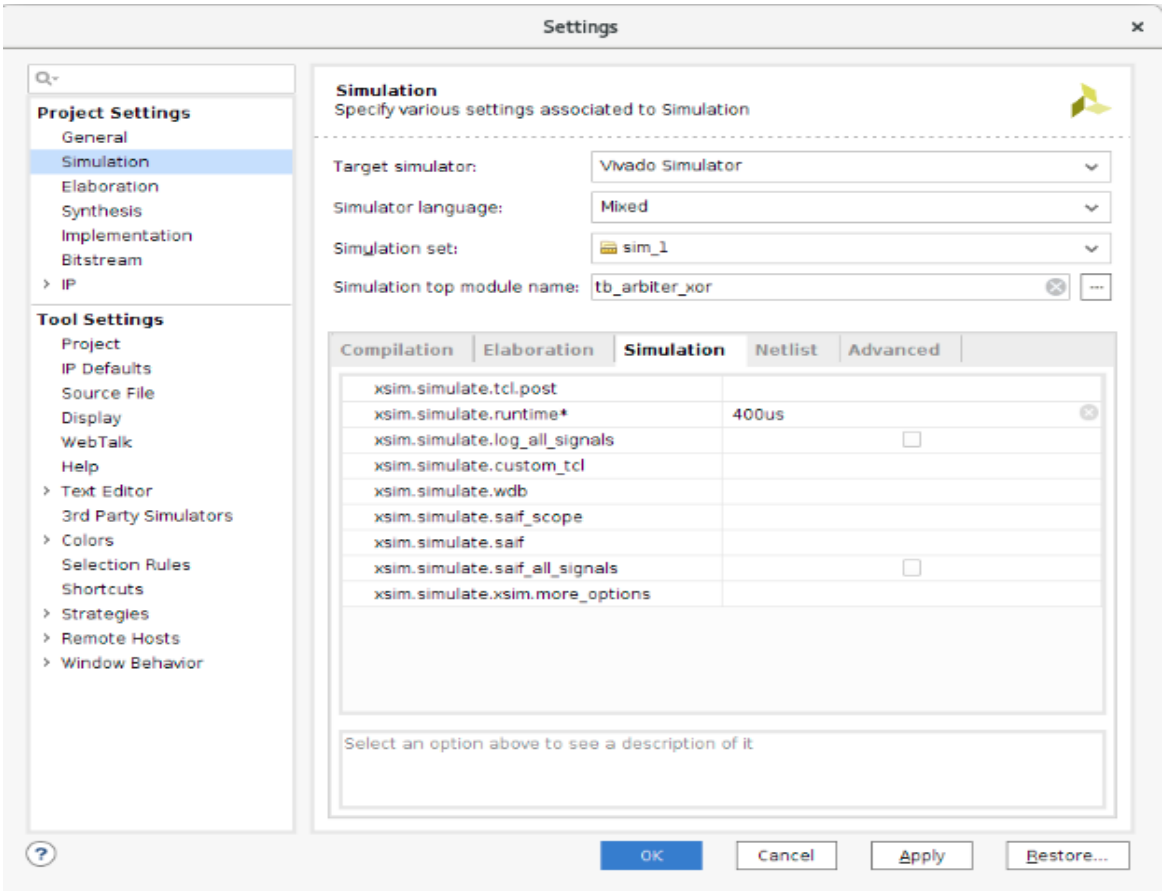

Figure 6.1 Simulation settings to change the runtime.

6.1.2 3-1 Randomness comparison 16 -bit input for 65K iterations

FPGAs	INC		DEC		RAND	
	0's	1's	0's	1's	0's	1's
Artix 7	31940	33596	23028	42508	23093	42443
	48.74%	51.26%	35.14%	64.86%	35.24%	64.76%
KCU1500	119905	45631	19761	45775	15705	49831
	30.37%	69.63%	30.15%	69.85%	23.96%	76.04%
VC707	16988	48548	17570	47966	17312	48224
	25.92%	74.08%	26.81%	73.19%	26.81%	73.5%

Table 6.6 Arbiter PUF randomness results for 65k responses

Based on Table 6.6, it shows randomness results for Arbiter PUF when presented with Random Challenges, the Artix7 device exhibits a randomness measure of approximately 36%. In contrast, the other two devices demonstrate a reduced degree of randomness.

