UNIVERSITY OF CALIFORNIA

Los Angeles

Enhance Energy-efficiency and Security of IoT using Hardware-oriented Approaches

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Teng Xu

2017

ABSTRACT OF THE DISSERTATION

Enhance Energy-efficiency and Security of IoT using Hardware-oriented Approaches

by

Teng Xu
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2017
Professor Miodrag Potkonjak, Chair

The rapid growth of Internet of things (IoT) including mobile phones, portable devices, and remote sensor network systems have imposed both conceptually and technically new challenges. Among them, the most demanding requirements for the widespread realization of many IoT visions are security and low power. In terms of security, IoT applications include tasks that are rarely addressed before such as trusted sensing, secure computation and communication, privacy, and data right management. These tasks ask for new and better techniques for the protection of hardware, software, and data.

On the other hand, most of the IoT systems suffer from the problem of limited power sources, which in turn require the security on IoT devices to be lightweight. While low energy design is crucial for the successful deployment of resource-constrained IoT devices, their often physically accessible nature, as well as low energy budget restriction, have also contributed to rendering traditional cryptographic approaches insufficient to address all the security concerns.

In this dissertation, we present hardware-oriented security designs and synthesis techniques with the aim to reduce the system energy overhead while maintaining the security and reliability. We first demonstrate our work to analyze and enhance the properties of hardware security primitives. We emphasize on physical unclonable functions (PUFs) and use them to enable a wide range of applications, including private/public key communication, authentication, and multi-party communication. We then present a unique system reliability design with the use of non-volatile memory (NVM) to create a fault-tolerance application specific architecture with almost no timing overhead and low energy overhead. Finally, we demonstrate novel energy reduction and synthesis techniques applied on integrated circuit subsystems of IoT applications. The techniques we have applied and improved include near-threshold computing, dual-supply voltage optimization, and pipelining.

The dissertation of Teng Xu is approved.

Gregory J. Pottie

Miloš D. Ercegovac

Mario Gerla

Miodrag Potkonjak, Committee Chair

University of California, Los Angeles

2017

To my family.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I have always thought acknowledgment is the most difficult section to write in the whole thesis. When I started typing the following paragraphs, I realized there is a reason for this. The fact is that I didn't work all by myself to accomplish all the work in this thesis, instead, it is the combination of wisdom and efforts from many individuals. I'd like to try my best to list all of you here and provide my most sincere gratefulness.

First and foremost, I would like to thank my adviser and doctoral committee chair, Professor Miodrag Potkonjak, for offering me a great chance to study at University of California, Los Angeles. He guided me through the world of academic research, witnessed and helped me step by step to get ready for the future research career. He shared his insightful research ideas with me selflessly and was patient to carefully listen to my proposals and discuss them with me. I would always remember he went through my papers word by word to correct my grammar mistakes and gave comments to every figure and table. He would always remind me to be self-discipline and steer me back to the right track when I became complacent. His great working attitude and critical thinking will impact me through my future career.

To my doctoral committee members, Professors Miloš D. Ercegovac, Mario Gerla, and Gregory J. Pottie, thank you for your visionary comments and advice on my prospectus, dissertation, and final defense, and for aiding me in the completion of my graduate studies.

Thank you all of my collaborators and colleagues: Sheng Wei, Nathaniel Conos, Jason Zheng, Saro Meguerdichian, James B. Wendt, Jia Guo, Hongxiang Gu, for sharing and discussing research ideas with me, for staying up with me for nights to fight for the papers and projects. I would like to offer my special thanks to Jason Zheng (chapter 1), Hongxiang Gu (chapter 2), James B. Wendt (chapter 4 and 6), and Nathaniel Conos (chapter 9) who directly helped me with finishing this thesis. I would like to extend my special thanks to James B. Wendt with whom I worked closely with to co-author a number of my publications. Thank you for sitting together with me to correct (rewrite) the English of my whole paper meticulously, and showing me an example of being a good researcher. And of course, the most sincere thank you goes to Professor Miodrag Potkonjak who supervised and co-authored all of the research work that constituted the chapters in this thesis.

Last but not least, I would like to thank my family and friends who have supported me throughout my doctor study. A special thank you to my parents, although they are physically half an earth alway from me, they always provide their support and unconditional love to me whenever I need. And to Song, for her company and encouragement to keep me through the doctor study. Thank you.

Vita

2008-2012	B.A. (Computer Science) University of Science and Technology of China
2012-2017	Ph.D. candidate (Computer Science) University of California, Los Angeles

Selected Publications

T. Xu, J. B. Wendt, and M. Potkonjak, “Digital Bimodal Function: An Ultra-Low Energy Security Primitive,” *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 292-297, 2013.

T. Xu, M. Potkonjak, “Robust and Flexible FPGA-based Digital PUF,” *International Conference on Field Programmable Logic and Applications (FPL)*, 2014.

T. Xu, J. B. Wendt and M. Potkonjak, “Secure Remote Sensing and Communication using Digital PUFs,” *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2014.

T. Xu, J. B. Wendt, and M. Potkonjak, “Security of IoT Systems: Design Challenges and Opportunities,” *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 417-423, 2014.

T. Xu, M. Potkonjak, “Digital PUF using Intentional Faults,” *International Symposium on Quality Electronic Design (ISQED)*, pp. 448-451, 2015.

T. Xu, D. Li, and M. Potkonjak, “Adaptive Characterization and Emulation of Delay-based Physical Unclonable Functions Using Statistical Models,” *Proceedings of the 52nd Annual*

Design Automation Conference (DAC), pp. 76-81, 2015.

T. Xu, M. Potkonjak, "The Digital Bidirectional Function as a Hardware Security Primitive: Architecture and Applications," *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 335-340, 2015.

T. Xu, H. Gu, M. Potkonjak, "Data Protection Using Recursive Inverse Function," *International Conference on Field Programmable Logic and Applications (FPL)*, 2015.

T. Xu, M. Potkonjak, "Digital bimodal functions and digital physical unclonable functions: architecture and applications," *Secure System Design and Trustable Computing*, Springer International Publishing, pp. 83-113, 2016.

T. Xu, M. Potkonjak, "Retiming and Dual-supply Voltage Based Energy Optimization for DSP Applications," *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1055-1059, 2016.

T. Xu, H. Gu, M. Potkonjak, "An Ultra-Low Energy PUF Matching Security Platform Using Programmable Delay Lines," *International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2016.

H. Gu, T. Xu, M. Potkonjak, "An Energy-Efficient PUF Design: Computing While Racing," *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 142-147, 2016.

T. Xu, M. Potkonjak, "Energy-efficient Fault Tolerance Approach for Internet of Things Applications," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 62-69, 2016.

T. Xu, M. Potkonjak, "Pipelining For Dual Supply Voltages," *International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, 2016.

CHAPTER 1

Introduction

1.1 The Internet of Things

According to the survey from L. Atzori [1], the basic idea of Internet of things (IoT) is the pervasive presence around us of a variety of things or objects such as Radio-Frequency identification (RFID) tags, sensors, actuators, mobile phones, etc. which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals.

Being first invented in 1985, the concept of IoT has greatly evolved over the years. Despite different visions and interpretations, IoT has revolutionized and will continue to revolutionize the ways in which individuals and organizations interact with the physical world as well among themselves. For example, the interactions with home devices, cars, customer items, industrial plants, and weaponry will be fundamentally altered. Many services, such as health, learning, and resource management, will be provided in new ways that are novel, better organized, and user-customized.

It is estimated that there are 6.4 billion connected IoT devices in use worldwide in 2016, and will reach 20.8 billion by 2020 [2]. The rapid growth of IoT devices not only provides a large potential market, but also draws many research attentions to resolve emerging problems due to the unique properties of IoT.

A few key characteristics of IoT include the following.

- connectivity

- security
- constrained energy
- enormous scale

One of the pivotal property of IoT is connectivity. An IoT device is always connected to the IoT network to communicate with the other IoT devices. It enables network accessibility and compatibility in the IoT. Such accessibility provides great convenience to IoT users, but also makes it vulnerable to the access of malicious parties.

IoT security is another key issue for IoT systems. As IoT provides intelligent experience, high efficiency, and other benefits to users, a major concern is regarding how to efficiently protect information that is stored and transmitted with IoT devices. It is necessary to secure not only the devices, but also the networks, and the data that is transferred across all the IoT.

IoT is also highly constrained by energy. This is mainly due to the fact that many IoT devices have limited size, weight, and require excellent mobility so that such devices can only afford highly limited battery. Ironically, to stay connected and secure, it normally takes a considerable amount of energy. Consequently, many research efforts focus on reducing the energy overhead of IoT devices.

Lastly, IoT employs an enormous scale, and the size is still growing dramatically. The management of such a huge network of devices, as well as the corresponding data generated, becomes critical. While IoT can borrow designs from protocols and technologies that have been well established in other domains, there are some fundamental differences between IoT systems and traditional computing systems, such as the differences in session lengths, processing power, storage capacity, and transfer methods [3].

1.2 Motivation and Challenges

The practical realization of IoT requires the development of a number of new versions of platforms and technologies including device identification and tracking, sensing and actuation, communication, semantic knowledge processing, coordinated and distributed control, and behavioral modeling [4]. The realization of IoT subsystems will be subjected to numerous constraints such as cost, energy, scalability, connectivity, security and so on. Among all the above issues, in this thesis, we focus on the two key aspects: security and energy efficiency.

It is widely acknowledged that the potential for malicious attacks can and will be greatly spread and actuated from the Internet to the physical world. Hence, the security of the physical IoT devices is of essential importance. One should also consider a great diversity of IoT systems from fully organized to small individual nodes [5][6]. For example, things such as cars, airplanes, and industrial equipment allow for much more expensive instrumentation with much high power and energy budgets in comparison to household and mobile IoT devices, such as sensors, remote controllers, personal phones and intelligent watches. Therefore, although for full impact, generic algorithms and protocols are required, different customized solutions are also mandatory. This is in particularly true for security solutions.

IoT security encompasses several layers of abstraction and a number of dimensions. The abstraction levels range from physical layers of sensors, computation and communication, and devices to the semantic layer in which all collected information is interpreted and processed. We expect that a majority of security attacks will occur at the software level because it is currently most popular and can simultaneously cover a large number of devices and processes. From a research point of view, most novel attacks are on physical signals and, in particular, semantic attacks during data processing and decision making steps. It is important to observe that the lowest level of protection at any level and at any dimension determines the overall security.

A significant percentage of IoT devices are operated in passive mode without batteries. Their energy will either be harvested or received using a wireless medium. Even equipped with batteries, a considerable portion of IoT devices require mobility so that only very limited size of the battery is allowed. For example, mobile phones and tablets, in which case the slow growing of battery capacity has become a major obstacle for the development of these devices. Another type of IoT system, the wireless sensor network only employs a lightweight power source, but normally requires relatively long lifespan since it is not only expensive but also inconvenient to keep changing the batteries. All the above mentioned systems often allow for only very minimal hardware, and thus, require an ultra compact security solution with an ultra-small footprint and energy budget.

We target to find techniques to increase the energy efficiency of IoT devices and especially focus on finding the lightweight security solutions. While the problem itself can be addressed from different aspects, such as network, hardware, software etc, we emphasize on the hardware level. Our main claim in this thesis is that hardware-oriented security is ideally suited to answer IoT security requirements. First of all, hardware-based security provides a natural starting point for the realization of IoT protocols and procedures due to its flexibility and potential to be designed and implemented with low area and energy requirements. It is also naturally more resistant against side-channel and physical attacks. Finally, it provides elegant and efficient solutions to several problems that classical cryptography has not been able to solve, such as to prevent cloning and physical access to devices.

However, in order to realize the full potential of hardware-oriented security, significantly additional research and engineering issues have to be addressed in novel and creative ways. Many challenges exist when actually incorporating hardware based solutions to the IoT designs. For example, a typical type of hardware security primitive is the physical unclonable function (PUF). It is a physical structure with input-output mapping that is easy to evaluate but hard to predict or clone due to the effect of process variation in the manufacturing process. A PUF naturally employs the properties of low-energy, small-area, and most importantly unclonability. A few IoT applications have started applying PUFs to authenticate devices and prevent hardware counterfeit [7][8]. However, its shortcomings are significant as well, among which low stability, insufficient confusion and diffusion are the most representative. Besides the defects from the hardware security solutions themselves, another obstacle is to actually incorporate the new hardware solutions to the existing hardware systems. Still take PUFs as an example, they take advantage of the process variation to create unclonability, in other words, the PUFs are analog in nature, which prevents them from being easily embedded to the current digital systems.

In this thesis, we focus on hardware oriented approaches to address the security and energy efficiency issues of IoTs systems. For security, we have improved, designed and implemented a set of hardware security solutions. We especially tackle their existing weakness as well as increase their compatibility with IoT systems. In terms of energy efficiency, on one hand, we always consider it as an important criterion when designing security primitives, on the other hand, we have also proposed novel synthesis algorithms to decrease circuit power of IoT devices.

1.3 Contributions and Organization

The dissertation is organized into three major categories: hardware security, hardware reliability, and energy optimization as shown in Table 1.1. The three categories target on different aspects of IoT systems and try to solve the issues not only from the perspective of hardware design, but also to address the emerging applications in the domain of IoT. Note that for hardware security and reliability, low energy overhead has also been a key design criterion for all the proposed approaches.

The first key category of the thesis is hardware security, in which we further split it into two subcategories: analyze&improve analog PUFs and design novel digital security primitives. As a typical type of hardware security primitive, a PUF is the abbreviation of a physical unclonable function, it is a physical entity with unpredictable input-output mapping and the entity itself is unclonable because of process variation. For example, the arbiter PUF is a typical type of PUF and is well analyzed in a lot of research. It takes advantage of the process variation on delays to create unique paths with unpredictable delays to compete and

Hardware Security	
Analyze&improve analog PUFs	**Novel digital security primitives**
• Characterize and emulate traditional arbiter PUFs (ch. 2) • Enable PUF matching using programmable delay lines (ch. 3)	• Ultra-low energy public key communication using digital bimodal function (ch. 4) • Ultra-low energy private key communication using digital bidirectional function (ch. 5) • Digital PUFs initialized with analog PUFs (ch. 6) • Digital PUFs with laser-based fault injection (ch. 7)
Hardware Reliability	
• An energy-efficient fault tolerance approach for IoT (ch. 8)	
Hardware Energy Optimization	
• Combine pipelining with dual-supply voltages (ch. 9)	

Table 1.1: Major contributions and organization of the dissertation.

generate output signals. Note that the majority of standard PUFs are analog PUFs since their unclonability depends on analog properties of circuits such as delay and frequency. Our work investigates the properties of the existing standard analog PUFs (mostly on arbiter PUFs), with emphasize on improving their stability and unpredictability. Two major efforts are taken in this part. In chapter 2, we first use statistical models to characterize arbiter PUFs. Then based on the retrieved delay information, we emulate an arbiter PUF using another random piece of arbiter PUF. The work proves that traditional arbiter PUFs are highly vulnerable to statistical attacks, and can be cloned with high accuracy. Motivated by the above work, in chapter 3, we further propose to match two arbiter PUFs and increase their matching accuracy with programmable delay lines (PDL). The matched PUFs can be directly applied as security keys in private key communication.

While analog PUFs have intrinsic disadvantages such as inconsistent against different temperatures and voltages that can not be completely eliminated, we take the next step to develop novel digital security primitives as a replacement for the traditional analog PUFs. The fact that the primitive is digital makes it resilient against environmental variations. More importantly, it can be naturally embedded with digital circuits which are used by the majority of IoT devices. To achieve the goal, we have proposed four types of digital security primitives. In chapter 4, we first discuss digital bimodal function (DBF) and the corresponding public key communication protocol. As the name suggested, the DBF has two forms with the same input-output mapping function, among which one form is fast, compact and the other form is slow and computationally expensive. We take advantage of the above properties and use the fast form as a private key, the slow form as a public key to design public key communication protocol. In chapter 5, we propose another type of security primitive named digital bidirectional function (DBirF). It is designed and implemented on a field-programmable gate array (FPGA), and is dedicated to be used as private keys in security key communication. The previous two proposals employ the digital property of circuits, but have totally abandoned the unclonability. Our next effort has been made to create a security primitive that is both digital and unclonable (a digital PUF). In chapter 6, we demonstrate an extension of the fast format of DBF and modify it to acquire unclonability. The key idea is to use the output from analog PUFs as a random source to configure the digital circuit. In chapter 7, we create another type of digital PUF with a totally different technique. The essential operation is to use lasers to cut the wires in the circuit layouts, thus to intentionally introduce faults in circuits. From the perspective of an attacker, the faults are not observable and are extremely difficult to detect, so that the faulty circuit is unclonable.

While reliability is essential in many IoT applications, in particular in the domains such as medical devices and automotive systems where a single fault in the system can lead to serious consequences, it is rarely addressed because the high energy and area cost are oftentimes not affordable by IoT devices. In chapter 8, we present our techniques to improve hardware reliability and the approach is designed to be energy efficient. Specifically, we combine the use of non-volatile memory (NVM) and classical CMOS transistors so that NVM judiciously stores selected complete states of the pertinent program. It allows the program to resume from the saved state in NVM when faults occur.

The last category of our thesis organization is the energy optimization techniques for IoT devices. We have analyzed, redesigned, and combined a set of traditional circuit optimization algorithms, and applied them to circuits of IoT applications. In chapter 9, we propose algorithms to combine dual-supply voltage optimization with pipelining. Although each individual techniques are proposed before, they have their own constraints that prevent them from being optimally combined together to jointly lower circuit energy. The objective goal of our work is to propose an approach to combine the above technologies to enable an even more effective energy optimization scheme while trying to leverage as many constraints as possible.

CHAPTER 2

Characterize and Emulate Traditional Arbiter PUFs

2.1 Motivation and Problem Formulation

Process variation is an important side effect in circuit manufacturing. Due to the effect of process variation, the physical attributes of transistors (channel length, delay, leakage) become unique when integrated circuits are fabricated. Such attributes can not be duplicated since they depend on the physical properties of each transistor and the randomness in the manufacturing process. On the other hand, PUFs take advantage of the process variation to form uniquely unclonable devices. PUFs are physical devices that have a random but deterministic mapping of inputs to outputs. The input to a PUF is defined as a challenge and the output is defined as a response. When given a challenge to a PUF, it produces a response based on both the given challenge and the intrinsic physical properties of the PUF itself.

As a type of security primitive, some properties of PUFs are ideal for securing IoT systems, such as low-power, fast-speed, small-area, unpredictability, and most importantly, unclonability. For example, the unclonability of PUFs can directly prevent IoT devices from being copied by malicious parties. Among the family of PUFs, the arbiter PUFs are widely studied and analyzed. They utilize the process variation on delays to create two theoretically identical paths to rival to each other. A challenge vector is used to modify and decide the two signal paths. To generate a response, an impulse signal is sent to both paths simultaneously, and an arbiter is at the end of the two paths taking them as two inputs. Depending on the delay of the two paths, the impulse signal from one path triggers the arbiter first to generate a response. Our work in this chapter will focus on arbiter PUFs.

PUFs enable a variety of security protocols, such as message encryption and decryption, authentication, and public key communication. They are all widely used in IoT systems. Unfortunately, a PUF has its own shortcomings when applied in security protocols. Such protocols usually require at least two parties to share the same challenge-response mapping function. However, a PUF can not be copied due to its unclonability, consequently, one of the parties can only simulate the PUF mapping function which is significantly slower and takes more energy when comparing to directly using the hardware of PUF.

In this chapter, we leverage the above issue by matching multiple PUFs in such a way all the PUFs own the same or at least similar challenge-response mapping functions. Thus, each party can hold a physical piece of PUF and apply it in the security protocols. We achieve our goal by using adaptive PUF characterization and emulation. Although it has been a common wisdom that a PUF is a system that is hard to be predicted, there have been many emulation attacks proposed to characterize the arbiter PUFs and to emulate the PUFs in software. Most of them collect and observe many challenge-response pairs (CRPs) and apply machine learning techniques to build a statistical model from it. Based on the statistical model, the PUF response can be predicted given any random challenge.

We have two major contributions. The first is that we propose an adaptive PUF characterization technique using a statistical model. Instead of purely statistically predicting the PUF response, our approach looks into the PUF structure and builds a delay model to estimate the delays in all the segments of the PUF. Such delay information is used as the premise for our PUF emulation. The second aspect is that instead of only using software to emulate the PUF, we use hardware based emulation. To be more specific, we utilize another piece of PUF to emulate the original PUF and the new PUF has a high probability to generate the same response as the original PUF when given the same challenge. In another word, the two PUFs are matched to each other.

Our workflow is shown in Figure 2.1. We first propose our statistical model for the adaptive PUF characterization. It acts as the premise of the later hardware emulation. We combine the traditional statistical approach with our adaptive test to improve the characterization accuracy. Then based on the characterization results, we propose our algorithm to emulate a PUF using another PUF so that the two PUFs are matched to each other. Using the matched PUFs, a set of security protocols for IoT communications will also be discussed. Finally, we implement and evaluate our approach on a Xilinx Spartan-6 field-programmable gate array (FPGA).

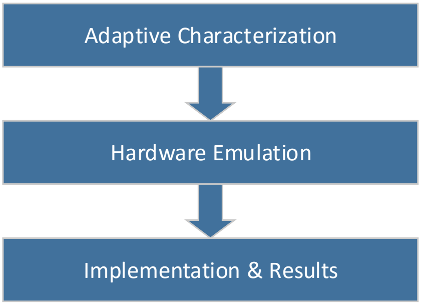


Figure 2.1: Work flow illustration.

2.2 Preliminaries

2.2.1 Process Variation

Process variation is a widely recognized phenomenon in modern CMOS technologies [9]. When the transistors are designed to be identical, due to manufacturing limitations, the real produced components are different and unique in terms of structural and operational properties, such as propagation delay and leakage power. Many factors can cause process variation, including wafer lattice structure imperfections, non-uniform dopant distribution, mask alignment, and chemical polishing.

2.2.2 PUFs

The concept of PUF is first proposed by Pappu et al. using mesoscopic optical systems [10]. Devadas et al. developed the first silicon PUFs through the use of intrinsic process variation in deep submicron integrated circuits [11]. A variety of PUFs are proposed after that, including arbiter PUFs [11], ring oscillator PUFs [12], and SRAM PUFs [13]. The arbiter PUF on FPGA is proposed in [14], with the core idea of taking advantage of the intrinsic delay difference between look-up tables (LUTs) to design the delay paths.

2.2.3 PUF based Protocols

Numerous traditional protocols can be interpreted using PUFs, ranging from the traditional security key communication and authentication [15] to more sophisticated public key communication [16]. The key idea is to employ the highly unpredictable PUF responses to secure the information. However, conventionally only one party has the actual hardware of PUF (e.g., decryption party). Thus, all the rest communication parties can only simulate the PUF (e.g., encryption party), which takes high timing/energy overhead compared to directly using the PUF hardware. Our new proposed PUF matching platform targets to leverage the above drawback.

2.2.4 PUF Emulation

Many technologies have been proposed to emulate PUFs. Lee first proposed to use a statistical model to extract unique secret key information from PUFs [17]. Rührmair et al. developed numerical modeling attacks combining with machine learning techniques to break various types of PUFs [18]. The core idea of previous PUF emulation is to collect a number of CRPs of the PUF and derive a statistical model from there. But all of the previously proposed techniques employ software-based emulation. Our work focuses on hardware based PUF emulation instead.

2.2.5 PUF Model

The arbiter PUF model we are using is shown in Figure 2.2. The basic structure of the n-bit PUF consists n delay segments. Two identically designed paths are generated by connecting delay components from each segment, and an arbiter is at the end of the two paths.

