Physical Unclonable Function-based Key Management Unit for RISC-V on FPGAs

Xiangyun Wang

Department of Electrical & Computer Engineering

McGill University

Montréal, Québec, Canada

July 4, 2024

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

The emergence of the Internet of Things (IoT), with an increasing amount of data transmission and storage, as well as a scope of physical system control, has placed a strong emphasis on data security. Encryption is the most widely adopted data security measure, which relies heavily on the secure storage of cryptographic keys. While the traditional key storage methods based on non-volatile memory (NVM) have been proven insecure, physical unclonable functions (PUFs) are a promising alternative. PUFs utilize inherent physical variations in devices to generate unique and unpredictable identifiers. Due to their outstanding cost-efficiency, low power consumption, low resource utilization, and unclonable nature, PUFs have become a favorable IoT security solution.

Concerning the security of implementations, Field Programmable Gate Arrays (FPGAs) are more trustworthy and flexible than Application Specific Integrated Circuits (ASICs), making them appealing platforms for implementing PUFs. Regarding integrations of PUF-based applications with IoT systems, RISC-V instruction set architecture (ISA) presents an excellent fit, especially on FPGAs, due to its modularity, customizability, and open-source

nature.

In this thesis, we explore the use of a more complex delay-based PUF compared to the traditional Arbiter PUF (APUF) to further enhance IoT security. We propose a key management unit (KMU) that relies on a time-to-digital converter (TDC) PUF. Its quality-driven design and implementation strategies on FPGAs are investigated to address the challenges arising from the complex structure of the TDC PUF. Our proposed KMU and the quality-driven methodologies yielded great results when tested with Xilinx Artix-7 FPGAs. We achieved a key regeneration success rate of approximately 90%, a doubling in the entropy, and around 4.5x more robust against small temperature variations. Compared to the traditional APUF, our TDC PUF is 12.5x and 2x more robust in uniqueness for same-model and cross-model FPGA implementations, respectively. Furthermore, our TDC PUF has more complex responses of 11 bits compared to the single-bit APUF response, making it more robust against modeling attacks. Lastly, we successfully integrated our KMU into an open-source RISC-V System-on-Chip (SoC).

Abrégé

L'émergence de l'Internet des objets (IdO), avec une quantité croissante de transmission et de stockage de données, ainsi qu'une portée de contrôle des systèmes physiques, met fortement l'accent sur la sécurité des données. Le chiffrement est la mesure de sécurité des données la plus largement adoptée, qui repose fortement sur le stockage sécurisé des clés cryptographiques. Alors que les méthodes traditionnelles de stockage de clés basées sur la mémoire non volatile (NVM) se sont avérées peu sécurisées, la fonction physique non clonables (PUFs) est une alternative prometteuse. Les PUF utilisent les variations physiques inhérentes aux dispositifs pour générer des identifiants uniques et imprévisibles. En raison de leur efficacité, de leur faible consommation d'énergie et de leur faible utilisation de ressources, les PUF sont devenues une solution de sécurité IdO favorable.

Concernant la sécurité des implémentations, les réseaux de portes programmables sur le terrain (FPGA) sont plus fiables et flexibles que les circuits intégrés spécifiques à une application (ASIC), ce qui en fait des plateformes attrayantes pour l'implémentation des PUF. En ce qui concerne les intégrations des applications basées sur les PUF avec les systèmes

IoT, l'architecture de jeu d'instructions (ISA) RISC-V présente une excellente adéquation, notamment sur les FPGA, en raison de sa modularité, de sa personnalisation et de sa nature open-source.

Dans cette thèse, nous explorons l'utilisation d'un PUF à retard plus complexe par rapport au traditionnel PUF arbitraire (APUF) pour renforcer encore la sécurité IdO. Nous proposons une unité de gestion de clés (KMU) qui repose sur un PUF de convertisseur de temps en numérique (TDC). Sa conception et ses stratégies de mise en œuvre axées sur la qualité sur les FPGA sont également étudiées pour relever les défis découlant de la structure complexe du PUF TDC. Notre KMU proposé et les méthodologies axées sur la qualité ont donné de bons résultats lorsqu'ils ont été testés avec des FPGA Xilinx Artix-7. Nous avons atteint un taux de réussite de régénération de clé d'environ 90%, un doublement de l'entropie, et environ 4,5 fois plus de robustesse contre de petites variations de température. Par rapport au APUF traditionnel, notre PUF TDC est 12,5 fois et 2 fois plus robuste en termes d'unicité pour des implémentations FPGA du même modèle et de modèles croisés, respectivement. De plus, notre PUF TDC présente des réponses plus complexes de 11 bits par rapport à la réponse APUF d'un seul bit, le rendant plus robuste contre les attaques de modélisation. Enfin, nous avons intégré avec succès notre KMU dans un système sur puce (SoC) RISC-V open source.

Acknowledgements

I would like to take this opportunity to acknowledge all those who have supported and inspired me throughout this invaluable research journey at McGill University. This endeavor would not have been possible without them.

First, I am deeply indebted to my supervisor, Professor Zeljko Zilic, for his constant guidance, inspiration, and financial support throughout my master's degree. His insightful feedback and academic mentorship have profoundly assisted me with my research and thesis.

Special recognition goes to my wonderful labmates, who emotionally supported me throughout my research and thesis writing, especially with those interesting stories. I extend my thanks to Katyayani Prakash for her assistance with my publication and Zice Tang for supporting me during the conference presentation. Meanwhile, I would also like to thank Tomas Langsetmo and KNOX for their expertise and financial support.

Furthermore, heartfelt thanks to my best buddies, Purui Chen, Lingzhi Zhang, and Yicheng Song. Whether it was our weekend hotpot gatherings, friendly tennis and squash matches, or simply having fun together, their presence has been a constant source of comfort

during those intense research moments.

Lastly and most importantly, I would like to express my deepest gratitude to my beloved parents, F.K. and H.W. Their inspiring guidance, unconditional support, and unwavering love have been the greatest sources of strength in my life, and indispensable for my current achievements.

Contents

List of Figures

List of Tables

List of Acronyms

AES	Advanced Encryption Standard.
APUF	Arbiter PUF.
ASIC	Application Specific Integrated Circuit.
BRAM	Block Random Access Memory.
CAD	Computer-Aided Design.
CRP	Challenge-Response Pair.
DES	Data Encryption Standard.
DSC	Differential Sequence Coding.
ECC	Error Correction Code.
FF	Flip-Flop.
FPGA	Field Programmable Gate Array.
FSM	Finite State Machine.
GUID	Globally Unique Identifier.
HD	Hamming Distance.

HT	Hardware Trojan.
ID	Identification.
IoT	Internet of Things.
IP	Intellectual Property.
ISA	Instruction Set Architecture.
KMU	Key Management Unit.
LUT	Look-Up Table.
MMIO	Memory-Mapped Input/Output.
MOS	Metal-Oxide-Semiconductor.
NVM	Non-Volatile Memory.
P&R	Placement & Routing.
PUF	Physical Unclonable Function.
RISC	Reduced Instruction Set Computer.
RoCC	Rocket Custom Coprocessor.
RSA	Rivest–Shamir–Adleman.
RTL	Register-Transfer Level.
SHA	Secure Hash Algorithm.
SoC	System-on-Chip.
SRAM	Static Random Access Memory.
TDC	Time-to-Digital Converter.

UART Universal Asynchronous Receiver-Transmitter.

UUID Universally Unique Identifier.

Chapter 1

Introduction

1.1 Motivation

As Internet usage grows exponentially and the number of electronic devices proliferates, an ever-growing array of devices is equipped with network connectivity, facilitating data exchange among devices. Environmental and personal data can be collected and wirelessly transmitted to terminal devices via sensors and wearable devices. Users can access and process these data to achieve remote monitoring and control functionalities. This network of interconnected devices is commonly referred to as the Internet of Things (IoT).

In recent years, there has been a dramatic increase in the number of IoT devices. It has been predicted that by 2025, over 30 billion devices will be connected to IoT networks, comprising over 75% of the total number of electronic devices [7]. This rapid insurgence of

IoT has surely given rise to a new paradigm for high-tech lifestyles. From household applications like smart home appliances, real-time surveillance cameras and health monitoring devices, to industrial applications such as public transportation systems, power grids and medical systems, IoT technologies are reshaping modern life, making it more intelligent and convenient.

As IoT services become more personalized and integral to critical infrastructures, the collected data becomes more sensitive, and more permissions are granted to IoT systems. This makes IoT devices increasingly targeted by attackers, especially low-power embedded devices. These devices often suffer from security weaknesses such as misconfigured ports, lack of security updates, and inadequate protection for communication channels [8].

The cornerstone of any IoT system lies in data storage and communication. Cryptography has been the conventional and widely adopted method to protect them. The fundamental principle is to convert plain texts into cipher texts through encryption. The original data can be recovered from the cipher texts with the reverse decryption process. Both encryption and decryption are achieved with cryptographic keys. Decryption keys are only distributed to authorized parties, thereby ensuring data security. However, this security model is based on the assumption that keys remain securely concealed from attackers. The traditional key storage methods are based on non-volatile memory (NVM), which has been proven insecure [9]. Numerous studies have revealed that data can be forcibly extracted from NVMs through invasive and non-invasive attacks.

Because of the limited power and hardware resources of IoT devices, the scope of IoT security solutions is limited. While alternative solutions are being explored for secure key storage, physical unclonable functions (PUFs) are deemed as a current favorable candidate.

The concept of PUF was first introduced in [10] as the physical one-way function. It exploits inherent device characteristics, such as signal propagation delays and transistor threshold voltages, to generate unpredictable responses to given inputs. Since PUFs depend on the nanoscale structural differences caused by uncontrollable manufacturing variations, this disorder is impossible to clone and is unique to each device.

There have been many studies on PUFs, mostly focused on Arbiter PUFs (APUFs). However, APUFs are known to be highly vulnerable to modeling attacks due to their simple structure and response [11]. Also, most of the PUF studies only focus on theory explanation and performance analysis, whereas the difficult design and implementation strategies required to achieve high-quality PUFs are rarely described.

In this thesis, we aim to explore PUFs with higher structural complexity, thereby increasing the resistance to modeling attacks. Taking the time-to-digital converter (TDC) PUF discussed in [12] as a baseline, we propose a TDC-PUF-based key management unit (KMU). We investigate its quality-driven design and implementation strategies for Field Programmable Gate Arrays (FPGAs). Furthermore, we showcase its real-world application by integrating our KMU into an open-source RISC-V System-on-Chip (SoC).

1.2 Statement of Contribution

The author, Xiangyun Wang, is responsible for all the content presented in this thesis, including the design, implementation, analysis, and integration of the system, as well as the code and illustrations. In cases where concepts and diagrams are adopted from prior research, proper citations have been provided to credit original authors. The TDC implementation on FPGA was conducted in collaboration with Yicheng Song, a fellow master's student in Prof. Zeljko Zilic's Laboratory. A portion of this work was previously published in [13], where Xiangyun Wang served as the sole first author. In this thesis, certain textual contexts have been reused from the aforementioned publication.

1.3 Document Structure

The rest of the thesis is structured as follows: Chapter 2 presents previous relevant research and necessary background knowledge to understand the context of this thesis. Chapter 3 describes the detailed design of the TDC PUF and KMU. Chapter 4 discusses their quality-driven implementation strategies for FPGAs. Chapter 5 provides performance analyses of our TDC-PUF-based KMU. Chapter 6 demonstrates the integration of our KMU with a RISC-V SoC. Chapter 7 discusses the conclusion and future work of this thesis.

Chapter 2

Background & Literature Review

2.1 IoT Data Security

2.1.1 Data At Risk

A generic IoT system can be divided into three major layers: Perception Layer, Transportation Layer, and Application Layer [1]. Fig. 2.1 shows the layered IoT structure, with some common technologies and devices used.

The perception layer is responsible for acquiring environmental data so that IoT systems can be monitored and controlled remotely and intelligently. Different sensors and actuators are deployed to perform various measurements such as temperature, location, acceleration, etc.

The transportation layer provides access to the Perception Layer. The collected data

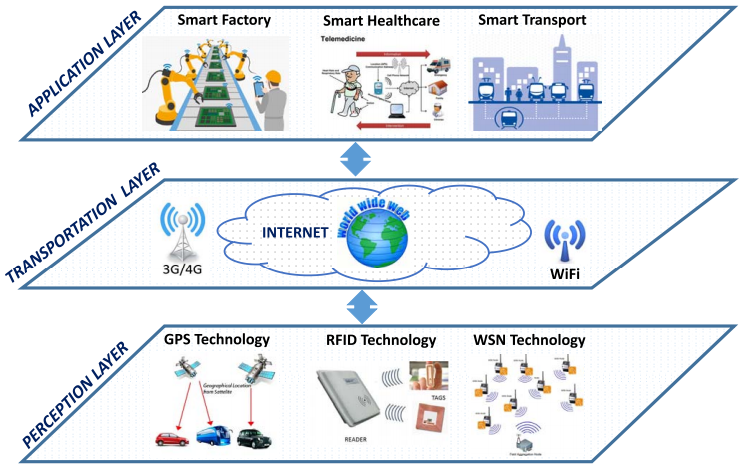

Figure 2.1: IoT System Layers & Data Flows [1]

are transmitted to different data consumers through the transportation layer with various communication technologies, such as cellular, Bluetooth, Ethernet, WiFi, Zigbee, etc. [14] Requests from the upper-layer applications are also handled and transmitted here. Furthermore, due to the limited hardware resources of IoT edge devices, data storage and processing are usually achieved with cloud platforms, such as Amazon Web Services and Microsoft Azure, which are also part of the transportation layer.

The application layer provides platforms for customers to interact with IoT systems. Many services can be implemented in this layer, such as smart homes, smart healthcare, smart factories, etc. Users can remotely access measurement data and control devices. They can also deploy different data processing algorithms and allocate resources on cloud platforms to achieve customized intelligent IoT systems.

Consequently, as the IoT system expands and becomes more intelligent, more endpoint

devices are deployed to monitor and control the system. Meanwhile, larger amounts of environmental and private data must be collected, transmitted, processed, and stored across all three layers. However, this also exposes sensitive data to higher risks. An increasing number of endpoint devices provides attackers with more potential system entry points. When an endpoint is hijacked, attackers can potentially alter and extract sensitive information, or even paralyze the entire IoT system. This could be disastrous to critical systems such as healthcare and public transportation. Also, the chance of data leakage increases with data transfers within the IoT system through attacks such as sniffing and man-in-the-middle [1]. Therefore, data security is a critical problem for reliable IoT systems and requires serious consideration.

2.1.2 Cryptographic Solution

Cryptography is a widely adopted data security solution to prevent sensitive data from unauthorized access or theft. Before data storage or transmission, the original data (plaintext) are transformed into unreadable texts (ciphertext) through encryption so that attackers cannot understand the actual content, even if there is a data leakage or interception. An authorized party can extract the original content from the ciphertext through decryption.

Symmetric & Asymmetric Cryptography

Encryption and decryption are achieved with cryptographic algorithms and keys, which can be categorized into two major types: symmetric and asymmetric. They possess distinct characteristics and should be applied based on demands. Examples of symmetric algorithms are Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Blowfish; and examples of asymmetric ones are Elliptic Curve, Diffie-Hellman, and Rivest–Shamir–Adleman (RSA) [15].

For the symmetric approach, encryption and decryption are performed with the same key. Symmetric algorithms tend to be computationally faster and more efficient than asymmetric ones [16], making them suitable for managing data with large amounts, such as local/cloud data storage. However, a key agreement is required in advance since the same key is shared between the encryption and decryption parties. This is particularly challenging for distributed systems with insecure communication channels, where attackers can make interceptions during the key distribution process and ultimately compromise data security.

In contrast, the asymmetric approach provides more secure key distributions than the symmetric approach. An asymmetric algorithm uses a public key for encryption and a different private key for decryption [16], where the public and private keys are mathematically related. The owner secretly keeps the private key, and only the public key is shared with authorized parties who want to communicate with the private key owner.

Since the distributed key is only used for encryption, the data security will not be jeopardized if attackers intercept the public key. However, asymmetric algorithms are computationally more intense than symmetric ones. As a result, they are often jointly used, where a secure communication channel is established with asymmetric encryption before exchanging symmetric keys so that both efficiency and security can be achieved.

Cryptographic Key Storage

Although sensitive data can be encrypted to avoid direct exposure to attackers, both the symmetric and asymmetric algorithms still require cryptographic keys to achieve this. Therefore, secure key storage is the basis of data security, which is particularly challenging for IoT devices with limited resources and power.

The traditional approach of key storage is based on NVMs. Since NVMs preserve their stored information indefinitely, probing the stored data during power-off is possible. It has been proven that NVMs are susceptible to various attacks, such as imaging attacks and side-channel attacks, resulting in data leakage [9] [17] [18]. The conventional software-based countermeasures are ineffective since these attacks can be deployed without powering on the device. While alternatives to NVM-based key storage are being explored, PUFs are deemed a current favorable candidate for IoT systems.

2.2 Physical Unclonable Function (PUF)

PUFs exploit inherent device characteristics resulting from manufacturing variations to produce unforeseeable and unclonable responses (R) to given challenges (C), jointly known as challenge-response pairs (CRPs). Practically speaking, no manufacturing process is perfect. Although there is hardly any impact on device operations, the nanoscale structural disorder caused by manufacturing variations exists nonetheless [19]. This disorder cannot be cloned or reproduced exactly, not even by its original manufacturer, and is unique to each device.

2.2.1 PUF Classification

There are various ways to categorize PUFs, such as based on their material, underlying mechanisms, and security levels. For instance, although the term "PUF" is most commonly used to refer to the silicon PUF, there are also PUFs based on other materials and properties such as coating PUFs and optical PUFs [20]. From a higher-level perspective, shown in Fig. 2.2, PUFs are usually classified into extrinsic & intrinsic PUFs, and strong & weak PUFs.

Extrinsic & Intrinsic PUF

PUFs can be categorized into extrinsic and intrinsic based on the source of uniqueness and the measurement method. For extrinsic PUFs, the unique characteristics are deliberately added into devices with extra manufacturing processes, such as semitransparent particles of

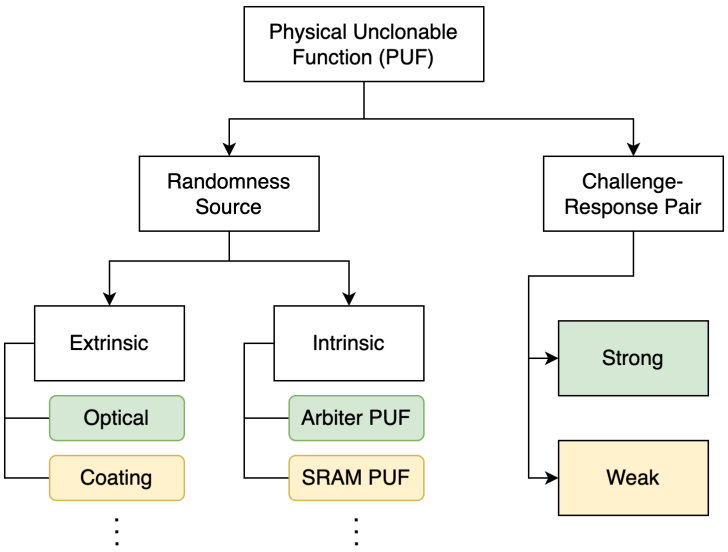

Figure 2.2: PUF Classification

optical PUFs and dielectric particles of metal coating PUFs [21]. Also, extrinsic PUFs require external evaluations, and their raw responses cannot be directly used for other purposes [22]. For example, speckle patterns are a type of optical PUF response [10], which needs to be observed with external devices and digitized before practical usage. As a result, the deliberate uniqueness injection induces extra costs to produce this type of PUFs, and the requirement of an external system for measurements makes it less accurate and unreliable.

In contrast, intrinsic PUFs are based on implicit uncontrollable side effects of standard manufacturing processes [22]. Also, all evaluations of the unique properties must be performed by embedded measurement equipment without any external assistance, which makes intrinsic PUFs more reliable than extrinsic ones. Most of the intrinsic PUFs are silicon PUFs [22].

Strong & Weak PUF

PUFs can also be categorized into strong and weak PUFs based on the number of CRPs available. Weak PUFs, initially termed Physically Obfuscated Keys, only have very limited CRPs available, and in the extreme case with just one [23]. The responses of weak PUFs are never meant to be accessed directly from the external world and are usually used for secret key derivation. Thus, weak PUFs are essentially a special form of non-volatile key storage, where extracting keys from them is harder than typical NVMs. Static random-access memory (SRAM) PUFs are an example of weak PUFs.