3-1 XOR Arbiter PUF randomness analysis

FPGAs	Challenges (All chains (1,2,3 random)	Number of 0's	Number of 1's
Artix7	**RAND**	32424 49.48%	33111 50.52%
VC707	**RAND**	32273 49.24%	33263 50.76%
KCU1500	**RAND**	32718 49.92%	32818 50.08%

Table 6.7 XOR Arbiter PUF randomness results for 65k responses

3-1 Double Arbiter PUF randomness analysis

FPGAs	Challenges (All chains (1,2,3 random)	Number of 0's	Number of 1's
Artix7	**RAND**	33230 50.72%	32305 49.28%
VC707	**RAND**	32472 49.55%	33064 50.45%

Table 6.8 Double Arbiter PUF Randomness results for 65k responses

From a careful examination of Tables 6.7 and 6.8, we can conclude that the 3-1 XOR APUF and 3-1 DAPUF exhibit notably superior randomness compared to the Arbiter PUF. Their randomness measures gravitate towards an impressive 50±1% threshold, positioning them proximate to the ideal benchmark. It suggests that the 3-1 XOR APUF and 3-1 DAPUF demonstrate robustness and reliability in generating unpredictable results, making them potentially more effective for applications that rely on high levels of entropy and randomness.

6.2 Uniqueness Results

```
Function CalculateUniqueness(filePath1, filePath2) -> uniquenessValue:
    responseSet1 = ReadResponses(filePath1)
    responseSet2 = ReadResponses(filePath2)

    If length of responseSet1 is not equal to length of responseSet2:
        Raise Error "Response sets must have the same length."

    hammingDistance = Count number of differing bits between responseSet1 and
                      responseSet2
    uniquenessValue = (hammingDistance / length of responseSet1) * 100

    Return uniquenessValue

End Function

Function ReadResponses(filePath) -> responses:
    Open file at filePath for reading, assign to fileID
    If fileID is invalid:
        Raise Error "Failed to open file."

    Initialize an empty list responses

    While not end of file:
        Read a line from fileID into tline
        Extract response value from tline using pattern matching.
        If response value exists:
            Append response value to responses list.

    Close field
    Return responses.

End Function
```

Table 6.9 Pseudocode for Uniqueness calculations

Uniqueness refers to the distinctiveness of the PUF responses generated by different PUF instances, even when subjected to the same set of challenges. Ideally, two distinct PUF devices, due to manufacturing variations, should produce different, unique responses to the same challenge. The degree of uniqueness is

often expressed as a statistical measure, indicating how different the responses are across a population of PUF devices. High uniqueness ensures that each PUF instance can be reliably distinguished from others, making it a crucial property for device authentication and individualized key generation.

We evaluate the uniqueness across various FPGA boards by supplying them with a consistent set of inputs and challenges. The uniqueness is then quantified by measuring the Hamming distance between the responses of individual boards.

3-1 XOR and 3-1 DAPUF Uniqueness comparison

FPGAs	3-1 X-OR PUF	3-1 Double APUF
Artix7 and KCU1500	46.36 %	49.51 %
KCU1500 and VC707	49.03 %	50.93 %
Artix7 and VC707	45.67 %	51.70 %

Table 6.10 Uniqueness result comparison between XOR and DAPUF for 16-bit input

From Table 6.10, it is evident that the Double Arbiter PUF exhibits a marginally more uniqueness compared to the X-OR Arbiter PUF. In these evaluations, both PUFs were tested using three 16-bit arbiter challenges: challenge_1_i and challenge_3_i progressed incrementally from 0 to 65535, while challenge_2_i was set in a decremental order, ranging from 65535 to 0.

Arbiter PUF uniqueness

FPGAs	INC	DEC
Artix7 and KCU1500	44.02 %	42.70 %
KCU1500 and VC707	45.02 %	40.13 %
Artix7 and VC707	41.51 %	38.11 %

Table 6.11 Uniqueness result comparison APUF for 16-bit input

The data from the three FPGA boards indicates that the arbiter PUF, when provided with an incremental challenge (0 to 65535). exhibits higher uniqueness compared to when given a decremental challenge (65535

to 0). The uniqueness is approximately around the 50% mark. Moreover, the arbiter PUF performance across these three devices consistently presents a commendable measure of uniqueness.