The two paths of the arbiter PUF can be modified using the control bit of each segment (c_i). When the control bit is 0, the two paths will not shuffle. When the control bit is 1, the two paths swap. In Figure 2.2, when $c_1 = 0$, d_1^0 connects to d_2^0 and d_1^1 connects to d_2^1. Meanwhile, when $c_1 = 1$, d_1^0 connects to d_2^1 and d_1^1 connects to d_2^0. If we denote the propagation delays of the ith segment as d_i^0 (upper delay) and d_i^1 (lower delay), then the two delays should be designed nominally equal to each other. After manufacturing, the effect of process variation will cause unpredictable delay difference between them.

When an n bit challenge ($c_1c_2...c_{n-1}c_n$) is given to a PUF, two identically designed paths are generated. To retrieve a response, an impulse signal is fed into the system to excite both paths simultaneously. Because of the delay difference caused by process variation, one path will reach the arbiter earlier, and an output bit is generated as the PUF response.

Our PUF characterization and emulation are discussed on the standard arbiter PUFs (Figure 2.2), they are not applicable to feed-forward PUFs or feed-back PUFs.

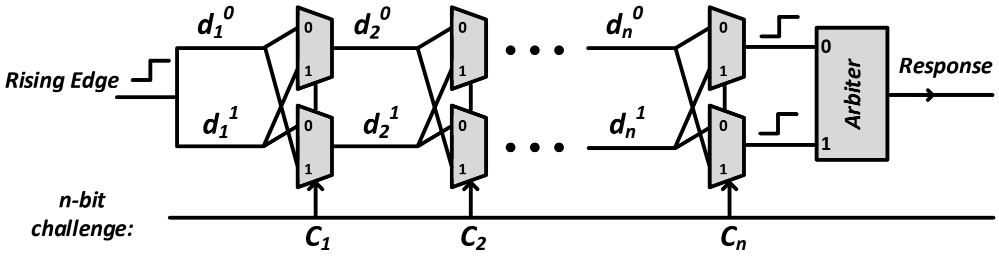


Figure 2.2: The model of an arbiter PUF with an n-bit challenge.

2.3 PUF Characterization

2.3.1 Objective and Challenges

The primary goal of PUF characterization is to observe the characteristics of a PUF, hence to build a PUF model to correctly predict the response given a random PUF challenge. When considering the arbiter PUF, the most important characteristics are the delay information of each PUF segment, given which the output of the PUF can be easily predicted. For example, based on our PUF model in Figure 2.2, the delay information includes d_i^0 and d_i^1 for each segment.

A widely used approach to characterize a PUF is to collect a certain number of CRPs of the PUF. Using the collected CRPs as a training set, machine learning approaches are adopted to observe the correlations between the PUF challenges and responses. Then such correlation is applied to the test set to make further predictions of PUF outputs. However, current approaches can not extract the exact delay information of each PUF segment which is important for the PUF emulation. Therefore, our goal in PUF characterization is not only to make an accurate response prediction, but also to extract the PUF delay information.

Another important challenge for PUF characterization is about how to build an accurate PUF model with a limited number of CRPs. To solve this, we specifically divide our PUF characterization into two stages. The first stage follows the standard PUF model building strategy which takes advantage of some number of CRPs to build a statistical model. As mentioned above, compared to standard PUF models, our model is specifically designed in such a way to reveal the delay information of the PUF. At the second stage, we propose two novel approaches respectively to use "adaptive challenges", and "compensate challenges". Both approaches only take advantage of a small number of selected CRPs with certain specific properties. They are applied on the top of the PUF model achieved in stage one to further tune the model to a better precision.

2.3.2 Statistical Model

Assume that it is an n-bit PUF that we are characterizing. For the *ith* segment, there exists 2 delays: d_i^0 and d_i^1 (i∈{1,2,...n}). After collecting some number of CRPs, we statistically calculate the following 2 probabilities for each segment as shown in Equation 2.1.

$$\begin{aligned} &P_i^0(path\ p_0\ is\ longer \mid p_0\ contains\ d_i^0) \\ &P_i^1(path\ p_1\ is\ longer \mid p_1\ contains\ d_i^1) \end{aligned} \tag{2.1}$$

We claim that when a delay in a segment is longer than its rival delay, e.g., d_i^0 and d_i^1 are rival delays, the path that contains the longer delay will have a larger probability to be longer than the opposite path. The intuition is that each path is a sum of single delays from each segment. Consequently, if e.g., d_i^1 in segment i is longer than d_i^0, because the delays in the rest segments are randomly assigned to the two paths, the path that contains d_i^1 will have a better chance to be longer than the opposite path which contains d_i^0. Therefore, there exists a correlation between the segment delays and the probabilities listed in Equation 2.1.

According to Figure 2.2, assume that if the path (p_0) that reaches the upper port of the arbiter is earlier, the output equals to 0. Otherwise, the arbiter output equals to 1. The delay of the two paths can be written in the format as shown in Equation 2.2. The definition of $parity(i)$ can be found in Equation 2.3. It represents the parity of the times of switching after segment i for a given challenge. Because the output is decided by the relation between the total delays in p_0 and p_1, we represent the PUF output by using only the difference between the rival delays in each segment as shown in Equation 2.4.

$$\begin{gathered}
p_0 = \sum_{i=1}^{n} t_i^0 \\
p_1 = \sum_{i=1}^{n} t_i^1 \\
t_i^0 = d_i^0,\ t_i^1 = d_i^1,\ when\ c_i = 0\ and\ parity(i) = 0; \\
t_i^0 = d_i^1,\ t_i^1 = d_i^0,\ when\ c_i = 1\ and\ parity(i) = 0; \\
t_i^0 = d_i^1,\ t_i^1 = d_i^0,\ when\ c_i = 0\ and\ parity(i) = 1; \\
t_i^0 = d_i^0,\ t_i^1 = d_i^1,\ when\ c_i = 1\ and\ parity(i) = 1;
\end{gathered} \tag{2.2}$$

$$parity(i) = \begin{cases} 0,\ even\ 1s\ in\ \{c_{i+1}...c_n\},\ i \in \{1, 2, ...n-1\} \\ 1,\ odd\ 1s\ in\ \{c_{i+1}...c_n\},\ i \in \{1, 2, ...n-1\} \\ 0,\ i = n \end{cases} \tag{2.3}$$

$$Output = \begin{cases} 0, \ \sum_{i=1}^{n} d_i^{diff} < 0, \\ 1, \ \sum_{i=1}^{n} d_i^{diff} > 0, \ d_i^{diff} = t_i^0 - t_i^1 \end{cases} \tag{2.4}$$

After collecting a number of CRPs, the probabilities in Equation 2.1 can be summarized statistically. Since Our goal is to characterize the PUF, therefore, the major question now is how to derive the real delays from the probabilities. To be more specific, since it is the delay difference that decides the PUF output, our target becomes to derive the d_i^{diff} of each segment in Equation 2.4 from the probabilities in Equation 2.1. To achieve this, we need to find a suitable regression model.

We simulate a 64-bit PUF after collecting 100,000 CRPs. We calculate the probabilities in Equation 2.1 for the delays in all the segments. By repeating the test on 1,000 simulated PUFs, we plot all the calculated probabilities in the x-axis and the corresponding normalized delay difference in the y-axis. Note that because of $P_i^0 + P_i^1 = 1$ (i∈{1,2,...n}), in order to avoid repeating, we only plot the probabilities which are larger than 0.5. Due to the same reason, we only plot the absolute value of d_i^{diff}. The regression plot in Figure 2.3 indicates a perfect linear mapping between the delay difference and the probability. The results strongly suggest that a statistical delay model can be derived from the collected probabilities using linear regression.

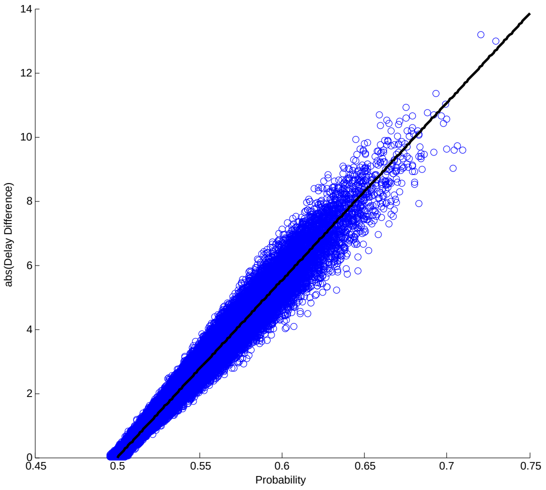


Figure 2.3: Regression plot between the delay difference and the probability.

The derived delay model can be used to predict PUF responses. When given a random challenge, the delay of the two paths can be calculated and compared using the statistically estimated d_i^{diff} of each segment.

2.3.3 Adaptive Challenges

On the base of the statistical model, we propose two approaches to improve the model accuracy. The first approach is to use adaptive challenges.

Figure 2.4 shows a motivational example. We first consider the challenge 0010, where the two paths are denoted in the red line and the blue line. Meanwhile, if we consider challenge 0100, the two paths almost stay the same except that d_3^0 and d_3^1 switch positions. However, the response changes from 0 to 1, which means that the red path is longer in the first case and the blue path is longer in the second case. It can only be caused by the switching between d_3^0 and d_3^1. Consequently, we can conclude that $d_3^0 > d_3^1$. To extend the example, we define challenges such as 0010 and 0100 as a pair of adaptive challenges, because they can be adaptively adjusted to test the relation between the upper and lower delay of a single segment.

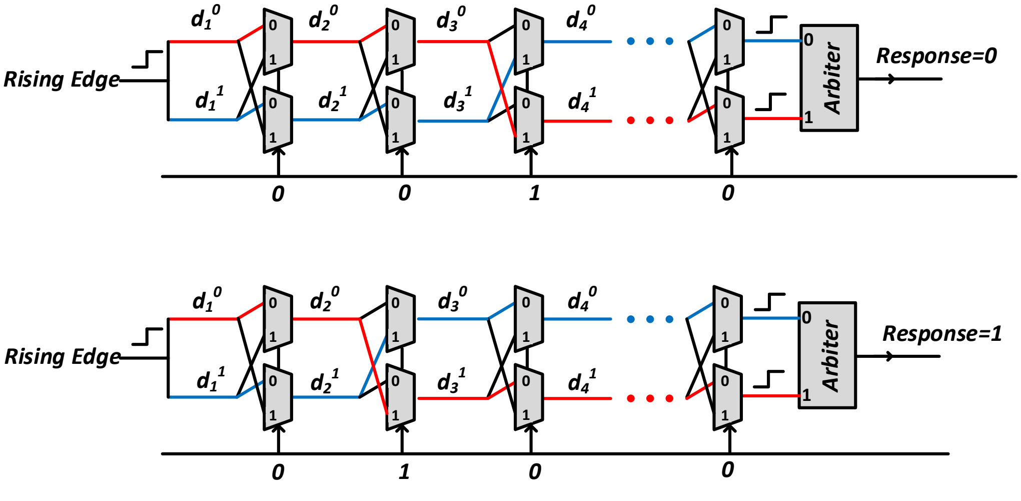


Figure 2.4: An example of adaptive challenges.

Adaptive challenges require that the two paths have a very small delay difference given the challenges so that the switching of only a single segment can change the PUF response. In order to find such challenges, we need to use the statistical model achieved from the previous step. Given the delay of each segment from the model, to find the challenge with a smallest total delay difference to cover all the segments, the time complexity is exponential. As an alternative, we use the greedy algorithm to find adaptive challenges. The basic idea is to first sort the delay difference of each segment. Then starting from the segment with the largest delay difference to the segment with the smallest delay difference, we put the challenge bits in such a way that the delay difference of current segment compensates the already considered segments' delay difference to make the sum of d_i^{diff} stays as close to 0 as possible. Note that adaptive challenges deterministically decide the sign of d_i^{diff} in segment i rather than to probabilistically estimate it from a statistical model.

2.3.4 Compensate Challenges

A drawback in the adaptive challenge is that it only compares the d_i^{diff} with 0. For instance, if the real delay difference is 2ps, and in the statistical model the delay difference is calculated as 10ps. Despite the error is large, the use of adaptive challenge can not test it out. As long as the real delay difference has the same sign as the predicted one, the response stays the same. Due to this reason, adaptive challenges miss many cases to reveal the errors in the statistical model.

In order to better quantify the delay difference, we propose our second method: compensate challenges. The basic idea is similar to the adaptive challenges, we still start from a motivational example. We consider the challenge $0c_2c_3...c_n$. The delay difference of the two paths can be represented as $(d_i^0 + t_{rest}) - (d_i^1 + t'_{rest})$. In the case of compensate challenges, instead of designing $c_2c_3...c_n$ to make $t_{rest} \approx t'_{rest}$, we make $t_{rest} - t'_{rest} \approx \beta$. Therefore, the delay difference can be simplified as $d_i^0 - d_i^1 + \beta$. Depending on the response of the PUF, the relation between d_i^0 and $d_i^1 - \beta$ can be concluded. We define the challenges that are intentionally designed to use a part of the segments to compensate the rest delay difference as compensate challenges.

When designing a compensate challenge, the primary goal is to compensate the target delay difference as accurate as possible, so that the real delay difference can be accurately tested. For example, in the example of $0c_2c_3...c_n$, we want to design β to be as close to $d_i^0 - d_i^1$ as possible. The algorithm to find such compensate challenges employs the same idea as the greedy algorithm to find adaptive challenges.

Both the idea of adaptive challenges and compensate challenges are built on the top of the basic statistical model, aiming at improving the model accuracy. Our characterization algorithm has the advantage of being fast, scalable, and can be used to derive delay information for each individual segment.

2.4 Hardware PUF Emulation

2.4.1 Objective and Challenges

Hardware PUF emulation is defined as to create a piece of hardware with the same functionality of the original PUF. Although PUF characterization provides an accurate PUF model, it is restricted to be used in simulation. However, there is a need to emulate PUF in actual hardware in order to achieve lower area/energy overhead and faster speed. On the other hand, a more demanding scenario for PUF emulation is to use another actual PUF to emulate the original PUF. It not only inherits the good properties of traditional hardware PUF emulation, but also guarantees the hardware itself is unclonable. Various security protocols which require synchronized functional blocks can be enabled based on it.

The major challenge to emulate PUF B using PUF A is that it is almost impossible to find a PUF A with exactly the same delay information of PUF B since the delay of a PUF is purely decided by process variation. To leverage the above challenge, we have proposed a set of techniques to modify the delay of segments in PUF A to create a PUF emulation as close to PUF B as possible. To facilitate demonstration, we split the hardware emulation procedures into two scenarios based on the size and the configuration of PUFs. We first list the basic assumption of our PUF emulation as following.

- Use PUF A to emulate PUF B.
- Both PUF A and PUF B are characterized.
- PUF A has m segments, and PUF B has n segments.

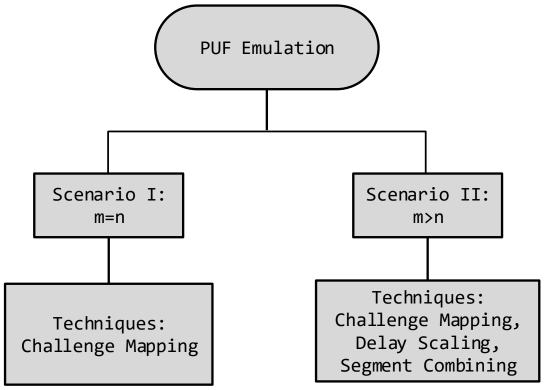

PUF Emulation
Scenario I: m=n
Scenario II: m>n
Techniques: Challenge Mapping
Techniques: Challenge Mapping, Delay Scaling, Segment Combining

Figure 2.5: Two categories of PUF emulation and the corresponding techniques.

The two scenarios of PUF emulation and the corresponding techniques are shown in Figure 2.5. Note that to emulate PUF B, PUF A must be at least the same size as PUF B ($m \geq n$). It is because the number of segments decides the mapping space between the PUF challenges and the PUF responses, it is impossible to use a smaller mapping space to emulate a larger mapping space. The first scenario considers the emulation strategy when PUF A and PUF B have the same size ($m = n$). It takes advantage of the technique "challenge mapping" to roughly emulate PUF B using PUF A. The second scenario is an extension to the first scenario, meanwhile also provides better emulation accuracy, where PUF A has more segments than PUF B ($m > n$). In addition to "challenge mapping", two more techniques are used, respectively "delay scaling", and "segment combining". In the following sections, we separately demonstrate our PUF emulation techniques for the two scenarios.

2.4.2 Scenario I: $m = n$

In this scenario, both PUF A and PUF B have the same number of segments. It is the simpler scenario among the two, and with less emulation accuracy. The major technique applied in this scenario is “challenge mapping”.

2.4.2.1 Challenge Mapping

The basic idea of challenge mapping is that given a challenge C_B for PUF B, there always exists a mapped challenge C_A for PUF A that has a high likelihood to produce the same output. Now assume that PUF A and PUF B have an equal number of segments. We use the notations in Equation 2.5 to represent the path delay difference. In the equation, the path delay difference is represented as a sum of plus and minus of the absolute value of each segment's delay difference. The sign before the absolute delay difference ($|d_i^{diff-A}|$ or $|d_i^{diff-B}|$) is decided by s_i which is the sum of the $parity(i)$ and the sign of d_i^{diff} as shown in Equation 2.6. $sign = 0$ when d_i^{diff} is positive, and $sign = 1$ otherwise.

$$\begin{aligned} D^{diff-A} &= \sum_{i=1}^{n} (-1)^{s_i^A} |d_i^{diff-A}| \\ D^{diff-B} &= \sum_{i=1}^{n} (-1)^{s_i^B} |d_i^{diff-B}| \end{aligned} \tag{2.5}$$

$$\begin{aligned} s_i^A &= parity^A(i) + sign(d_i^{diff-A}) \\ s_i^B &= parity^B(i) + sign(d_i^{diff-B}) \end{aligned} \tag{2.6}$$

The motivation of challenge mapping starts from a question: for a d_j^{diff-B} in PUF B, is it possible to find a d_i^{diff-A} in PUF A to replace its position when emulating? It requires the contribution of d_i^{diff-A} to D^{diff-A} is similar to the contribution of d_j^{diff-B} to D^{diff-B}. Our research suggests that for every d_i^{diff-A} in PUF A, there exists such a d_j^{diff-B} in PUF B to match it.

An example of challenge mapping is shown in Figure 2.6. The number in each segment represents the delay difference ranking of a segment among all the segments in the PUF. For example, 1 means that the delay difference in the segment is the largest among all the segments. The color shows the delay difference of a segment contributes to either the blue

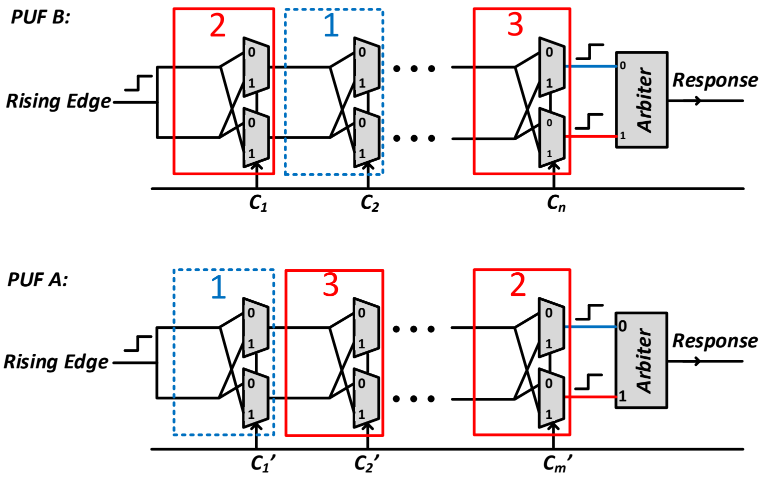


Figure 2.6: An example of challenge mapping.

path or the red path. In this example, the blue path equivalents to the upper path and the red path equivalents to the lower path. Our mapping works in the following way: we adjust the challenge in PUF A to guarantee that the same ranking segments in PUF A and PUF B contribute to the same path.

The algorithm of challenge mapping in shown in Algorithm 1. To guarantee that the same ranking segment contributes to the same path, we only need to guarantee that two segments have the same s_i as shown in Equation 2.6. To achieve this, given a challenge of PUF B, we find a mapped challenge of PUF A by adaptively assigning challenge bits from the last segment to the first segment in PUF A. Because of the order of assignment, when deciding a specific challenge bit c_i^A in PUF A, all the bits after have been assigned, so that the only variable in the s_i^A calculation is c_i^A. Therefore, by placing an appropriate value of c_i^A (either 0 or 1), s_i^A is decided so that the ith segment in PUF A matches the same ranking segment in PUF B.

2.4.3 Scenario II: $m > n$

The second scenario considers emulating the target PUF B using PUF A, and PUF A has more segments than PUF B ($m > n$). Compared to the scenario I, the key question is how

Algorithm 1 Challenge Mapping

```
Input: A challenge for PUF $B$: $C_B = c_1^B ... c_n^B$,
        $d_i^{diff-A}$ for segment $i$ in PUF $A$,
        $d_i^{diff-B}$ for segment $i$ in PUF $B$,
Output: A mapped challenge for PUF $A$: $C_A = c_1^A ... c_n^A$,
  1. Sort all the $|d_i^{diff-A}|$ in PUF $A$.
  2. Sort all the $|d_i^{diff-B}|$ in PUF $B$.
  3. For $i$ from $n$ to 1:
  4.     Find the ranking $R_i$ of $d_i^{diff-A}$ in PUF $A$.
  5.     Find $d_{a_i}^{diff-B}$ which has ranking $R_i$ in PUF $B$.
  6.     Calculate $s_{a_i}^B$ for $d_{a_i}^{diff-B}$ using $C_B$.
  7.     Set $c_i^A$ to guarantee $s_i^A = s_{a_i}^B$.
  8. Endfor
  9. Return $C_A = c_1^A ... c_n^A$.
```

to utilize the extra $m - n$ segments to improve the emulation accuracy. To achieve this, on the top of "challenge mapping", we additional introduce two more techniques, respectively "segment combining" and "delay scaling".

2.4.3.1 Segment Combining

One problem for challenge mapping is that the segments with the same ranking in two PUFs may have very different delays. For example, the segment with the largest delay difference for PUF A has $100ps$ difference, while for PUF B, the largest delay difference is $70ps$. Therefore, the highest ranking segment in PUF A actually has a much larger delay difference compared to the corresponding segment in PUF B. Such gap in delay differences has a direct influence on the matching accuracy. To leverage the above problem, we propose the technique of segment combining.

Instead of using only one segment in PUF A to match one segment in PUF B, segment combining proposes to use a few segments in PUF A to match a single segment in PUF B. As shown in Figure 2.7, the highest delay difference segment in PUF B has $70ps$ difference.

In PUF A, the same ranking segment has a delay difference of $100ps$. We assume that there exists another segment with delay difference $30ps$ adjacent to the PUF A segment. By properly assigning challenge bits, we can achieve delay difference $100ps - 30ps = 70ps$ over the 2 segments together. If we regard the above two segments together as a single segment to match the largest delay difference segment in PUF B as circled in blue (dashed line) in Figure 2.7, we can have the exact match of delay difference $70ps$.