In contrast, strong PUFs possess a much larger number of CRPs compared to weak PUFs. It must have enough CRPs so that it is impossible for attackers to perform a complete measurement of all CRPs within a limited amount of time. The number of CRPs of a strong PUF should also increase exponentially with its size [24]. Furthermore, it must be difficult for attackers to establish a numerical model for effective CRP predictions, even if many CRPs are exposed to attackers. Examples of strong PUFs include APUFs and optical PUFs. Their typical applications include key establishment and identification protocols [25].

2.2.2 PUF Performance Evaluation

Many different parameters have been proposed for PUF evaluations, such as in [26] [27] [28]. Among them, many of the parameters describe similar properties but are given different names. This thesis focuses on the three most essential criteria for PUFs:

uniqueness, randomness, and reliability. Uniqueness represents the ability of a PUF to uniquely identify a device from a set of devices of the same type/model. It is usually evaluated with Hamming Distances (HDs) (higher the better) between two responses generated with the same challenge but on two different devices. Randomness shows how chaotic and unpredictable the responses are. This is usually evaluated with the probability distribution of all possible responses. An ideal PUF should have a uniform response distribution. Reliability captures how consistent a PUF is in reproducing a response with the same challenge using the same device. PUF responses are subject to internal or external factors, such as circuit meta-stability, temperature, and voltage. To evaluate PUF reliability, responses under different operation conditions are collected, and HDs (lower the better) between them and the reference responses are calculated. The reference responses are collected under nominal device operation conditions.

2.2.3 PUF Design

As mentioned, PUFs can be based on different materials and properties. Since this thesis aims to enhance IoT security with PUFs, we only focus on silicon PUFs. In the rest of this thesis, the term "PUF" is used interchangeably with "silicon PUF". Many PUF designs have been proposed, which can be further classified into memory-based PUFs and delay-based PUFs. Examples of delay-based PUFs include APUFs and TDC PUFs, and SRAM PUFs are memory-based PUFs.

SRAM PUF

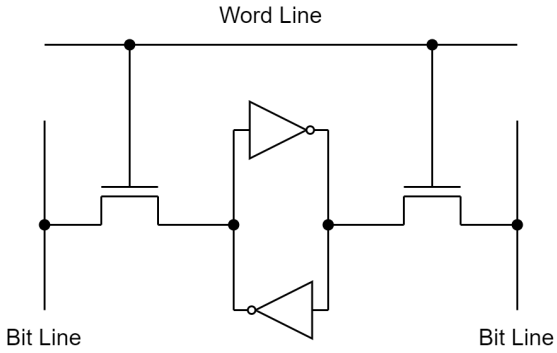

Figure 2.3: SRAM PUF Structure

SRAM PUFs exploit the random initial state of memory cells during device start-up to generate unique device signatures [29]. Fig. 2.3 shows the basic structure of an SRAM cell. Upon power-up, the back-to-back inverter configuration becomes briefly metastable. An inverter consists of a PMOS and an NMOS transistor connected in series. Depending on which set of transistors for the two inverters has a higher threshold voltage, the SRAM cell settles on either the high or low logic state at start-up [30]. While some SRAM cells might not have a strong logic state preference, those with strong preferences can be used as unique fingerprints. Due to the limited number of SRAM cells in an embedded device, SRAM PUFs are classified as weak PUFs.

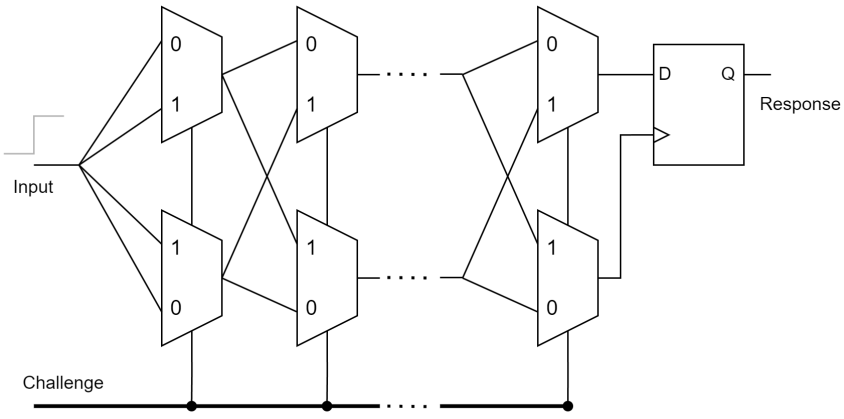

Figure 2.4: Arbiter PUF Structure

Arbiter PUF (APUF)

APUFs are the most extensively studied delay-based PUFs [31]. Fig. 2.4 shows the basic structure of an APUF. It consists of symmetric circuit paths of the same stage length. Each stage of the APUF has two 2-to-1 multiplexers, where signals can be configured to swap [30]. APUFs exploit the slight delay differences in the circuit paths between stages. The two signals arrive at the "arbiter", often a flip-flop (FF), with a slight timing difference. APUF responses are based on the arrival order of the FF data-in and clock signals. For an APUF with n stages, there are 2^n possible signal path configurations, which increase exponentially with the stage number. The implementations of APUFs are easy and lightweight with both ASICs and FPGAs due to their simple structure. This makes them currently the most widely adopted PUFs.

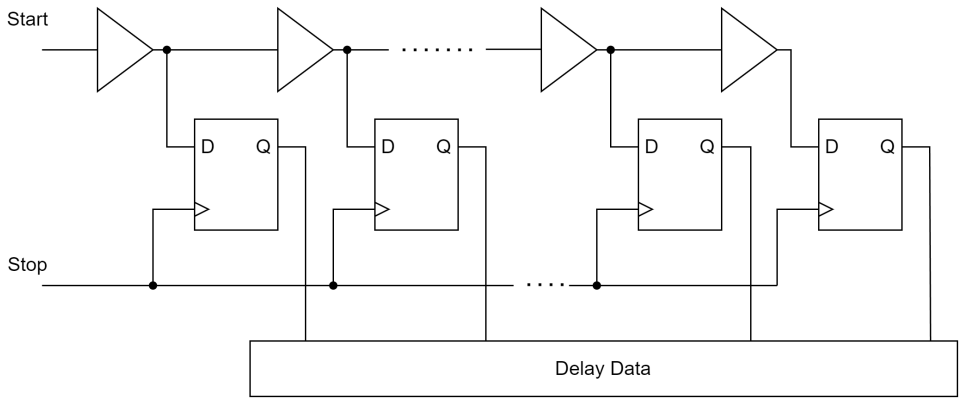

Figure 2.5: Time-to-Digital Converter Structure

TDC PUF

A less common delay-based PUF design has been discussed in [12] and [32], which utilizes a time-to-digital converter (TDC). Unlike APUFs, which only identify the arrival sequence of two signals, TDC PUFs measure the delay difference between them. Fig. 2.5 shows the basic structure of a TDC. Ideally, the *Stop* signal should arrive at FFs when the *Start* signal has not fully propagated through all the buffers, and the number of 1s locked in the FFs is the PUF response. Compared to the single-bit response of APUF, the TDC PUF generates more complicated multi-bit responses. Also, the circuits connecting to the *Start* and *Stop* of the TDC can have asymmetric and complex structures, as discussed in [12]. However, TDC PUFs require more careful design and implementation than APUFs. The arrival time of the *Start* and *Stop* signals should be tightly controlled to avoid invalid delay measurements.

2.2.4 PUF Quality

PUF-based key generations and authentications often require PUF responses to be regenerated in a bit-exact fashion. Since PUF responses are sensitive to variations such as changes in temperature and voltage, helper data are generated during enrollment to identify and correct errors for future response regeneration.

The most common form of helper data is error correction codes (ECCs) generated with syndrome coding. Various syndrome coding scheme techniques have been proposed for PUF applications, such as Code-Offset Syndrome [33], Index-Based Syndrome [34], and Differential Sequence Coding (DSC) [35]. As ECCs occupy extra memory spaces, data compression techniques are also being explored to reduce this overhead. For example, [36] demonstrated a data compression technique for DSC, which lowered the helper data size by 43%.

While the ECC approach can adapt to most PUF designs, PUF-type-specific helper data also exist. For the SRAM PUF discussed in [37], helper data were used to identify SRAM cells without a strong state preference. For the APUF described in [38], helper data were used to identify PUF configurations with large delay differences between FF data-in and clock signals, as these configurations are more tolerant to delay variations.

2.2.5 Modeling Attack

Modeling attacks are a major security threat to PUFs. It is possible to build a predictive model of a PUF when a sufficient number of CRPs are acquired by an attacker. This type of attacks usually targets strong PUFs due to their lack of protection against CRP extractions [11]. While APUF is one of the most widely applied strong PUFs, it is also the least resistant to modeling attacks due to its simple symmetric structure and limited possible responses [11]. For instance, effective APUF attacks have been demonstrated in [39] with a simple Logistic Regression model, obtaining a prediction accuracy of over 99%. It has also been proven in [40] that, by using neural-network-based modeling attacks, APUFs can be attacked faster with far fewer CRPs than previously known. There has been research focused on modeling-resistant APUF-based designs, such as in [41] [42] and [43]. However, given that these designs are still based on the same simple APUF structure, they are likely vulnerable to existing modeling attacks with slight model modifications. Ultimately, a more effective countermeasure involves exploring PUFs that are structurally more complex than APUFs, such as TDC PUFs. While not entirely immune to modeling attacks, it can significantly increase the complexity and difficulty of the attack.

2.3 Hardware Security - ASIC vs. FPGA

Malicious modifications to hardware, also known as hardware Trojans (HTs), can lead to chip malfunctions and backdoors, ultimately compromising data security [44]. Compared to software Trojans, HTs are more difficult to fix, as altering a fabricated silicon chip is hardly possible. HT insertions often happen during the fabrication, assembly, and testing phases of the modern IC supply chain [2], as shown in Fig. 2.6. Design and manufacturing of ICs are often performed by different off-shore suppliers due to budget constraints. This increases the chance of HT insertions, especially with an untrusted foundry, as the IC design companies have no control over the actual fabrication.

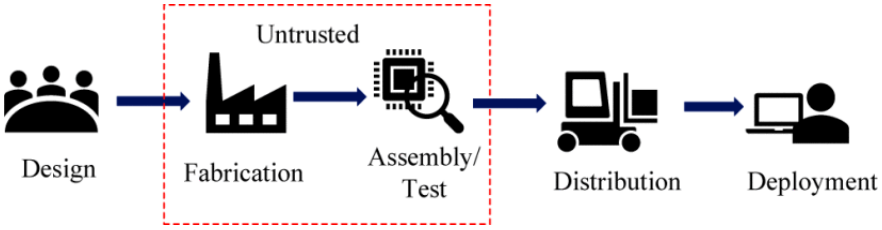

Figure 2.6: Possible HT Insertion in Modern IC Supply Chain [2]

Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) are the two major types of ICs used in IoT systems. The ASICs are designed and optimized for specific functionalities, which have better performance and power efficiency than FPGAs. In contrast, FPGAs can be reprogrammed after manufacturing for different demands, providing better flexibility. Both types of ICs are susceptible to HT insertions during manufacturing. However, the traditional HT is less effective for FPGAs compared to

ASICs, as attackers do not have prior knowledge of the actual functionalities and resource allocations of the FPGA designs [45].

Due to the reconfigurability of FPGAs, there exists a type of FPGA-specific HT called bitstream Trojan. The bitstream file is like a blueprint for FPGAs, configuring internal components to achieve specific functionalities. When a malicious party gains access to the bitstream file, they can reverse engineer it to identify FPGA designs and insert Trojans accordingly. However, unlike HTs inserted during manufacturing, this type of Trojan is non-invasive and non-permanent, which can be prevented with proper management of bitstream files. Some detection and prevention techniques are summarized in [46] and [47].

Furthermore, IoT systems can also benefit from FPGAs' reconfigurability. New functionalities might be preferred as an IoT system evolves and expands. They often require specialized hardware designs to maximize their performances. FPGAs can easily achieve these due to the shorter development cycles compared to ASICs and the ability to make remote bitstream updates. With all these advantages and safety characteristics, FPGAs are increasingly used in IoT systems [48]. However, one major drawback of FPGAs, regarding delay-based PUF implementations, is their limited placement and routing (P&R) flexibility. Since delay-based PUFs exploit the transient behavior of signal propagations, their circuit P&Rs should be carefully controlled, which is more challenging for FPGAs compared to ASICs.

2.4 RISC-V Instruction Set Architecture

The "brains" of smart IoT systems are embedded processors, and most of these processors are currently based on the Arm instruction set architecture (ISA). However, the Arm ISA is proprietary. It is usually sold as licenses, intellectual property (IP) cores, and off-the-shelf silicons. Their prices tend to be expensive, and their full register-transfer level (RTL) implementations are strictly hidden, making them unfriendly to researchers and start-up companies. Furthermore, commercially available IP cores and silicons are usually implemented with the full ISA, whereas the actual application might properly run with just a reduced ISA. Since there is no access to the underlying RTL implementation, it is impossible to eliminate the unused part of the ISA, which induces unnecessary costs for users.

To solve these drawbacks, the new RISC-V ISA has been introduced in [49] and is becoming an attractive alternative ISA for embedded systems. Since RISC-V ISA is open-source and patent-free, users can freely modify it. The RISC-V ISA is split into modular parts, and users may include only a subset of RISC-V ISA based on actual demands. It is also designed to support extensive customization. The base RISC-V ISA is encoded to utilize only a small fraction of the encoding space, and leave enough space for efficient and tight integration of specialized co-processors [50].

Although RISC-V ISA possesses great modularity and customizability, these advantages only exist in the design phase for ASICs, as it is impossible to make further modifications

after manufacturing. Flexibility is also preferred for its hardware to maximize the benefits of RISC-V ISA. The reconfigurability of FPGAs makes them perfect platforms to deploy RISC-V cores. Whenever IoT suppliers want to add new features requiring the support of another ISA module, this can be easily achieved with new FPGA designs and remote bitstream updates.

Chapter 3

Proposed System Design

3.1 KMU System Design Overview

The overview of our proposed KMU system design is shown in Fig. 3.1. From the external perspective (RISC-V side), six sets of signals are accessible, including *Key Request*, *Key Valid*, *Key Enroll*, *Key ID*, *Helper Data/Address*, and *Key*. To request a key, the requester must assert the *Key Request* signal and provide the corresponding 128-bit identification (ID), such as GUID, UUID, etc. With the *Key Enroll* signal, the requester must also indicate if this is a first-time key request (key enrollment). Upon a key enrollment, in addition to the standard key generation, the KMU will also generate helper data so that the reliability of future key regeneration can be guaranteed. After a successful key generation/enrollment, the *Key Valid* signal will be asserted by the KMU, performing a handshake with the external

system.

Storing the helper data of all enrolled keys within the KMU system requires additional storage resources, thereby imposing a limit on the maximum number of keys that can be enrolled by the host system. For our TDC-PUF-based KMU, helper data are used to differentiate between stable and unstable delay circuit paths without revealing the actual PUF responses. Consequently, it is deemed safe to expose them to the external world. Therefore, in our design, each key requester stores the helper data individually and externally in the NVM.

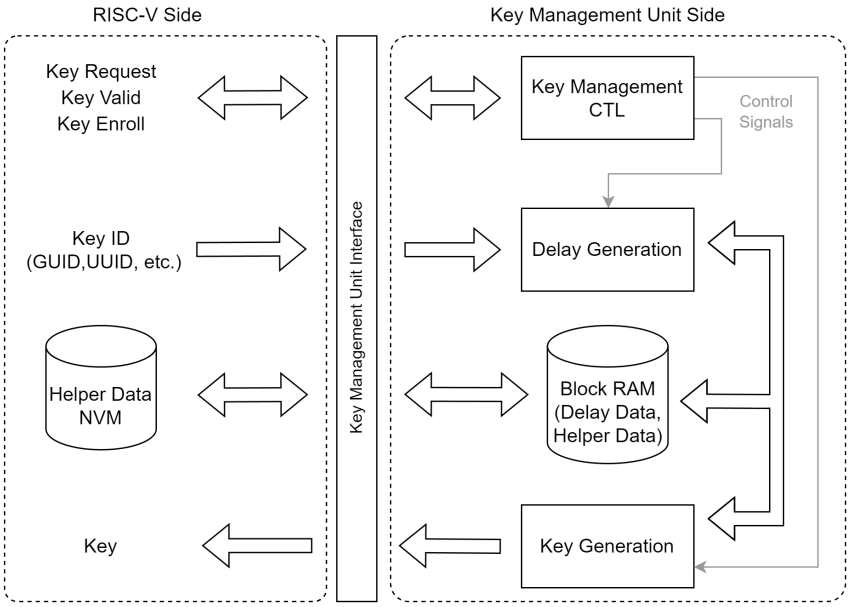

Figure 3.1: Overview of the Proposed Key Management Unit

The right side of Fig. 3.1 provides a simplified flow of the proposed KMU system. *Delay Generation* and *Key Generation* blocks are control by *Key Management CTL*. When

a key request is received, the *Delay Generation* block will start extracting delay data (TDC PUF responses) and store them in the block random access memory (BRAM). Meanwhile, helper data will also be generated for key enrollments. When sufficient delay data have been collected, the Key Generation block will generate a key by hashing stable delay data together using a Keccak hash function. More detailed explanations of the mechanisms and control flow for each part of the system are provided in the following sections.

3.2 KMU System Control Flow

The control flow of the KMU system is shown in Fig. 3.2. Upon a key request, the system starts off with extracting delay data. For key enrollment, the system also generates helper data to identify the stability of delay data. Both delay data and helper data are stored in BRAMs. Multiple rounds of delay and key generation are executed during the key enrollment process. The delay data generated in the first round are assumed to be all stable and are used as references. The regenerated delay data in later rounds are compared with the reference delay data. If they are different, the associated helper data are revised, and the respective delay data are excluded from the key generation. This iterative process persists until the key stability converges.

Once sufficient delay data are collected, the system generates the key by compressing the stable delay data with a hash function. Upon successful key generation, the system directly outputs the key for non-enrollment key requests. In case of enrollment requests, the system

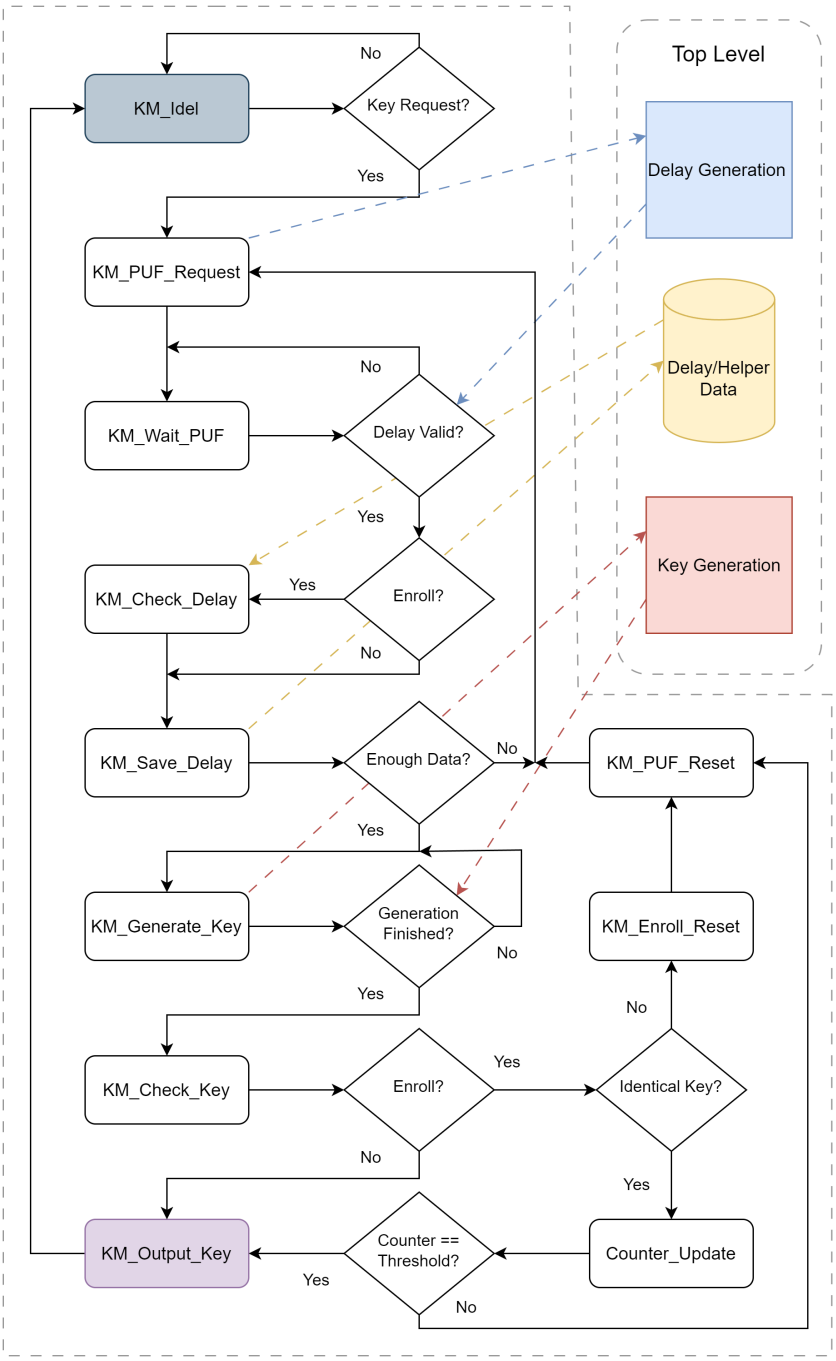

Figure 3.2: KMU Top-Level Control Flow

proceeds to the key stability check.