2-1 XOR Arbiter PUF uniqueness

FPGAs	Artix7	KCU-1500	VCU1525	ZC702	ZCU102-es2
KCU1500 reference	51.81 %	-	44.63 %	49.29 %	50.70 %
VCU1525 reference	43.69 %	44.63 %	-	57.89 %	50.59 %
ZC702 reference	26.35 %	59.29 %	57.88 %	-	52.37 %
ZCU102-es2 reference	53.73 %	50.71 %	50.59 %	42.37 %	-

Table 6.12 Uniqueness result comparison 2-1 XOR APUF for 64-bit input

The Table 6.12 provided offers a uniqueness comparison for the 2-1 XOR arbiter PUF across five FPGA boards. In the 2-1 XOR configuration, there are two APUF chains. Consequently, mixed values are utilized for challenge_1_i, while challenge_2_i features incrementing values ranging from 1501 to 6499. We have observed that uniqueness across different FPGA boards is 45 to 55 percent. It is evident from the observations that the uniqueness across most FPGA boards lies in the range of 45% to 55%.

6.3 Temperature Impact Simulation Module and Stability analysis

This part of report presents an in-depth analysis of the Temperature Impact Simulation module, a specialized component designed for assessing the impact of temperature on digital signal processing. The module's primary focus is on evaluating the uniqueness and randomness of response signals under different temperature conditions.

6.3.1. Functionality and Design

The module is intricately designed to process digital signals, mainly focusing on how temperature variations influence signal characteristics. The module's core functionality lies in its ability to detect variations in signal patterns and correlate these changes with temperature fluctuations.

```
module TemperatureImpactSimulation (
    input clk_i,
```

```
    input reset_i,
    input [0:0] response1_i, // Single bit response input
    input [0:0] response2_i,
    input [7:0] temperature_i,
    output reg uniqueness_flag_o, // Output flag for uniqueness
    output reg randomness_flag_o  // Output flag for randomness
);

// Internal variables
reg [31:0] hammingDistance;
reg [31:0] count_0, count_1;

// File I/O variables
integer fd; // File descriptor
integer i;   // Loop variable for file writing

always @(posedge clk_i or posedge reset_i) begin
    if (reset_i) begin
        hammingDistance <= 0;
        count_0 <= 0;
        count_1 <= 0;
        uniqueness_flag_o <= 0;
        randomness_flag_o <= 0;
    end else begin
        // Process responses for uniqueness
        if (response1_i !== response2_i)
            hammingDistance <= hammingDistance + 1;

        // Process response1_i for randomness
        if (response1_i == 0)
            count_0 <= count_0 + 1;
        else
            count_1 <= count_1 + 1;

        // Determine flags based on temperature_i and counts
        uniqueness_flag_o <= (temperature_i > 150) ? 1'b1 : 1'b0;
        randomness_flag_o <= (count_0 > count_1) ? 1'b1 : 1'b0;
    end
end

// File writing logic
initial begin
   // Open the file for writing at the specified path
   fd = $fopen("C:/.. /temperature_impact_simulation_results.txt", "w");
   if (fd == 0) begin
      $display("Error: unable to open file.");
```

```
  end

  // Write the header
  $fwrite(fd, "Hamming Distance, Count 0, Count 1, Uniqueness Flag, Randomness Flag\n");

  // Assuming a simulation range - modify this as needed
  for(i = 0; i < 1000; i = i + 1) begin
    @(posedge clk_i); // Wait for clock edge

    // Write the results to the file
    $fwrite(fd, "%d, %d, %d, %b, %b\n",
          hammingDistance, count_0, count_1, uniqueness_flag_o, randomness_flag_o);
  end

  // Close the file
  $fclose(fd);
end


endmodule
```

Table 6.13 Stability analysis Verilog code

6.3.2. Stability analysis Verilog port description

The `TemperatureImpactSimulation` module interfaces with its environment through several inputs and outputs. The primary inputs include a clock signal (clk), a reset signal (reset), two single-bit response inputs (response1 and response2), and an 8-bit temperature input. The outputs consist of two flags: `uniqueness_flag` and `randomness_flag`, which provide feedback on the signal characteristics based on the current inputs and temperature changes.