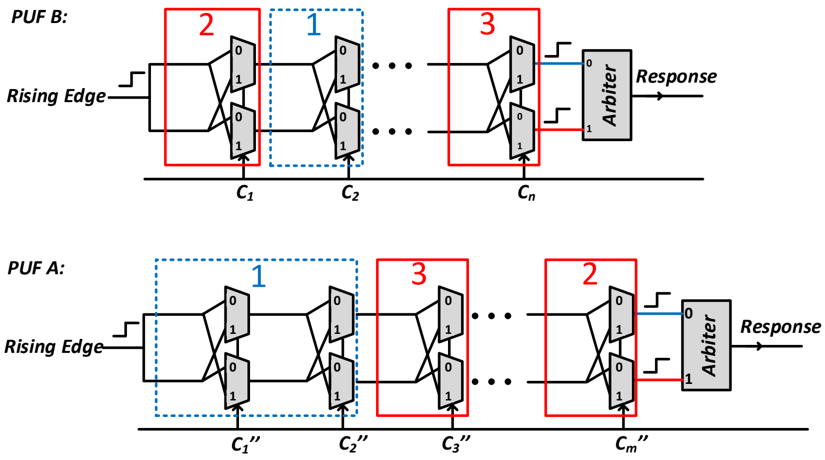


Figure 2.7: An example of segment combining.

Segment combining can significantly decrease the deviation between the delay difference of two PUF segments, but it requires a longer PUF A to emulate PUF B. However, one side-effect of using a longer PUF A is that there will be some segments left if not all the segments are used to match PUF B. Our solution is to put the challenge bits of those segments in such a way that their delay differences compensate to each other, thus the total effect is as close to 0 as possible.

However, it is extremely hard to decide an optimal way of segment combining since there are exponential possibilities. We leverage this problem by proposing the heuristic in Algorithm 2. The key idea is that we first find an optimal combining scheme for every single segment in PUF B. Then we check what will happen if we apply the optimal combining scheme for one of the PUF B segment d_B^j. Since a segment in PUF A can not be used for more than once, thus, the optimal combining scheme for the other PUF B segments will be

Algorithm 2 Segment Combining

```
Input: $d_A^i$ for segment $i$ in PUF $A$, $i \in \{1, 2..., m\}$.
    $d_B^j$ for segment $j$ in PUF $B$, $j \in \{1, 2..., n\}$.
    $S_A$ - a set initially has all the segments in PUF $A$.
    $S_B$ - a set initially has all the segments in PUF $B$.
    $t$ - constant.
Output: A combining scheme $C(d_B^j)$ for each segment $j$ in PUF $B$.
While $size(S_B) > 0$:
    For $d_B^j$ in $S_B$:
        Find the top combining scheme $\hat{C}(d_B^j)$ for segment $d_B^j$ using at most $t$ segments from $S_A$.
        $S'_A$=$S_A$-segments used in $\hat{C}(d_B^j)$
        $S'_B$=$S_B$-$d_B^j$
        $Error(d_B^j)$=0
        For $d_B^i$ in $S'_B$:
            Find the top combining scheme $\hat{C}(d_B^i)'$ for segment $d_B^i$ using at most $t$ segments from $S'_A$.
            $Error(d_B^j)$+=$|d_B^i - \hat{C}(d_B^i)'| - |d_B^i - \hat{C}(d_B^i)|$
        Endfor
    Endfor
    Sort $Error(d_B^j)$ in non-decreasing order.
    Find Min($Error(d_B^j)$) when $j = J$.
    Delete $d_B^J$ from $S_B$.
    Delete $\hat{C}(d_B^J)$ from $S_A$.
    $C(d_B^J) = \hat{C}(d_B^J)$.
Endwhile
Return $C(d_B^j)$ for $j \in \{1, 2..., n\}$.
```

influenced as they may share the same PUF A segments in their combining schemes with the ones to match d_B^j. We use function *Error* to represent such influence. After iterating through all the optimal combining schemes, we choose the one with the least influence over the other segments to really apply. This is because we want to minimize the loss of accuracy on the other PUF B segments' combing schemes caused by applying the optimal scheme on the current segment. In the next iteration, we update the available segments in PUF A (S_A) and the target segments in PUF B (S_B) and repeat the previous step.

2.4.3.2 Delay Scaling

Another key technique for scenario II PUF emulation is delay scaling. Equation 2.7 shows the delay difference notation for each segment in PUF A and PUF B. The delay difference of the ith segment in PUF B is a scaling of the delay difference of the ith segment in PUF A with the ratio α. We claim that PUF A and PUF B generate the same response when given the same challenge. According to Equation 2.4, the PUF response is decided by the sign of the sum of delay difference. When scaling all the delay segments by α at the same time, the path delay difference is also scaled by α. But this will not change the sign of the path delay difference, thus the response keeps the same.

$$\begin{gathered} PUF\ A = \{d_i^{diff-A}\} \\ PUF\ B = \{d_i^{diff-B}\}, \\ d_i^{diff-B} = \alpha * d_i^{diff-A}, \alpha > 0 \end{gathered} \tag{2.7}$$

We combine delay scaling with segment combining in the following way. Based on delay scaling, we know that the segments of PUF A don't need to be exactly combined to match each d_i^{diff-B}. Instead, they can be combined to $\alpha * d_i^{diff-B}$ as long as α is consistent for all the segments. In order to find a suitable α, we iterate through the candidate values from our defined lower bound to upper bound, and for each possible α, we repeat the process of segment combining. The above process repeats until we find a α with which the segment combining provides the best modeling accuracy.

To summarize the process of PUF emulation in scenario II, after the characterization of PUF A and PUF B, we find the best scaling ratio and the corresponding arrangement for segment combining. In the last step, for any random challenge C_B given to PUF B with a response R_B, we use challenge mapping to form a mapped challenge C_A to PUF A. This challenge has a high likelihood to generate $R_A = R_B$.

2.5 Security Protocols

PUF emulation allows multiple PUFs to have matched challenges to generate the same response. Using this as a starting point, various security protocols are enabled using our PUF matching. The major advantage is that now all the parties have a physical "unclonable" copy of hardware to encrypt or decrypt, which is much faster and less power consuming compared to applying software simulation.

2.5.1 Message Encryption and Decryption

One of the most commonly discussed security protocol is message encryption and decryption. It can be easily implemented using our proposed PUF matching. We assume that Alice and Bob are the two communicating parties where each of them owns a unique PUF. After Alice shares her PUF A characterization (includes all the segment delay information) with Bob who owns PUF B, PUF B is used to emulate PUF A based on our emulation methods. With the XOR operation, any message M sent from Alice can be decrypted using PUF B. The detailed flow is presented in Figure 2.8.

The above-proposed message exchange is an extension of the traditional PUF based encryption and decryption. In the traditional protocol, Bob can only simulate PUF A for decryption which is much slower and more energy consuming than the hardware emulation. However, a major concern of identity protection still exists. With the knowledge of PUF A characterization, Bob can pretend to be Alice. It can not be prevented if Bob needs to emulate PUF A himself. Consequently, we decide to introduce a trusted third party (TTP) to finish the job of PUF emulation in such a way that the characteristics of PUFs are invisible to all the rest parties.

Our modified message encryption and decryption protocol is shown in Figure 2.9. The TTP is in charge of managing the characteristics of all the PUFs. When Alice wants to

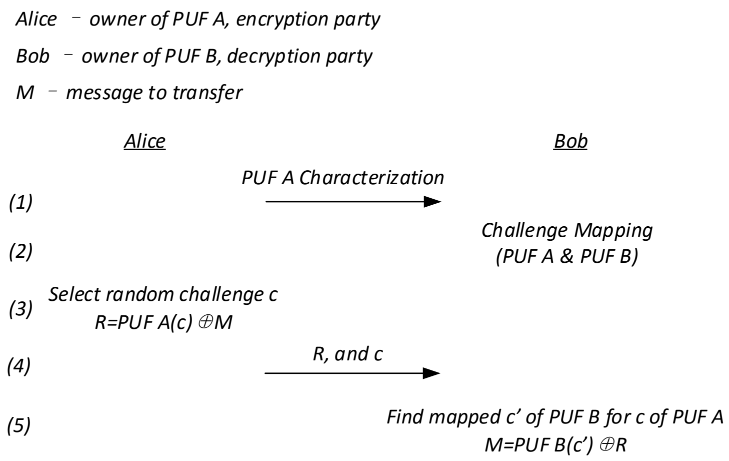

Alice – owner of PUF A, encryption party
Bob – owner of PUF B, decryption party
M – message to transfer
Alice
Bob
(1)
PUF A Characterization
(2)
Challenge Mapping
(PUF A & PUF B)
(3)
Select random challenge c
R=PUF A(c) ⊕M
(4)
R, and c
(5)
Find mapped c' of PUF B for c of PUF A
M=PUF B(c') ⊕R

Figure 2.8: PUF based message encryption and decryption.

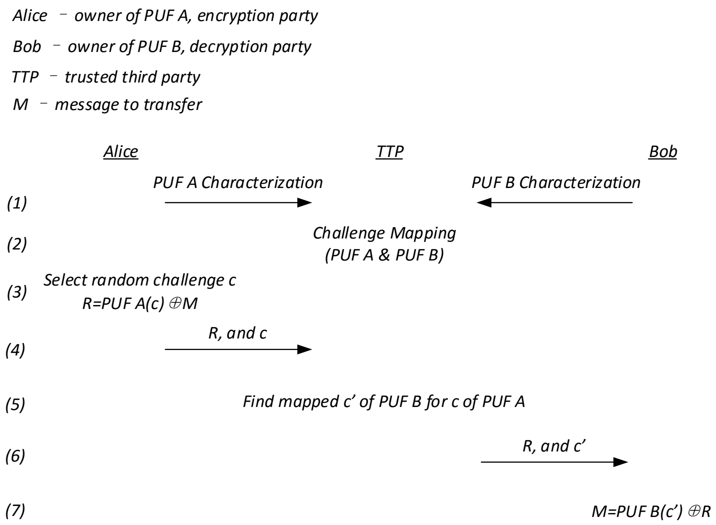


Figure 2.9: A modified PUF based message encryption and decryption using a trusted third party.

send a message M to Bob, she needs to first send the random challenge c together with the XORed message R to TTP, then the TTP is responsible for finding out the mapped challenge c' for PUF B and sends it together with the encrypted message R to Bob. In this protocol, the TTP serves as a centralized party. It manages the PUF emulation originally done by the decryption party to prevent identity theft.

2.5.2 Multi-party Communication

The protocol of multi-party communication assumes that N trusted parties want to exchange information mutually ($N \geq 2$). The message exchange should be acted in such a way that when a message is sent, all the other $N-1$ parties are able to retrieve the message, but any untrusted parties are incapable of doing so.

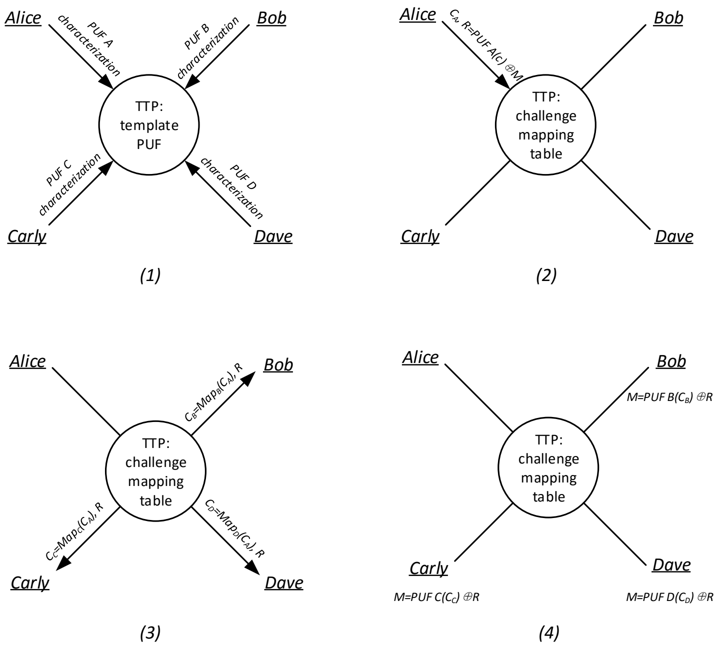


Figure 2.10: PUF based multi-party communication.

We present our PUF based multi-party communication scheme in Figure 2.10. We assume $N = 4$, and each party owns a unique PUF. The first step is that all the parties need to send their PUF characterizations to the TTP. The TTP will create a random template PUF which is used as the target PUF for all the PUFs to match to. Using the characterization information of each PUF, TTP does the matching and thus generates a challenge mapping table for all the PUFs to the template PUF. Note that the matching by TTP follows the process of PUF emulation, but is purely software based. For example, for challenge C in template PUF, it equivalents to challenge C_A in PUF A, challenge C_B in PUF B, challenge C_C in PUF C, and challenge C_D in PUF D. In the second step, when Alice wants to send message M to the other parties, similarly to the two party encryption and decryption, she sends a random challenge C_A and R to TTP. Then TTP will look up the challenge mapping table to find out the corresponding challenge C_B, C_C, and C_D for PUF B, C, and D. In the third step, TTP sends the challenge C_i and R to party i. Eventually, each party uses their own PUF to decrypt message M.

With the existence of TTP, all the parties do not need to expose their own PUF characterizations to the other parties. Meanwhile, with all PUFs match to a single template PUF in the TTP, it facilitates the process of PUF matching as well as the required amount of information exchange among all the parties. Note that the template PUF of TTP does not have to be a real PUF, it can simply be a virtual PUF model, serving as the target of PUF matching.

2.6 Results

In the following experiments, we repeat each test on 10 individually implemented PUFs or pairs of PUFs and take the average value to present. In the following parts, we first demonstrate our PUF implementation on FPGA, then we illustrate the results of PUF characterization and emulation respectively.

2.6.1 PUF Implementation on FPGA

Our arbiter PUF is implemented on a Spartan-6 LX45 FPGA. The entire PUF, including two 32-input delay paths and the arbiter, occupies only 9 logic slices (an FPGA such as the Spartan-6 LX45 has over 6800 slices).

Conceptually, a shuffle segment can be viewed as a pair of two-to-one multiplexers with shared input signals in opposite orders (see Figure 2.11). On the other hand, the Xilinx Spartan-6 FPGA uses six-input LUTs (LUT6) as the basic logic element, and each LUT6 can be used as two LUT5 units if they share the same inputs in the same order. To accommodate the opposite input order, the memory contents of each LUT5 are adjusted accordingly. This architecture allows each segment to be efficiently mapped to a single LUT6.

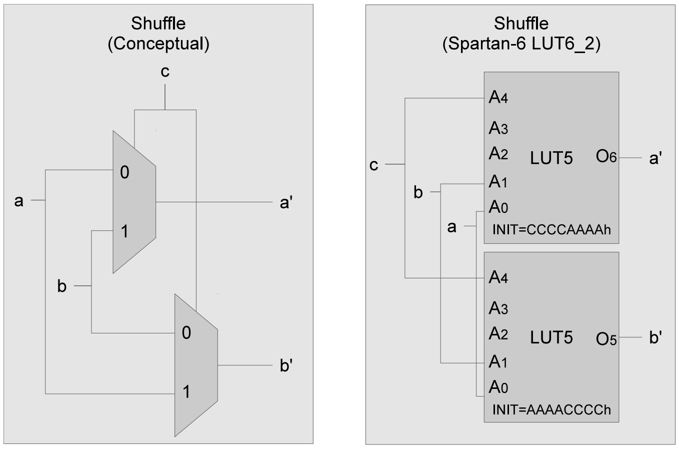


Figure 2.11: Conceptual shuffle segment and LUT6 realization on Spartan-6.

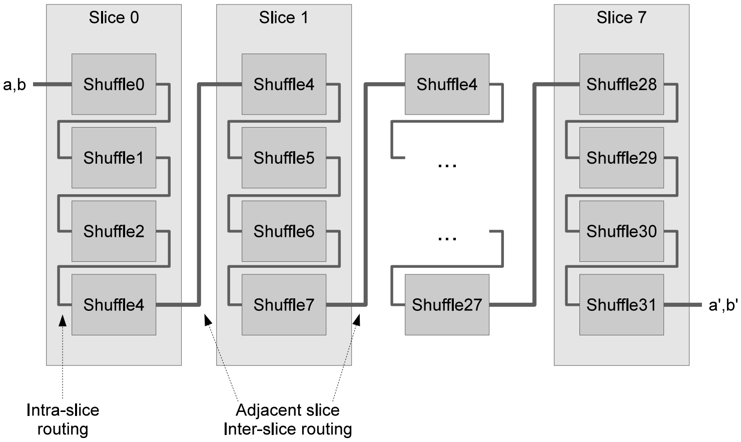


Figure 2.12: 32-segment PUF delay paths mapped on Spartan-6.

Besides using fewer logic resources, packing efficiently is also crucial to implementing balanced arbiter paths on an FPGA platform, where routing is usually outside the control of the designer. A Spartan-6 logic slice contains 4 LUT6 units, and the signal routing within the slice goes through a locally attached switching matrix. By packing four adjacent segments into one slice, routing between these four segments is minimized. Figure 2.12 illustrates a packing scheme that packs 32 shuffle segments into 8 adjacent logic slices. Notice that the routing within each 4-segment cluster utilizes the low-delay intra-slice routing channels. The slices are location constrained to be adjacent to each other to minimize the routing between the 4-segment clusters.

2.6.2 Characterization Results

Table 2.1 shows the prediction accuracy using our PUF characterization algorithm. For each instance, we vary the number of CRPs applied to build the statistical model as well as the size of PUFs. Then we random apply 10,000 challenges to compare the real response of the PUF and the predicted response based on the statistical model. From there, we calculate the correct rate of our response prediction. We also separately test the prediction accuracy with and without using adaptive/compensate challenges. From the results in Table 2.1, we have the following observations.

PUF size	Characterization Tech.	CRPs			
		1,000	10,000	100,000	1,000,000
32-bits	statistical model	91.9%	96.6%	97.3%	97.5%
32-bits	stats. model+adap./comp. challenges	93.1%	97.1%	97.9%	97.9%
64-bits	statistical model	90.7%	95.6%	96.4%	96.9%
64-bits	stats. model+adap./comp. challenges	92.9%	96.8%	97.0%	97.2%
128-bits	statistical model	89.6%	94.7%	96.0%	96.3%
128-bits	stats. model+adap./comp. challenges	91.5%	95.2%	96.3%	96.5%

Table 2.1: Response prediction accuracy using PUF characterization model under various number of CRPs.

Firstly, we can clearly see that the prediction accuracy based on PUF characterization can easily reach 96%-97% when enough CRPs are used. Besides, longer arbiter PUFs are obviously harder to predict given a fixed number of CRPs since more delay information needs to be retrieved. Secondly, adaptive/compensate challenges can improve the accuracy of PUF characterization model, although the improvement gets insignificant as the number of CRPs increases.

2.6.3 Emulation Results

Table 2.2 shows the prediction accuracy using PUF emulation. Assume that PUF A is used to emulate PUF B. Each row is the number of bits of PUF B (n), and each column is the number of bits of PUF A (m). Note that $m \geq n$. The characterization model we are using is based on the statistical model achieved using 1,000,000 CRPs.

We can clearly see that the accuracy of PUF emulation increases as we increase the size of PUF A. It suggests that our algorithms of segment combining and delay scaling have worked. However, when comparing the PUF emulation results to the PUF characterization results as shown in Table 2.1, the prediction accuracy significantly drops. It is because there always exists delay difference gaps between PUF A and PUF B segments which can not be completely eliminated through hardware emulation.

# of bits	m=32	m=64	m=128	m=256
n=32	89.6%	90.8%	91.6%	93.2%
n=64	-	90.3%	91.1%	92.6%
n=128	-	-	90.9%	92.5%

Table 2.2: Response prediction accuracy using PUF emulation: emulate PUF B (n-bits) using PUF A (m-bits).

2.7 Summary

In this chapter, we have proposed a new approach to characterize an arbiter PUF by combining the statistical model with adaptive challenges and compensate challenges. On the base of our characterization results, we demonstrate our approach to emulate an arbiter PUF using another arbiter PUF. By applying our proposed techniques, two arbiter PUFs can be matched with high accuracy. Consequently, a few security protocols are enabled such as message encryption and decryption, and multi-party communication.

We evaluate our algorithms with PUFs implemented on FPGA. Our result suggests that the accuracy of response prediction based on our characterization model can reach 96%-97% with $1,000,000$ CRPs. Meanwhile, the prediction accuracy based on PUF emulation is around 89%-93%.

CHAPTER 3

Enable PUF Matching using Programmable Delay Lines

3.1 Motivation and Problem Formulation

In the previous chapter, we have shown that through PUF emulation, two random PUFs can be matched in such a way that they have a high likelihood to produce the same output when given a pair of matched input vector. A variety of security protocols that are widely used in IoT systems can be built using the emulated PUFs, such as message encryption and decryption, and multi-party communication. However, according to the implementation results, the matching accuracy of two PUFs is only around 92-93% in the best case, which is still far from the standard of being applied in the actual secure communication.

In this chapter, our technical goal is to demonstrate a new platform for PUF matching to boost the matching accuracy. We achieve our goal by using the look-up table (LUT) based programmable delay lines (PDL). The PDL are applied to tune and modify the delay of each segment in the standard PUFs, and eventually in hope to match multiple PUFs. The generated matched PUFs should preserve all the properties of the standard PUFs, and the probability of a third party creating a same copy of the PUF must be negligible. By creating a set of PUFs that are matched to each other, each party is able to possess one PUF copy from the set, thus facilitating low-cost communication.

3.2 Preliminaries

3.2.1 Programmable Delay Lines

Programmable delay lines are a series of digital delay lines with electrically programmable and trimmable delay times [19]. They take advantage of the lookup table (LUT) internal structure on FPGA to create delay bias and use it to generate controllable delays.

To measure the delay bias, a high precision delay testing technique with picosecond resolution on FPGA is required. Some previous work includes: Raychowdhury et al. [20] proposed on-chip delay measurement hardware techniques to estimate the segment path delay. Tsai et al. [21] proposed a vernier delay lines based built-in delay measurement circuit with a small area overhead that can provide high-resolution delay measurement. Majzoobi et al. [22] designed a timing characterization circuit with clock synthesis that can measure pico-second resolution on FPGA.

3.2.2 PDL Implementation on FPGA

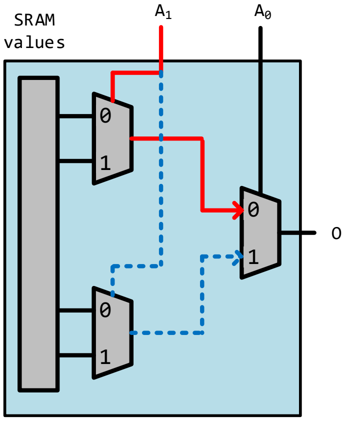


Figure 3.1: The internal structure of a 2-input LUT.

The PDL design on FPGA is proposed by Majzoobi et al [14]. It is implemented by a single LUT. Figure 3.1 shows an example LUT with 2 selection bits A_1 and A_0. Now consider two scenarios respectively when $A_0 = 0$ and $A_0 = 1$. The propagation delay from

A_1 to O when $A_0 = 0$ is depicted in the solid red line, and the propagation delay when $A_0 = 1$ is marked in the dashed blue line. From the figure, we can clearly see that due to the asymmetric structure in the LUT, the propagation delay in the blue line is slightly larger than the propagation delay in the red line. Many modern FPGAs have on board 6 selection-bit LUTs. By assigning different selection bits, the LUT will have slightly different delays, and the delay difference caused by such asymmetricity is in picosecond resolution. We take advantage of the small delay difference to tune delay segments in the arbiter PUFs.