For the stability check, the first generated key is preserved as a reference for future comparison. A new key is generated after each round of delay data generation and helper data update. This new key is compared with the reference key to check if the key stability converges. To determine the key convergence, we introduced a threshold called the Key Regeneration Threshold. If the same key can be regenerated consecutively for a number of times exceeding the threshold value, this key is considered stable, and the enrollment process ends. The key reliability and the length of enrollment time are closely related to the choice of the threshold value, which is further investigated in Section 5.1.

3.3 Delay Measuring Mechanism

The delay data generation mechanism of the TDC PUF is shown in Fig. 3.3, taking the design discussed in [12] as the baseline. Delay data of different circuit paths within an entropy circuit are measured with a TDC. The selected output from the entropy circuit propagates through the 128-stage buffer chain, shown in green. At each stage of the buffer chain, the output is connected to a FF to extract the signal propagation status. Ideally, the rising edge of the clock (*Launch*) signal should be precisely controlled to arrive at the FFs when the selected output has not fully propagated through all the stages of the buffer chain. The propagation status is subsequently passed to a decoder, and the number of 1s being captured in the FFs is the actual delay data of the selected circuit path. The late

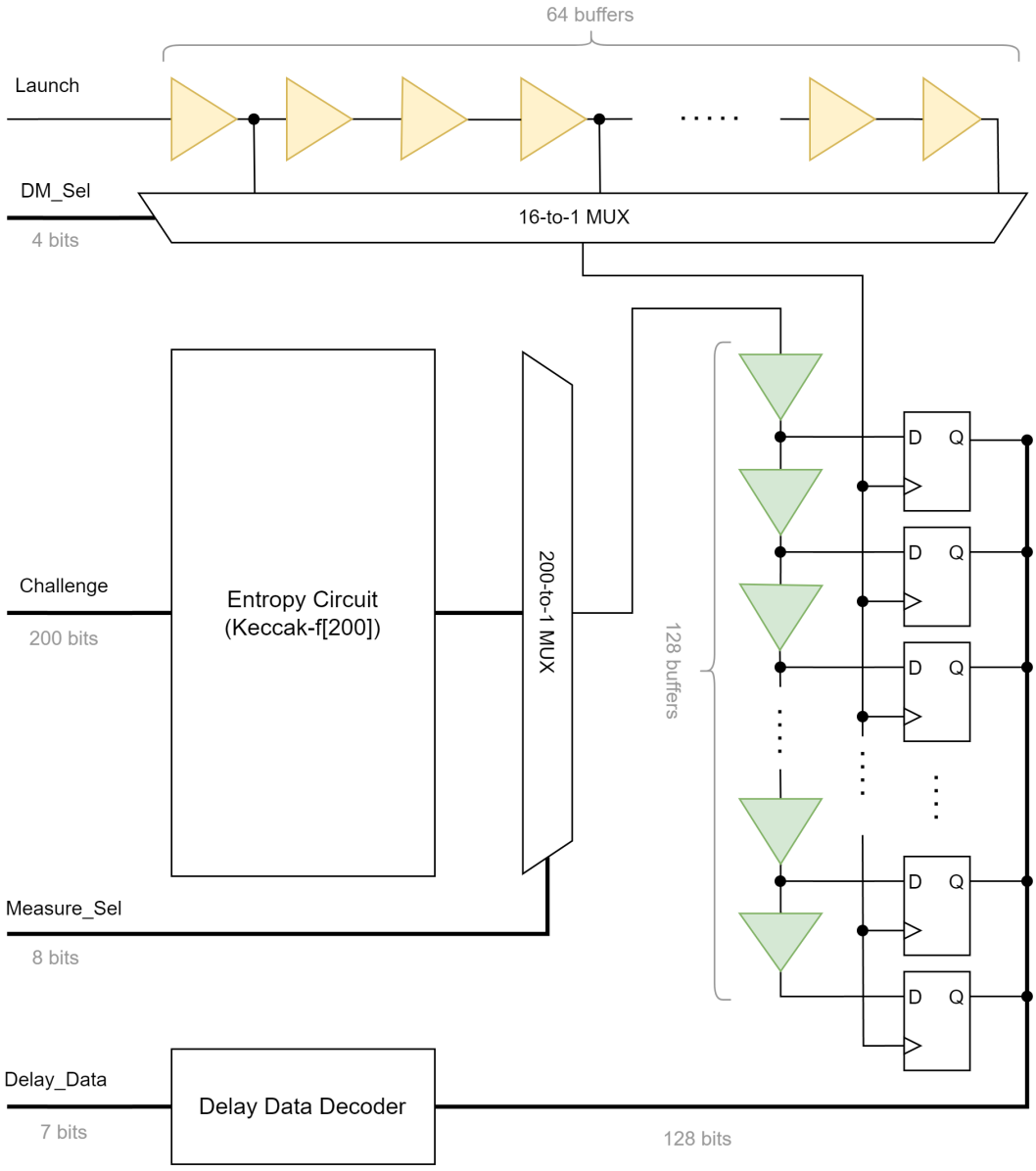

Figure 3.3: TDC PUF Delay Data Generation Mechanism

or early arrival of the clock signal might result in all 1s (overflow) and 0s (underflow) in the FFs, which are both invalid delay data. This signal propagation balance is important but difficult to achieve, which needs to be addressed with careful FPGA P&R during the implementation.

To ensure delay data validity, a delay matched to the selected circuit path must be added before the rising edge of the *Launch* signal. The delay matching block consists of a 64-stage delay chain and a multiplexer. The inputs of the multiplexer are connected to 16 different extraction positions on the delay chain, highlighted in yellow in Fig. 3.3. By choosing different extraction points, desired delay amounts can be added before the *Launch* signal to avoid overflow or underflow. When measuring the delay of a chosen circuit path, the *Launch* signal and the challenge to the entropy circuit must be asserted simultaneously. As a result, a complete piece of delay data comprises 11 bits, with 4 bits for extraction point selection and 7 bits for the 128-stage buffer chain.

The appropriate selection of the delay extraction point and the P&R of the delay matching chain are critical for high-quality delay measuring, both of which are further discussed in Section 4.1 and 4.3. Also, the clock routing of FFs on FPGAs must go through a global clock buffer before getting distributed to FFs, to minimize clock skew. However, this adds extra delay to the *Launch* signal, which is unpleasant for our TDC. Consequently, in our actual FPGA implementation, the FFs shown in Fig. 3.3 are replaced with NAND latches for better control of the *Launch* signal arrival, which is further discussed in Section 4.2.

A desired entropy circuit should possess great complexity and randomness. Meanwhile, it should also be lightweight to accommodate the limited hardware resources of FPGAs. This makes lightweight cryptographic functions preferred candidates. Our design uses a Keccak-f[200] hash function as the entropy circuit, which is further explained in Section 3.7. A 200-in-1 multiplexer selects the circuit path to be measured from the entropy circuit. Moreover, the same hash function is used for PUF challenge generation and key generation to further minimize hardware resource consumption.

3.4 Delay Data Generation Flow

The complete control flow of delay data generation is depicted in Fig. 3.4. The system can be activated by either of the two flags: the internal Relaunch request or the external PUF request, with the latter being the most common scenario. As mentioned in the previous section, delay data are generated by allowing the signal to propagate through a buffer chain. Consequently, only circuit paths with high (1'b1) output can be selected for delay measurement. To achieve this, the output vector of the entropy circuit must be saved when a new challenge is used for delay generation (when $Measure\_Sel == 0$).

Using the saved output vector, the system inspects the value of the selected circuit path. If the output is high, the system initiates the delay-measuring process. Otherwise, the system continues to check subsequent bits until it finds a high-output one or reaches the last bit of the vector. After checking all output bits, the system generates a new challenge and repeats

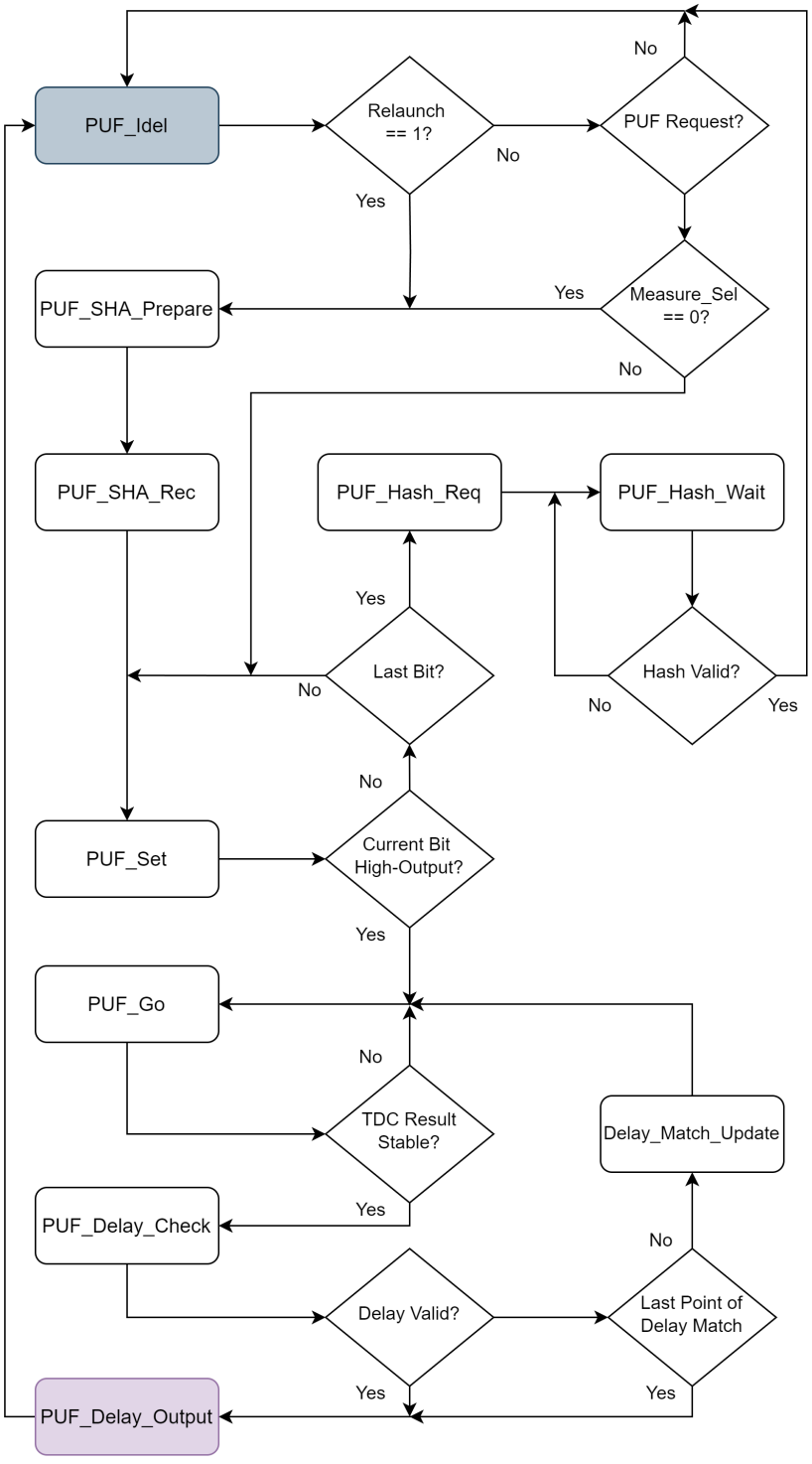

Figure 3.4: Delay Data Generation Control Flow

the process until enough delay data are collected.

With a high-output circuit path, the delay measuring process is initiated by simultaneously asserting the challenge and the *Launch* signal to the entropy circuit and the TDC, respectively. As aforementioned, overflow or underflow could occur if undesired delay matching is performed. When an underflow happens, indicating that the *Launch* signal arrives at the FFs too soon, a larger delay must be selected for delay matching. Since underflow and overflow are easily detectable in our design, the delay matching selection is done through a straightforward trial-and-error process. The system begins with the first delay extraction point (the lowest delay), and keeps increasing until no underflow happens, indicating an appropriate delay matching selection. In rare cases, the delay data might change directly from underflow to overflow due to bad placement of extraction points on the 64-stage delay chain. This problem can be mitigated with careful circuit P&R, which is discussed in Section 4.1. However, if this happens, the system directly outputs the overflowed delay data and labels it as unstable with the helper data.

When all paths with valid outputs are measured, the system requires a new challenge for the entropy circuit so that different circuit paths can be activated and measured. Considering the key request only takes a single 128-bit ID as a seed to generate the key, a mechanism to auto-generate challenges based on the key ID is needed, preferably with a low collision rate to avoid generating the same challenge for different IDs. We benefit from using a Keccak hash function as the entropy circuit in our design. The initial PUF challenge is the Key ID,

and new challenges are created by hashing the previous challenge with the key ID. Once a new challenge is generated, the *Relaunch* signal is asserted, and the system returns to the normal delay measuring routine.

3.5 Key Generation Flow

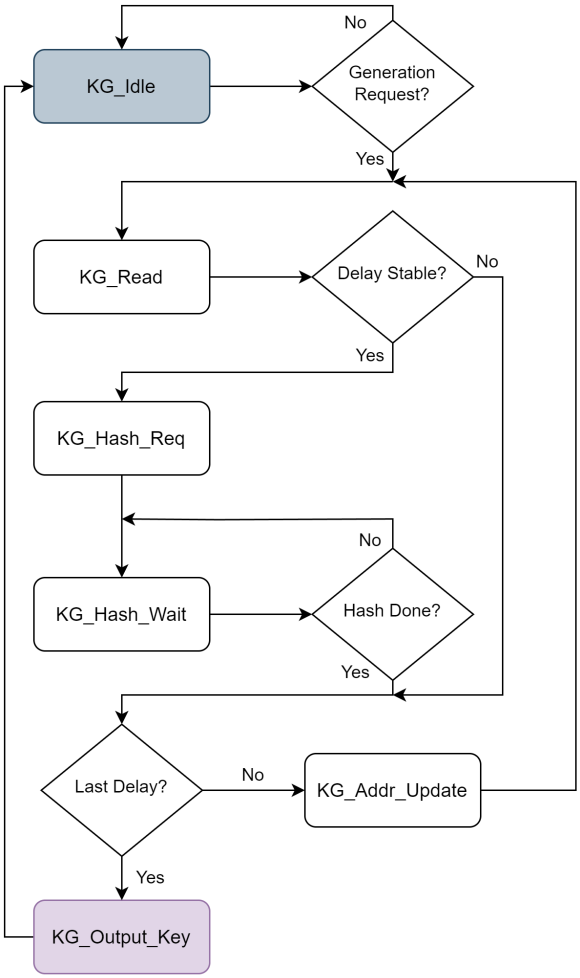

Figure 3.5: Key Generation Control Flow

The key generation flow is shown in Fig. 3.5. The top KMU control module activates this flow once a sufficient number of delay data have been collected. Each piece of delay data is read into the system and examined with its corresponding helper data, with only the stable ones being used for key generation. The Keccak hash function is used to hash and compress stable delay data together. Once all delay data has been examined and hashed, the system outputs the result of the hash function as the key.

3.6 Hash (Entropy) Circuit Control

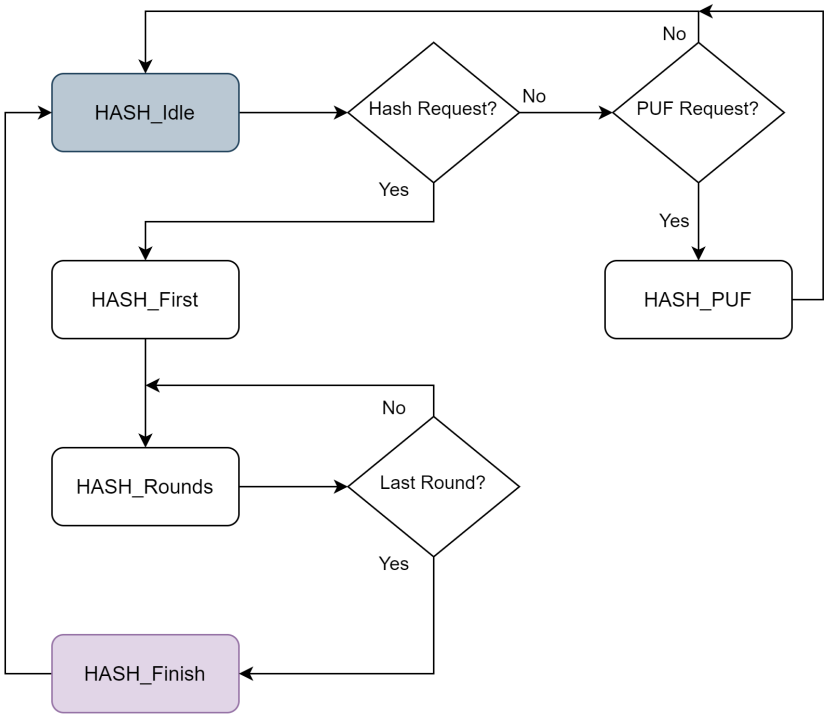

Figure 3.6: Hash (Entropy) Circuit Control Flow

The hash circuit plays an important role in our design. The circuit is used for key and challenge generation under normal hash function mode, and for delay data generation under PUF mode. Extra control logic must be implemented to control and switch between these two modes, as shown in Fig. 3.6.

The circuit operates in the normal hash mode when a hash request is received. To hash a piece of data with the Keccak hash function, the data go through the hash circuit for several rounds, where the round number depends on the chosen input size of the hash function. Only during the first round is the input data XORed with the current output of the hash circuit. In the PUF mode, where a selected circuit path of the entropy circuit is to be measured, the controller asserts the challenge to the hash circuit. Simultaneously, the *Launch* signal is also asserted to the TDC.

Since both the Delay Generation and Key Generation blocks require access to the hash circuit, they must share the input ports. Fig. 3.7 shows the port sharing and control signal connections with other system modules. The connections are highlighted in blue for Delay Generation, red for Key Generation, green for Key Management CTL, yellow for delay data storage, and gray for TDC. The hash circuit controller has four inputs, with *Hash_Req* and *MSG_In* being shared, *PUF_Req* from Delay Generation and feedback from the hash output. The two shared ports are managed with two multiplexers, both being controlled by *PUF_Busy* and *KG_Busy*.