Port	Bit Width	Direction	Description
clk_i	1 bit	Input	Clock signal
reset_i	1 bit	Input	Reset signal
response1_i	1 bit	Input	First single-bit response input
response2_i	1 bit	Input	Second single-bit response input
Temperature_i	8 bits	Input	Temperature values can vary from 0 to 255
uniqueness_flag_o	1 bit	Output	Indicates uniqueness in response signals

randomness_flag_o	1 bit	Output	Indicates randomness in response1

Table 6.14 Stability analysis design port description

6.3.3. Reset Functionality

Upon activation of the reset signal, the module initializes all outputs and internal counters to zero. It ensures that each measurement cycle starts from a known baseline, free from the influence of previous operations. That is active high reset.

The stability of the `TemperatureImpactSimulation` module is significantly highlighted by its response to reset conditions. A crucial aspect of this response is the Arbiter PUF's ability to revert to a predefined initial state promptly. This immediate reset response is essential for ensuring that subsequent operations start from a reliable and known state, not delayed by residual data or effects from previous operations. Equally important is the recovery time post-reset, where a shorter duration denotes a more stable module, allowing for swift resumption of normal operations.

A pivotal aspect of the PUF's stability is its sensitivity to temperature changes, particularly its accuracy in detecting and responding to the predefined temperature threshold of 150 units. This threshold sensitivity is a vital indicator of the PUF performance in temperature-variable environments. Furthermore, the module's stability is also assessed through its consistent response to gradual and abrupt temperature fluctuations. It is imperative that the module accurately adjusts its output flags in response to these temperature changes, thereby reliably reflecting the true impact of temperature on signal uniqueness and randomness.

6.3.4 Sensitivity to Temperature Variations

The module employs a Hamming distance algorithm to compare response1 and response2, thereby assessing the uniqueness of the responses. Additionally, it counts the occurrences of 0 and 1 in response1 to evaluate the randomness of the signal. These measurements are crucial in understanding the signal behavior under different temperature scenarios.

We thoroughly evaluate the stability of all three PUF variants, focusing on their response to reset conditions, sensitivity to temperature variations, and consistency in output flags across different scenarios at different temperatures. By providing different temperatures in the simulation environment, we are checking for the changes that can affect if the temperature of the FPGA device changes.

The reliability of the `uniqueness_flag` under various input conditions is a cornerstone of the physical unclonable function stability. This flag should consistently and accurately reflect the actual state of the input signals, indicating signal uniqueness only when there is a genuine difference in responses. In parallel, the `randomness_flag` must also demonstrate high accuracy, particularly in detecting and indicating the dominance of 0’s or 1’s in response1.The module's ability to assess signal randomness accurately under different input patterns is a critical measure of its stability.
Assessing the Physical unclonable functions long-term stability involves endurance testing, where its

performance is monitored over extended operational periods under varying conditions. This testing helps to ensure that the module's performance remains consistent and reliable over time. Additionally, stress testing plays a significant role in evaluating the PUF's robustness. By subjecting it to extreme and rapid environmental changes, or continuous input signal variations, insights into the Arbiter PUF's ability to maintain stability under challenging and stressful conditions can be observed.

6.3.5 Temperature module implementation in Xilinx Vivado

The internal logic of the `TemperatureImpact` design module utilizes custom-designed digital signal processing methods to analyze input signals as shown in the Figure 6.2 , consider temperature-related variations, and assess how the output deviates from the ideal temperature. This assessment contributes to understanding the impact of temperature fluctuations on the stability of all three variants of Arbiter PUF.