3.3 A Motivational Example

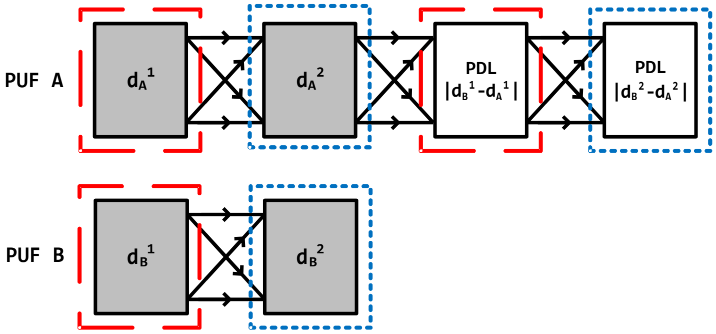


Figure 3.2: An example of PUF matching using PDL.

We present a motivational example of our PUF matching scheme in Figure 3.2. Both PUF A and PUF B originally have two segments. In order to match their first segments, respectively with d_A^1 delay difference in PUF A and d_B^1 in PUF B, we add an additional segment built by the PDL to PUF A with the delay of $|d_B^1 - d_A^1|$. This can be done by using two LUTs in a PUF segment while each LUT uses a set of different selection bits in such a way that the delay difference is exactly $|d_B^1 - d_A^1|$. By properly combining the existing first segment in PUF A (d_A^1) with the additional PDL segment ($|d_B^1 - d_A^1|$), they together can represent the first segment in PUF B which has the delay of d_B^1. The similar matching process can be repeated on the second segment of PUF A and B. To summarize, the basic idea for matching two $n-bit$ PUFs is to create n additional segments using PDL and attach them to the end of a PUF. The delays of PDL are designed in such a way that each one of them can be combined with an existing ith segment in the current PUF so that they together realize the same delay difference as the ith segment in the other PUF.

3.4 PUF Matching

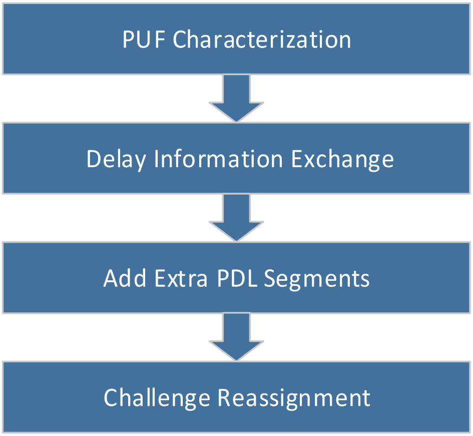


Figure 3.3: The flow of PUF matching.

The PUF matching process can be divided into four steps as shown in Figure 3.3. We demonstrate each procedure separately in the following parts. For simplification, we start with PUF matching between two parties. We assume that Alice owns an n-bit arbiter PUF A and Bob owns an n-bit arbiter PUF B. The goal is to match PUF A with PUF B.

3.4.1 PUF Characterization

In chapter 2, we have proven that standard arbiter PUFs can be characterized using statistical methods. To match two PUFs, the first step is to characterize the two PUFs. If the ith segment has two delays, d_i^0 and d_i^1, we denote the delay difference between d_i^0 and d_i^1 as d_i^{diff} as shown in Equation 3.1. We assume both PUFs have n segments

$$d_i^{diff} = d_i^0 - d_i^1, i \in \{1, 2, ...n\} \tag{3.1}$$

Using the notation of d_i^{diff}, the total delay difference (T_{diff}) between two PUF paths can be represented in Equation 3.2. The parity function denotes the number of times that switching happens after the current segment, which is decided by the number of ones in the challenge. The total delay difference of the two PUF paths can be denoted as the plus or minus of the delay difference of each segment.

$$
\begin{aligned}
T_{diff} &= \sum_{i=1}^{n} (-1)^{parity(i)} * d_i^{diff} \\
parity(i) &= \begin{cases} 0, & even\ 1s\ in\ \{c_i ... c_n\} \\ 1, & odd\ 1s\ in\ \{c_i ... c_n\} \end{cases}
\end{aligned}
\tag{3.2}
$$

We rewrite the Equation 3.2 to Equation 3.3 using vectors formats. Note that each challenge corresponds to a unique P vector which consists of only 1 and -1.

$$
\begin{aligned}
T_{diff} &= P \cdot D \\
P &= \{(-1)^{parity(1)}, (-1)^{parity(2)}, ...(-1)^{parity(n)}\} \\
D &= \{d_1^{diff}, d_2^{diff}, ...d_n^{diff}\}
\end{aligned}
\tag{3.3}
$$

In our characterization, we follow the notations in Equation 3.3 and measure a set of T_{diff} given different P values, so that to create linear equations regarding D which is a vector of d_i^{diff}. We solve the equations and retrieve the delay difference ($d_i^0 - d_i^1$) for each PUF segment.

3.4.2 Delay Information Exchange

Bob and Alice exchange the characterized delay difference information in the second step. This process is only one-time and can be done through the standard cryptographic approaches. Based on the exchanged characterization, Alice and Bob need to define a PUF matching template, which is used as the target PUF that PUF A and PUF B are matched to. Note that the template PUF has no physical entity; it is purely a conceptual function.

To facilitate demonstration, we denote the delay difference of each segment in PUF A, PUF B, and the template PUF in Equation 3.4. There are multiple ways to define the delays of the template PUF, we have shown an example definition in Equation 3.5.

$$\begin{aligned} PUF\ A &= \{d_A^1, d_A^2, ...d_A^n\} \\ PUF\ B &= \{d_B^1, d_B^2, ...d_B^n\} \\ PUF\ Template &= \{d_T^1, d_T^2, ...d_T^n\} \end{aligned} \tag{3.4}$$

$$d_T^i = max(d_A^i,\ d_B^i),\ i \in \{1, 2, ...n\} \tag{3.5}$$

3.4.3 Appending PDL Segments

The goal of this step is to modify PUF A and PUF B to match the template PUF using PDL. With the template PUF defined in Equation 3.5, two situations may happen during the matching process. Take the PUF A matching as an example:

(1)$d_A^i < d_T^i$. An extra segment with delay difference $|d_T^i - d_A^i|$ needs to be added to PUF A. This segment together with the ith segment in PUF A are equivalent to the ith segment in the template PUF.

(2)$d_A^i = d_T^i$. In this situation, nothing needs to done to modify the ith segment in PUF A.

We build the extra delay segments using PDL. In each segment, one LUT with selection bit combination C_{upper} is used as the upper delay and the other LUT with selection bit combination C_{lower} is used as the lower delay. Due to the delay bias caused by the asymmetric internal structure of LUTs, different input combinations to LUTs will generate distinct delays. Therefore, by properly assigning C_{upper} and C_{lower}, target delay difference ($|d_T^i - d_A^i|$) can be created. In some cases, $|d_T^i - d_A^i|$ is larger than the maximum delay difference that can be generated using C_{upper} and C_{lower}, then multiple LUTs can be applied as the upper delay/lower delay to boost the delay difference proportionally.

3.4.4 Challenge Reassignment

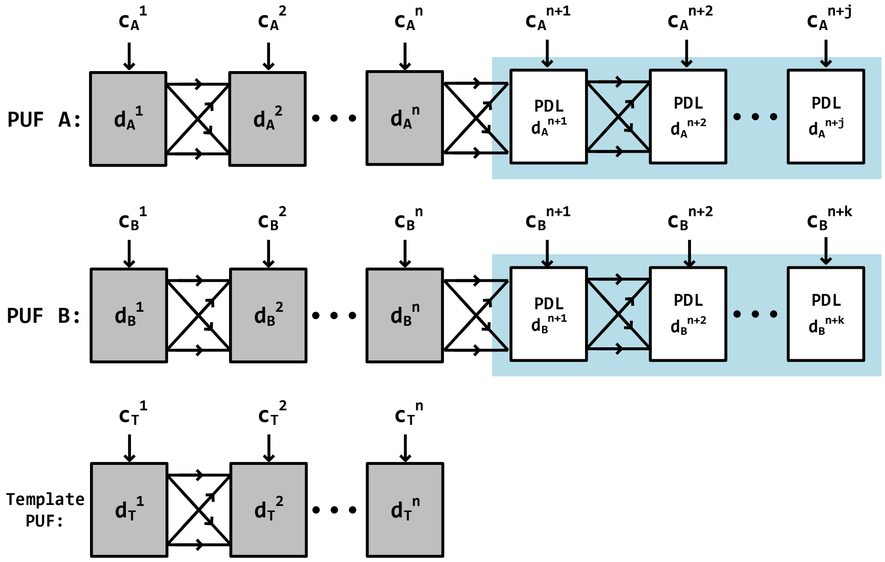


Figure 3.4: Structures of PUF A and PUF B after matching using PDL.

After the previous steps, both PUF A and PUF B now are matched to the template PUF, thus are expected to have the same challenge-response mapping function. The structures of matched PUF A and PUF B are shown in Figure 3.4. Assume that PUF A has j additional segments and PUF B has k additional segments implemented using PDL. Then $k + j$ should equal to n according to our defined template PUF, where n is the size of the original PUF. Now consider a random n-bit challenge C_T is fed to the template PUF to generate a response R. The key question is how to adjust the $(n + j)$-bit challenge C_A to be fed to PUF A and the $(n + k)$-bit challenge C_B to be fed to PUF B to generate the same response R.

We take PUF A as an example to illustrate our challenge reassignment algorithm. We start with the rightmost segmental delay difference d_A^{n+j} and check the corresponding segmental delay difference d_T^i that d_A^{n+j} is matched to in the template PUF. The segmental delay difference d_T^i, which equals to d_B^i in PUF B should be the sum of d_A^{n+j} and d_A^i based on the matching policy. Then according to the challenge C_T, we check whether the segmental delay difference d_T^i contributes to the upper path or the lower path before the arbiter. Based on the observed path P in the template PUF, we assign the $(n + j)th$ challenge bit C_A^{n+j} in C_A

in such a way that d_A^{n+j} also contributes to the same path P in PUF A. The above process is repeated with all the rest segments in PUF A. Note that when we assign a challenge bit to the ith segment in PUF A, all the bits after the ith segment have been assigned already. Therefore, by assigning the ith challenge bit to either 0 or 1, according to Equation 3.2, the delay difference in the segment contributes to either the upper path or the lower path. With the above process, for any random challenge to the template PUF, we can find a matched challenge to PUF A and another matched challenge to PUF B to make them all generate the same response.

3.4.5 Discussion

The above PUF matching flow can be easily extended to the multi-party PUF matching. As long as all the parties are synchronized with the template PUF, they can always add new PDL segments and follow the same algorithm to reassign challenges for matching. This provides the potential for message encryption/decryption between multiple parties.

The above matching process maintains the properties of standard PUFs. On one hand, the extra area and delay overhead are reasonably small. In the case of matching two PUFs, the total number of additional segments equals to n, which is the length of an original PUF. On the other hand, the generated matched PUFs remain unclonable. Although an attacker can reproduce the PDL segments by applying the same selection bits to the LUTs, he/she is unable to duplicate the segments of the original PUF because of process variation. This also explains the reason we can not directly use PDL to build a whole new PUF since then the unclonability of the function will be compromised.

3.5 Implementation

We demonstrate our implementation and evaluation of the PDL-based PUF matching mechanism in this section. All implementations and measurements are on Spartan-6 XC6SLX45 FPGAs.

3.5.1 Delay Measurement Setup

In order to measure and verify the delay of PDL on the FPGA, we use the circuit describe by Majzoobi et al [14]. The delay characterization circuit is shown in Figure 3.5. The entire measurement system is constructed by three parts: circuit under test (CUT), flip-flops and external modules.

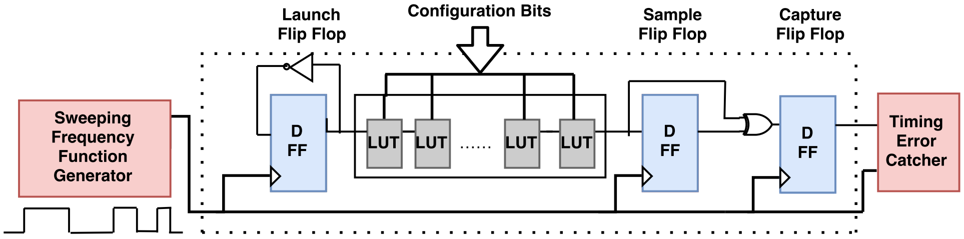


Figure 3.5: Delay characterization circuit.

3.5.1.1 Circuit Under Test

Our objective is to measure the delay of PDL when providing different input vectors. However, such delays are too small to measure individually, thus a chain of LUTs is used as our target CUT. We use on-board LUT6 to build our CUT and each LUT in the chain implements an inverter. To be specific, only the most significant selection bit fed to the LUT is inverted and the remaining 5 bits are used as configuration bits to configure the internal signal routing path inside the LUT. The configuration bits are identical for every LUT in the CUT so that we are able to estimate individual delay by calculating CUT delay over the number of LUTs. We measure the delays of CUT that consists of 10 LUTs when providing all possible configuration bits from “00000” to “11111”.

3.5.1.2 Flip-flops

There are three D flip-flops used in the delay characterization circuit: launch flip-flop, sample flip-flop, and capture flip-flop. All three flip-flops share the same clock signal generated by a sweeping frequency function generator. The launch flip-flop generates a low-to-high signal through the CUT at the rising edge of the clock. Then the signal arrives at the sample flip-flop. If the signal arrives before the sampling action takes place (at the rising edge of the clock) then the correct signal value is successfully sampled, otherwise, the sampled value would be different indicating a timing error. Such a mismatch can be detected by a simple XOR gate. If the sampled value and the correct signal value are the same, the XOR gate would produce a 0 indicating no timing violation occurred. Otherwise, the capture flip-flop will receive a 1 from the XOR gate indicating a timing error.

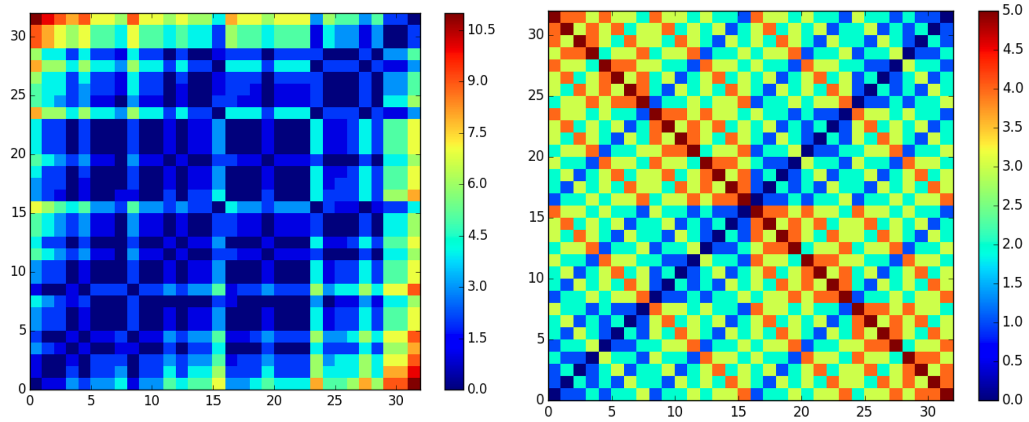
3.5.1.3 External Modules

There are two external modules used in the delay measurement process. A sweeping frequency function generator is used to generate a square wave from 2MHz to 100MHz. We then shift the frequency up by 32 times using the phased lock loop on the FPGA. For each frequency, the generator produces a fixed number of 10,000 pulses which are used to drive the flip-flops. A timing error catcher module takes the output of the capture flip-flop and the clock signal as inputs and calculates the probability of a timing error occurrence. By monitoring the timing error probability, the CUT delay can be inferred at picosecond resolution.

3.5.2 Delay Measurement Results

We have measured the average delay of our CUT under 25 °C operating temperature and then calculate the individual LUT delay. Figure 3.6a shows the delay difference between any pair of selection bits. Figure 3.6b demonstrates the hamming distance between selection bits.

The largest difference is approximately 11 ps, which occurs between selection bits 00000 and 11111. This case is found at location (x,y) = (0,31) and location (31,0) in Figure 3.6a. We also notice that some patterns shown in Figure 3.6a can be observed in Figure 3.6b. In many cases, if the two configuration vectors have a large delay difference in PDL, these two vectors will also have a large hamming distance. We believe this is a valid observation because large hamming distance indicates that the corresponding internal signal paths share very few common routes, consequently it is more likely to generate a larger delay difference.



(a)

(b)

Figure 3.6: (a) The absolute value of delay difference between any pair of selection bits (00000 to 11111). Delay difference unit: ps. (b) The hamming distance between all pairs of selection bits.

3.5.3 Process Variation

Process variation is not avoidable when we measure delays at picosecond resolution. We have run experiments on 3 difference FPGA boards as well as different locations on each board to test the effect of process variation. We have measured the delays of PDL on 5 different locations on each board when providing 00000 and 11111 as configuration bits. The average delays of PDL on three boards are compared and presented in Table 3.1. The results indicate that despite the delays under the same selection bits of the three FPGAs are distinctive, the delay difference between 00000 and 11111 is relatively stable.

	00000 (ns)	11111 (ns)	Difference (ns)
FPGA 1	1.253	1.265	0.012
FPGA 2	1.248	1.259	0.011
FPGA 3	1.257	1.267	0.010

Table 3.1: Delay measurement results on three different FPGAs.

Based on our measurement, we believe it is safe to assume that the time difference between different routes within PDL is larger than the difference caused by the effect of process variation.

3.6 Results

We implement our PDL-based PUF matching platform on Spartan-6 XC6SLX45 FPGAs. We first test the matching accuracy of our matching scheme. We then examine the system overhead to prove that our design is lightweight.

3.6.1 Matching Accuracy

When applying the same challenge vector to a pair of matched PUFs, the probability that the two PUFs generate the same response is defined as matching accuracy.

We follow our proposed matching scheme to implement and match two 64-bit PUFs. We have generated 1,000,000 random challenges and applied them to the two PUFs. The test is repeated 10 times and the average matching accuracy reaches 98.64%. Compared to the PUF emulation result in Table 2.2, the accuracy has been significantly improved.

3.6.2 System Overhead

We measure the delay, area and energy consumption for the PDL-based PUF matching platform and show the results in Table 3.2. Note that we have measured the average overhead of a single PUF owner in our matching scheme. We further compare our design with several popular low energy block ciphers as shown in Table 3.3. The comparison indicates that our design is competitive regarding the size of the design while is the best in terms of energy consumption (due to small delay overhead).

Type	Overhead
LUTs	196
Slices	145
Max Delay (ns)	116.712
Energy (μJ)	9.54×10^{-4}

Table 3.2: The overhead of PDL-based matched PUF.

Type	Energy(μJ)	LUTs
TinyXTEA-3 [23]	5.45×10^{-3}	364
Present [23]	3.16×10^{-3}	159
HIGHTs [23]	1.07×10^{-3}	132
Matched PUF using PDL	9.54×10^{-4}	196

Table 3.3: The energy and the area overhead comparison.

Statistical Test	Lowest Success Ratio
Frequency	100%
Block Frequency (m=128)	97.6%
Cusum-Forward	98.1%
Cusum-Reverse	98.3%
Runs	98.5%
Longest Runs of Ones	96.5%
Rank	98.2%
Spectral DFT	95.6%
Non-overlapping Templates ($m = 9$)	95.9%
Universal	100%
Approximate Entropy ($m = 8$)	96.5%
Rand. Excursions ($x = 1$)	98.2%
Rand. Excursions Variant ($x = -1$)	97.6%
Serial ($m = 16$)	98.7%
Linear Complexity ($M = 500$)	97.8%

Table 3.4: The average success ratio for the NIST statistical test suite. The test passes for p-value$\geq \sigma$, where σ is 0.01.

3.6.3 Randomness Test

As a security primitive, the output randomness is an important criteria to evaluate the security property. We quantify the statistical randomness of the matched PUF outputs by applying the industry-standard statistical test suite of the National Institute of Standards and Technology (NIST) [24]. We generate a stream of outputs in the following way: a random seed is used as the primary inputs to the matched PUF after configuration and the corresponding outputs are generated. In each subsequent clock cycle, the outputs are first shuffled and then XORed with the previous inputs to generate the inputs for the next clock cycle. We repeat the process until we have collected enough outputs required by the benchmark suite. For each test, we use 10 instances of matched PUFs, the results in Table 3.4 show the lowest passing ratio of each subtest over all the instances.

3.7 Summary

In this chapter, we have proposed an ultra-low energy PUF matching scheme by using PDL. Our core idea is to modify/add arbiter PUF segments in such a way that multiple PUFs have the same challenge-response mapping function. On the top of our PUF matching platform, a variety of low overhead security protocols between multiple parties are enabled. Furthermore, we have implemented our design on the Spartan-6 FPGA platform. The experiment results indicate that our design allows the PUFs to be matched with high accuracy while requiring ultra-low energy and area overhead.

CHAPTER 4

Ultra-low Energy Public Key Communication using Digital Bimodal Function

4.1 Motivation and Problem Formulation

Classic algorithmic cryptography has been widely used in the past few decades, covering both private key and public key communications. However, due to its relatively slow speed, high energy consumption, and large hardware footprint, it is not well suited for the IoT devices. It is also often susceptible to side channel and physical attacks. Furthermore, classic cryptographic algorithms have only been retrofitted for emerging tasks such as remote trusted sensing and computation, but are not optimized in terms of energy and speed for such applications.

On the other hand, new types of security primitives are proposed and analyzed. Many of them are specifically designed to meet the requirements of modern systems. However, with most efforts made to apply the above security primitives to private key communication, public key cryptography is still a domain that is rarely addressed. Some previous efforts to improve the speed and energy of public key communication include the following. A compact implementation of RSA on FPGA is proposed by Oksuzoglu et al. [25]. While it is novel to combine RSA with FPGA, the design is still highly restricted by the algorithmic flow of RSA, preventing a larger save in energy. The other effort is related to PUF. A derivative of PUF called public PUF is designed by Beckmann et al. [26] to be used for public key cryptography. A problem of the design is that it requires a lot of computational efforts for at least one involved party (normally are the parties who use public key since they need to simulate the original PUF).

The objective of this chapter is to design and implement a new type of security primitive named digital bimodal function (DBF) that allows ultra-low energy consumption for all the legitimate parties in public key communication. The key idea of DBFs is to have a set of Boolean functions presented in two forms, $f_{compact}$ and $f_{expanded}$, with which $f_{compact}$ can be calculated ultra-fast and low-energy while $f_{expanded}$ is the opposite. We then take advantage of the above difference and use the two forms separately as the private key and the public key in the protocol. The properties of DBFs are summarized as following.

- Low-energy consumption for all involved parties in public key communication.
- Small-area implementation on FPGA.
- Excellent security properties in terms of confusion and diffusion.

The rest chapter is organized in the following way. We start with a motivational example of the DBF. Then we formally explain its design and architecture. Afterwards, we analyze the security properties of DBFs by applying various statistical tests. Lastly, we discuss how to apply DBFs on the public key communication protocol and compare its overhead with other popular cryptographic approaches.