During delay generation, the Delay Generation block raises *PUF_Busy* flag to claim

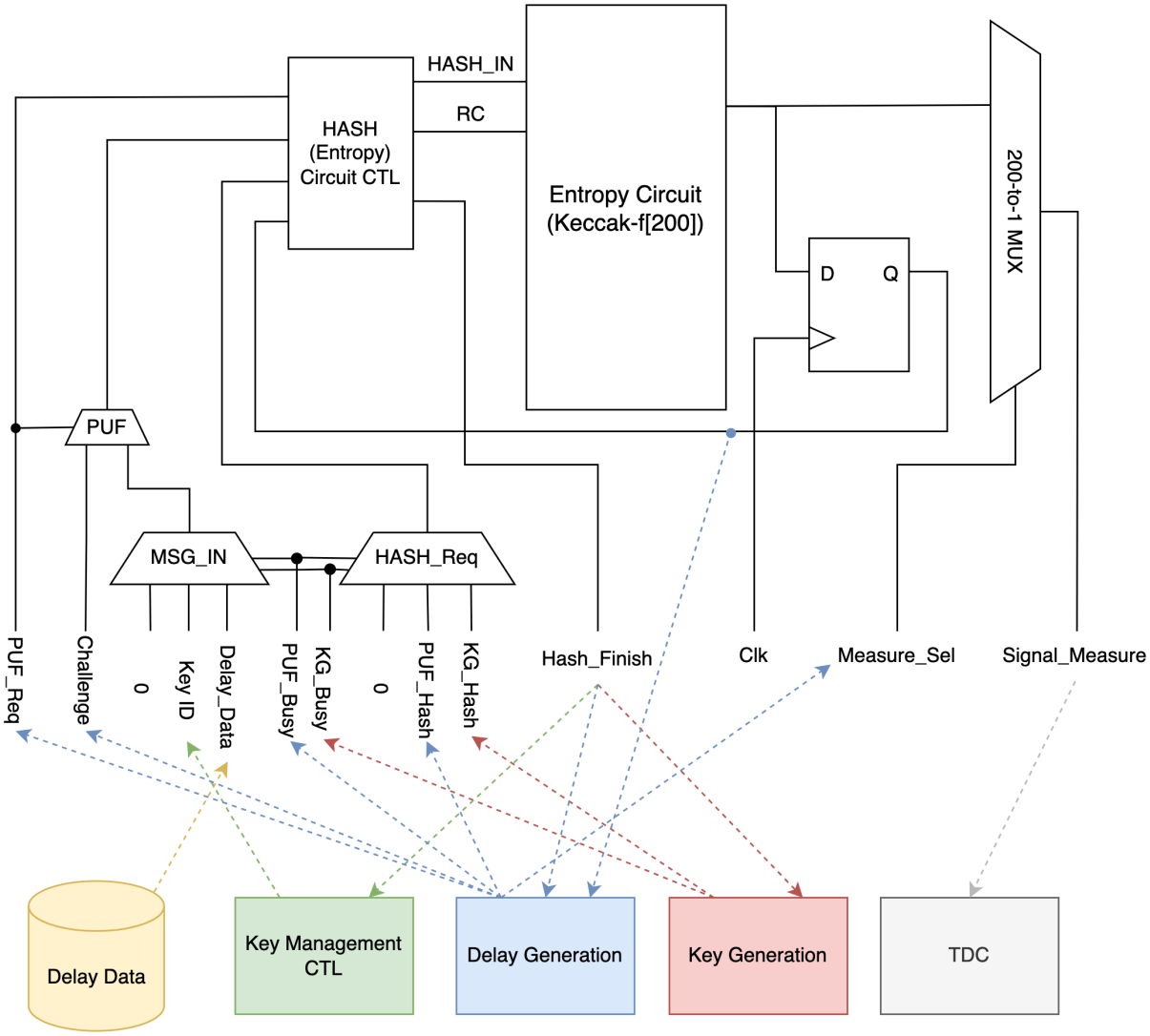

Figure 3.7: Hash (Entropy) Circuit Peripheral Logic & Connections

control over the entropy circuit. For delay data requests, *PUF_Req* is raised to operate the circuit in PUF mode and to select the PUF challenge as the circuit input. When the system requires a new challenge, *PUF_Hash* is raised, and the *ID* is selected as the hash input. When delay generation finishes, the control of the entropy circuit is handed over to the Key Generation block by lowering *PUF_Busy* and raising *KG_Busy*. Once all stable delay data have been hash together to form a key, the shared ports are released.

3.7 Keccak Hash Function

The Keccak hash function is used not only as the entropy circuit but also as the challenge and key generation algorithms in our KMU design. While the primary focus of this thesis is not on the actual theories behind the Keccak hash function, a brief explanation, based on the original Keccak documentation [3], is still provided in this section.

The Keccak hash flow has two major phases: absorb and squeeze, as shown in Fig. 3.8. During the absorb phase, data pieces are XORed and compressed with the hash function. After absorbing all data, a desired number of outputs can be squeezed out by asserting the hash output to the hash input.

There are 7 Keccak permutations, with state widths $b = \{25, 50, 100, 200, 400, 800, 1600\}$. The state width represents the input/output size of the hash function. The latest Secure Hash Algorithm 3 (SHA-3) uses the 1600-bit Keccak permutation (Keccak-f[1600]) to maximize security. Due to the lightweight requirement of IoT systems, our KMU design

uses the Keccak-f[200] version. Its resource consumption is only around 10% of the Keccak-f[1600] version [51], and it is the smallest version where a reasonable level of security can be obtained [3].

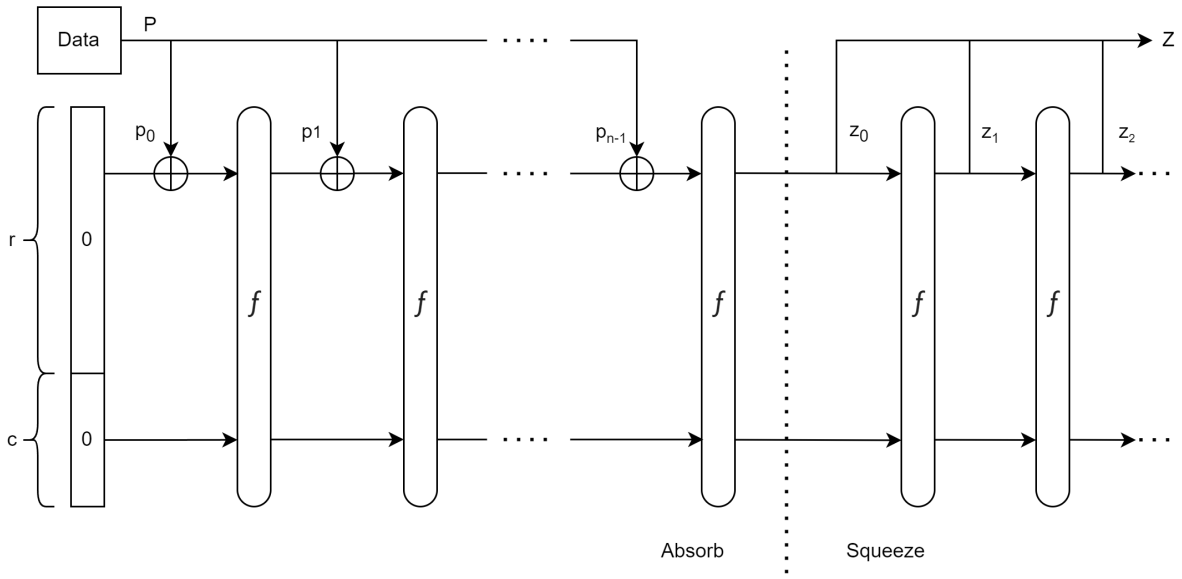

Figure 3.8: Absorb & Squeeze Phase of Keccak Hash Function

The state width is further divided into rate (r) and capacity (c) blocks, as shown in Fig. 3.8. The rate (r) represents the number of bits that can be hashed in a single operation. The higher the r, the sooner the hash process ends, but the lower the security. For our design, the rate is set to 128-bit during challenge generation to accommodate the 128-bit ID size and is later changed to 16-bit for key generation to maximize the security level. The state is constructed as a $5\times5\times w$ block in the actual hash function implementation. Fig. 3.9 shows the graphical view of the state block and different state pieces.

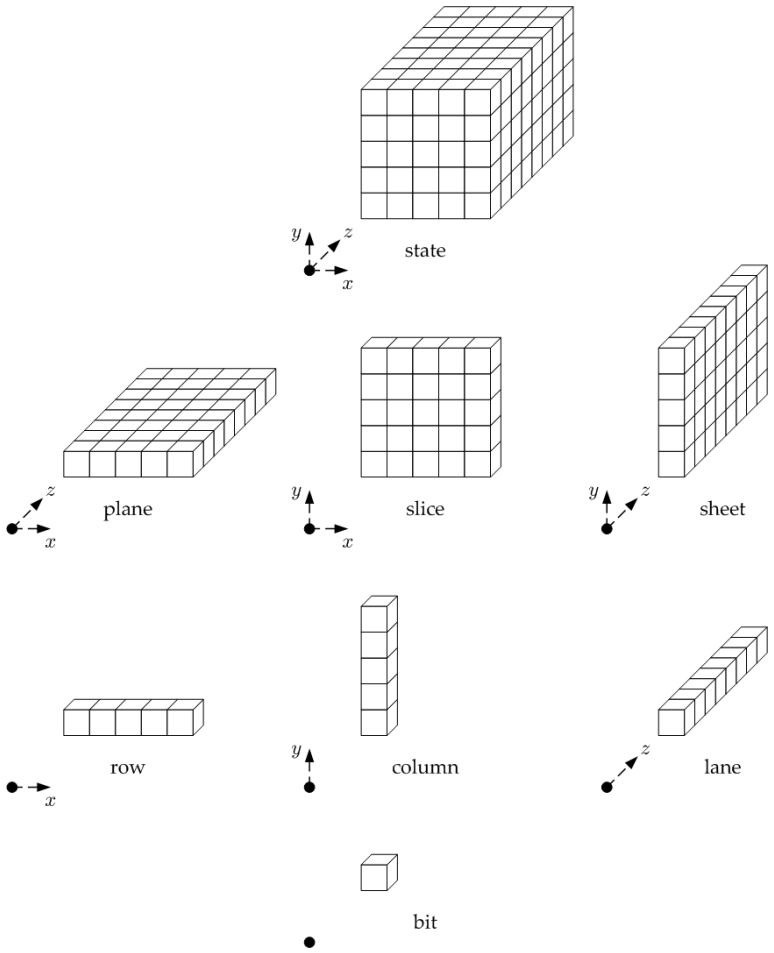

Figure 3.9: The Keccak-f State Block and Pieces of State

$$n = 12 + 2l, \; where \; 2^l = w \tag{3.1}$$

For a single hash operation, the data goes through the same manipulations for several rounds. Only during the first round is the new data XORed with the state block. The round number n is related to w, as shown in Eq. 3.1. For the Keccak-f[200] permutation, $w = 8$ and $n = 18$. Each round consists of five steps in the order of θ, ρ, π, χ, and ι. Their detailed

operations are given in Eq. 3.2 to 3.6 [3], where all expressions of x and y positions are in modulo of 5, and z positions in modulo of w.

The step θ is a linear mapping to provide significant diffusion for the Keccak function. Its operation is shown in Eq. 3.2, and Fig 3.10 gives a more intuitive schematic representation. The (x, y, z) bit is updated with the XOR result of the (x - 1, z) and (x + 1, z - 1) column. The step ρ consists of translations within the lane. As shown in Eqn. 3.3, each lane's translation relies on its (x, y) coordinates to introduce inter-slice dispersion. The step π changes the lane positions within the state. The lane transposition follows Eqn. 3.4, without any operation in the z-direction. The step χ is the only non-linear mapping in the Keccak function and is translation-invariant in all directions. As shown in Eqn. 3.5, the bit is updated with itself and adjacent bits using AND, NOT, and XOR operations. Fig. 3.11 provides a more intuitive schematic representation of χ. The step ι consists of the round constant addition at lane (0, 0) to disrupt symmetry, as shown in Eqn. 3.6. The round constant changes based on the round number. Table 3.1 shows the pre-defined round constants for the Keccak-f[200] hash function.

$$\theta\text{: } A[x][y][z] = \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1] \tag{3.2}$$

$$\rho\text{: } A[x][y][z] = a[x][y][z-(t+1)(t+2)/2], \tag{3.3}$$

$$\text{where } 0 \leqslant t < 24 \text{ and } \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}^t \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}, \text{ or } t = -1 \text{ if } x = y = 0$$

$$\pi\text{: } A[x][y] = a[x'][y'], \text{ where } \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{3.4}$$

$$\chi\text{: } A[x] = a[x] + (a[x+1]+1)a[x+2] \tag{3.5}$$

$$\iota\text{: } A[0][0] = a[0][0] + RC[i_r], \text{ where } i_r \text{ is the round number} \tag{3.6}$$

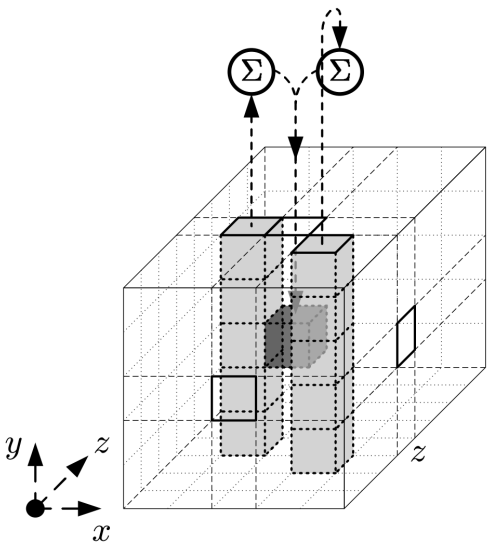

Figure 3.10: Keccak θ Step [3]

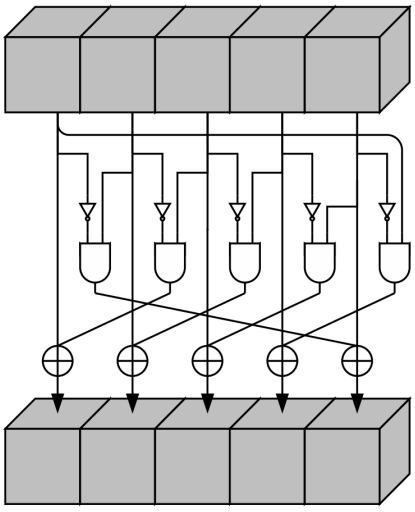

Figure 3.11: Keccak χ Step Applied to a Single Row [3]

Table 3.1: Round Constants for Keccak-f[200] [6]

RC[0]	0x01	RC[6]	0x81	RC[12]	0x8B
RC[1]	0x82	RC[7]	0x09	RC[13]	0x8B
RC[2]	0x8A	RC[8]	0x8A	RC[14]	0x89
RC[3]	0x00	RC[9]	0x88	RC[15]	0x03
RC[4]	0x8B	RC[10]	0x09	RC[16]	0x02
RC[5]	0x01	RC[11]	0x0A	RC[17]	0x80

Chapter 4

Quality-Driven TDC PUF & Arbiter PUF Implementation

Although detailed explanations of the system design have been provided in Chapter 3, its actual implementation on FPGAs still faces many challenges. This chapter aims to discuss quality-driven PUF implementation techniques on FPGAs.

4.1 TDC Measuring Range

The TDC needs a continuous and reasonable measuring range with respect to the entropy circuit to ensure delay data validity. To achieve this, the positions of extraction points on the delay-matching chain should be carefully managed. Too far a distance between two extraction points leaves a gap in the measuring range, resulting in jumping directly from

underflow to overflow. On the other hand, a distance too close results in too much overlap between measuring range segments, which shortens the overall measuring range. Ideally, the delay between two extraction points should be slightly smaller than the total delay of the 128-stage buffer chain, to provide a continuous measuring range without too much overlap. Also, the delay of buffers in the 128-stage buffer chain is preferred to be smaller than the ones used for delay matching, to provide a finer resolution within each measuring range segment. Unlike ASIC designs with complete P&R flexibility of circuits, FPGAs have limited types of instances and signal routing paths available. Thus, it is difficult to achieve and maintain this delay balance, and extra P&R optimizations are required.

Our implementation mapped the 128-stage buffer chain to CARRY4 instances, and the delay-matching buffers to LUT1 instances on Artix-7 FPGAs. Since each CARRY4 instance can be used as 4 cascaded buffers, as shown in Fig. 4.1, a total of 32 instances were used. The theoretical delays of CARRY4 and LUT1 instances were found with post-implementation timing simulations. The input-to-input delay between two tightly cascaded LUT1 instances is approximately 450 ps, and around 120 ps between CARRY4 instances. Therefore, extraction points can theoretically be at most 8 LUT1 buffers apart. However, due to manufactural variations, the actual signal propagation delay can be inconsistent. In our final implementation, we decided to separate extraction points with half of the theoretical maximum to ensure TDC stability. Extraction points were separated by 4 LUT1 instances, with a total of 16 extraction points and 64 LUT1 instances for the

delay matching block.

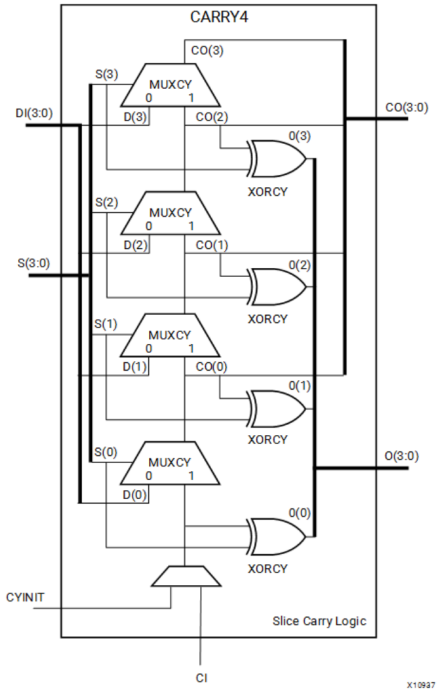

Figure 4.1: CARRY4 Instance of Xilinx 7-Series FPGAs [4]

4.2 Flip-Flop vs. Latch

The delay data generation mechanism shown in Fig. 3.3 captures the signal propagation status with FFs, with the FF clock signal coming from the delay matching block. The delay should be minimized between the delay matching block and the FF clock ports so that the clock arrival time can be tightly controlled solely with the delay matching block. ASICs can achieve this by carefully designing the local clock tree routing. However, FPGAs have clock tree routings fixed for FFs without any flexibility. The clock signal is

routed to a global buffer (BUFG) before getting redistributed to FFs. This lowers clock skews in normal FPGA designs but significantly increases the overall clock signal delay, which is unpleasant for our TDC PUF implementation. The additional delay introduced by the BUFG will raise the limit on the minimum delay that the TDC can measure. If the delay of the selected circuit is smaller than the BUFG delay, the TDC will never be able to perform a valid delay measurement on it.

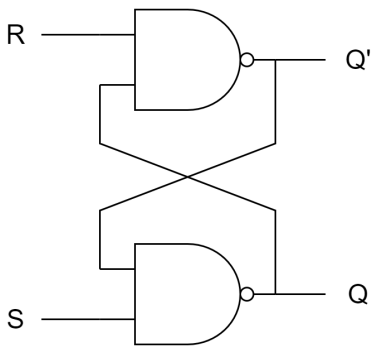

Figure 4.2: NAND Latch Structure

Table 4.1: NAND Latch Truth Table

Set	Reset	Output
1	1	No Change
0	1	Q=1
1	0	Q=0
0	0	Invalid

To solve this problem, we replaced FFs with NAND latches, which support more flexible P&R on FPGAs. A NAND latch is constructed by two cross-coupled NAND gates, as shown in Fig. 4.2. There are three valid states and one invalid state for a NAND latch, as shown in Table 4.1. The latch is locked when both *Set(S)* and *Reset(R)* are high, preventing any changes to the output Q. The latch is in transition states when S and R have different values. The latch is in the invalid state when both S and R are low, with Q' and Q both having high outputs.

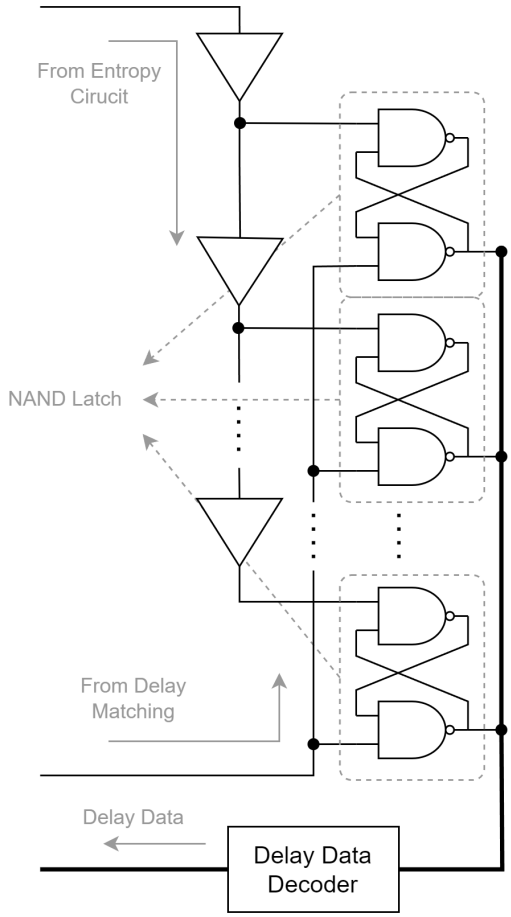

Figure 4.3: TDC with NAND Latch

The revised TDC design is shown in Fig. 4.3, with FFs replaced by NAND latches. The *Reset* ports are connected to the buffer chain, and the *Set* ports are connected to the lock signal from the delay matching block. The signal propagation direction in the CARRY4 chain is opposed to the latch-locking direction. This prevents the signal in the 128-stage buffer chain from racing against the lock signal, thereby avoiding "bubbles" (0s) in the locked propagation status. One drawback of using latches is the extra settling time

compared to FFs. When a latch changes states, the output experiences a brief period of metastability, where the signal bounces between the cross-coupled NAND gates before settling down. Therefore, the system must wait for extra clock cycles before getting stable outputs from the latches. The number of clock cycles to be waited depends on the chip model and the system clock speed. For our implementation on the Artix-7 FPGA chip with a clock frequency of 25 MHz, 2 extra clock cycles were waited per delay measuring operation.