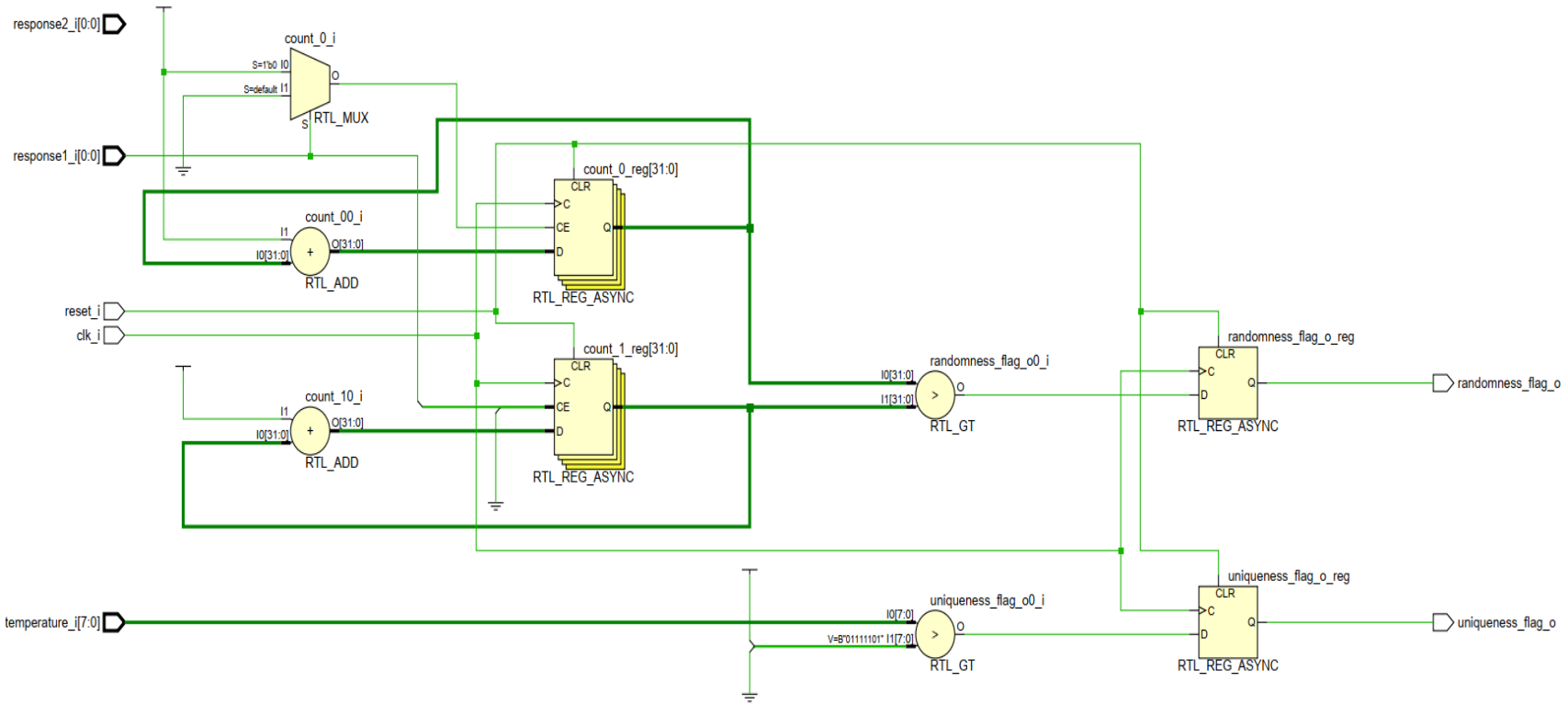

Figure 6.2 Schematic of Temperature Impact module

6.3.6 Average Stability Result Analysis

In this section, we will discuss the impact of temperature on the performance of Physical Unclonable Functions (PUFs). Temperature fluctuations can influence the outcome of PUFs due to alterations in the signal path caused by varying temperatures. Specifically, temperature changes can lead to variations in the propagation delay of signals within the PUF.
It is important to note that this implementation uses the Vivado synthesizer on an Artix 7 FPGA device. The choice of Vivado and the Artix 7 FPGA indicates the specific tools and hardware used for this study, highlighting the analysis of temperature effects on Arbiter PUF and its variants.