4.2 A Motivational Example

A DBF is defined as a set of randomly generated Boolean functions that have two forms: $f_{compact}$ and $f_{expanded}$. Given the same inputs, both forms generate the same outputs, however, the size and calculation expense vary greatly between the two. We design these functions such that $f_{compact}$ can be computed rapidly and with only a small amount of energy, while $f_{expanded}$ can only be computed using an exponentially higher amount of energy, hardware resources, and longer time. The functions are naturally designed to enable public key communication where $f_{compact}$ is used as the private key and $f_{expanded}$ is used as the public key.

$$
\begin{gathered}
Inputs : \mathbf{A} = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\} \\
Outputs : \mathbf{C} = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7\}
\end{gathered}
$$

$$
f_{compact_1}
\begin{cases}
b_0 = a_0'a_3 + a_2a_6' & b_4 = a_3'a_4' + a_5a_7 \\
b_1 = a_1a_7 + a_4a_5 + a_1'a_4' & b_5 = a_2a_6 + a_4a_5' + a_5'a_6' \\
b_2 = a_0'a_2' + a_3a_6' & b_6 = a_3a_5' + a_2'a_7 \\
b_3 = a_4a_6 + a_1a_7 & b_7 = a_3'a_7 + a_5a_6' + a_3'a_5'
\end{cases}
\tag{4.1}
$$

$$
f_{compact_2}
\begin{cases}
c_0 = b_0'b_3 + b_2b_6' & c_4 = b_3'b_4' + b_5b_7 \\
c_1 = b_1b_7 + b_4b_5 + b_1'b_4' & c_5 = b_2b_6 + b_4b_5' + b_5'b_6' \\
c_2 = b_0'b_2' + b_3b_6' & c_6 = b_3b_5' + b_2'b_7 \\
c_3 = b_4b_6 + b_1b_7 & c_7 = b_3'b_7 + b_5b_6' + b_3'b_5'
\end{cases}
\tag{4.2}
$$

$$
f_{expanded}
\begin{cases}
c_0 = a_0a_1a_6a_7 + a_0a_1a_2'a_7 + a_0a_4a_6 + a_1a_2'a_3'a_7 + a_1a_3'a_6a_7 + a_2a_3a_5a_6' \\
\qquad + a_3a_5a_6'a_7' + a_0'a_2'a_5a_7 + a_0'a_2'a_3'a_7' + a_3'a_4a_6 \\
c_1 = a_1a_3a_4'a_7' + a_1a_5'a_6'a_7' + a_2a_5a_7 + a_2a_3'a_4'a_6 + a_3a_5a_6' + a_4a_5a_6' + a_4a_5'a_7' \\
\qquad + a_1'a_4a_5' + a_1'a_5a_6' + a_1'a_3'a_5' + a_3'a_7 \\
c_2 = a_4a_5a_6a_7' + a_0a_2'a_3' + a_1a_2a_5a_7 + a_2a_3'a_6 + a_2a_4a_5a_6 + a_0a_6 + a_3'a_4a_6a_7' \\
\qquad + a_1a_2a_3'a_7 \\
c_3 = a_1a_3'a_7 + a_4a_5a_6' + a_5a_6'a_7 + a_1'a_5a_6' + a_1'a_3'a_4'a_5' + a_3'a_5a_7 + a_2'a_5a_7 \\
c_4 = a_2a_3'a_6a_7 + a_2a_4'a_5'a_7' + a_3a_4'a_7' + a_4a_6'a_7' + a_1'a_3a_4'a_5' + a_1'a_5'a_6' + a_3'a_4a_5' \\
\qquad + a_3'a_5'a_6' \\
c_5 = a_3a_6' + a_5a_6' + a_0'a_2'a_3 + a_0'a_2'a_7 + a_2'a_5 + a_2'a_3'a_4'a_6 \\
c_6 = a_0a_3'a_7 + a_0a_3'a_5' + a_0a_3'a_6' + a_1a_5a_6'a_7 + a_1a_2'a_4'a_6a_7 + a_2a_3'a_7 + a_2a_3'a_5' \\
\qquad + a_2a_3'a_6' + a_2'a_4a_5a_6 \\
c_7 = a_2a_5a_6 + a_2a_3'a_5' + a_5a_6'a_7' + a_1'a_5a_6' + a_1'a_2'a_4'a_6 + a_1'a_3'a_6' + a_2'a_4'a_6a_7' \\
\qquad + a_3'a_5'a_7'
\end{cases}
\tag{4.3}
$$

We present a motivational example consisting of Equations 4.1, 4.2, and 4.3, which comprise a DBF. In this example, each sub-function, b_i, in Equation 4.1 is created by selecting four inputs from $\mathbf{A}$ at random, enumerating a subset of the minterms, and minimizing the resulting switching function. Each sub-function, c_i, in Equation 4.2 is created by duplicating the structure of the corresponding sub-function in Equation 4.1, selecting four b_i variables at random, and mapping them as inputs to each c_i.

By substituting Equation 4.1 into Equation 4.2, we create new equations for each c_i as functions of **A**. We minimize the switching function, resulting in $f_{expanded}$ listed in Equation 4.3. This substitution and expansion create a larger function with more operations than the original $f_{compact}$ functions even after minimization. Note that together, $f_{compact_1}$ and $f_{compact_2}$ produce the same outputs as $f_{expanded}$, but are much more compact in size than the algebraically simplified $f_{expanded}$. In this example, both $f_{compact_1}$ and $f_{compact_2}$ together require a total of 38 AND operations and 22 OR operations while $f_{expanded}$ requires 156 AND operations and 59 OR operations.

While Equations 4.1, 4.2, and 4.3 offer insight into how substitutions and expansions can create large size difference between $f_{compact}$ and $f_{expanded}$, our example only uses eight inputs and two levels of sub-functions in $f_{compact}$ for substitution (from $f_{compact_1}$ into $f_{compact_2}$). To show the impact of using more inputs and levels of sub-functions in $f_{compact}$, we test and compare $f_{compact}$ and $f_{expanded}$ of larger DBFs. We define our DBF setting for comparisons as following.

1. Both $f_{compact}$ and $f_{expanded}$ are in the form of a simplified sum of products and are compared with the total number of products. In our motivating example, $f_{compact}$ contains 38 product terms and $f_{expanded}$ contains 67 product terms.

2. Each sub-function in $f_{compact}$ has at most four input variables. This purposefully limits the size of $f_{compact}$.

3. The number of outputs is the same as the number of inputs in each sub-function level of $f_{compact}$. In the motivational example, $i \in \{0, 1, ...7\}$ for all a_i, b_i, and c_i.

4. The number of sub-function levels is set to be the half of the number of inputs.

Based on the above setting, we randomly generate a group of functions defining $f_{compact}$ and subsequently generate the corresponding $f_{expanded}$ expression.

The difference in computation complexity between $f_{compact}$ and $f_{expanded}$ can be determined by comparing their size differences in the form of their number of products. Table 4.1 shows the comparison result based on 10 instances of randomly generated DBFs. The

Sub-function levels	Inputs	Avg. # of Products ($f_{compact}$)	Avg. # of Products ($f_{expanded}$)
4	8	92.7 ± 11.2	259 ± 16
5	10	143.4 ± 15.4	766 ± 69
6	12	207.6 ± 21.2	3820 ± 250
7	14	273.5 ± 26.5	$11\,500 \pm 760$
8	16	368.9 ± 31.6	$49\,200 \pm 2700$
9	18	468.5 ± 34.8	$142\,000 \pm 7500$
10	20	578.0 ± 46.1	$370\,000 \pm 19\,000$

Table 4.1: Size comparison between $f_{compact}$ and $f_{expanded}$.

number of products in $f_{compact}$ grows polynomially as inputs and sub-function levels increase, while the number of products in $f_{expanded}$ grows exponentially. For example, when the number of inputs is 8, the average number of products in $f_{expanded}$ is approximately 3× larger than the average number of products in $f_{compact}$, while for 20 inputs, the average number of products in $f_{expanded}$ is 640× larger. By increasing the number of inputs as well as the number of sub-function levels proportionally, it is very easy to create a huge size gap between $f_{compact}$ and $f_{expanded}$.

4.3 DBF Architecture on the FPGA

We use FPGA as the platform to implement our DBF. The goal is to design a low energy, low delay, and small area implementation of $f_{compact}$ while demonstrating that there does not exist a fast and compact implementation of $f_{expanded}$, thus ensuring that it can only be simulated with a large overhead.

4.3.1 $f_{compact}$ Implementation

The configurable logic blocks (CLBs) in FPGA containing LUTs and flip-flops are the fundamental logic units we are using. For a 4-input LUT, the relation between its inputs (a_0, a_1, a_2, a_3) and output (b) can be expressed by a Boolean function $b = f(a_0, a_1, a_2, a_3)$.

Each sub-function in Equation 4.1 and Equation 4.2 is a Boolean function with four

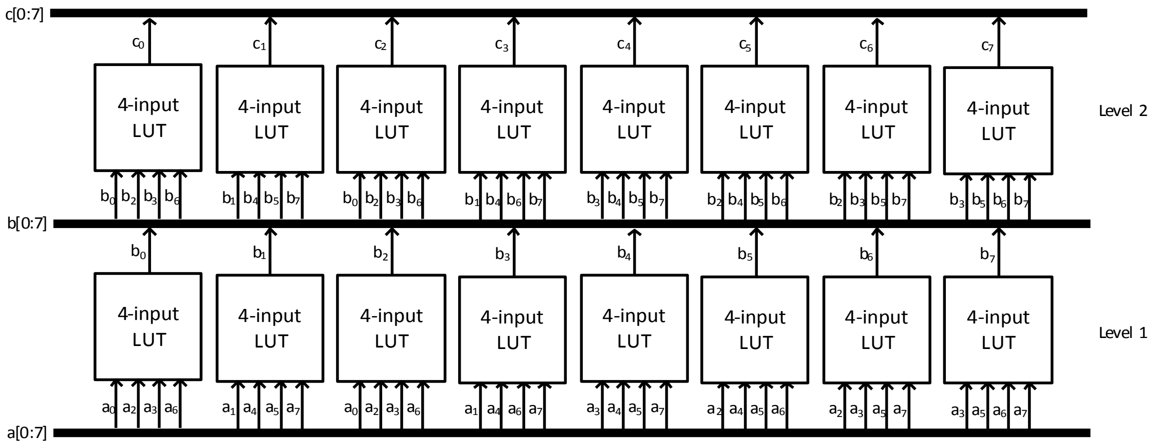


Figure 4.1: Combinational logic implementing $f_{compact_1}$ and $f_{compact_2}$.

inputs, thus it can be expressed using a LUT. Therefore, we can use eight 4-input LUTs to implement $f_{compact_1}$ and another eight to implement $f_{compact_2}$. The structure is depicted in Figure 4.1.

To extend our motivational example to a more general case, any $f_{compact}$ that consists of multiple levels of Boolean functions can be implemented using a similar LUT network. With the above architecture, each sub-function level in $f_{compact}$ requires an additional level of LUTs.

4.3.2 Synthesis Analysis of $f_{expanded}$

We show that there does not exist a fast and compact implementation of $f_{expanded}$ on FPGA. For a randomly generated $f_{compact}$, we calculate the corresponding $f_{expanded}$. Then we use the Xilinx ISE design suite to synthesize the function mappings of $f_{expanded}$ and compare the resources it requires with that of $f_{compact}$. Table 4.2 enumerates the amount of resources for $f_{compact}$ and $f_{expanded}$ under varying numbers of inputs and sub-function levels. The number of LUTs of $f_{expanded}$ increases exponentially with the linear growth of the number of inputs and the sub-function levels of $f_{compact}$.

Sub-function levels	Inputs	$f_{compact}$ (LUTs)	$f_{expanded}$ (LUTs)
4	8	32	81
5	10	50	396
6	12	72	1,620
7	14	98	4,350
8	16	128	13,600
9	18	162	31,200
10	20	200	98,300

Table 4.2: Average number of LUTs required for a DBF with particular input sizes and sub-function levels of $f_{compact}$ synthesized using the Xilinx ISE design suite.

4.3.3 Time Gap Between $f_{compact}$ and $f_{expanded}$

Because of the huge hardware resources that $f_{expanded}$ requires, implementing a system with a standard cryptographic size (e.g., 64-bit input) representing $f_{expanded}$ is unrealistic. Hence, with large DBFs, we can only simulate $f_{expanded}$. This further separates the time difference between the FPGA implementation of $f_{compact}$ and the simulation of $f_{expanded}$. Table 4.3 provides a comparison of the two. The huge time gap between the two function forms naturally enables the application of public key communication. We demonstrate the detailed protocol in Section 4.6.

Sub-function levels	Inputs	$f_{compact}$ (implemented)	$f_{expanded}$ (simulated)
4	8	29.2 ± 3.7	$(1.18 \pm 0.08) \times 10^4$
5	10	37.0 ± 4.0	$(4.33 \pm 0.49) \times 10^4$
6	12	45.1 ± 5.5	$(1.63 \pm 0.12) \times 10^5$
7	14	53.9 ± 5.5	$(5.77 \pm 0.48) \times 10^5$
8	16	61.4 ± 6.5	$(2.31 \pm 0.25) \times 10^6$
9	18	69.5 ± 7.4	$(6.75 \pm 0.49) \times 10^6$
10	20	77.2 ± 8.0	$(1.61 \pm 0.19) \times 10^7$

Table 4.3: Average execution time (measured in nanoseconds) for $f_{compact}$ and $f_{expanded}$.

4.4 DBF Design Optimization

With our basic architecture of DBFs proposed in the previous section, there still exist many detailed design questions unanswered regarding the DBFs. We list the following ones.

- Is it possible to design a sequential DBF?
- What is the desired size of a DBF?
- What is the optimal way to interconnect the LUTs of a DBF?
- How to initialize the LUT contents of a DBF?

In the following part of this section, we discuss each of the above questions. Meanwhile, we consider three major criteria when answering each question, respectively energy minimization, footprint reduction, and security enhancement. By finding the solution of each question that fits best for all the criteria, our goal is to picture an optimized DBF structure. The experimental results that are related to each question will be presented in Section 4.5.

4.4.1 DBF Size

There are two size parameters of the DBF design that need to be decided. The first is the number of input variables, and the second is the number of sub-function levels in $f_{compact}$. As for the prior, a larger number of inputs usually represent a more secure system. In the case of DBFs, the time consumption to calculate $f_{compact}$ grows polynomially with the number of inputs while the time overhead to calculate $f_{expanded}$ grows exponentially. Thus it is ideal to increase the number of inputs of a DBF because a larger gap between the two function forms is more beneficial for the design of the public key protocol.

On the other hand, it is more challenging to decide what is the desired number of sub-function levels of a DBF. Now we assume that the number of inputs of a DBF is m. If we consider the DBF as a set of Boolean functions, the maximum number of minterms of each output function is 2^m. In other words, no matter how many levels of sub-functions are applied, there is an upper bound for the size of $f_{expanded}$ to reach if m is set. As a result, after a certain point, increasing the number of sub-function levels will no longer increase the security of DBFs. The best-suited number of sub-function levels for any given m can be found out experimentally.

4.4.2 A Sequential DBF

An observation from the motivational example is that $f_{compact_1}$ and $f_{compact_2}$ employ the same function format with the only change of the variable names. Hence, there is actually no need to use two levels of LUTs to implement the $f_{compact}$. An alternative design is shown in Figure 4.2 which only requires a total of eight 4-input LUTs to implement the $f_{compact}$. In the first cycle, $f_{compact_1}$ is computed and the outputs of that cycle are mapped as inputs to the next cycle, which computes $f_{compact_2}$.

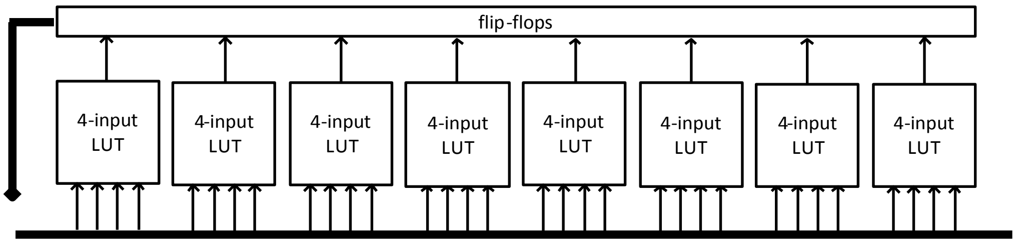


Figure 4.2: Sequential logic implementing $f_{compact}$. At initialization, input variables (e.g. a_i) are loaded into the flip-flops. The outputs of the previous cycle are mapped as inputs for the next cycle.

The sequential logic design is more advantageous than the combinational architecture in terms of the hardware footprint. For a DBF with N sub-function levels, the sequential design reduces the number of required LUTs by N times. However, a security concern of sequential DBFs is that without changing the format of functions in each level, the outputs of a sequential DBF can become more predictable and less random. We leave more quantitative security analysis to Section 4.5.

4.4.3 DBF Connections

For a DBF implemented in combinational logic as shown in Figure 4.1, a most straightforward way of interconnecting LUTs is to have all the inputs of level i LUTs from the outputs of level $i - 1$ LUTs. However, there are ways to increase the complexity of connection, e.g., feedforward and feedback structures. In the feedforward structure, the inputs of level i LUTs are from all the previous LUT levels instead of only level $i - 1$. In the feedback structure, the key idea is that the outputs from higher LUT levels (level $i + 1$ etc.) can be feedback as the inputs of previous LUT levels (level i etc.) in the next cycle of computation.

Both the feedforward and feedback structures increase the variety of DBFs without changing the area and energy cost. Consider an attacker who tries to break a DBF by brute forcefully simulating all possible LUT interconnections. With the possibility of feedforward and feedback structures, he/she has exponentially more interconnections to simulate. Thus we conclude that feedforward and feedback connections are powerful against brute force simulation attacks. We will discuss their performance against more types of attacks through experiments in the next section.

4.4.4 DBF Initialization

The Boolean functions in $f_{compact}$ are implemented using LUTs on FPGA. The way we initialize the LUT contents directly decides the functionality of the DBF. An intuitive proposal is to initialize all the LUT cells in a completely random way. However, a concern with totally random assignment is that a frequency bias of 0s and 1s may appear at the outputs of the DBF. For example, for a LUT with 6 selection bits, $2^6 = 64$ cells need to be initialized. While the perfect case for random initialization is to have half 1s and half 0s among the 64 cells, the real scenario can be far from perfect. Consider a LUT has more cells with 1s than with 0s, its output will have a larger probability to be 1 rather than 0 if we assume every cell has an equal chance to be selected. From an attacker's point of view, after collecting some historical outputs of the LUT, he/she can easily observe the existence of the frequency bias, thus he/she can now predict the LUT output correctly with a probability of more than 0.5.

The above problem can be leveraged by adding restrictions on the LUTs' initialization. Still take a 6 selection bit LUT as an example, during the initialization, if we enforce that half LUT cells need to be assigned to 1s and the other half to 0s, the frequency bias problem will no longer exist. The drawback of the above approach is that it will exponentially reduce the solution space of the LUT initialization. In other words, an attacker can try fewer times if he/she wants to brute forcefully discover the contents of the LUT.

4.5 Security Analysis

In this section, we explore the resilience of DBFs against a variety of potential statistical attacks. We first propose a few security tests we will use for the evaluation. Then we present the experimental results on DBFs. We also experiment to explore the effect of DBF structure optimization proposed in the previous section.

4.5.1 Security Tests

Confusion and diffusion are the two most important criteria for security ciphers [27]. Confusion refers that the relation between the inputs and the outputs of a security cipher should be as non-linear as possible. On the other hand, diffusion suggests that when the inputs to a security cipher change by a smallest amount (e.g., one bit), the corresponding outputs should change completely in a pseudo-random way. To test the confusion and the diffusion of DBFs, we have designed the following three standards.

4.5.1.1 Conditional Correlation

This standard represents the confusion of DBFs. The basic idea is to build a bitwise correlation model via the construction of per-bit input-output and per-bit output-output conditional probability distributions. To be more specific, by observing some historical inputs-outputs of a DBF, a potential attacker targets to build a model that calculates $P(O_i = c_1|I_j = c_2)$ and $P(O_i = c_1|O_j = c_2)$, where O_i is an output bit i, I_j is an input bit j, and c_1 and c_2 are 0 or 1. Assume that there exists some correlation between an input bit and an output bit, then attackers can predict the output bit by observing the corresponding input bit, and success with a probability of more than 0.5. The ideal secure system will have a probability of approximately 0.5 for all conditions.

4.5.1.2 Avalanche Effect

We use avalanche effect to test the diffusion of a DBF. If the avalanche effect is evident in a cryptographic system, then there is an ultra low probability that an attacker can predict any subsequent outputs using the knowledge of outputs of similar inputs. The avalanche effect can be measured by observing the hamming distance between the corresponding outputs of two inputs which differ by a minimal amount. In the case of our DBF, the minimal amount is one bit. For a cipher with perfect diffusion, the distribution of output hamming distance should form a normal distribution.

4.5.1.3 Frequency Prediction

A special case of the conditional correlation test is not to consider the conditions, but only to calculate the probability that an output bit is 1 or 0. A potential attacker can build a model to predict $P(O_i = c)$, where $c = 0$ or 1. A frequency biased output bit can be dangerous as an attacker can extract a part of output information with a high probability purely by guessing the more frequent case. An ideal secure system will produce each output bit as 0 or 1 with an equal chance.

4.5.2 Test Results on Standard DBFs

We present the security test results on a standard DBF. The setting of a standard DBF is defined as following. The DBF has 64 inputs and 64 outputs. All the LUT contents are randomly initialized without controlling the frequency of 1s and 0s. It is a combinational DBF with 32 levels of LUTs neither using feedforward nor feedback structures. The interconnection between LUTs is assigned in a completely random way.

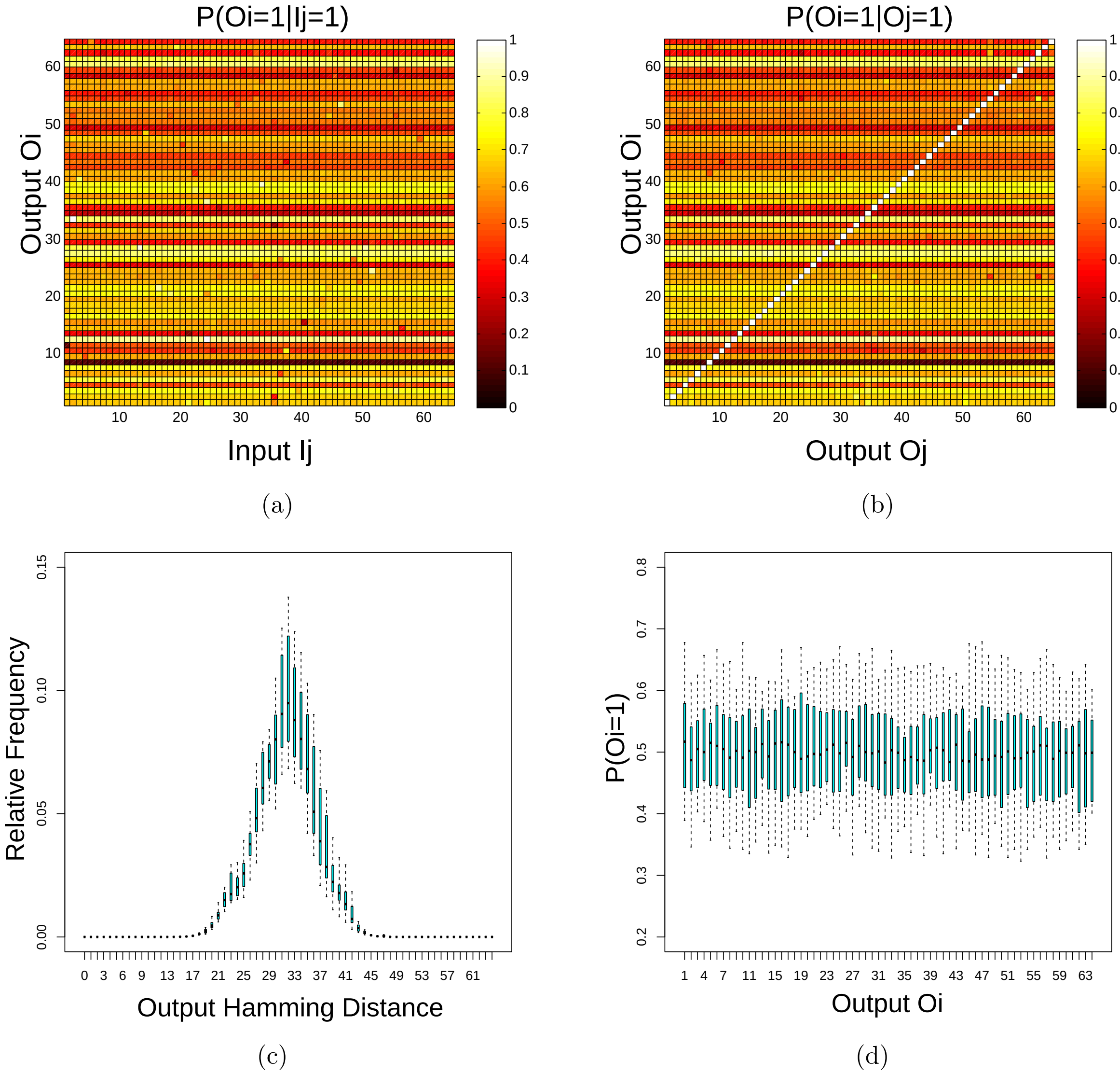
We start with the test of conditional probabilities. Figures 4.3a and 4.3b depict the $P(O_i = 1|I_j = 1)$ and $P(O_i = 1|O_j = 1)$ for a single DBF instance. The first observation is that most of the probabilities stay close to 0.5. Secondly, we can clearly see the pattern of horizontal lines with similar colors in the colormap, which suggests that each output bit does not have a particularly strong correlation with any input bits or the other output bits.