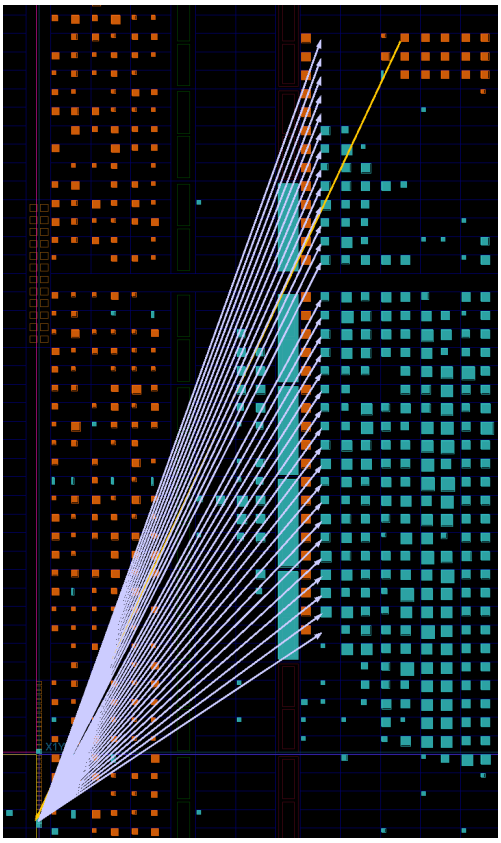

Figure 4.4: Clock Routing for FF-based TDC Implementation

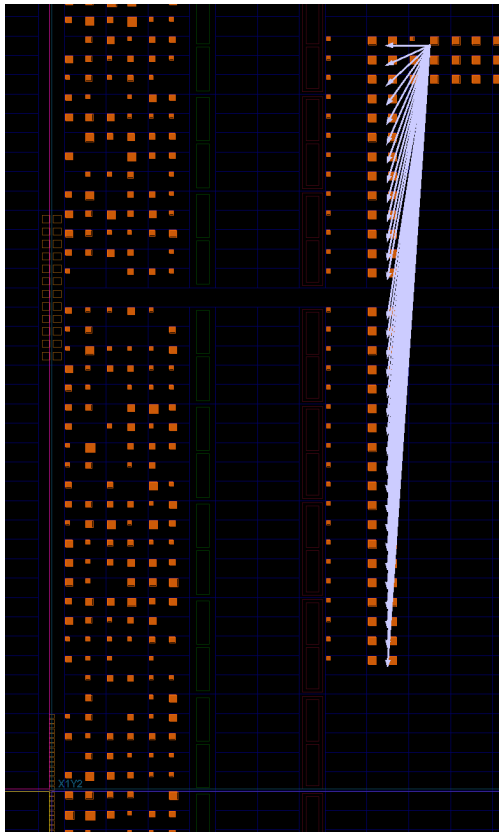

Figure 4.5: Clock Routing for Latch-based TDC Implementation

The actual P&R of FF-based and latch-based implementations are shown in Fig. 4.4 and 4.5. For FF-based implementation, the signal from the delay matching block was routed to the BUFG (yellow) before being redistributed to each FF (purple). In contrast, the lock signal was directly distributed to each latch in the latch-based implementation. This significantly reduced the clock routing distance and retained tight control over the delay matching mechanism. According to the post-implementation timing analysis, the clock delay of the latch closest to the delay matching block was reduced by nearly 85% (from 2132 ps to 327 ps). Also, the CARRY4 chain and the lock signal output were placed and routed to have opposite signal propagation directions (bottom-up for CARRY4 chain signal, top-down for latch lock signal) to avoid "bubbles" in the signal propagation status. Fig. 4.6 showed a more detailed P&R of a latch. NAND gates were mapped to LUT2 instances on FPGAs, and they were cross-coupled to form latches. The top (*Reset*) and bottom (*Set*) latch inputs were connected to the CARRY4 chain and the delay matching block, respectively.

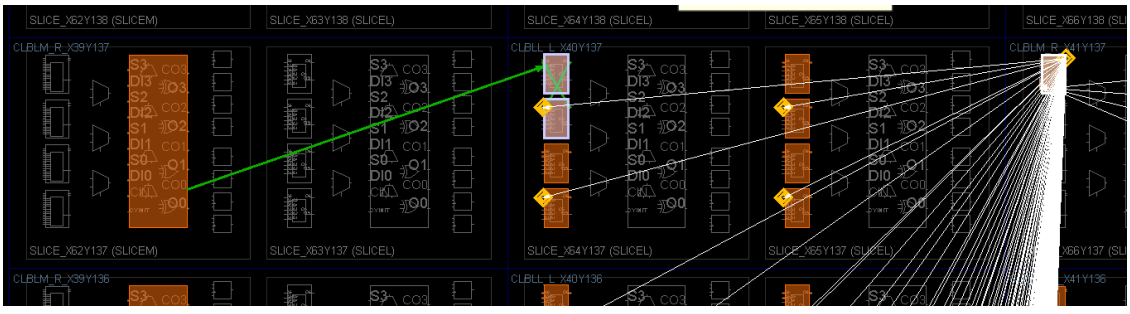

Figure 4.6: P&R for NAND Latch Implementation

4.3 TDC Placement & Routing

TDC is the most crucial component of the delay measuring mechanism. If the Computer-aided design (CAD) tool fails to generate a low variability TDC P&R, the quality of the delay data generation cannot be guaranteed. When adjacent delay matching buffers are placed too far such that the delay between them is longer than the total delay of the CARRY4 chain, gaps will exist in the measuring range. Similarly, bad P&R of CARRY4 instances will cause uneven measuring resolution and biased delay data. For a pair of adjacent CARRY4 instances with a large inter-instance signal propagation delay, the signal takes longer to reach the next CARRY4 instance and jump to the next delay data. In this case, the delay data will be more biased to the front CARRY4 instance.

CAD tools usually provide built-in P&R strategies for setup/hold-time and power-performance-area (PPA) optimizations. However, most of them are goal-driven and do not carefully control the signal transient behavior. Since our TDC PUF relies heavily on the actual signal propagation within the circuit, manual TDC P&R optimization is required for high-quality delay measuring.

The automated and manually-optimized P&Rs for the delay matching block are shown in Fig. 4.8 and 4.7. Compared to the automated P&R, the optimized one had LUTs more compactly and orderly placed to minimize inter-LUT delay variations. For the optimized P&R, the signal propagation starts from the bottom right register, and gets extracted to the multiplexer in the top left corner after propagating through every set of 4 LUTs.

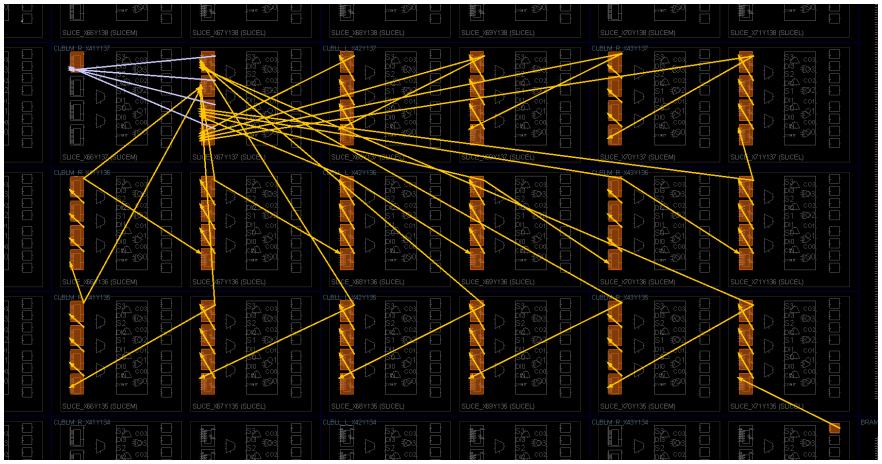

Figure 4.7: Optimized P&R of Delay Matching Module

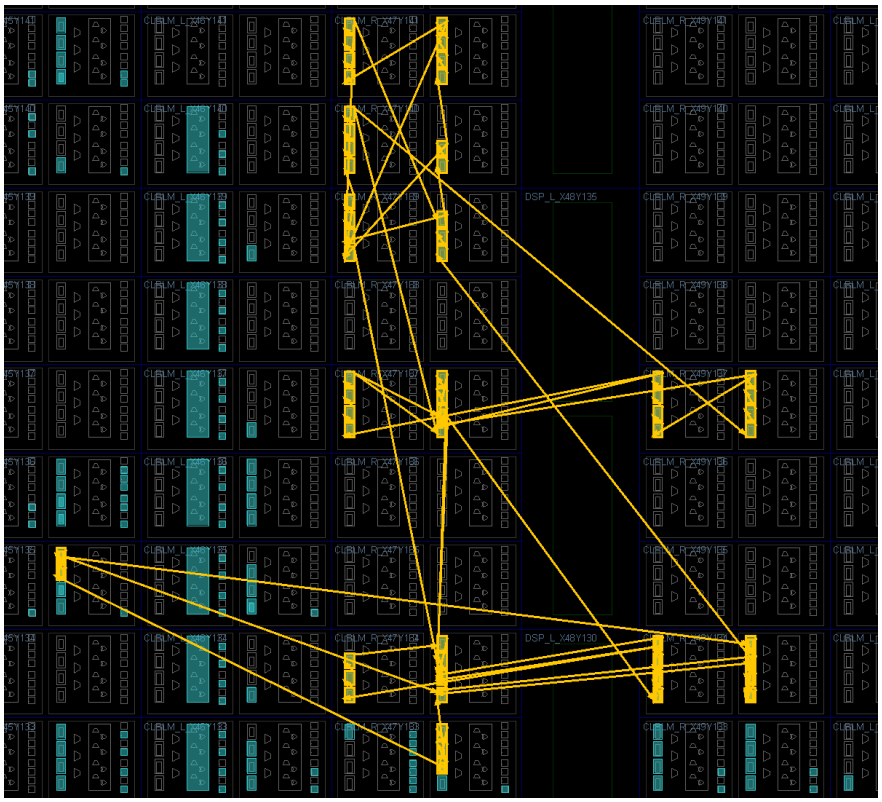

Figure 4.8: Automated P&R of Delay Matching Module

The CARRY4 chain had a default P&R strategy to minimize delay, as highlighted in purple shown in Fig. 4.9. However, the latches of the automated P&R were scattered around the chain. As aforementioned, the signal propagation direction of the CARRY4 chain must be opposite to the latch locking direction to ensure high-quality delay data generation, which was not achieved with the automated P&R. In the optimized P&R shown in Fig. 4.10, latches were closely and sequentially placed beside their corresponding CARRY4 instances. By placing the output of the delay matching block at the top, the latch locking can be controlled to follow the top-to-bottom sequence, opposing the CARRY4 chain direction.

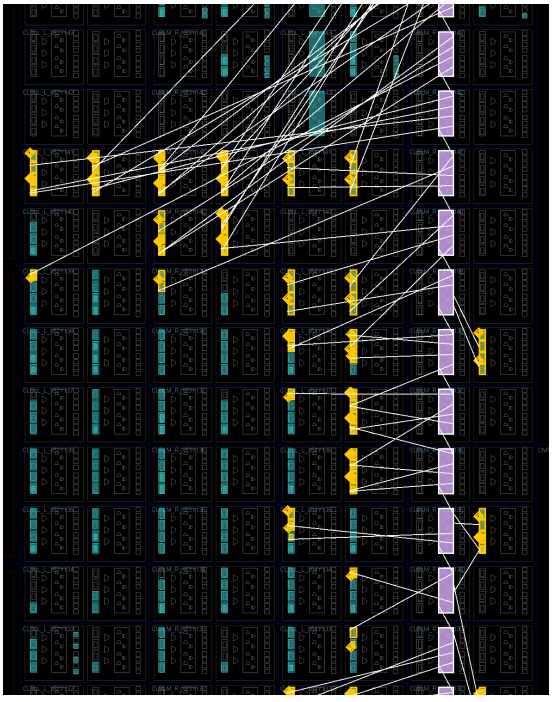

Figure 4.9: Automated P&R of CARRY4 Chain and Latches

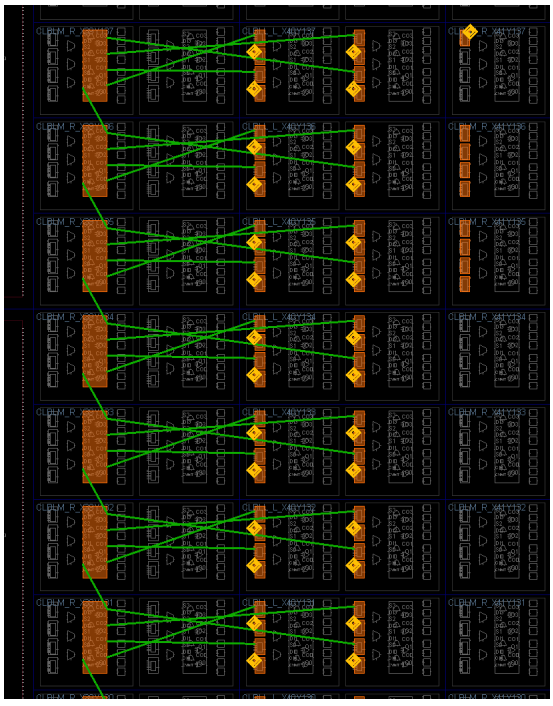

Figure 4.10: Optimized P&R of CARRY4 Chain and Latches

4.4 Entropy Circuit Routing & Placement

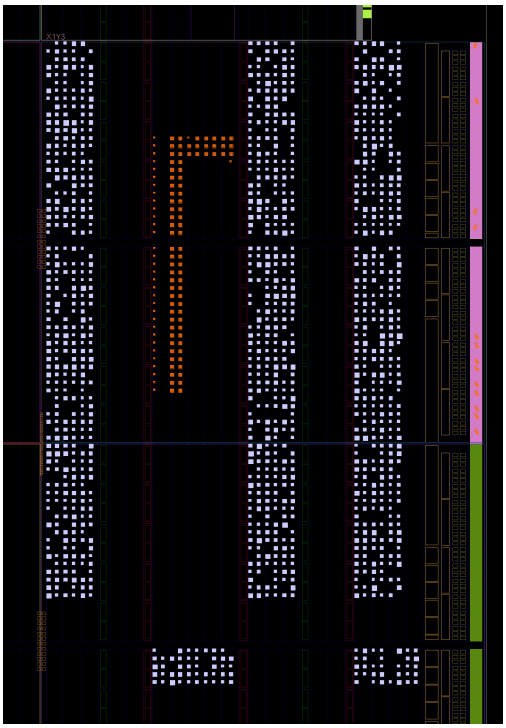

Figure 4.11: Randomized Entropy Circuit Placement

Unlike the low-variable P&R strategy for the TDC, the entropy circuit is preferred to be as chaotic as possible while maintaining the correctness of its basic logical operations (Keccak hash function). With an ideally randomized entropy circuit, the signal propagation delay should uniformly distribute across but not exceed the available TDC measuring range to maximize the PUF randomness. For our implementation, a separate Python program has been created to randomize the entropy circuit placement. This program randomly assigns the required FPGA resources of the entropy circuit to specific user-defined locations. An example of the randomized circuit placement is highlighted in purple as shown in Fig. 4.11,

with resources scattered over the five user-defined FPGA locations. The randomnesses of the entropy circuit with automated and randomized P&Rs are evaluated and compared in Section 5.3.

4.5 Debugging

For FPGA designs, modules focused on logic operations, such as finite-state machines (FSMs) and decoders, can be easily verified with testbenches and simulations. However, timing-related modules are more difficult to debug. Since the performance of delay-based PUFs is closely related to the actual timing characteristics, it is crucial to find efficient timing-related monitoring and debugging techniques.

Most FPGA CAD tools, such as Vivado for Xilinx, can perform post-implementation timing analysis with built-in timing models. However, due to manufacturing variations, actual timing characteristics vary slightly. Using safety factors to cope with these variations is only adequate for making general implementation choices, such as choosing extraction points on the delay-matching buffer chain. Certain optimizations, such as for randomness and uniqueness, requiring analyses of the actual FPGA delay data are impossible to achieve with simulations.

One possible solution is to use the integrated logic analyzer to monitor the signal propagation status within the FPGA. This method is effective when detailed signal propagation status is required, as logic analyzers can usually monitor at high frequencies.

However, logic analyzers have limited memory space and recording time, which makes it inefficient for large-quantity delay data extractions.

When requiring a substantial volume of delay data, a more effective approach is to read from the BRAM after delay data generation. Our implementation achieved this with a universal asynchronous receiver-transmitter (UART) module. Each UART data packet can transmit one byte of data. Thus, each delay read operation requires two UART transactions to transfer the 11-bit delay data and its 1-bit helper data. A Python program was also created to automate the read operations, to further increase the efficiency of delay data extraction.

4.6 Optimization - Aging Effect

The aging effect is common for IC chips and can impact signal propagation delay. Since it is impossible to mitigate aging effects completely, more worth considering problems are the detection of aging and its handling once detected.

The most obvious symptom of the aging effect is the inability to regenerate the correct keys. For our KMU, key requesters should verify key correctness at the software level. Each key requester saves a small amount of meaningless data along with its encrypted version. After retrieving the key from the KMU, the correctness can be verified by decrypting the ciphertext and comparing it with the original data.

The key regeneration success rate is expected to be high right after enrollment. For

each key request, the key requester can record the regeneration times taken before getting the correct key. Meanwhile, a threshold can be set for it, and exceeding this threshold indicates the probable happening of aging. Besides aging, temperature and voltage are the two most significant variation factors for signal propagation delay. Since these two factors are externally observable and controllable, their impacts can be excluded with careful management. Ultimately, it is reasonable to assume that aging is the only factor left.

Fortunately, aging usually happens gradually. The system will not experience a sudden failure, which provides a time margin to repeat the enrollment process. The key will change correspondingly if the affected circuit path has a new stable delay. If the circuit path becomes unstable, it will be flagged in the helper data and excluded from key generation. Since this thesis focuses on hardware-level implementations, this aging-effect solution is only proposed here to provide a software-level implementation guide for key requesters.

4.7 Optimization - Temperature Variation

Signal propagation delay is extremely susceptible to temperature variations. To ensure the reliability of delay-based PUFs, it is important to find approaches to minimize the impact of temperature variations.

The most direct solution is to create a feedback loop between the FPGA temperature sensor and the heat dissipation unit, trying to maintain the ideal operational temperature. However, this can only filter out impacts of large temperature fluctuations, whereas more

refined enhancements are still required.

One possible approach is to compensate for delay data errors with offsets. This approach requires the collection of delay variations at various temperature points and later making corrections according to the real-time chip temperature. However, temperature variations are not uniform across the chip. The impact of variations needs to be collected under a refined temperature resolution for each circuit path, which is not practical in terms of time and memory cost. Another more practical and effective approach is to perform error corrections on PUF responses with ECCs as aforementioned. However, the robustness of error correction is closely related to the amounts of ECCs generated. This ECC data overhead will increase significantly when more keys are enrolled with the KMU.

We proposed an efficient hardware-level optimization for our TDC PUF, as shown in Fig. 4.12, to enhance its temperature variation resistance. The 128-stage buffer chain is divided into groups of 4. The signals after each stage within the same group are connected to an OR gate. The delay data consistency can be preserved as long as the delay variation stays within the same group. However, delay data originally landing on the edge stages of a group might jump to adjacent groups under temperature variations. Therefore, only delay data landing in the middle two stages of a group are reliable.