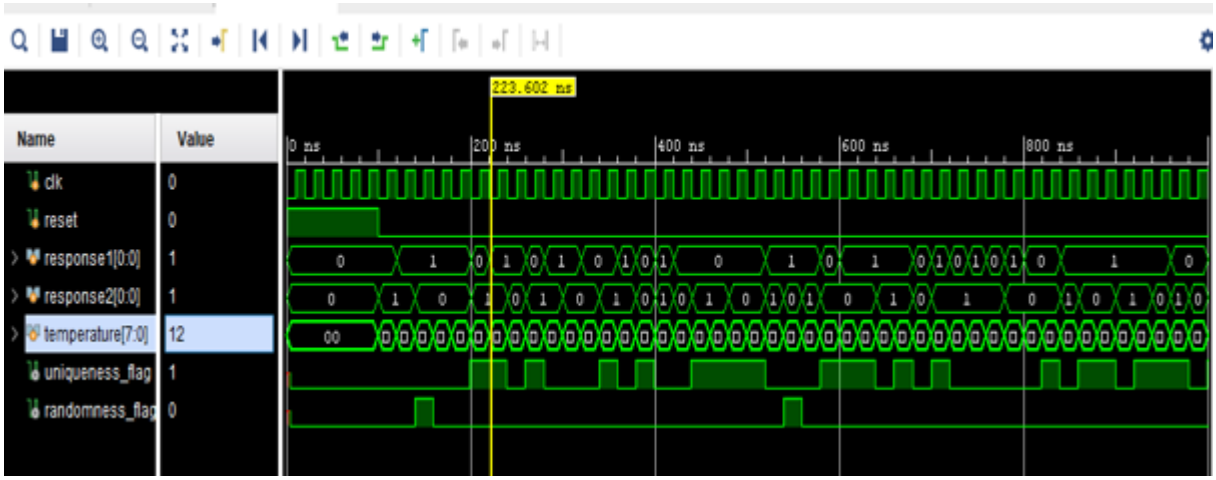

Figure 6.3 Post Implementation Timing simulation of Temperature Impact

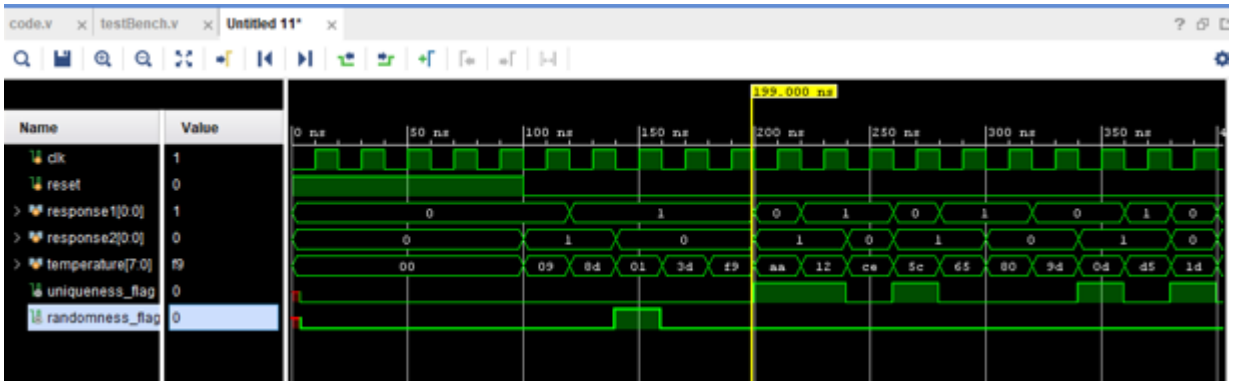

Figure 6.4 Post Implementation Timing simulation of Temperature Impact: closer look

1. The Figure 6.3 and Figure 6.4 displays the results of effect of temperature variations, and it is represented by two flag bits:

- **clk_i**: This is typically the clock input signal. A clock signal is used to synchronize the operations of a digital system. The waveform for clk_i shows regular high (1) and low (0) states, indicating the clock's ticking.

- **reset_i**: This is typically the reset input signal. It is used to bring the system into a known state. A high state (1) on this line indicates the reset is active.

- **response1_i[0:0]** and **response2_i[0:0]**: These are single-bit ports representing the response or output from the PUF design. We use this for analyzing temperature.

- **temperature_i[7:0]**: This is an 8-bit port represents temperature. The range [7:0] indicates bits 7 through 0, making up an 8-bit binary number.