The avalanche effect is measured with the hamming distance between output vectors when changing one bit of the input vector. Ideally, the distribution of output differences should form a normal distribution with the peak centered over 32. The avalanche effect result of DBFs is depicted in Figure 4.3c. It is the average result over 100 randomly generated standard DBFs. Each DBF is tested with 100,000 pairs of randomly generated input vectors. The curve in the figure shows a nearly perfect normal distribution, indicating that our DBF has an excellent property of diffusion.

Lastly, Figure 4.3d presents our investigation of the output frequency. We can see that each output bit has a frequency centered close to 0.5, indicating a low probability of a successful attack via frequency prediction. The error bars also suggest that for a single DBF instance, some output bits indeed have frequency bias due to the random initialization on the DBF LUTs.

4.5.3 Comparisons of DBF Optimizations

We aim to experimentally analyze the performance of different DBF optimization techniques. We consider the 3 security tests listed in section 4.5.1. To facilitate presenting and comparing results, instead of plotting figures, we choose to calculate a quantitative result value for each test. The calculated value should serve as an indicator of the performance on the test. Our selected presenting value for each test is shown as below.



(a)

(b)

(c)

(d)

Figure 4.3: (a) Conditional probability $P(O_i = 1|I_j = 1)$ for a single DBF instance. (b) Conditional probability $P(O_i = 1|O_j = 1)$ for a single DBF instance. (c) Distribution of output hamming distance when changing the input by one bit. Error bars represent maximum, 75th percentile, mean, 25th percentile, and min. (d) Probability that an output bit equals to 1.

- Conditional correlation (input-output): average pair-wise correlation of $P(O_i = 1|I_j = 1)$. Ideal case: 0.5.

- Conditional correlation (output-output): average pair-wise correlation of $P(O_i = 1|O_j = 1)(i \neq j)$. Ideal case: 0.5.

- Avalanche effect: average output hamming distance when flipping 1 bit of the input vector. Ideal case: 32.

- Frequency prediction: average output frequency of $P(O_i = 1)$. Ideal case: 0.5.

Each value is tested on 100 randomly generated DBF instances. We have calculated the average value of each test together with the standard deviation (the values presented in the table are the average of average). We consider 4 optimized DBF derivatives respectively sequential DBF, feedforward DBF, feedback DBF, and equal 1s/0s DBF. To facilitate the comparison, we only change one DBF property per derivative. For example, the only change from the standard DBF to the sequential DBF is the switch from the combinational structure to the sequential structure. Meanwhile, all the rest properties of the standard DBF remain unchanged in the sequential DBF. The results are shown in Table 4.4 (see the last page of this chapter). Note that the sub-function levels for sequential DBF refer the number of cycles as the sequential DBF only has a single level of LUTs. For the equal 1s/0s DBF, we restrict the initialization of LUTs in such a way that each LUT in the DBF has an equal number of 1s and 0s in its cells. But the positions of 1s and 0s are randomly assigned. We have the following observations based on Table 4.4.

First of all, increasing the number of sub-function levels in the standard DBF can improve avalanche effect. However, the improvement becomes less significant as the number of sub-function levels increases. For example, from 4 levels to 8 levels, the average hamming distance increases from 13.9 to 25.1. Meanwhile, from 16 levels to 32 levels, the average hamming distance only increases from 28.4 to 29.1.

Secondly, the sequential DBF has a very similar security performance compared to the standard DBF. In other words, the change from the combinational structure to the sequential structure does not compromise the DBF security.

Thirdly, the feedforward and feedback structures improve the DBF security in terms of the avalanche effect. Especially, with the feedback structure, the average hamming distance of avalanche effect reaches almost the ideal value 32.

Lastly, although restricting an equal number of 1s and 0s in LUTs comprises the solution space, the equal 0s/1s DBF derivative indeed has the best performance regarding the conditional correlation and the output frequency prediction. The standard deviations in both tests are the smallest, which indicates equal 1s/0s DBF are the most consistent against conditional attacks and frequency attacks.

4.6 Public Key Communication

In this section we present the DBF based public key communication protocol and analyze its performance. We present the following definitions for use in our discussion:

- K_{priv}: the private key represented by $f_{compact}$.
- K_{pub}: the public key represented by $f_{expanded}$.
- Alice: the owner of K_{priv}.
- Bob: the communicating party.
- TTP: the trusted third party that administrates K_{pub}.

Protocol 1 presents the flow of public key communication. K_{priv} exists only as an implementation of $f_{compact}$ on an FPGA from which $f_{expanded}$ is constructed. K_{pub} resides with the TTP and is stored in two forms: as a set of sum of products (SOP), and as a set of product of sums (POS). Assume that K_{pub} consists of sub-functions f_i, where $i \in \{1, 2, ..., t\}$. When a party, such as Bob, requests access to K_{pub} he can request individual product terms from the SOP representation or individual sum terms from the POS representation. With these products and sums, he constructs the binary vectors to encrypt his message.

We have two reasons to use the TTP to store K_{pub}. The first is that $f_{expanded}$ as the K_{pub} is huge in size which requires a considerable amount of storage. Meanwhile, to communicate with Alice, Bob only needs a very small portion of $f_{expanded}$. TTP significantly reduces Bob's

Protocol 1 Public Key Communication

```
for $j \in \{1, 2, ..., N\}$ do
    Bob chooses a random binary vector, $\mathbf{r}$, of length $t$.
    $\mathbf{R_j} = \mathbf{r}$
    for $i \in \{1, 2, ..., t\}$ do
        if $r_i == 1$ then
            Bob selects one product at random from the SOP form of $f_i$ in $K_{pub}$.
            Using the selected product, Bob generates a binary input vector $\mathbf{p_i}$ such that $r_i = f_i(\mathbf{p_i}) = 1$.
        else if $r_i == 0$ then
            Bob selects one sum at random from the POS form of $f_i$ in $K_{pub}$.
            Using the selected sum, Bob generates a binary input vector $\mathbf{p_i}$ such that $r_i = f_i(\mathbf{p_i}) = 0$.
        end if
        Bob flattens and appends $\mathbf{p_i}$ onto $\mathbf{P_j}$ such that $\mathbf{P_j} = \mathbf{p_1 p_2 ... p_i}$.
    end for
end for
Bob calculates $E = m \oplus \mathbf{R_1} \oplus \mathbf{R_2} \oplus ... \oplus \mathbf{R_N}$, where $m$ is the message Bob wishes to transmit securely.
Bob broadcasts $E$ and all $\mathbf{P_j}, j \in \{1, 2, ..., N\}$.
for $j \in \{1, 2, ..., N\}$ do
    for $\mathbf{p_i}$ in $\mathbf{P_j}$ do
        Alice computes $r_i = f_i(\mathbf{p_i})$ using $K_{priv}$.
        Alice appends $r_i$ onto $\mathbf{R_j}$ such that $\mathbf{R_j} = r_1 r_2 ... r_i$.
    end for
end for
Alice computes $m = E \oplus \mathbf{R_1} \oplus \mathbf{R_2} \oplus ... \oplus \mathbf{R_N}$.
```

memory footprint requirements. The second reason is that attackers need the whole $f_{expanded}$ to simulate and decrypt the message. However, with TTP, no other party owns a complete copy of $f_{expanded}$. It will create the difficulty for attackers to collect all the pieces of $f_{expanded}$.

4.6.1 Execution Time Gap

Protocol 1 takes advantage of the calculation time difference between $f_{compact}$ and $f_{expanded}$. Because K_{pub} is so large, only the owner of the physical device, K_{priv}, can calculate $\mathbf{R_j}, j \in \{1, 2, ..., N\}$ given E and $\mathbf{P_j}, j \in \{1, 2, ..., N\}$, in a short amount of time. In order to successfully attack this protocol, an attacker would need to simulate the sub-functions in K_{pub} in order to compute $r_i, i \in \{1, 2, ..., t\}$ for each $\mathbf{R_j}, j \in \{1, 2, ..., N\}$. However, as previously discussed in Section 4.3.3, the calculation of $f_{expanded}$ via simulation takes an exponentially longer time than calculation of $f_{compact}$ via implementation.

We estimate the decryption time gap between the private key holder and the attacker. Suppose that the private key holder uses a sequential DBF with 64 inputs and 32 cycles to implement $f_{compact}$ and the attacker simulates the corresponding $f_{expanded}$. Since the full simulation of $f_{expanded}$ is too expensive to execute, we estimate its timing overhead by extrapolating on our experimental simulations times in Table 4.3. We fit an exponential model to the lower bounds of our experimental results with an R^2 value of 0.9961. Using this model, we compute the estimated simulation time of $f_{expanded}$ to be approximately $1.16 \times 10^{19} ns$, which corresponds to about 370 years. Meanwhile, the implementation of $f_{compact}$ only takes approximately $239ns$.

The previous estimation assumes that the simulation is executed sequentially. In order to combat the possibility that an attacker utilizes parallel computation to reduce the total time, we increase the number of rounds, N. If we increase the rounds to 10^3, this changes the private key calculation time to $239 \times 10^3 = 2.39 \times 10^5 ns$, while an attacker must now attempt to parallelize 370×10^3 years worth of computation.

4.6.2 Performance Comparison

We measure the overall performance of the DBF by evaluating and comparing its speed and energy consumption against other existing and similar cryptographic systems. Table 4.5 compares the DBF against traditional hardware-based block ciphers (private key) and hardware-based implementations of RSA (public key). In the table, the overhead is presented on the side of the private key holder who uses the physical implementation of $f_{compact}$ to decrypt the message. The encryption party's overhead is almost negligible as the only thing he/she has to do is to select and create N binary vectors. The energy result of sequential DBF is retrieved using Xilinx power estimator (XPE). Based on the table, the DBF is orders of magnitude more compact in area, faster in speed, and more energy efficient than RSA (and is even competitive to the energy efficiency of secret key block ciphers).

4.7 Summary

We have developed and evaluated the DBF, an ultra low energy cryptographic primitive that enables public key communication. The proposed protocol requires low energy consumption for all legitimately involved parties. The primitive is composed of two forms of Boolean functions, one is created in a compact format and has a low-energy FPGA implementation while the other is ultra complex for both implementation and simulation. Our experimental result concludes that the DBF exhibits excellent statistical properties as well as a resilience to a variety of security attacks. A comprehensive performance evaluation further indicates that the energy consumption, area footprint, and speed of DBFs outmatch current public key cryptographic hardware implementations.

DBF settings			Correlation	Correlation	Avalanche criterion	Frequency
Design	Inputs	Sub-function levels	(input-output)	(output-output)	(hamming distance)	
Standard DBFs	64	4	0.50±0.09	0.50±0.10	13.9±1.6	0.50±0.08
Standard DBFs	64	8	0.50±0.10	0.50±0.10	25.1±1.5	0.50±0.09
Standard DBFs	64	16	0.50±0.08	0.50±0.09	28.4±1.1	0.50±0.07
Standard DBFs	64	32	0.50±0.08	0.50±0.08	29.1±1.3	0.50±0.08
Sequential DBFs	64	32(cycles)	0.50±0.09	0.50±0.08	28.8±1.5	0.50±0.09
Feedforward DBFs	64	32	0.50±0.09	0.50±0.07	29.9±1.4	0.50±0.09
Feedback DBFs	64	32	0.50±0.08	0.50±0.08	**31.1±1.2**	0.50±0.07
Equal 1s/0s DBFs	64	32	**0.50±0.03**	**0.50±0.02**	29.0±1.3	**0.50±0.02**

Table 4.4: Security comparisons of different DBF optimizations.

Design	Flip-flops	LUTS	Area (slices)	Max. cycle delay (ns)	Cycles	Energy (μJ)	Block size	Throughput at f_{max} (Mbps)	Device
Present [23]	114	159	117	8.78	256	3.16×10^{-3}	64	28.46	xc3s50-5
HIGHT [23]	25	132	91	6.12	160	1.07×10^{-3}	64	65.48	xc3s50-5
AES [23]	338	531	393	14.21	534	3.58×10^{-2}	128	16.86	xc3s50-5
RSA [25]	1870	2811	1553	7.62×10^3	907	128.80	-	0.15	xc3s500e
RSA radix-2 [25]	7564	11496	6282	8.21×10^3	1058	654.80	-	0.12	xc2v6000
RSA radix-4 [25]	9944	14907	8328	4.23×10^3	560	236.73	-	0.43	xc2v6000
Sequential DBF	64	64	32	239.04	1000	$9.18{\times}10^{-2}$	64	267.74	xc6slx45

Table 4.5: Comparing the DBF with traditional block ciphers and RSA.

CHAPTER 5

Ultra-low Energy Private Key Communication using Digital Bidirectional Function

5.1 Motivation and Problem Formulation

While the digital bimodal function has addressed the public key communication, an even commonly used security protocol among IoT systems is private key communication. In the previous chapters, we have proposed PUF emulation and PUF matching to enable private key communication. However, the system is analog thus has the problem of being unstable against environmental variations. Consequently, the PUF matching accuracy can never reach 100%, which is often required by private key communication. In this chapter, we further propose a new type of security primitive purely on the digital level, the digital bidirectional function (DBidirF) which addresses private key communication with even lower power, lower bandwidth, and higher stability.

The core idea of DBidirF is to use FPGA-based structure to produce a pair of functions, respectively $f_{original}$, and $f_{inverse}$. As the name suggested, the two functions have completely inverse mappings. Assume that $\mathbf{x}$ and $\mathbf{y}$ are two n-bit vectors, the mappings realized by $f_{original}$ and $f_{inverse}$ can be defined in Equation 5.1.

$$\begin{aligned} f_{original} &: \mathbf{x} \rightarrow \mathbf{y} \\ f_{inverse} &: \mathbf{y} \rightarrow \mathbf{x} \end{aligned} \tag{5.1}$$

In order to guarantee the mappings are invertible, the **x** to **y** mapping must be a one-to-one mapping. The major difficulty of designing a one-to-one mapping system in hardware is to guarantee the scalability and the flexibility of the system. To be more specific, the scalability means that the system needs to be invertible even when the number of inputs and outputs is large. Because in such cases, it has been impossible to elaborate the whole mapping. As for flexibility, it denotes that the mapping implemented by the system needs to be easily reconfigured.

We meet both requirements by using the FPGA-based hierarchical LUT networks. For each level of LUT network in $f_{original}$, we allocate the LUT contents to guarantee that the inputs and outputs form a one-to-one mapping. Meanwhile, we use another LUT network to implement the inverse one-to-one mapping for $f_{inverse}$. We also use invertible multiplexer/demultiplexer based interstage shuffling to increase the randomness of outputs. On the top of this, we connect small LUT networks of one-to-one mapping both in parallel and in series to build large LUT networks to achieve scalability while still maintaining the property of one-to-one mapping. In terms of flexibility, we directly take advantage of the reconfigurability of the LUTs on FPGA, so that the DBidirF can be reconfigured every time before use.

The technical goal of this chapter is to propose DBidirF as a new type of low-power hardware security primitive that is specialized for private key communication. Compared to the traditional cryptographic ciphers, DBidirF has the advantage of low-power, low-area, and high-speed. E.g, using our proposed structure of LUT networks, the encryption/decryption only requires one clock cycle computation. Compared to the traditional analog PUFs, DBidirF resolves the problem of instability by completely operating in the digital domain. It utilizes the digital logic functions to build the inputs-outputs mapping. Consequently, it is resilient to the environmental and operational variations. In terms of security protocols, DBidirF requires completely no additional assistant information to facilitate secure message transfer. For example, a random seed is required in the encryption using traditional PUFs, but no longer required when using the DBidirF.

5.2 A Motivational Example

A prerequisite of the DBidirF is that the mapping from **x** to **y** must be a one-to-one mapping. Correspondingly, the mapping from **y** to **x** will also be a one-to-one mapping. Our solution to build such a mapping is to use hierarchical LUT connections.

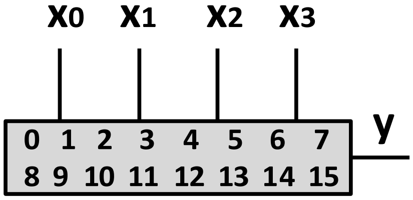
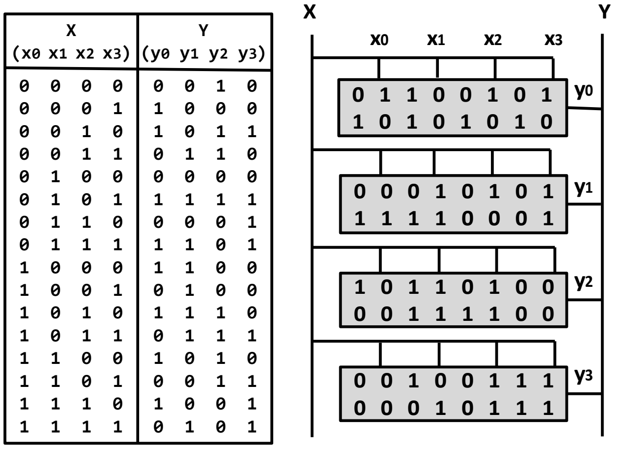


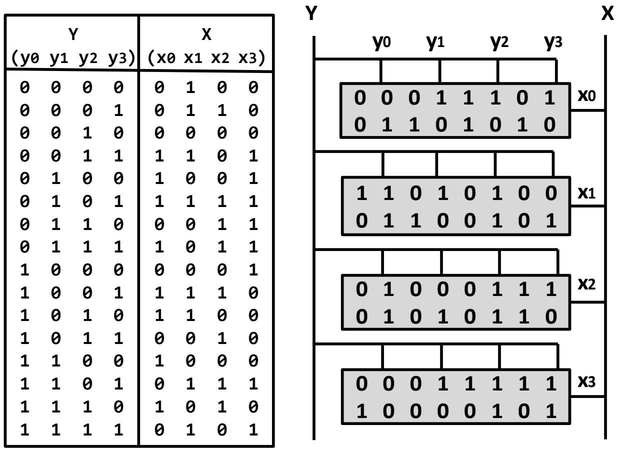
Figure 5.1: The memory location of a 4-input LUT.



X				Y			
(x0	x1	x2	x3)	(y0	y1	y2	y3)
0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	0
0	0	1	0	1	0	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	1
0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	1
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	1	1	1	0
1	0	1	1	0	1	1	1
1	1	0	0	1	0	1	0
1	1	0	1	0	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	0	1	0	1

Figure 5.2: The mapping and the LUT implementation for $f_{original}$: $\mathbf{x} \rightarrow \mathbf{y}$.

We use a motivational example to explain our design. We consider the mapping between two 4-bit vectors. Given the mapping of $f_{original}$, we use four 4-input LUTs for implementation as shown in Figure 5.2. The memory location of each LUT cell is depicted in Figure 5.1 .We use $x_0x_1x_2x_3$ as the inputs for each LUT, then based on the mapping, we allocate values to each memory location on the LUTs to implement the mapping. For example, in the mapping shown on the left side of Figure 5.2, when given the inputs as 1000, the corresponding outputs are 1100, thus, we assign $1, 1, 0, 0$ to the memory location 8 of the 4



Y				X			
(y0	y1	y2	y3)	(x0	x1	x2	x3)
0	0	0	0	0	1	0	0
0	0	0	1	0	1	1	0
0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	1
0	1	0	0	1	0	0	1
0	1	0	1	1	1	1	1
0	1	1	0	0	0	1	1
0	1	1	1	1	0	1	1
1	0	0	0	0	0	0	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	0	0	1	0
1	1	0	0	1	0	0	0
1	1	0	1	0	1	1	1
1	1	1	0	1	0	1	0
1	1	1	1	0	1	0	1

Figure 5.3: The mapping and the LUT implementation for $f_{inverse}$: $\mathbf{y} \rightarrow \mathbf{x}$.

LUTs separately. By repeatedly filling all the memory locations of the LUTs, a specified one-to-one mapping can be implemented. For $f_{inverse}$, exactly the same procedures can be followed. The mapping and the LUT implementation for $f_{inverse}$ can be found in Figure 5.3.

There exist many derivatives of the structure. For example, for each individual LUT, the order of the inputs does not have to be fixed. In the motivational example, instead of using $x_0x_1x_2x_3$ as the inputs for all of the LUTs, we can switch the inputs to be any combination of $x_0x_1x_2x_3$, e.g., $x_2x_0x_1x_3$. Meanwhile, the LUT locations to fill in the values need to be adjusted because the positions have switched. As long as the derivatives still keep the property of one-to-one mapping, they can be applied.

5.3 Architecture

5.3.1 Global Architecture

Combining the proposals from the motivational example, Figure 5.4 depicts the global architecture of a DBidirF. Note that the $f_{original}$ and the $f_{inverse}$ own separate pieces of the structures, realizing two mappings of opposite directions. The system in Figure 5.4 has n inputs/outputs and m levels of k-input LUTs. Each level of LUTs implements a one-to-one mapping using the approach described in the motivational example. Between levels of

LUTs, the outputs of previous level LUTs are fed as the inputs to the next level LUTs after interstage shuffling. The interstage shuffling introduces more randomness to the system. The design detail of the shuffling will be explained in the next subsection. Both the LUT contents and the LUT connections can be customized by the users as long as the structure generates a one-to-one mapping. From the functional perspective, this architecture implements a mapping between two n-bit vectors. In order to build inverse mappings for $f_{original}$ and $f_{inverse}$, assume that the ith level LUTs in $f_{original}$ implement x to y mapping, then the y to x mapping should be implemented by the $(n+1-i)th$ level LUTs in $f_{inverse}$. Since each level of LUTs in $f_{original}$ has a corresponding level of LUTs in $f_{inverse}$ at symmetric position implementing the inverse mapping, the overall architecture of $f_{original}$ and $f_{inverse}$ also forms inverse mappings.