To achieve this, more complicated logic than in Fig. 4.12 is required, and the actual implementation of this optimization is shown in Fig. 4.13. All the operations above are integrated into the delay data decoder. The 128-bit propagation status is decoded into 5-bit

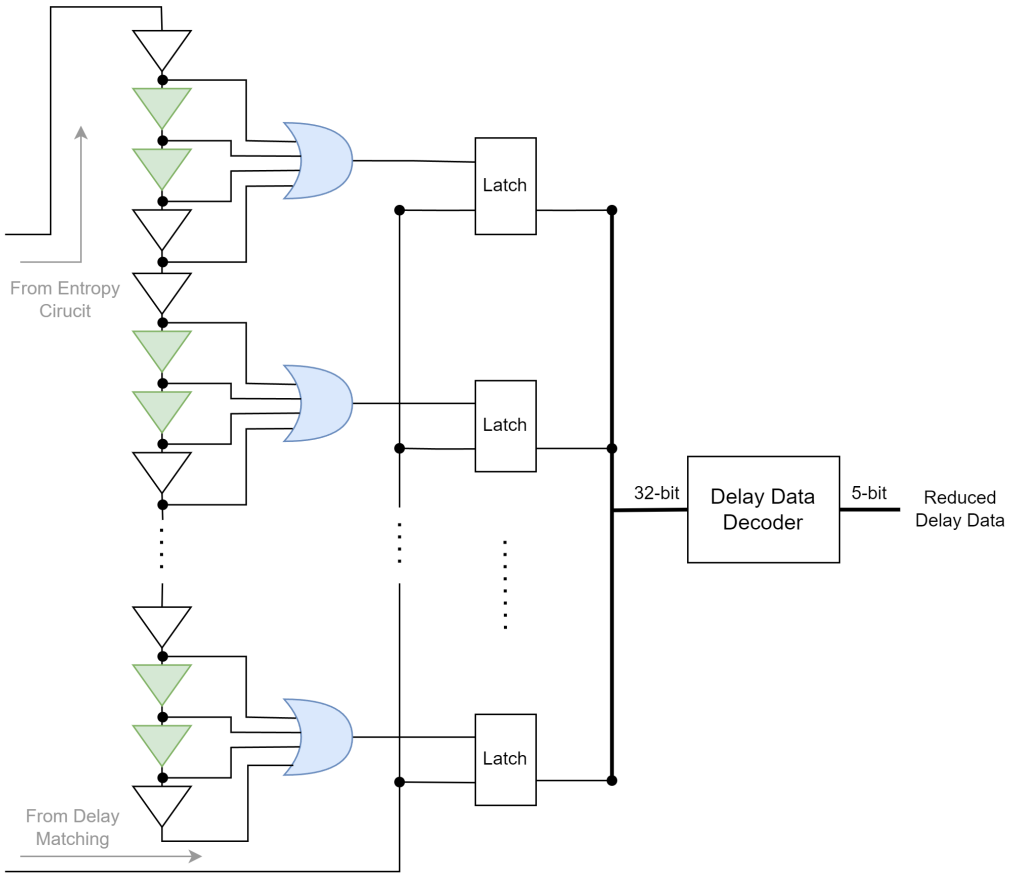

Figure 4.12: Proposed TDC Optimization for Temperature Variation

delay data according to the aforementioned technique. The decoder also generates an extra helper data bit to indicate if the measured delay lands in the middle two stages of a group.

The trade-off of this method is the reduction in PUF entropy. The length of the decoded signal propagation status decreases from 7 bits to 5 bits. An additional switch can be added to the KMU so that users can enable/disable this optimization feature based on their demand. Although this method creates extra hardware overhead, this overhead is one-time and will not increase with the number of keys. Since this technique makes raw delay data

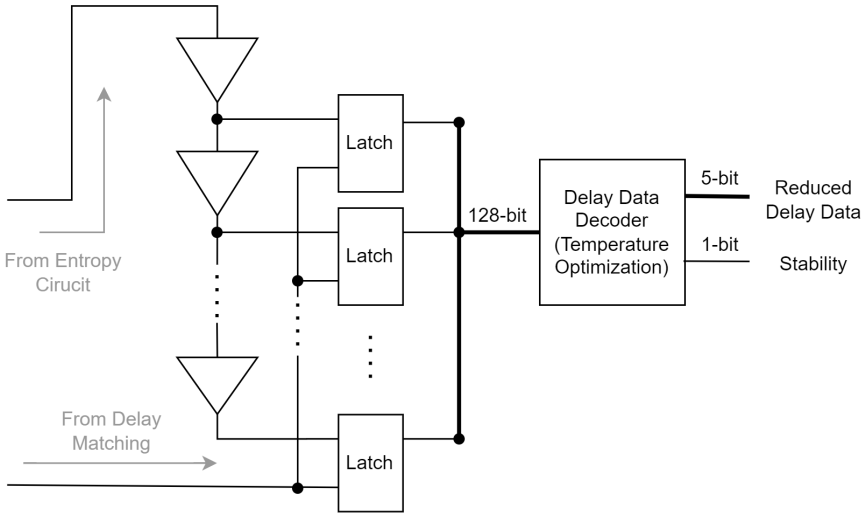

Figure 4.13: Actual Implementation of TDC Optimization for Temperature Variation

more robust against temperature variations, it can also reduce the ECC overhead while maintaining the same level of error correction robustness, which makes it a great add-on to the ECC approach. The performance of this hardware-level optimization is evaluated in Section 5.4.

4.8 Arbiter PUF Implementation

The traditional APUF was chosen as a performance reference for our TDC PUF since it is one of the most commonly applied PUFs. To match with the 128-bit key ID input of our TDC-PUF-based KMU (the only challenge externally observable is the 128-bit key ID), we implemented a 128-stage APUF.

The 2-to-1 MUXs of the APUF were mapped to LUT3 instances. Due to the restricted

clock signal routing, the same latch-based optimization was used for the APUF implementation, as shown in Fig. 4.14. Fig. 4.15 shows the actual implementation of the APUF on the FPGA. The P&R of the APUF was tightly controlled to prevent significant delay differences between any two stages, which would otherwise cause biased delay data.

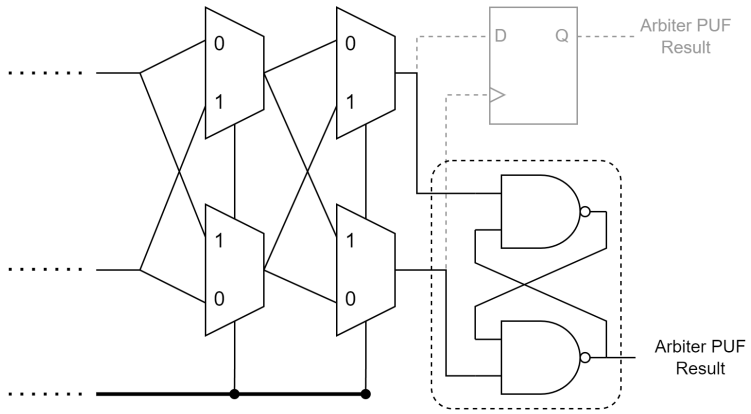

Figure 4.14: Arbiter PUF with Latch-based Implementation

Figure 4.15: Arbiter PUF P&R

Chapter 5

KMU Results & Analysis

This chapter aims to evaluate the reliability, uniqueness, and randomness of our TDC-PUF-based KMU. Where necessary, the TDC PUF's performance is also compared to the reference 128-stage APUF. Finally, the effectiveness of the hardware-level optimization against temperature variations is also evaluated.

5.1 Reliability

Enrollment time and key consistency are the two most critical reliability aspects for our KMU. The enrollment time highly depends on the key regeneration threshold and the amount of delay data required. The KMU security level increases with the amount of delay data used. However, this also requires a larger key regeneration threshold to ensure helper data quality, leading to longer and unstable enrollment time. Therefore, the delay data amount and the

key regeneration threshold should be carefully chosen for a reliable KMU implementation.

Fig. 5.1 shows the impact of delay data amount on enrollment time with different key regeneration thresholds under a clock frequency of 25MHz. The enrollment times are stable and steadily increase with the key regeneration threshold until the delay data amount reaches 30K. To maximize the KMU security level while maintaining a consistent enrollment time, we chose the delay data amount of 10K for our implementation.

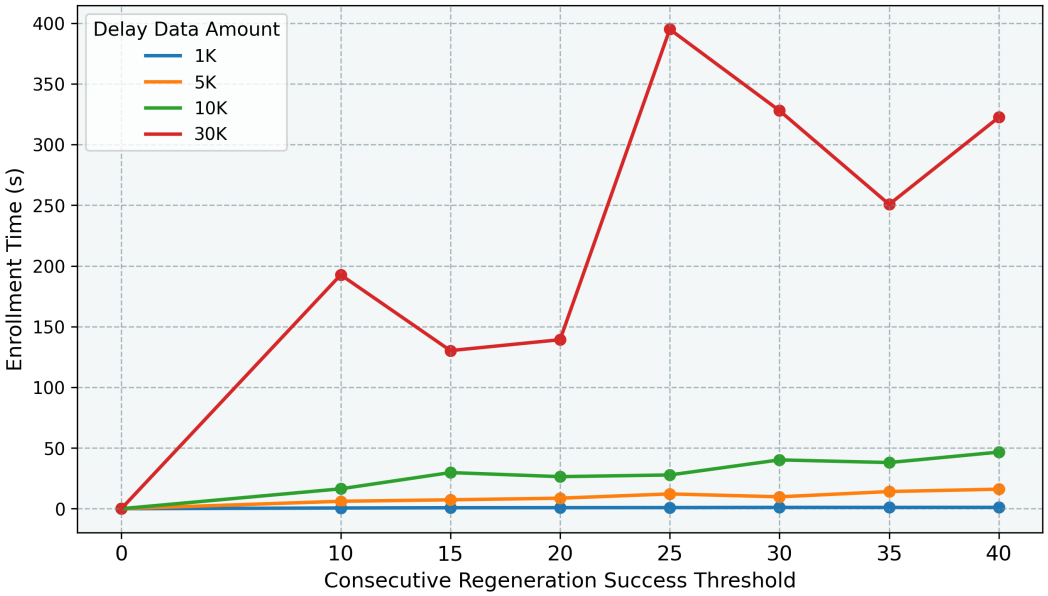

Figure 5.1: Enrollment Time with Varying Delay Data Amount under 25MHz Clock

With a desired delay data amount, the subsequent task is to find the optimal key regeneration threshold. Although enrollment time increases with the key regeneration threshold, the helper data are also more effective in identifying unstable circuit paths. Therefore, the balance of this trade-off should be carefully considered. The effectiveness of

helper data is evaluated with Eqn. 5.1. It is a modified version of the formula proposed in [26], making it more suitable for our TDC PUF. This equation calculates the average HD between the reference delay data and the regenerated ones. The delay data are collected in batches, with 10K delay data per batch. Variables m and n represent the number of batches and the number of responses (delay data) per batch, respectively. $HD(R_i, R'_{i,t})$ represents the HD between the reference response R_i and the regenerated response $R'_{i,t}$. The HD is then multiplied by its stability flag $S(i)$ according to the helper data (1 if stable, 0 if not) so that the effectiveness of helper data can be evaluated. To regenerate the same key, the ideal HD value is 0. Since there is no error correction mechanism in our current implementation, non-zero HD cannot be tolerated. When an incorrect key is generated, the system needs to issue another key request.

$$HD_{INTRA} = \frac{1}{m}\sum_{t=1}^{m}\sum_{i=1}^{n} HD(R_i, R'_{i,t})S(i) \tag{5.1}$$

The delay data used for this analysis were generated with combinations of 10 different key IDs and 8 different key regeneration thresholds. Delay data for each combination were generated 10 times. The key regeneration success rates were also collected to evaluate the helper data effectiveness, where each key was regenerated 1K times. Fig. 5.2 shows the average enrollment time trend. The enrollment time increases linearly from 0 s to around 50 s, with the key regeneration threshold from 0 to 40. Deploying the key regeneration threshold effectively reduces HDs. The average HD per 10k delay data decreases drastically

from approximately 1.4K to 15 with just the first threshold value, as shown in Fig. 5.3. Also, the key regeneration success rate increases and gradually converges to around 90% as the key regeneration threshold increases, with relatively low standard deviations (error bars). Consequently, we chose 30 as the optimal key regeneration threshold for our KMU system to ensure the balance between the enrollment time and key regeneration consistency.

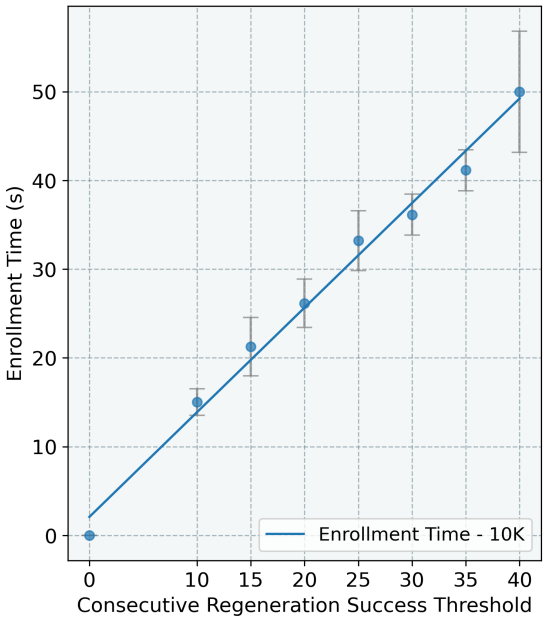

Figure 5.2: Enrollment Time with Varying Regeneration Threshold for 10K Delay Data

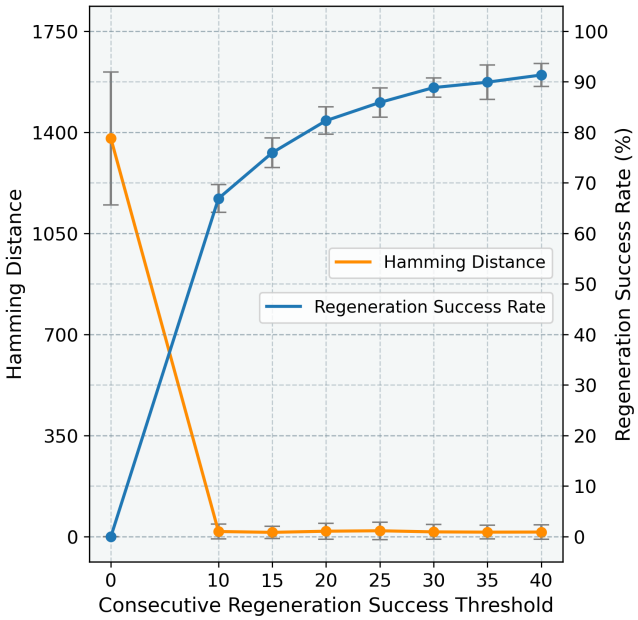

Figure 5.3: Delay Data HD & Key Regeneration Success Rate with Varying Regeneration Threshold for 10K Delay Data

5.2 Uniqueness

Uniqueness has always been an important criterion for PUF-based systems. Even if the same KMU system is deployed on FPGA chips of the same model, the generated keys are expected to be unique. To investigate the PUF uniqueness, responses generated with the same challenges must be collected from similar devices. The uniqueness can be quantified with Eqn. 5.2 [26], which calculates the average normalized HD of the two sets of responses. Variable n represents the number of collected responses, and b is the bit length of each response.

$$Uniqueness = \frac{1}{n}\sum_{i=1}^{n}\frac{HD(R_i, R_i^{'})}{b} \times 100\% \tag{5.2}$$

We investigated the uniqueness of our TDC PUF using the Artix-7 FPGA family. A rigorous uniqueness test was conducted with two chips of the same XC7A35T model. A more relaxed uniqueness test was also conducted between the XC7A35T and XC7A100T models, considering that system migrations within the same FPGA family are common. For each FPGA chip, 10K delay data were generated and collected with each of the 10 different key IDs. The uniqueness of the 128-stage APUF was also evaluated for comparison, where 100K CRPs were collected for each FPGA chip. For the TDC PUF, $b = 11$ and $n = 10$K, and the uniqueness was determined with the average of 10 key IDs. For the APUF, $b = 1$ and $n =$ 100K. To maintain test fairness, the relative positions of the hardware resource placements

were kept the same when migrating the KMU system and the APUF from XC7A35T to XC7A100T.

The uniqueness test results are shown in Table 5.1 and 5.2. For the uniqueness test of the same FPGA model, the TDC PUF has an average uniqueness of 27.64%, approximately 12.5x the APUF uniqueness of just 2.21%. For the cross-model test, the uniqueness of both PUFs increases. However, the uniqueness of the TDC PUF is still over 2x the APUF, with TDC PUF reaching 46.69% and APUF reaching 22.34%. Overall, the uniqueness of the TDC PUF used in our KMU system is significantly higher than the traditional APUF, which effectively ensures the uniqueness of keys generated with our KMU across different FPGA chips.

Table 5.1: Uniqueness Test - Artix-7: 35T vs. 35T

Artix-7: 35T vs. 35T							
Key ID	0	1	2	3	4	Average	Arbiter PUF
Uniqueness(%)	27.47	27.03	22.11	24.21	24.92	27.64%	2.21%
Key ID	5	6	7	8	9		
Uniqueness(%)	27.16	28.91	32.60	33.00	29.00		

Table 5.2: Uniqueness Test - Artix-7: 35T vs. 100T

Artix-7: 35T vs. 100T							
Key ID	0	1	2	3	4	Average	Arbiter PUF
Uniqueness(%)	45.03	46.97	48.77	39.33	33.76	46.69%	22.34%
Key ID	5	6	7	8	9		
Uniqueness(%)	51.81	55.14	43.63	49.33	53.14		

5.3 Randomness

Randomness is also a critical criterion for PUFs. As mentioned, the entropy circuit P&R is preferably as chaotic as possible so that the generated delay data have a more uniform distribution and are harder to predict. To achieve this, a randomness enhancement method has been proposed, and its performance is evaluated and discussed in this section.

Shannon's entropy is the widely adopted method of randomness evaluation, and it was also used for our PUF randomness analyses. We evaluated both the bit-wise and response-wise Shannon's entropy of the TDC PUF to provide analyses from two distinct perspectives.

Bit-wise entropy can better represent the resistance to bit-by-bit modeling attacks. This is the most straightforward attack approach without knowing the PUF type, where a numerical model is built for each bit of the response. The bit-wise entropy H_{bit_wise} is evaluated with Eqn. 5.3. It first calculates the entropy of each delay data bit, and then sums them together to get the total entropy of the delay data. The values of $p_{1,j}$ and $p_{0,j}$ refer to the occurrence frequencies of 1s and 0s for the j-th bit of the response, and n is the total number of bits of each response.

On the other hand, the response-wise approach can more effectively represent the resistance to modeling attacks when the response bits are known to be correlated. Each piece of delay data is viewed as a whole during the entropy calculation, as shown in Eqn. 5.4. Variable r represents the number of possible responses, and p_k refers to the occurrence frequency of the k-th response.

$$H_{bit\_wise} = (-1)\sum_{j=1}^{b} p_{1,j}\log_2 p_{1,j} + p_{0,j}\log_2 p_{0,j} \tag{5.3}$$

$$H_{response\_wise} = (-1)\sum_{k=1}^{r} p_k \log_2 p_k \tag{5.4}$$

Since each delay data is 11-bit in our TDC PUF implementation, b=11 in Eqn. 5.3, and r=2^{11} in Eqn. 5.4. The bit-wise and the response-wise entropy have the same theoretical maximum of 11 if the delay data uniformly distributes across the TDC measuring range.

To compare the entropy between the automated and optimized P&R of the entropy circuit, 10K delay data were generated with each key ID for both P&Rs. The comparison results are shown in Fig. 5.4. The variances of the entropy values are low across all key IDs. With the optimized P&R, both the bit-wise and response-wise entropy increase by more than two-fold. The average bit-wise entropy increases from 4.60 to 9.73, and the average response-wise entropy increases from 2.71 to 7.06.

The entropy of the APUF was also evaluated to compare with the TDC PUF. Since the APUF response is 1-bit, the theoretical maximum entropy is 1, and the entropy of the actual implementation reaches 0.998. To ensure a fair comparison between the APUF and the TDC PUF, the entropy of the TDC PUF must be normalized, which equals 0.885 for bit-wise entropy and 0.642 for response-wise entropy. Although the normalized entropy of the APUF is higher than the TDC PUF, the latter has more possible responses than the APUF.

APUFs can be duplicated and placed in parallel to generate multi-bit responses. However, since these bits are independent, the attackers can perform bitwise attacks in parallel using the same numerical model as the traditional APUF attack. Therefore, this approach does not effectively enhance the resistance to modeling attacks. On the other hand, since the response bits of the TDC PUF are interdependent, the simple bit-by-bit modeling attack is ineffective. Instead, a significantly more complex multi-class classification machine learning model will be needed to attack the TDC PUF effectively.