2. **Flags**:

Uniqueness Flag:

This flag indicates uniqueness in terms of response at the ideal temperature and response at other temperatures.

1: Simulations in Figure 6.4 indicate that the event or condition at 200ns timestamp is unique. For example, observing a temperature value that has yet to be encountered before or paying attention to a specific combination of Response1 and Response2 is critical.

- 0: This suggests that the event or condition is not unique, meaning it has likely occurred before or is a standard part of the simulation's expected outcomes.

Randomness Flag:

This flag indicates randomness, based on effect on the temperature changes.

- 1: from the simulations, 1 Implies that the temperature or responses are generated randomly. it is crucial for testing how the system behaves under unpredictable conditions.
- 0: Indicates that no randomness is involved in the values at that timestamp, meaning the conditions are pre-determined or follow a specific pattern within the simulation.

Temperature variation effect on the PUF response

Iteration	Response1	Response2	Temperature	Uniqueness Flag	Randomness Flag
0	0	1	9	0	0
1	1	1	141	0	1
2	1	0	1	0	0
44	0	0	253	1	0

Table 6.15 Stability analysis results representation table

Average Stability of all three Arbiter PUF variants

Iteration on Artix 7	Arbiter PUF	3-1XOR arbiter PUF	3-1 Double arbiter PUF
65,536	64.86 %	50.52 %	45.63 %

Table 6.16 Average Stability analysis results

The reported percentage from the above Table (64.86%) represent the success rate of obtaining a consistent and reliable response from the Arbiter PUF over 65,536 iterations. This means that, on average, the Arbiter PUF produced the correct response about 64.86% of the time across the tested iterations. As arbiter chain has very simple structure with only one chain so it remains consistent over different temperature compared to other variants.
As 3-1 XOR arbiter and Double arbiter has more than one arbiter chain so if one of chain response affects because of the temperature variation the whole system gets affected and as a result final XOR 'ed response differs. 3-1 XOR Arbiter PUF and 3-1 Double arbiter is less stable than conventional arbiter PUF.

Test Scenarios for temperature Variations

Each test scenario in the testbench is crafted to challenge and evaluate different aspects of the module's functionality. These scenarios range from basic functionality checks, such as response to the reset signal, to more complex situations involving varying temperature conditions and response patterns. An exhaustive list of test cases is provided, each designed to target specific functionalities of the module. The test cases include equal and unequal responses at different temperature levels, varying patterns of 0s and 1s in response1, and critical temperature threshold testing. On reset, all flags and counters should reset to zero. The `uniqueness_flag` should react to changes in temperature and differences in response signals. The `randomness_flag` should indicate 0's or 1's dominance in response1. At the edge temperature value (125 units), the module's response is critical for assessing its sensitivity to temperature changes. For the Virtex-5 FPGAs, the system monitor (SYSMON) hard macro is calibrated for accurate and reliable measurements over a temperature range of -40° C to +125° C. This range indicates the safe operational temperature limits for the FPGA [17].

CHAPTER 7 Conclusion and Future Work

In this paper, a proposed new mode of operation for APUF that was determined by the connection method of the wires to the conventional arbiter demonstrated a higher degree of performance in terms of all evaluation metrics of the Arbiter PUF. Conventional arbiter PUF with only one arbiter chain gives an average of 30 percent randomness. We compared DAPUFs and APUFs of m = 2 with two selector chains such as 2-2 DAPUF and 2-1 APUF, and evaluated these PUFs regarding their uniqueness, randomness, and steadiness. Further, we proposed 3-1 DAPUF using three selector chains, an improved version of DAPUF. We compare 3-1 DAPUF to 3-1 XOR APUF, which have three selector chains, and evaluate these PUFs. From our experimental results, the uniqueness of responses from 3-1 XOR APUF and 3-1 DAPUFs was approximately 49% and 50% with 65,536 iterations with a number of Multiplexer *ci (0 < I < 64)*, closely aligning with the ideal 50%. On general 7 series FPGAs, we showed that we could improve the uniqueness and randomness by using the new mode of operation for APUF and responses obtained by XORing responses from more duplicated selector chains.