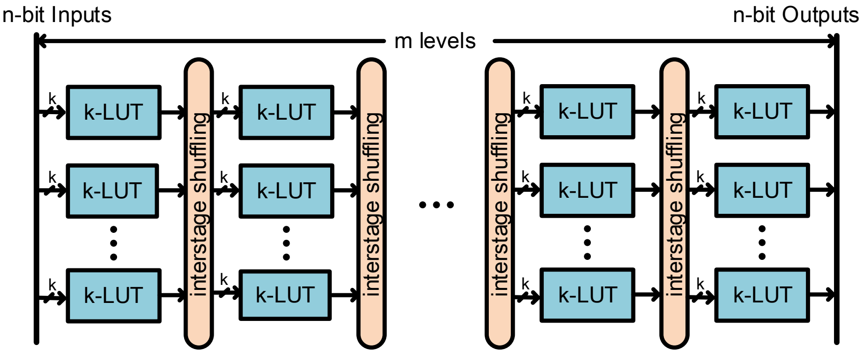


Figure 5.4: The architecture of digital bidirectional function.

5.3.2 Interstage Shuffling

The interstage shuffling is implemented using multiplexers and demultiplexers. Figure 5.5 shows an example of a 4-bit shuffling network. On the left side, it depicts a single shuffling network implemented using multiplexers for $f_{original}$. Each multiplexer takes 4-bit data inputs a, b, c, d. By configuring the selection bits of the 4 multiplexers as $10, 11, 01, 00$, the outputs of the 4 multiplexers will be shuffled as c, d, b, a. Meanwhile, the demultiplexer network on the right side is for $f_{inverse}$ which takes in the selection bits in the same order and implements an inverse shuffling from c, d, b, a to a, b, c, d. To get rid of the $0s$ in the demultiplexer outputs, one way is to OR the ith output of each demultiplexer, thus only a, b, c, d will be shown as the final outputs.

Since the inputs and the outputs must form a one-to-one mapping, no duplicated selection bits are allowed in the network. The reason that we choose multiplexers and demultiplexers to build the shuffling network is because of the inverse property between them, they can naturally generate inverse mappings without designing additional logic. Note that the structure has low delay and area overhead. The delay is equal to a single multiplexer/demultiplexer delay and the area of the network only grows linearly with the number of inputs.

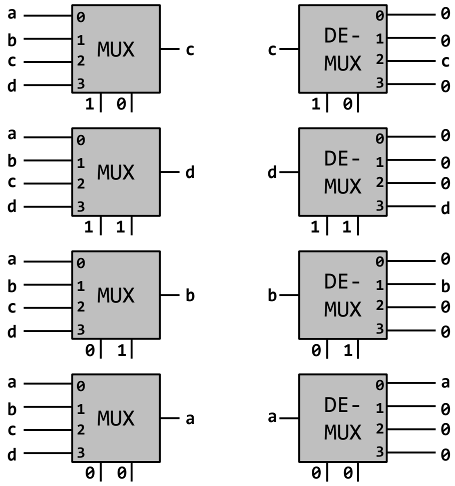


Figure 5.5: The multiplexer/demultiplexer based interstage shuffling network.

5.3.3 Scalability and Flexibility

The scalability of the DBidirF design is a vital criterion. The motivational example above shows a 4-bit inputs-outputs DBidirF. A key question is how to build a large-scale DBidirF from there, e.g., to implement a DBidirF with a 64-bit inputs-outputs mapping. Our idea is to connect the small scale mapping network both in parallel and in series. As for the parallel connection, we can increase the number of LUTs as well as the number of inputs. For instance, we can duplicate the structure in the motivational example with a new set of 4-bit inputs-outputs LUTs. If we put them in parallel, the system will become an 8-inputs-outputs system composed of 8 4-input LUTs. However, only using one level of LUTs can lead the structure to be easy to break. Therefore, to increase the complexity to our system, we connect the LUTs in series to form a hierarchical structure. The structure can be formed by connecting the outputs of previous level LUTs as the inputs of next level LUTs.

The flexibility of DBidirF refers that the structure should be easily reconfigured. It is solved through two aspects. The first is that the contents of the LUTs in the DBidirF can be easily reconfigured as long as the two functions in DBidirF form a bidirectional mapping. The second is due to the fact that the interstage shuffling can be customized by the DBidirF users. Thus, different users will have their own flexibility to build their unique mappings of DBidirF.

5.4 Protocols

Private key communication is one of the most commonly used protocols in cryptography. We propose the DBidirF-based private key communication in this section and compare it with the protocol based on traditional ciphers.

Protocol 2 Private Key Communication - DBidirF

```
1: Alice has $f_{original}$, Bob has $f_{inverse}$.
2: Alice wants to send a message $m$ to Bob.
3: Alice calculates $n = f_{original}(m)$.
4: Alice sends $n$ to Bob.
5: Bob calculates $m = f_{inverse}(n)$.
```

Protocol 3 Private Key Communication - traditional ciphers

```
1: Both Alice and Bob have $f$.
2: Alice wants to send a message $m$ to Bob.
3: Alice generates a random seed: $s$.
4: Alice calculates $S = f(s)$, $R = m \oplus S$.
5: Alice sends $s$ and $R$ to Bob.
6: Bob calculates $m = f(s) \oplus R$.
```

Protocol 2 shows the steps of communication using the DBidirF. Note that in the protocol, before communication, Alice and Bob need to split the DBidirF to own the $f_{original}$ and the $f_{inverse}$ separately. We also show the traditional cipher based protocol using a single mapping function f in Protocol 3. In comparison, the DBidirF-based protocol has the following two advantages. Firstly, both the encryption and the decryption parties are to use either the $f_{original}$ or the $f_{inverse}$ for only one-time calculation. Meanwhile, in the traditional cipher based protocol, it requires extra XOR operations. The second advantage is that the DBidirF saves the bandwidth in the message transferring. In Protocol 2, the required bandwidth equals to the length of $n = f_{original}(m)$, because of one-to-one mapping, it equals to the length of message m. However, in Protocol 3, the required bandwidth is $length(s)+length(R)$ which is around two times of the required bandwidth in Protocol 2.

5.5 Summary

In this chapter, we have proposed another hardware security primitive that is specially designed for private key communication: digital bidirectional function (DBidirF). The core idea of DBidirF is to implement a pair of functions that forms bijective mappings. We have proposed the architecture of DBidirF using LUTs on FPGA. The DBidirF based private key communication requires less computational efforts and only half of the bandwidth compared to the traditional secure ciphers.

CHAPTER 6

Digital PUFs Initialized with Analog PUFs

6.1 Motivation and Problem Formulation

In the previous two chapters, we separately discussed the digital bimodal function and the digital bidirectional function to enable public and private key communication. Both designs are proven to be compact and energy-efficient compared to traditional cryptography. However, when compared to standard PUFs, the advantages and disadvantages are both significant. As the name suggested, the advantage of the digital design is that everything is operated on the digital level, thus is completely stable when the temperature and the voltage change. The disadvantage is equally obvious, the digital logic can be easily replicated by malicious parties, thus it is no longer unclonable.

Oftentimes IoT devices are exposed to the open environment, thus vulnerable to malicious physical access. The property of unclonability is a valid approach to prevent the device or the data stored from being duplicated. Therefore, a security primitive that is both digital and unclonable is desired. In other words, we need to design a digital PUF. It must be stable in the same sense that digital logic is stable against environmental and operational variations and must produce deterministic outputs for all input vectors. The digital PUF must integrate with existing combinational logic without requiring additional clock cycles or logic to use its outputs. And lastly, the digital PUF must be flexible in the sense that its structure can be altered for different tradeoffs between security, energy, and delay as required by the pertinent task.

We demonstrate a novel type of digital PUF built on FPGA in this chapter. It is composed of two parts, the digital logic part and the analog PUF part. The core idea is to use the responses of the analog PUF to initialize the LUT contents of the digital logic so that the digital logic obtains unclonability. Then the users can directly use the initialized digital logic to encrypt and decrypt messages as needed. A high-level abstraction of the proposed digital PUF is depicted in Figure 6.1.

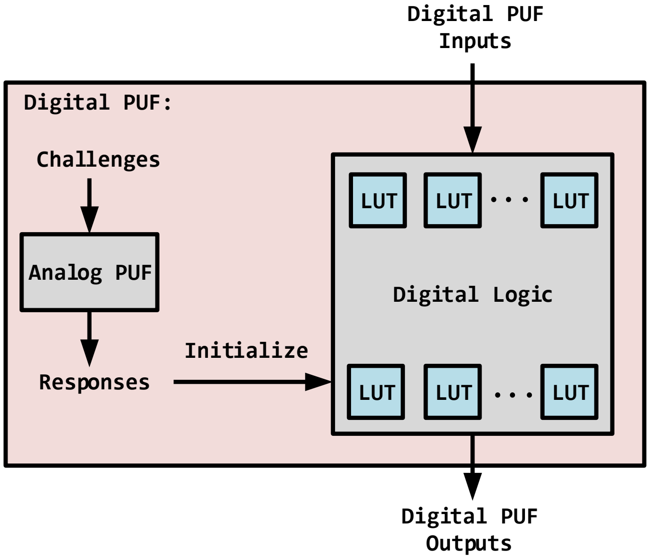


Figure 6.1: A high level abstraction of digital PUF.

Before diving into more details about the design and properties of digital PUFs, we organize the content of this chapter with two major questions to answer.

- How does a digital PUF work?
- Why is the design digital and unclonable?

With the first question, we aim to demonstrate the detailed architecture of the digital PUF and the operations needed to use the digital PUF in security applications. The second question is asked to confirm that the structure is indeed a “digital and physical unclonable function”. We organize the following sections by presenting answers to the above questions.

6.2 Architecture and Operations

A digital PUF is the combination of the analog PUF and the digital logic. In terms of the analog PUF, it can be any type of standard analog PUF that can create an analog response under a given challenge. The analog responses need to be then converted to digital signals (0s and 1s) so that to be used for the initialization of digital logic. For example, arbiter PUFs as a typical type of PUF can be used in the design by using arbiters to convert analog signals to digital signals.

In terms of the digital logic portion of the design, it also has the freedom of choosing different digital functions as long as they can be built upon LUTs on the FPGA. Considering digital PUFs are to be used for security tasks, functions such as the AES block cipher, the RSA design, and the $f_{compact}$ of DBF are preferred.

The operational flow of a digital PUF is stated as following. Upon power up, the user needs to feed challenges to the analog PUFs to generate responses. Then the retrieved responses are used to initialize the LUT cells of the digital logic. On a FPGA device, assume that the analog PUF is also embedded on the same device, such initialization occurs automatically during the FPGA's configuration phase as the initialization vectors are embedded in the FPGA's configuration bitstream. Afterwards, the digital logic is free to use according to the predetermined functionality.

We assume that attackers can not break into the FPGA device to read the bitstream, so the only information that can be potentially exposed to attackers is the user defined challenges for the analog PUF. However, only by knowing the challenges, attackers have no way to figure out the analog PUF responses since the PUF itself is not duplicable. Hence the digital logic initialized with the PUF responses remains a black box to the attackers.

One possible attack is that attackers can reverse engineer the digital logic by observing the digital PUF inputs and outputs. This issue can be easily leveraged by reinitializing the content of the digital logic every time before use. For example, if the digital logic implements the LUT network of a DBidirF, then the LUT contents can be randomly assigned as long as the network generates a one-to-one mapping. With the reconfiguration, the model reverse engineered from the previous digital logic will no longer be valid for the next time use.

The above operation process assumes that users should know the expected analog PUF response given a challenge so that they are able to configure the digital logic with desired initialization. This requirement can be met through the PUF characterization. After embedding an analog PUF on the FPGA, the manufacturers should characterize the PUF to retrieve a PUF model and share it with users. Depending on the statistical model, users can generate the challenge-response mapping of the PUF.

A critical issue exists with the above process. An analog PUF can be highly unstable thus the challenge-response mapping from the PUF model can be inaccurate. In fact, a large portion of the mapping will change dramatically with even a small variation of temperature and voltage, making the initialization process difficult to control. We discuss and solve this problem in the next chapter by taking a close look at the PUF stability.

6.3 PUF Stability

6.3.1 Stable Challenge Response Pairs (CRPs)

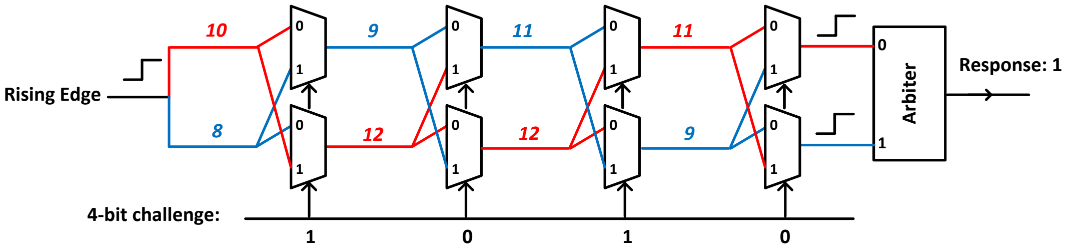


Figure 6.2: A 4-bit arbiter PUF with challenge 1010.

We take the arbiter PUF as an example. The starting point is from the key observation that a portion of challenges for an arbiter PUF can bring about more stability than other input challenges. Figure 6.2 shows an example of an arbiter PUF with a 4-bit challenge. If we use 1010 as the 4-bit challenge, the delay difference between the two paths is maximized. The path in red corresponds to the element (can be a transistor, a LUT etc.) with a larger delay at each stage, while the path in blue is the opposite. Now suppose the temperature increases, the delay of each element will also increase. However, since the challenge 1010 already provides a large difference in path delay, even though the delay increases, there is a very high probability that the red path will still have a larger delay in comparison to the blue path and, hence, the PUF response is stable. We define such challenges which can stand variations in environmental conditions as stable challenges.

We use delay ratio, as defined in Equation 6.1, to evaluate the relative delay difference between the two PUF paths. In the following test, we assume that the element delays of the PUF follow a normal distribution due to process variation.

$$Delay\ Ratio = \frac{Delay_{p1} - Delay_{p2}}{min(Delay_{p1}, Delay_{p2})} \tag{6.1}$$

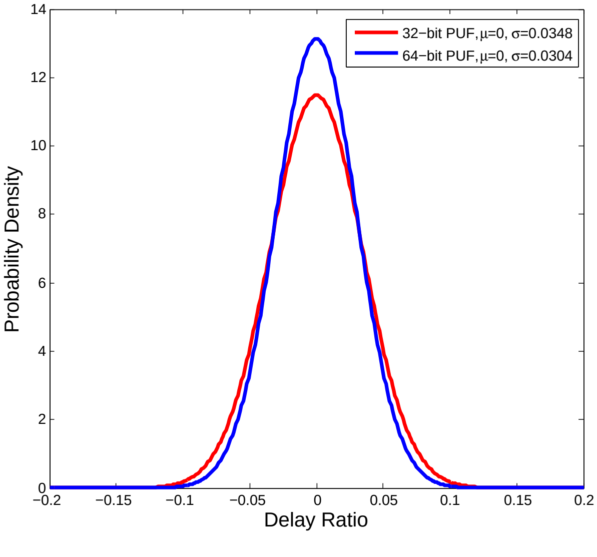


Figure 6.3: Distributions of delay ratios for a 32-bit PUF and a 64-bit PUF.

The distribution of the delay ratio for random challenges on a FPGA implemented 32-bit arbiter PUF and a 64-bit arbiter PUF are depicted in Figure 6.3. In both cases, they follow

	P(DR≥0.04)	P(DR≥0.06)	P(DR≥0.08)	P(DR≥0.1)
32-bit PUF	12.51%	4.27%	1.07%	0.21%
64-bit PUF	9.34%	2.44%	0.43%	0.05%

Table 6.1: Probability that the delay ratio (DR) is larger than threshold values for a 32-bit PUF and a 64-bit PUF.

Delay Ratio (Original T=300K)	0.04	0.05	0.06	0.07	0.08	0.09	0.1
32-bit: prob. to stay @ T=250K	0.979	0.987	0.994	0.997	1	1	1
32-bit: prob. to stay @ T=350K	0.969	0.975	0.989	0.994	0.997	1	1
32-bit: prob. to stay @ T=400K	0.937	0.951	0.959	0.977	0.988	0.996	1
64-bit: prob. to stay @ T=250K	0.984	0.986	0.996	0.998	1	1	1
64-bit: prob. to stay @ T=350K	0.982	0.986	0.993	0.998	1	1	1
64-bit: prob. to stay @ T=400K	0.954	0.974	0.986	0.991	0.997	1	1

Table 6.2: Probability that the output of a 32-bit PUF and a 64-bit PUF is stable over varying temperature conditions for different original delay ratios. Assume the original temperature is 300K, we test under 250K, 350K, and 400K respectively.

a normal distribution with their means at 0. The standard deviation of the 64-bit PUF is smaller than the 32-bit PUF. To better visualize the probability that the delay ratio is larger than some value, we use Table 6.1 to show the quantified result.

We test the stability of PUF challenges of various delay ratios under different environmental temperatures. After changing the temperature, we measure the probability that the same challenge produces the same response. Table 6.2 shows the results on a 32-bit arbiter PUF and a 64-bit arbiter PUF. We draw the conclusion that with a higher original delay ratio, the chance that the PUF output remains stable is higher. For example, when the original delay ratio reaches 0.1, the probability that the output remains stable is 1 no matter how the temperature changes (250K to 400K). Compared to the 32-bit test case, the 64-bit test case demonstrates a similar trend and even exhibits a better stability under the same condition. Therefore, we conclude that as long as the original delay ratio reaches some threshold (e.g., 0.1 according to this test), the output will be stable for a wide range of temperature conditions (250K to 400K). Challenges that match this threshold are selected as the stable challenges.

6.3.2 Benefits and Limitations

When using an arbiter PUF to initialize the digital logic, users should only use stable responses of arbiter PUFs for initialization. The key benefit is that the approach effectively avoids the instability of PUF model. And users can now have a full control over the configuration of digital logic.

However, limitations also exist. On one hand, in practice, it is time-consuming to identify all the stable CRPs for each individual PUF. It requires the manufactures or users to test exponential CRPs under various temperatures and voltages. On the other hand, the requirement of using only stable CRPs greatly compromises the space of available CRPs of arbiter PUFs. For example, as shown in Table 6.1 and 6.2, only 0.21% challenges in 32-bit PUFs and 0.05% challenges in 64-bit PUFs are completely stable across all the tested temperatures. With such a small space of stable CRPs, potential attackers can more effectively attack the PUF, even with brute force method.

Our design has created a conflict between the need of stability in the arbiter PUF responses versus the main point of an arbiter PUF having an exponential number of CRPs. This problem can not be completely eliminated due to the instability nature of analog PUFs. We are motivated to create a new digital PUF in chapter 7 where the design completely gets rid of the analog PUFs.

6.4 A Digital and Unclonable Design

With the presenting of the architecture and the operations of the digital PUF design. We answer the second question in this section: why is the design digital and unclonable?

First of all, the design is digital in the following sense. Although we also utilize analog PUFs to generate responses, they are only applied to initialize the LUT cells of the digital logic at the device power up stage. Afterwards, when users start to feed inputs to the digital PUF, everything is purely operated on the digital level, and the digital PUF outputs only depend on the functionality of digital logic. In other words, the outputs are not subject to operational and environmental conditions.

On the other hand, our design is also unclonable. It is solely attributed to the existence of the analog PUF in the design. Note that we assume there exists no interface for attackers to physically access the analog PUF responses because the response generation and the logic initialization happen automatically during the configuration phase. Otherwise, attackers can build a statistical PUF model to emulate the analog PUF.

6.5 Security Attacks

6.5.1 Cloning Attack

The most intuitive type of attack is to clone an identical piece of the FPGA-based digital PUF. As mentioned above, it is not possible because of the analog PUF. It takes advantage of process variation in the FPGA. Although many efforts are being taken to minimize the effect of process variation as technology improves, as long as the bias still exists, the analog PUF will continue to work and is even more unpredictable. Therefore, due to its unclonability, there is no way for the attacker to figure out the digital PUF configuration.

6.5.2 Side-channel Attack

Side channel attack is not a threat to analog PUFs. For example, in the arbiter PUF, regardless of the challenge bits, two similar delay paths will always be generated, and the response purely depends on the delay property of the physical entity.

6.5.3 Brute-force Simulation

Suppose the attackers create a huge LUT to store all the possible digital PUF inputs and outputs. On one hand, the size of the security cipher implemented as the digital logic can be easily boosted, making the number of pairs which need to be enumerated grows exponentially.

On the other hand, since our digital PUF will be reinitialized every time before use, as a result, even if the digital logic on the FPGA after configuration is simulated by an attacker, it will not be a threat because the same configuration is not to be used for the next time.

6.5.4 Special Purpose Hardware

This type of attack requires the attacker to use a very fast processor, e.g., ASIC, to simulate the FPGA-based digital PUF. The key thing to note here is that our design can not be reversed engineered and it is easy to create an exponentially larger security cipher (in terms of simulation effort).

6.6 Summary

We have proposed a FPGA-based digital PUF by intentionally choosing stable challenge-response pairs from the analog PUF and using them for digital logic initialization. Our design inherits the unclonability from analog PUF and the digital property from digital circuits. It enables a complete elimination of the traditional PUF vulnerabilities, such as susceptibility to operational and environmental variations while maintaining unclonability.

CHAPTER 7

Digital PUFs with Laser-based Fault Injection

7.1 Motivation and Problem Formulation

We continue our efforts to create digital PUFs in this chapter. In chapter 6, we have created a valid digital PUF. However, the problem with the design is that it still depends on analog PUFs for initialization which creates the stability problem. There is no way to completely solve the stability problem without compromising system security. Therefore, our goal in this chapter is to create a digital PUF that completely operates on digital level and does not require the use of analog PUFs. A well-known wisdom that is widely and strongly established is that integrated circuit (IC) defects and their functional faults are intrinsically a phenomenon that should be detected, diagnosed and, if possible, eliminated. Essentially faults in circuits are unwanted. Our objective is to rebut exactly the above well-established postulate. Specifically, we intentionally introduce faults in circuits to create digital PUFs. Three key observations are that (i) faults can be intentionally produced. For example, using laser-based fault injection. (ii) large VLSI ICs with partial faults can produce highly unpredictable outputs. (iii) Multiple faults in ICs are extremely difficult to be detected and positioned. The first one indicates the feasibility to introduce random faults in circuits. The second observation can prevent a large family of security attacks from statistical level. And the last one suggests that ICs with faults are hard to be duplicated. Based on the above observations, we claim that the faulty circuit can serve as a natural PUF that operates on the digital level. We call our proposed digital PUF the laser-based PUF.

Since at least 1997, laser-based fault injection has been recognized and demonstrated as a powerful security attack on cryptographic devices [28]. Numerous fault injection-based security attacks have also been reported and are surprisingly successful. A comprehensive survey of fault injection techniques as tools for compromising security devices and protocols is presented by Barenghi etc. [29]. Models to evaluate the circuit sensitivities to random defects are proposed by Stapper [30]. The key difference between the surveyed research and our efforts is that for the first time we intentionally introduce faults in circuits and advocate the positive use of faults for security.

In the following sections, we introduce the concept of faulty circuits and further evaluate the security properties of our laser-based PUF. When a circuit has one fault or very few number of faults, the fault detection is still possible. However, as the number of the faults increases, the detection becomes exceptionally hard, consequently, the device itself becomes unclonable. An essential step in exploiting faults is the creation of structures so that the faults in circuits can maximize the output randomness. By simulating the faulty circuits, we analyze commonly used adders, multipliers and xor networks based laser-based PUFs in terms of their security properties.

The laser-based PUF is another type of digital PUF since its unclonability does not depend on the analog properties of the circuit. With the injected faults, the circuit itself is still operated on the digital level thus being stable against temperature and voltage variations.