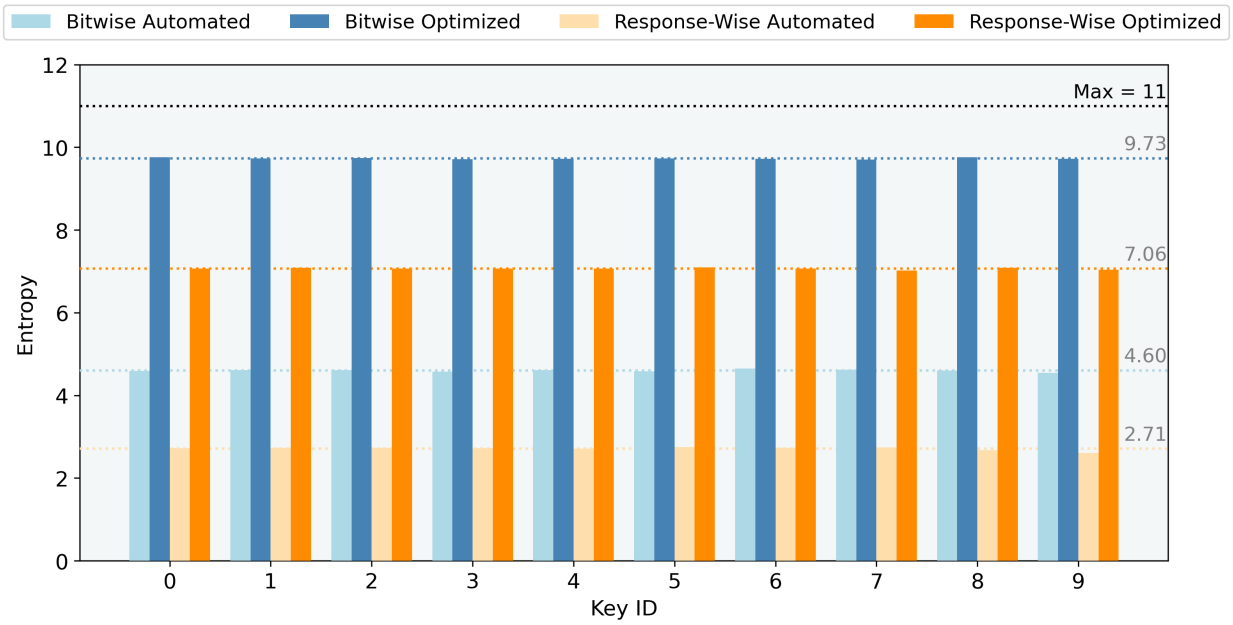

Figure 5.4: Bitwise & Response-Wise Entropy for Automated & Optimized P&R

5.4 Temperature Optimization

We also assessed the performance of the hardware-level optimization proposed in Section 5.4. The reference FPGA chip temperature for key enrollment was set to 30 °C. Optimization effectiveness was evaluated with the key regeneration success rates collected at [30, 32, 34, 36, 38] °C for both the pre-optimized and post-optimized implementations. Each key was regenerated 1K times, and 10 key IDs were used.

As shown in Fig. 5.5, for a 2 °C temperature increase, the optimized design is much more reliable than the pre-optimized design. There is a significant increase in key regeneration success rate, from around 10% to about 45%. This proves that the proposed hardware-level optimization can effectively enhance the resistance to small temperature variations. Meanwhile, larger temperature variations can be handled with a heat dissipation unit.

Furthermore, this optimization has a positive side effect of enrollment time reduction. As shown in Fig. 5.6, the average enrollment time reduces from 36.14 s to 12.53 s. The higher enrollment time of the pre-optimized design might be due to the small temperature increase during the enrollment process, and its impact has been reduced with the proposed optimization.

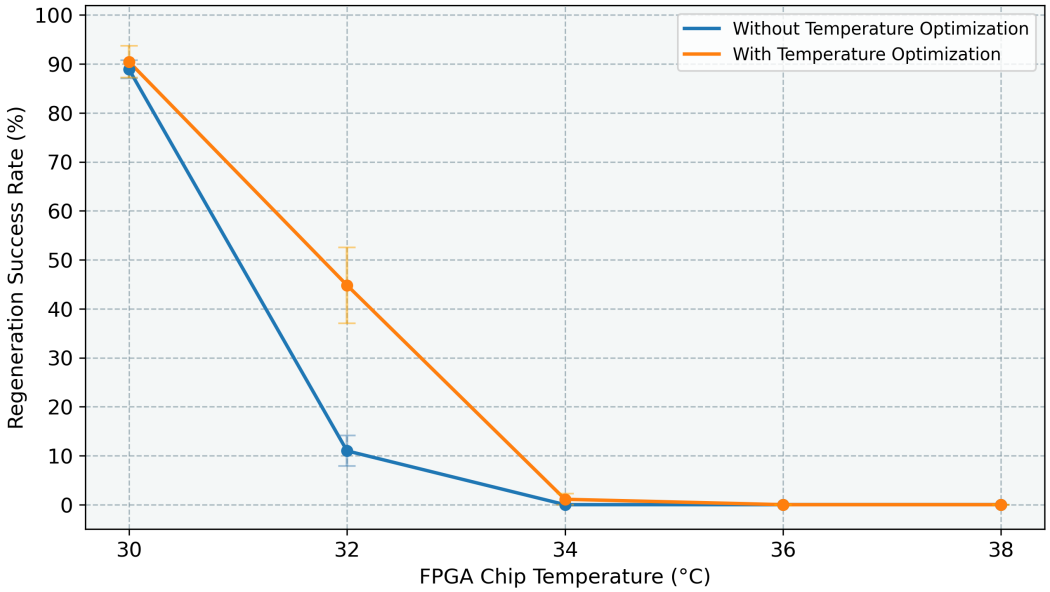

Figure 5.5: Effect of Temperature Optimization on Key Regeneration Success Rate

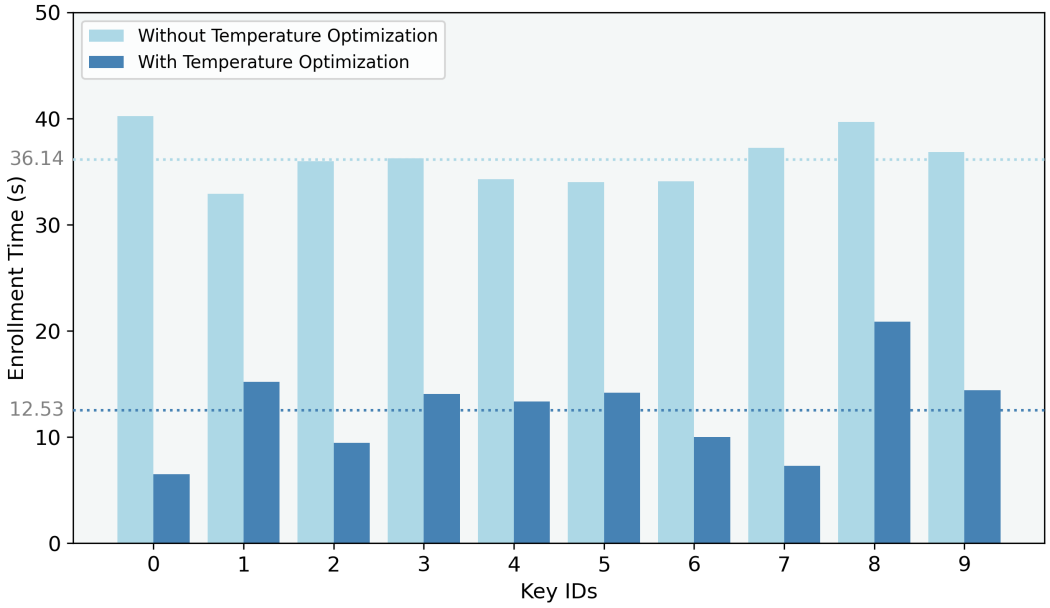

Figure 5.6: Effect of Temperature Optimization on Enrollment Time

Chapter 6

KMU RISC-V Integration

Building a complete RISC-V SoC from scratch is extremely complex and out of the scope of this thesis. For our KMU integration, we utilized existing RISC-V ecosystems of the Rocket Chip Generator [52] and Chipyard framework [53]. They were developed at UC Berkeley and are actively being maintained and used in academia and industry. Rocket Chip Generator is an open-source RISC-V SoC generator that allows engineers to customize SoC features, such as CPU cores and memory hierarchy. It is later integrated into the Chipyard framework, which provides a unified framework and workflow for agile SoC development.

The Chipyard framework allows two ways to integrate custom co-processors into the SoC: Rocket Custom Co-processor (RoCC) and memory-mapped I/O (MMIO). This chapter demonstrates the integration of our KMU using both approaches. The RoCC approach is for Rocket-Chip only. Although lacking generality, this approach fully utilizes

the customizability of the RISC-V ISA, where vacant instruction codes are assigned to different KMU functionalities. In contrast, the MMIO approach can support most SoC frameworks. The KMU is attached to the SoC as a peripheral, and dedicated memory locations are assigned to its I/Os for control and data access. The following sections discuss mechanisms and implementations for both RoCC and MMIO approaches.

6.1 KMU Memory Access Flow

The most critical part of the KMU integration is to read the required data for key generation and write back keys and helper data. The data access flows for MMIO and RoCC approaches are similar, as shown in Fig. 6.1. Before key generation/enrollment, the required information for memory access must be decoded and prepared, such as memory locations of helper data and key IDs. Subsequently, the key ID and the helper data are transferred from the main memory to KMU internal registers, providing faster data access during KMU operations. When the required data are available, the system initiates a KMU key request and waits for the key generation to complete. Once the key is generated, the system writes the key to the assigned memory location, and transfers the helper data back into the main memory if it is under key enrollment.

Although the RoCC and MMIO approaches share a similar data access flow, their underlying mechanisms are completely different. The next two sections discuss their implementations and results separately.

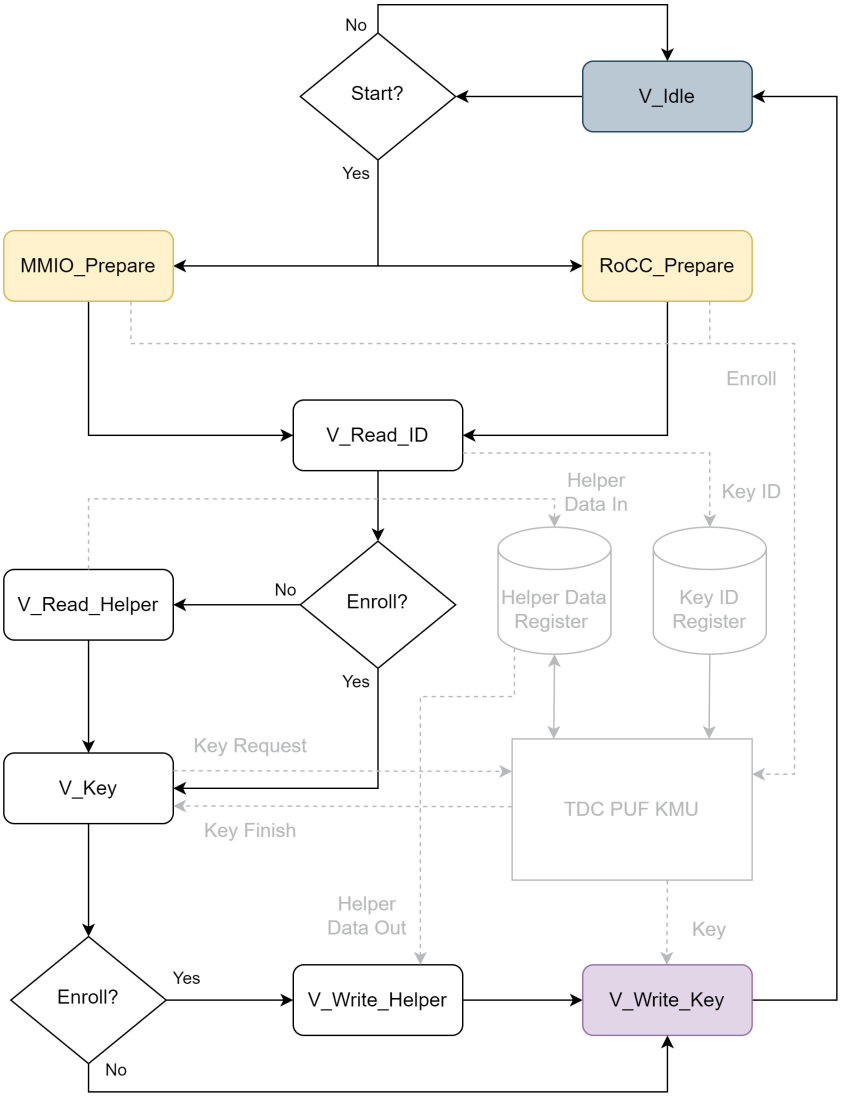

Figure 6.1: KMU Memory Access Flow for SoC Integration

6.2 RoCC Implementation

The RoCC interface is specifically designed to integrate co-processors with RISC-V Rocket cores, and Fig. 6.2 shows a simplified view of it. Custom instructions can be passed with the *cmd* channel to activate the co-processor, and its response can be returned with the *resp* channel. The co-processor can access the memory through the L1 Cache using the *mem.req* and *mem.resp* channels.

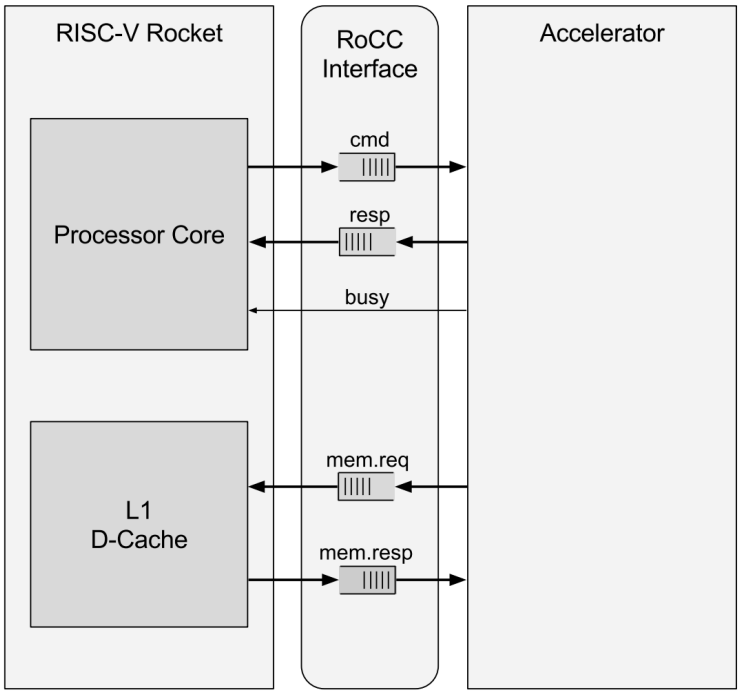

Figure 6.2: Simplified View of RoCC Interface [5]

The RoCC custom instructions follow the R-Type instruction format of the RISC-V ISA, as depicted in Fig. 6.3 for 32-bit instructions. The KMU is mapped to the *opcode* of 7'b0001011, which is one of the pre-assigned values for RoCC. KMU functionalities are

indicated by *funct7*, with 7'b0000000 for key generation and 7'b0000001 for key enrollment. The memory locations of the required data are passed by registers, with *rs1* for Helper Data, *rs2* for Key ID, and *rd* for key output. The *funct3* is used to indicate which of the *rs1*,*rs2*, and *rd* registers are used.

Adding custom RoCC instructions results in a non-standard RISC-V ISA, thus requiring a custom toolchain for code compilations. Fortunately, this has already been included in the Chipyard framework, which eases the integration process. The KMU driver for RoCC is shown in Alg. 1. Custom instructions are encoded and invoked with the pre-defined *ROCC_INSTRUCTION_DSS()* function of the Chipyard framework.

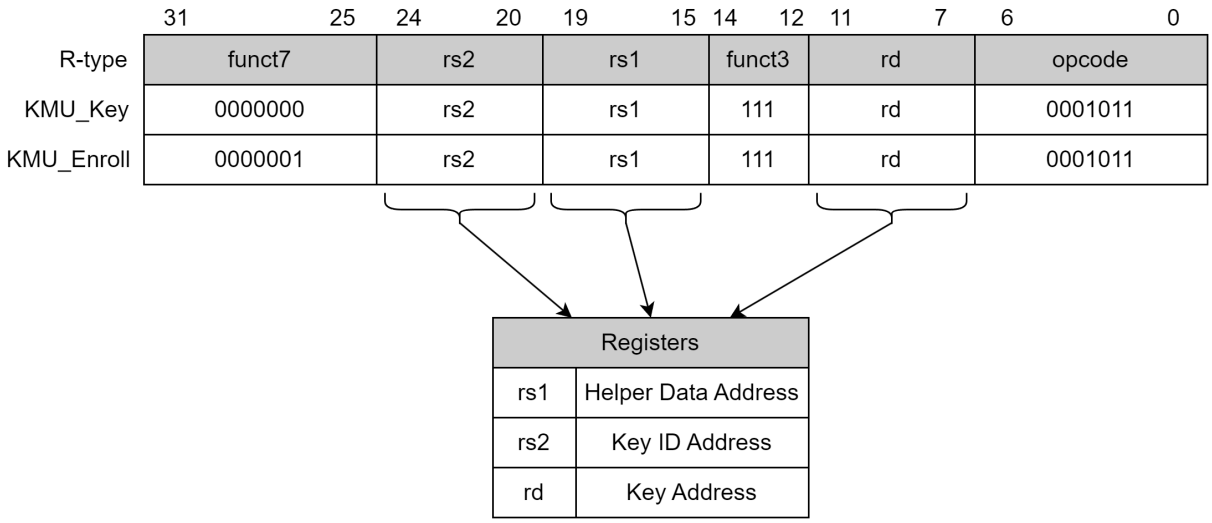

Figure 6.3: Customized RISC-V Instructions for KMU

Algorithm 1 KMU RISC-V RoCC Driver

Input: Enroll (E), Helper_Data_Addr (H), ID_Addr (I), Key_Addr (K).

```
1: function RoCC_Key_Request(E, H, I, K)
2:     asm volatile ("fence")          ▷ //Force to wait for Mem Operation Completion
3:     if E = 1 then
4:         ROCC_INSTRUCTION_DSS(opcode = 7'b0001011, K, H, C, funct7 = 7'h01)
5:     else
6:         ROCC_INSTRUCTION_DSS(opcode = 7'b0001011, K, H, C, funct7 = 7'h00)
```

To verify the integration, simulation files were created to mimic the memory access behaviors of our KMU during key enrollment and generation. Instead of the 10K helper data amount used in the actual KMU implementation, the simulation used only 1K, which was sufficient for behavioral verification. The simulation consisted of a key enrollment and a key generation with the same Key ID and helper data, as shown in Fig. 6.4 and Fig. 6.5. The key enrollment and generation were triggered by the correct opcode and funct, as shown in the last five wave lines of both figures. The memory access behavior of the enrollment process was simulated by writing an alternative bit pattern to the helper data buffer. The simulation result is as expected. Extensive write operations were performed on the helper data buffer during key enrollment. For the subsequent key generation, as shown in Fig. 6.5, the helper data from the same memory location had a hexadecimal pattern of 5s, equivalent to 4'b0101. This confirms the correctness of the helper data write-backs during key enrollment.

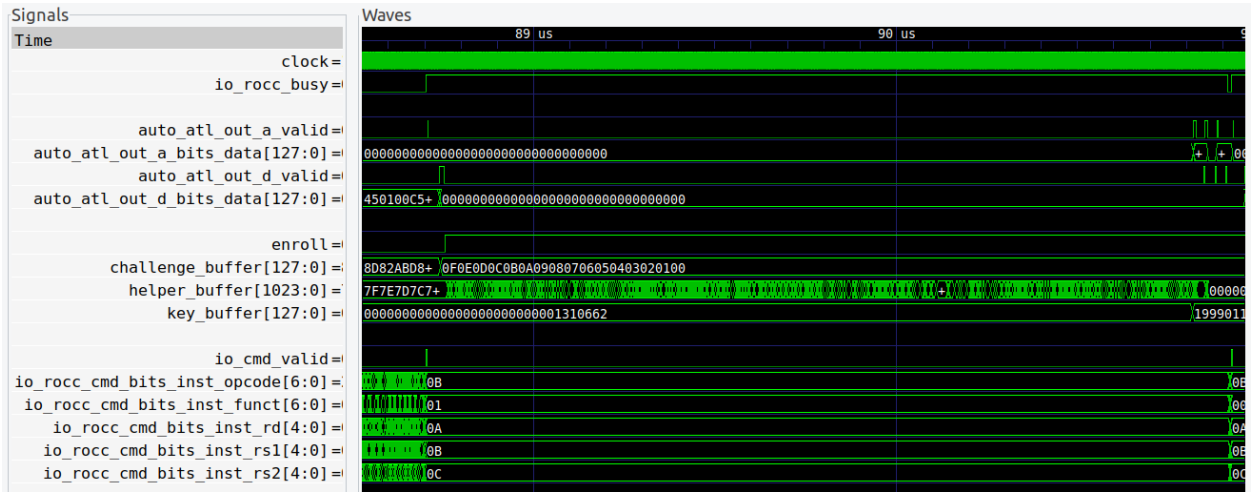

Figure 6.4: Simulation Result of RoCC Key Enrollment

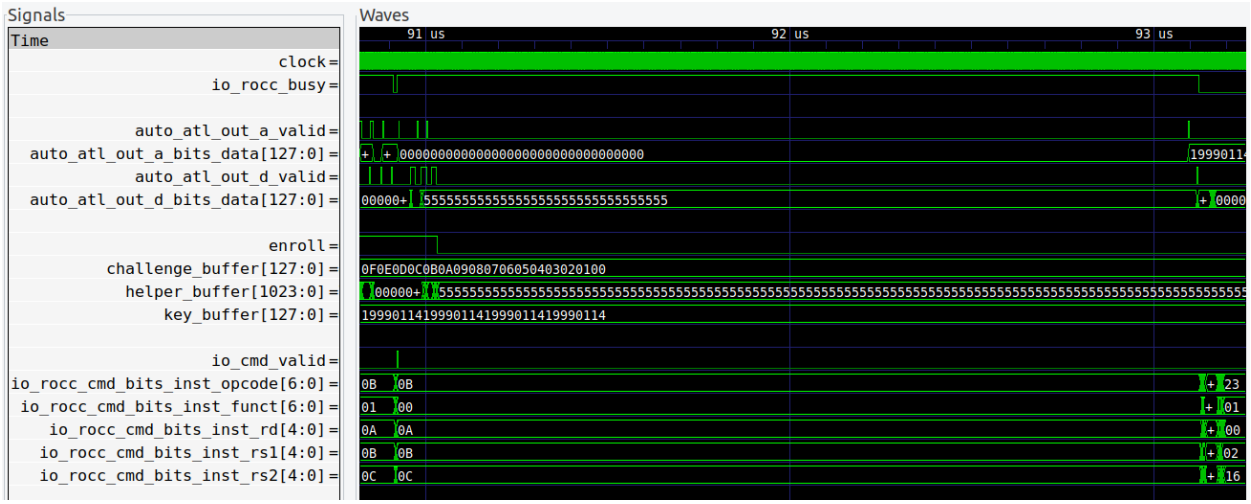

Figure 6.5: Simulation Result of RoCC Key Generattion

6.3 MMIO Implementation

Another integration approach is to attach the KMU as an MMIO peripheral. For the MMIO approach, a memory segment is allocated to the peripheral, and its I/O ports are mapped to different locations within the assigned memory range. This allows the system to treat the peripheral interactions as simple memory accesses, which makes the driver implementation compatible with standard toolchains.