The Temperature Impact on all three PUFs provides a good result and is a crucial part of understanding how temperature changes affect the behavior of all three arbiter PUFs. Arbiter PUF, 3-1 XOR arbiter, and 3-1 double arbiter PUF give average stability results on 7 - series FPGA devices at 64 %, 50%, and 45 %, respectively.

In the context of uniqueness, the data suggests that all three PUF types deliver impressive uniqueness. In the referenced study [14], the authors say that the uniqueness of the Arbiter PUF on 5 series FPGA devices, 2-1 XOR arbiter PUF, and 3-1 XOR arbiter PUF stands at approximately 5%, 6%, and 7% respectively. Based on the analysis of the 7-series FPGA devices in this paper, the 3-1 XOR and 3-1 DAPUF have significantly higher values of 45% and 50%, respectively. We demonstrated that by employing a novel operational mode for APUF utilizing responses obtained through XORing responses from additional duplicated selector chains and making the structure more complex by using the same number of resources for 3-1 Double Arbiter PUF, we were able to enhance uniqueness, randomness, and stability.

The prospective phase of this research involves the implementation of 4-1 or 5-1 DAPUFs, followed by assessing their responses in terms of uniqueness, randomness, and stability. Additionally, we will conduct a comparative analysis between these newly implemented DAPUFs and existing 3-1 DAPUFs based on the obtained results.

REFERENCES:

[1] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in Proceedings of DAC, 2007, pp. 9–14, http://dx.doi.org/10.1145/1278480.1278484.

[2] B. Gassend, D. Lim, D. E. Clarke, M. van Dijk, and S. Devadas,"Identification and authentication of integrated circuits." Concurrency and Computation: Practice and Experience, pp. 1077–1098, 2004,

[3] R. Maes, "Physically Unclonable Functions - Constructions, Properties and Applications," in Springer, 2013, http://dx.doi.org/10.1007/ 978-3-642-41395-7.

[4] J. H. Anderson, "A PUF design for secure FPGA-based embedded systems," in Proceedings of ASP-DAC, 2010, pp. 1–6, http://dx.doi.org/10.1109/ASPDAC.2010.5419927.

[5] K. Seki, Y. Hori, and H. Imai, "Implementation and Evaluation of Physical Unclonable Function on SASEBO-GII (in Japanese)," in Symposium record of SCIS, 2010.

[6] M. Barbareschi, "Notions on silicon physically unclonable functions," in Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment, N. Sklavos,

[7] R. Chaves, G. Di Natale, and F. Regazzoni, Eds. Springer International Publishing.
T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A Study on Uniqueness of Arbiter PUF Implemented on FPGA (in Japanese),"in Symposium record of SCIS, 2014.

[8] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A Study on Uniqueness of Arbiter PUF Implemented on FPGA (in Japanese)," in *Symposium record of SCIS*, 2014.

[9] U. R¨uhrmair, F. Sehnke, J. S¨olter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," in *Proceedings of CCS*, 2010, pp. 237–249, http://dx.doi.org/10.1145/1866307.

[10] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in CryptoGraphic Hardware and Embedded Systems- CHES 2007, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[11] A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare. the performance of physical unclonable functions," in Embedded systems design with FPGAs. Springer, 2013

[12] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of RO-PUF," in 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). IEEE, 2010.

[13] NIST Special Publication 800-22. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Information Technology Laboratory of the National Institute of Standards and Technology, May 2000.

[14] T. Machida, D. Yamamoto, M. Iwamoto, and K. Sakiyama, "A new mode of operation for arbiter PUF to improve uniqueness on FPGA," *2014 Federated Conference on Computer Science and Information Systems*, Warsaw, Poland, 2014, pp. 871-878, doi: 10.15439/2014F140.

[15] Trusted Computing Group. Available online: https://trustedcomputinggroup.org (accessed on 11 August 2021).

[16] Herder, C.; Yu, M.D.; Koushanfar, F.; Devadas, S. Physical Unclonable Functions and Applications: A Tutorial. Proc IEEE 2014, 102, 1126–1141. [CrossRef]
[17] S. Kulkarni, Designing FPGA Based Reliable Systems Using Virtex-5 System Monitor," Design & Reuse