The memory allocation of the KMU integration is shown in Fig. 6.6. The base address starts from 0x1000, with four I/O ports mapped in the assigned memory segment. The *KMU_Status* is a read-only output port to monitor KMU status so that the SoC system can properly interact with the KMU. The *KMU_Enroll* port is a write-only port to select the KMU function, with 0x0000 for key generation and 0x0001 for key enrollment. This port is also associated with a valid/ready interface, to indicate the data receive readiness and data validity. The KMU is activated with the valid flag of the *KMU_Enroll* port, which is asserted when the function mode is written. The *KMU_Data_In* and *KMU_Data_Out* ports are used for data accesses, including helper data, key IDs, and generated keys. Both ports are associated with valid/ready interfaces and are 64-bit in width, which is the maximum size supported by the SoC driver.

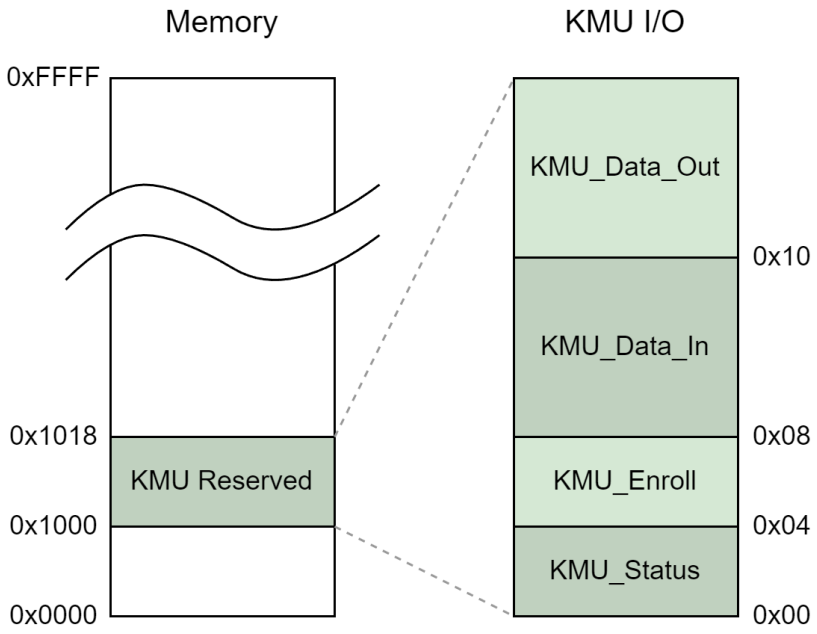

Figure 6.6: MMIO Mapping for KMU

Compared to the RoCC approach, the MMIO approach requires a more complex driver implementation. As shown in Alg. 2, appropriate memory read and write sequence must be followed. For a key request, the system first checks the availability of the KMU by reading the *KMU_Status* port. If the KMU is available, the *KMU_Enroll* port is written to select the function mode and initiate the process. Subsequently, the system transfers the required data (key IDs and helper data) to the KMU and waits for the key generation/enrollment to complete. Once finished, the helper data are written back to the main memory if under enrollment mode, followed by the key output.

Algorithm 2 KMU RISC-V MMIO Driver

Input: Enroll (E), Helper_Data_Buffer (H), Helper_Size (S), ID_Buffer (I), Key_Buffer (K).

```
1: function MMIO_Key_Request(E, H, S, I, K)
2:     while (MMIO_Read(0x1000, 1) & 0x1 = 1) do Wait()
3:     MMIO_Write(0x1004, E, 1)
4:     MMIO_Write(0x1008, I, 16)
5:     if E = 1 then
6:         MMIO_Write(0x1008, H, S)
7:     while (MMIO_Read(0x1000, 1) & 0x2 = 0) do Wait()
8:     if E = 1 then
9:         H ← MMIO_Read(0x1010, S)
10:    K ← MMIO_Read(0x1010, 16)
```

The MMIO simulation results are shown in Fig. 6.7 and 6.8 for key enrollment and generation, respectively. The same simulation files as the RoCC verification were used to mimic the KMU memory access behaviors. The KMU was activated as expected by the *enroll_valid* signal, and extensive write operations to the helper data buffer were performed during key enrollment. The helper data and the generated key were written back to the memory with the data output port once enrollment finishes (when *io_keydone* was asserted). For the following key generation, the system accessed helper data from the same memory location as the key enrollment. The read-back helper data had a hexadecimal pattern of 5s (equivalent to 4'b0101), verifying the correctness of write-back operations during MMIO key enrollment.

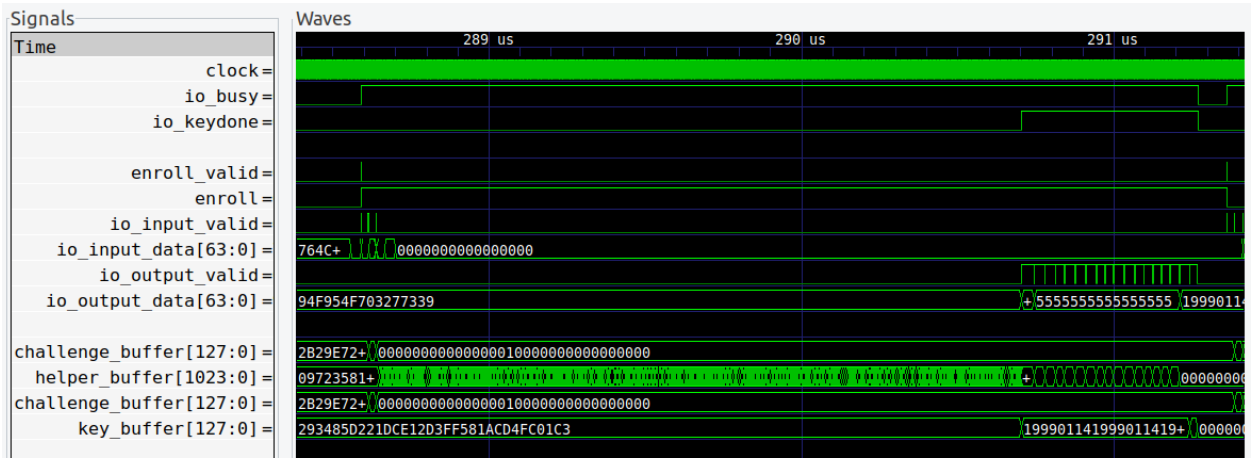

Figure 6.7: Simulation Result of MMIO Key Enrollment

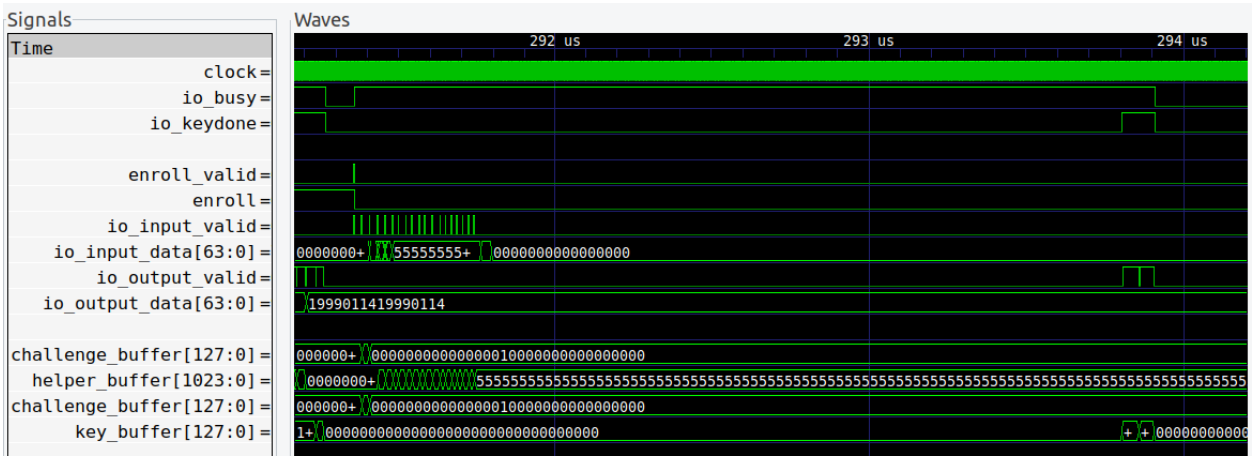

Figure 6.8: Simulation Result of MMIO Key Generation

Chapter 7

Conclusions & Future Work

In conclusion, this thesis presented the detailed design and implementation of a KMU on FPGAs, utilizing the more complex TDC PUF compared to the traditional APUF. With quality-driven design and implementation strategies, we tackled various challenges concerning the reliability, uniqueness, and randomness of the TDC-PUF-based KMU.

We achieved a key regeneration success rate of approximately 90%, facilitated by high-quality delay data generation and TDC-PUF-specific helper data. Compared to the traditional APUF, our TDC PUF demonstrated an approximately 12.5x increase in uniqueness for same-model FGPAs and over a 2x increase for cross-model FPGAs. Furthermore, our P&R optimization for the entropy circuit led to significant improvement in the randomness. The bit-wise entropy increased from 4.60 to 9.73 and response-wise entropy from 2.71 to 7.06, both reaching closer to the theoretical maximum of 11. Although

our TDC PUF had a lower normalized entropy than the APUF, it has a much larger and bit-correlated response space of 11 bits compared to the single-bit APUF response, making it more robust against modeling attacks. Furthermore, our optimization for temperature variation resulted in an approximately 4.5x increase in key regeneration success rate under a temperature variation of 2°C. This optimization also led to a significant 65% enrollment time reduction. Lastly, we demonstrated successful KMU integrations with the Chipyard framework's RISC-V SoC. We utilized both the RISC-V-specific RoCC and the ubiquitous MMIO approaches, and verified their functions with simulations.

While we have made significant progress in the high-quality design and implementation of the TDC-PUF-based KMU, there remain areas for future exploration and improvement. Besides using the TDC-PUF-specific helper data, the reliability can be further enhanced with ECC techniques. Additionally, more thorough studies need to be conducted on the resistance of the TDC PUF to modeling attacks. Actual modeling attacks with different machine learning models should be performed on the TDC PUF to explore deeper insights into its security strengths and vulnerabilities. Although the integration of the KMU with RISC-V SoC has already been verified through simulation, it should be further validated on real FPGA platforms to confirm its feasibility and effectiveness for real-world applications, including the incorporation in the scalable non-uniform memory access (NUMA) multiprocessors [54].

Bibliography

[1] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating critical security issues of the IoT world: Present and future challenges," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2483–2495, 2018.

[2] A. Jain, Z. Zhou, and U. Guin, "Survey of recent developments for hardware Trojan detection," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[3] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "KECCAK sponge function family main document," 2008, accessed: 05-05-2024. [Online]. Available: https://keccak.team/obsolete/Keccak-main-1.0.pdf

[4] AMD, "Vivado design suite 7 series FPGA and Zynq 7000 SoC libraries guide (UG953)," 2023, accessed: 05-05-2024. [Online]. Available: https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries/CARRY4

[5] J. Martin, "RISC-V, Rocket, and RoCC," 2017, accessed: 05-05-2024. [Online]. Available: https://inst.eecs.berkeley.edu/~cs250/sp17/disc/lab2-disc.pdf

[6] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak specifications summary," 2023, accessed: 05-05-2024. [Online]. Available: https://keccak.team/keccak_specs_summary.html

[7] F. Duarte, "Number of IoT devices (2024)," 2024, accessed: 05-05-2024. [Online]. Available: https://explodingtopics.com/blog/number-of-iot-devices

[8] Y. R. Siwakoti, M. Bhurtel, D. B. Rawat, A. Oest, and R. C. Johnson, "Advances in IoT security: Vulnerabilities, enabled criminal services, attacks, and countermeasures," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 224–11 239, 2023.

[9] M. N. I. Khan and S. Ghosh, "Information leakage attacks on emerging non-volatile memory and countermeasures," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18. New

York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3218603.3218649

[10] P. S. Ravikanth, "Physical one-way functions," Ph.D. dissertation, MIT, Mar 2001.

[11] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[12] D. Owen Jr., D. Heeger, C. Chan, W. Che, F. Saqib, M. Areno, and J. Plusquellic, "An autonomous, self-authenticating, and self-contained secure boot process for field-programmable gate arrays," *Cryptography*, vol. 2, no. 3, 2018.

[13] X. Wang, Y. Song, K. Prakash, Z. Zilic, and T. Langsetmo, "Quality-driven design methodology for PUFs in FPGAs for secure IoT," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, 2023, pp. 1–8.

[14] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 336–341.

[15] M. B. Yassein, S. Aljawarneh, E. Qawasmeh, W. Mardini, and Y. Khamayseh, "Comprehensive study of symmetric key and asymmetric key encryption algorithms," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–7.

[16] Q. Zhang, "An overview and analysis of hybrid encryption: The combination of symmetric encryption and asymmetric encryption," in *2021 2nd International Conference on Computing and Data Science (CDS)*, 2021, pp. 616–622.

[17] N. Huynh, H. Cherian, and E. C. Ahn, "Hardware security of emerging non-volatile memory devices under imaging attacks," in *2021 International Conference on Applied Electronics (AE)*, 2021, pp. 1–4.

[18] S. Ghosh, M. N. I. Khan, A. De, and J.-W. Jang, "Security and privacy threats to on-chip non-volatile memories and countermeasures," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–6.

[19] J. Miskelly, "Intrinsic PUFs for commodity devices," Ph.D. dissertation, Queen's University Belfast, 2022.

[20] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, "A survey on physical unclonable function (PUF)-based security solutions for Internet of things," *Computer Networks*, vol. 183, p. 107593, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620312275

[21] I. Papakonstantinou and N. Sklavos, *Physical Unclonable Functions (PUFs) Design Technologies: Advantages and Trade Offs*. Cham: Springer International Publishing, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-58424-9_24

[22] R. Maes, *Physically Unclonable Functions: Concept and Constructions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. [Online]. Available: https://doi.org/10.1007/978-3-642-41395-7_2

[23] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 237–249. [Online]. Available: https://doi.org/10.1145/1866307.1866335

[24] F. Zerrouki, S. Ouchani, and H. Bouarfa, "A survey on silicon PUFs," *Journal of Systems Architecture*, vol. 127, p. 102514, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122000832

[25] U. Rührmair, H. Busch, and S. Katzenbeisser, *Strong PUFs: Models, Constructions, and Security Proofs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 79–96. [Online]. Available: https://doi.org/10.1007/978-3-642-14452-3_4

[26] A. Maiti, V. Gunreddy, and P. Schaumont, *A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions*. New York, NY: Springer New York, 2013, pp. 245–267. [Online]. Available: https://doi.org/10.1007/978-1-4614-1362-2_11

[27] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing techniques for hardware security," in *2008 IEEE International Test Conference*, 2008, pp. 1–10.

[28] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann, "A formalization of the security features of physical functions," in *2011 IEEE Symposium on Security and Privacy*, 2011, pp. 397–412.

[29] E. I. Vatajelu, G. Di Natale, and P. Prinetto, "Towards a highly reliable SRAM-based PUFs," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 273–276.

[30] M. Al-Haidary and Q. Nasir, "Physically unclonable functions (PUFs): A systematic literature review," in *2019 Advances in Science and Engineering Technology International Conferences (ASET)*, 2019, pp. 1–6.

[31] S. Hemavathy and V. S. K. Bhaaskaran, "Arbiter PUF—a review of design, composition, and security aspects," *IEEE Access*, vol. 11, pp. 33 979–34 004, 2023.

[32] K. Katoh, S. Yamamoto, Z. Zhao, Y. Zhao, S. Katayama, A. Kuwana, T. Nakatani, K. Hatayama, H. Kobayashi, K. Sato, T. Ishida, T. Okamoto, and T. Ichikawa, "A physically unclonable function using time-to-digital converter with linearity self-calibration and its FPGA implementation," in *2023 IEEE International Test Conference in Asia (ITC-Asia)*, 2023, pp. 1–6.

[33] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM Journal on Computing*, vol. 38, no. 1, p. 97–139, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1137/060651380

[34] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[35] M. Hiller, M. Weiner, L. Rodrigues Lima, M. Birkner, and G. Sigl, "Breaking through fixed PUF block limitations with differential sequence coding and convolutional codes," in *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, ser. TrustED '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 43–54. [Online]. Available: https://doi.org/10.1145/2517300.2517304

[36] M. Hiller and G. Sigl, "Increasing the efficiency of syndrome coding for PUFs with helper data compression," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2014, pp. 1–6.

[37] E. I. Vatajelu, G. Di Natale, and P. Prinetto, "Towards a highly reliable SRAM-based PUFs," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 273–276.

[38] Z. He, W. Chen, L. Zhang, G. Chi, Q. Gao, and L. Harn, "A highly reliable arbiter PUF with improved uniqueness in FPGA implementation using bit-self-test," *IEEE Access*, vol. 8, pp. 181 751–181 762, 2020.

[39] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[40] N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, "Neural network modeling attacks on arbiter-PUF-based designs," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2719–2731, 2022.

[41] F. Dan, Y. Xu, Z. Li, J. Wen, B. Liu, S. Chen, and B. Li, "A modeling attack resistant R-XOR APUF based on FPGA," in *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, 2018, pp. 577–581.

[42] Z. Chang, S. Shi, B. Song, W. Fan, and Y. Wang, "Modeling attack resistant arbiter PUF with time-variant obfuscation scheme," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 60–63.

[43] C. Xu, J. Zhang, M.-K. Law, Y. Jiang, X. Zhao, P.-I. Mak, and R. P. Martins, "Modeling attack resistant strong PUF exploiting obfuscated interconnections with ¡0.83

[44] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[45] S. Trimberger, "Trusted design in FPGAs," in *2007 44th ACM/IEEE Design Automation Conference*, 2007, pp. 5–8.

[46] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten years of hardware Trojans: a survey from the attacker's perspective," *IET Computers & Digital Techniques*, vol. 14, no. 6, pp. 231–246, 2020. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cdt.2020.0041

[47] V. Jyothi and J. J. Rajendran, *Hardware Trojan Attacks in FPGA and Protection Approaches*. Cham: Springer International Publishing, 2018, pp. 345–368. [Online]. Available: https://doi.org/10.1007/978-3-319-68511-3_14

[48] M. Elnawawy, A. Farhan, A. A. Nabulsi, A. Al-Ali, and A. Sagahyroon, "Role of FPGA in Internet of things applications," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2019, pp. 1–6.

[49] A. S. Waterman, "Design of the RISC-V instruction set architecture," Ph.D. dissertation, University of California, Berkley, 2016.

[50] A. S. Waterman and K. Asanovi, "The RISC-V instruction set manual - volume I: User-level ISA," 2017, accessed: 05-05-2024. [Online]. Available: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[51] E. Kavun and T. Yalcin, "A lightweight implementation of Keccak hash function for radio-frequency identification applications," vol. 6370, 06 2010, pp. 258–269.

[52] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[53] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[54] A. Grbic, S. Brown, S. Caranci, R. Grindley, M. Gusat, G. Lemieux, K. Loveless, N. Manjikian, S. Srbljic, M. Stumm, Z. Vranesic, and Z. Zilic, "Design and implementation of the NUMAchine multiprocessor," in *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 1998, pp. 66–69.