7.2 A Motivational Example

We start with the simple one-bit adder circuit represented by logical gates. We assume that the fault in the circuit is a gate-level stuck-at fault which means that the output of a gate is tied to logical 1 or 0 regardless of the inputs. Note that when using the laser to cut in the circuit, if we connect the position being cut to V_{dd}, it is equal to stuck at 1, and if we connect it to ground, the output is equal to 0. Figure 7.1 shows the four potential fault positions in a one-bit adder. For each position, the output can be stuck at either 1 or 0. Table 7.1 compares the outputs of the circuit with different fault positions to a fault-free adder. The results in Table 7.1 show that even if there is only a single stuck-at fault in the circuit, the impact on the outputs is significant and difficult to predict. This provides the intuition that the outputs of the faulty circuit can show excellent randomness and unpredictability after the appropriate configuration.

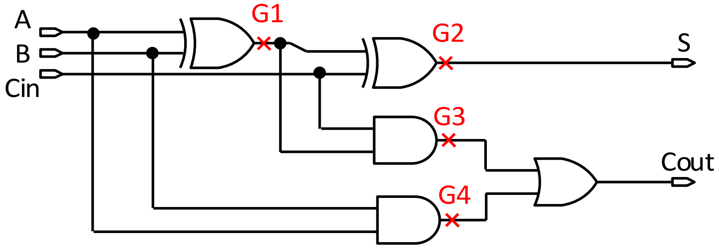


Figure 7.1: Stuck-at faults in a one-bit adder. Gi indicates different positions of faults.

$A/B/Cin$	Cout / S								
	Fault Free	$G1 \rightarrow 1$	$G1 \rightarrow 0$	$G2 \rightarrow 1$	$G2 \rightarrow 0$	$G3 \rightarrow 1$	$G3 \rightarrow 0$	$G4 \rightarrow 1$	$G4 \rightarrow 0$
0 0 0	0 0	0 1	0 0	0 1	0 0	1 0	0 0	1 0	0 0
0 0 1	0 1	1 0	0 1	0 1	0 0	1 1	0 1	1 1	0 1
0 1 0	0 1	0 1	0 0	0 1	0 0	1 1	0 1	1 1	0 1
0 1 1	1 0	1 0	0 1	1 1	1 0	1 0	0 0	1 0	1 0
1 0 0	0 1	0 1	0 0	0 1	0 0	1 1	0 1	1 1	0 1
1 0 1	1 0	1 0	0 1	1 1	1 0	1 0	0 0	1 0	1 0
1 1 0	1 0	1 1	1 0	1 1	1 0	1 0	1 0	1 0	0 0
1 1 1	1 1	1 0	1 1	1 1	1 0	1 1	1 1	1 1	0 1

Table 7.1: The impact of a single stuck-at fault on the outputs of a one-bit adder. Values in red indicate the changed bits in the faulty outputs compared to the fault-free outputs.

7.3 Architecture

In order to create the laser-based PUF, we have two key operations. The first is to randomly use the laser to introduce faults in circuits. As a result, for different implementations, the position and type of the faults would be different. The second is that since the faults are randomly created, it is only by gate level characterization that the position and the type of the faults can be measured and, thus, potentially enable an attacker to clone the device. We eliminate this possibility by physically removing (e.g. burning) those pins on the circuit which enable gate level characterization. Therefore, the physical unclonablity of the faulty circuit is guaranteed.

Now consider an attacker who attempts to clone our PUF. He/she is not able to execute a hardware level attack to look into the structure of laser-based PUF due to the burning of the pins. Instead, the attacker can test all the possible input vectors on the faulty circuit to get the corresponding outputs, thus to create an input-output mapping. However, for a large scale circuit, the attacker can not reverse engineer the circuit structure just by acquiring the mapping.

In this section, we propose the architecture of our laser-based PUF based on a few common circuits. The desiderata is that although the laser injection can be applied to any type of circuit, we define an architecture to have "good performance" only when it guarantees excellent statistical properties against potential attacks. We first propose the laser-based PUF structure based on commonly used adders and multipliers. Then we propose a customized XOR network with a better performance.

7.3.1 Adders

An adder is one of the most commonly used circuits. We have applied faults on the most fundamental carry-ripple adder to create a digital PUF.

7.3.2 Multipliers

Multipliers can also be found in many circuits, but generally take more area and power than adders. We use the multiplier built with carry ripple adders in our design. Our intuition is that since a multiplier has more depth and a larger number of gates than adders, the faults in multipliers are easier to propagate. Consequently, they will alter the outputs more significantly.

7.3.3 XOR Networks

The XOR network architecture shown in Figure 7.2 has w inputs, u outputs and h stages of XOR gates. Each stage is comprised of u XOR gates. Between two stages, the outputs of the previous stage are randomly shuffled and used as the inputs for the next stage of XOR gates. The total number of XOR gates used in this design is $u * h$.

In this design, on average, an output from a previous stage needs to be used as the input of two gates in the next stage, e.g., the connection indicated by the red line in Figure 7.2. Now suppose a fault occurs at the first stage of this XOR network. As a result, in stage 1, the output of one gate is possibly changed. In stage 2, since the wire of the faulty output from the previous stage is connecting to two gates in this stage, the outputs of two gates in stage 2 are influenced. In the final stage (stage h), on average, 2^{h-1} outputs are influenced. Therefore, we conclude that a fault in XOR network propagates exponentially as the number of stages grows.

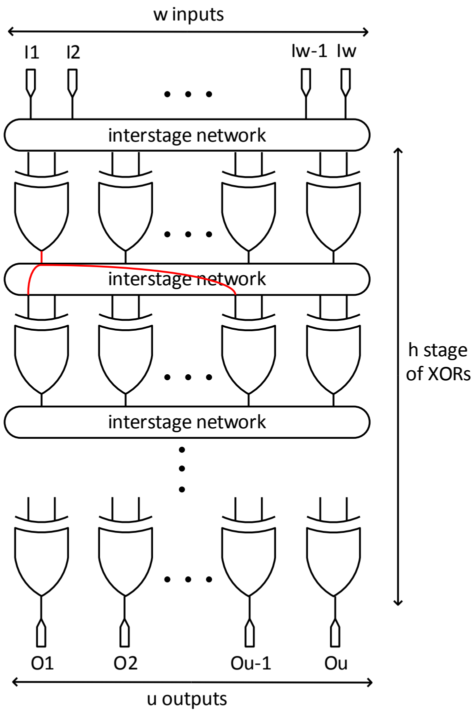


Figure 7.2: XOR network with w inputs, u outputs and h stages of XOR gates. Interstage network interconnects only the cells between neighboring layers of gates. The red line shows an example of interstage connection.

7.4 Security Attacks and Evaluation

In this section, we analysis the security properties of the laser-based PUF based on adders, multipliers, and XOR networks respectively. The basic approach is to identify their resistance against statistical attacks. In the attacks, the attacker observes a number of challenge-response pairs and tries to statistically analyze them in order to predict the response to an unseen challenge.

For each type of attack, we conducted comprehensive tests using a 64-bit carry-ripple adder, a 32-bit array multiplier and a XOR network with $w = 64$, $u = 64$, and $h = 8$. All of these circuits have 64 outputs for the sake of comparison (we do not consider the last carry out bit in the case of adder and multiplier). For each simulation, we present the results using 10,000 input vectors. In each type of circuit, we suppose 2 percent of the gates have stuck-at faults, and the positions of the faults are randomly assigned.

7.4.1 Predict with Fault-free Circuits

In this type of attack, the attacker tries to predict the outputs of a faulty circuit by using the outputs of the corresponding fault-free circuit given the same inputs. We simulate to analyze their average output hamming distance on adders, multipliers and XOR gates respectively. Ideally, the result should be around half of the number of outputs, which is 32 in our test. Table 7.2 shows the average output hamming distance results between faulty and fault-free circuits. It is obvious that the XOR network has the best performance, followed by the multiplier and the adder has the worst performance.

	Adder	Multiplier	XOR network
Avg. Distance	2.7 ± 1.8	22.9 ± 4.3	31.92 ± 4.56

Table 7.2: Average output hamming distance between the faulty circuit and the fault-free circuit.

7.4.2 Predict with Similar Inputs

If the avalanche effect is evident in a cryptographic system, then there is an ultra low probability that an attacker can predict any subsequent outputs using the knowledge of outputs of similar inputs. The avalanche criterion can be measured by observing the outputs of two inputs that differ by a minimal amount. In the case of our laser-based PUF, the smallest amount that an input vector can change is by one bit.

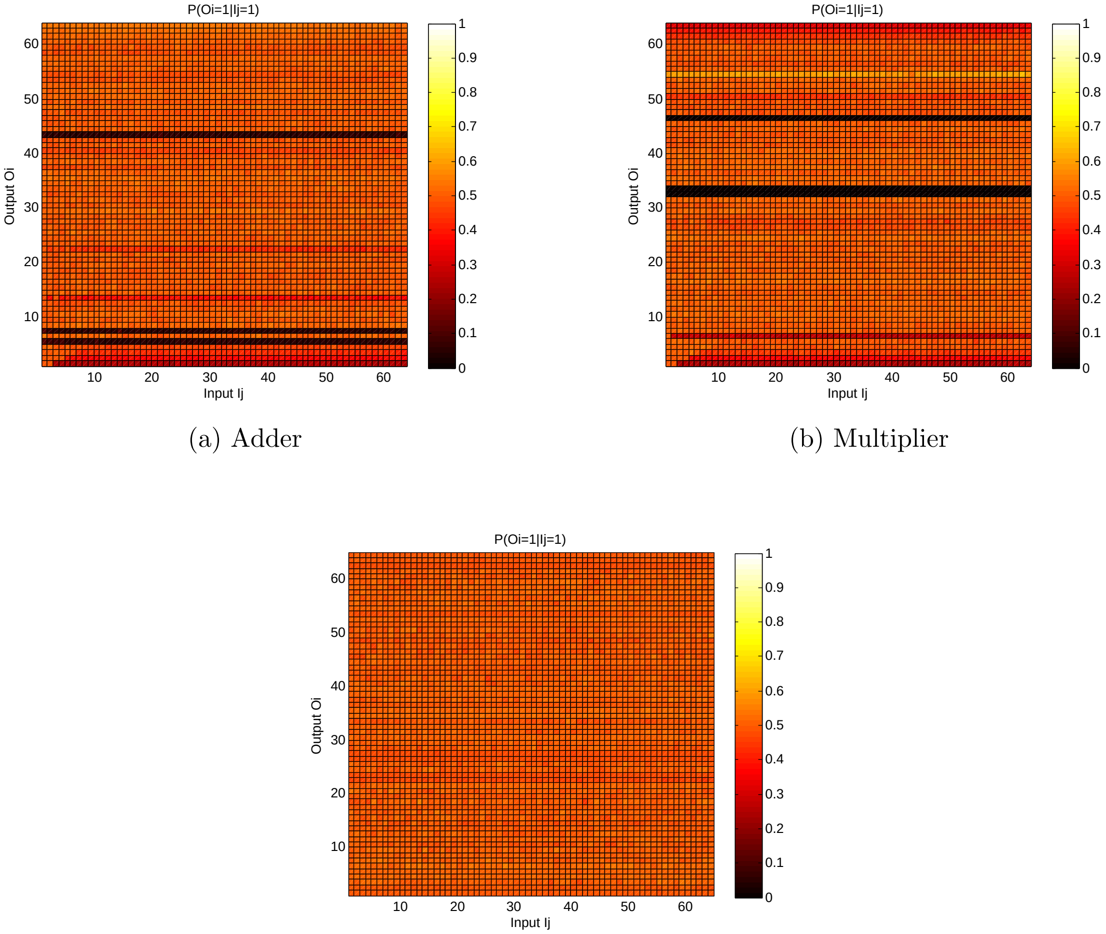
Thus, we measure the hamming distance between output vectors when changing one bit of our input vector over 10,000 inputs. Ideally, the average hamming distance should be 32. Table 7.3 presents the results on the three architecture, still, the XOR network performs the best, then the multiplier and the adder.

	Adder	Multiplier	XOR network
Avg. Distance	1.54 ± 0.51	16.41 ± 3.15	31.24 ± 4.02

Table 7.3: Average output hamming distance for changing faulty circuits input vector by one bit.

7.4.3 Predict with Conditional Probabilities

Another type of attack is the bitwise correlation modeling via the construction of per-bit input-output conditional probability distribution. The goal of the attacker is to predict an output bit by observing the inputs and the probability: $P(O_i = c_1|I_j = c_2)$, c_1, c_2=1 or 0. An ideal secure system will have a probability of 0.5 for all conditionals. Figure 7.3 depicts the conditional probability $P(O_i = 1|I_j = 1)$ across adders, multipliers and xor networks. Despite the fact that some output bits in adders and multipliers are always 0 because of the positions and types of the faults, the overall performance of the three structures is excellent. Note that the majority of conditional probabilities are around 0.5 and for the few “black lines” in Figure 7.3a and Figure 7.3b, users can simply avoid using the corresponding output bits.



(a) Adder

(b) Multiplier

(c) XOR network

Figure 7.3: Conditional probabilities between input bits I_j and output bits O_i: $P(O_i = 1|I_j = 1)$ on (a) adder, (b) multiplier, and (c) XOR network.

7.5 Summary

We have developed another type of digital PUF: the laser based PUF by introducing intentional faults in circuits. In addition to complete elimination of standard analog PUF vulnerabilities such as susceptibility to operational and environmental variations, the overall structure is still unclonable given the fact that multiple faults in large ICs are extremely difficult to detect. The test on simple models (adders, multipliers and xor network) indicates that the xor networks show the best security properties against statistical attacks compared to adders and multipliers.

CHAPTER 8

An Energy-efficient Fault Tolerance Approach for IoT

8.1 Motivation and Problem Formulation

The prevailing of Internet of Things (IoT) enables a variety of applications including smart homes, wearable devices, intelligent automotive, and so on. Such IoT applications have imposed new requirements on fault tolerance (FT) and energy management. On one hand, in some IoT applications such as implanted devices and airplane control systems, a single fault can be ultra dangerous and even life threatening. Therefore, to detect and to correct faults in such IoT applications have become especially important. On the other hand, energy/power has become a primary design criterion for many IoT applications. It is because such IoT devices are normally highly constrained by their limited battery life. To summarize, the FT in IoT applications has to be addressed in a way with high energy efficiency.

Non-volatile memory (NVM) is a type of memory that can retain the stored data even after external power has been turned off. Common types of NVM include phase-change memory (PCM), magnetoresistive random-access memory (MRAM), resistive RAM (RRAM), and memristors. Such resistive memory technologies share the properties of high scalability, non-volatility, and high density. Due to the unique properties of NVM, it is commonly used to improve system reliability. The principal idea is to store program execution data in NVM as a checkpoint. Thus, the system can be recovered from a wide range of transient and permanent faults using the stored information.

NVM based FT approach provides a possible solution to enhance the reliability of IoT devices. It is especially suitable for the IoT applications that calculate real-time data flow.

For example, in applications such as self-driving cars, airplane navigation systems, or even fall detection systems for elderly or disabled people, they all require their data to be processed in realtime. And when a fault indeed occurs in the process, the system has to be recovered with an ultra-short period. NVM based FT is ideal for these IoT applications because it is much faster to restart a program execution with the most recent checkpoint and the set of inputs instead of falling behind trying to recover an older state from the beginning of the program.

However, one major problem of the FT mechanism using NVM is that it employs large timing and energy overhead. As shown in Table 8.1, NVM employs a significant cost in writing time and writing energy when compared to other memory technologies. Consequently, when used in FT, it is crucial to reduce the number of write access to the NVM. Meanwhile, it is also an important issue to schedule the writes at proper clock cycles to avoid congestion.

Features	SRAM	eDRAM	MRAM	PRAM
Density	Low	High	High	Very high
Speed	Very fast	Fast	Fast read Slow write	Slow read Very slow write
Dyn.Power	Low	Medium	Low read High write	Medium read High write
Leak.Power	High	Medium	Low	Low
Non-volatile	No	No	Yes	Yes

Table 8.1: Comparison of different memory technologies.

Our goal is to build an energy efficient FT approach using NVM. The key idea is to store some states of a program in the NVM cache in such a way that the program can be resumed from the saved state after faults occur. We have addressed and optimized two major issues in our approach, respectively:

- Where should each state be?
- How to schedule the write of states to the NVM cache?

Regarding the first question, the key requirement of state selection is to minimize the overhead. To address this issue, we have applied the min-cut max-flow algorithm on the data flow model of the program. It is straightforward in the sense that each write to the NVM takes large timing and energy overhead. Therefore, the fewer data to write to NVM, the less overhead the system will have. Furthermore, to enable multiple states in a single program, we extend the standard min-cut algorithm to the “multiple min-cuts algorithm”, where the total overhead of all the states is minimized.

The second question targets to solve the issue of slow write to NVM cache. According to Table 8.1, the write to NVM takes heavy timing overhead. Thus, it is important to minimize the influence of states writing to the overall execution speed of the program. In other words, the program execution should not be halted to wait for the write to NVM to finish. To addressed this issue, we combine the use of NVM and classical CMOS register files so that when multiple variables are generated at the same time, they are first temporarily stored in some dedicated register files which are fast and low-energy consuming. Then the program can continue without being influenced while the write from register files to NVM cache is executed simultaneously.

We summarize our contributions in the following.

- We have proposed an NVM based architecture for FT by storing recoverable program states.
- We leverage the issue of large energy and timing overhead to write to NVM by proposing min-cuts for state selection, so that the amount of data needs to be written to NVM is minimized. By introducing register files for temporary storage, the extra timing overhead is almost eliminated.
- Our approach is generic in a sense that it can be used to recover systems from a wide range of permanent and transient faults on many underlying IoT architectures and computational models.

8.2 Preliminaries

We survey the related work along several research and development directions: non-volatile memories, fault tolerance, program slicing techniques, and min-cut max-flow algorithms. These efforts are intrinsically multi-disciplinary and due to space limitations, we consider only most directly related technologies and mechanisms.

8.2.1 Non-volatile Memory

Several types of non-volatile memory have been developed and studied as potential alternatives for CMOS flip-flops, registers, cache, main memory as well as for long term storage including flash, ferroelectric RAM, phase change, and MRAM [31]. In our simulation, we have used spintronic memories, specifically spin-transfer torque RAM (STT-RAM) due to its attractive properties regarding integration flexibility, size, and relatively low latency and energy overhead [32]. Hybrid memory systems have seen also widely studied, and it has been demonstrated that they are an attractive alternative in several types of systems and workloads [33].

8.2.2 Fault Tolerance

The design of reliable and fault-tolerant circuits and systems has a long history since a landmark pioneering paper by Moore and Shannon was published in 1956 [34]. There is a number of excellent related books including recent ones [35][36]. Three dominant aspects of fault-tolerant systems are fault abstraction into models (e.g. soft upsets [37], delay faults, and power supply reduction), fault detection, and fault recovery. Our focus is on fault recovery. We consider all types of faults such as failure of the power supply and soft upsets that can be fully resolved through the use of NVM. More recently, a memory architecture to enable NVM-based checkpoints for FT is proposed by Kannan et al. [38]. Compared to their work, our paper has emphasized on using an algorithmic way to find optimal positions of NVM-based checkpoints.

8.2.3 Program Slicing

A slice of a program can be defined as an executable subset of the program that computes the same functions as the initial program for a selected subset of variables [39]. Techniques for program slicing have emerged in the seventies as popular mechanisms for program debugging [40][41]. They are still essential procedures for the analysis of software products [42].

The availability of several systems such as shared memory multiprocessors with global checkpoints is optimized for on-line processing systems [43]. A checkpoint consists a set of variables required for a program to restart after a fault detection. In computer-aided design literature, a special type of checkpoint named cut was used for the export of "difficult to observe variables" and for the injection of "difficult to control variables", such that to enable high speed debugging of complex integrated circuits. Under the assumption of the static synchronous dataflow computational model, Kirovski et al. defined a program cut as a system of variables that are all stored in a minimal number of registers and require minimal additional interconnect network for their complete controllability and observability [44].

8.2.4 Min-cut Max-flow Algorithm

Our main optimization step is related to the min-cut max-flow theorem. The theorem has been independently discovered by Elias, Feinstein, and Shannon and by Ford Fulkerson in 1956. A comprehensive coverage of the algorithms for a variety of network flow problems is given in a book by Ahuja, Magnati, and Orlin [45] as well as many other books. We have used the linear programming formulation from Papadimitriou and Steiglitz [46] that is both simple and very flexible with the respect of modifying the types of min-cuts that are required. Finally, it is interesting to notice that retiming for minimization of the number of flip-flops by Leiserson and Saxe can also be used for this tasks and sometimes leads to even more flexible formulations [47].

8.3 System Overview

Our approach works on the platform of application-specific integrated circuits (ASIC). The program execution is performed on the regular datapath while the outputs of functional units can be stored in the cache. Figure 8.1 shows the overview of our system. States of the executed program are captured with some particular frequency and are stored in the NVM based cache. When any fault occurs, the program can be recovered from the stored states. Note that we use two NVM spaces in the cache to store program states interchangeably. Each state is only written to a single NVM space, and it overwrites the previous state stored in the space. Such mechanism guarantees that when faults occur in the middle of states write, at least one NVM space in the cache stores a complete state. We also assume that there exists a single port to write data from the datapath to the cache. As a result, to avoid congestion, the schedule of data write becomes crucial.

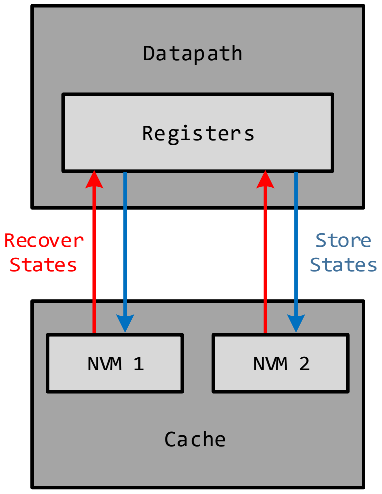


Figure 8.1: System overview.

8.4 A Motivational Example

The graph G in Figure 8.2 shows an example data flow graph where I_1 to I_4 are inputs and O_1 to O_2 are outputs. A state in the graph is defined as a set of variables that cuts

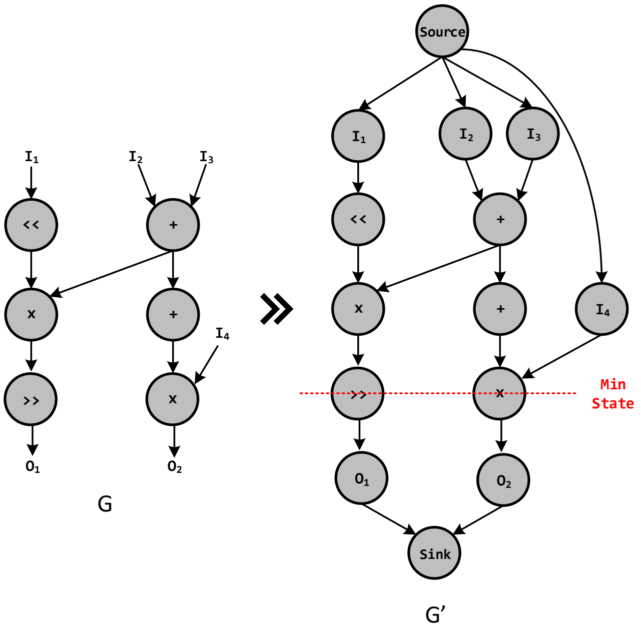


Figure 8.2: A motivational example of dataflow graph and the corresponding min cut.

all possible paths from primary inputs to outputs in the computational flow. Graph G' depicts the adjustment of the original data graph G in order to fit for the min-cut max-flow algorithm to find a state. Two major changes are made. The first is to add a source node and a sink node to the graph. The second is to create nodes for each arithmetic unit as well as primary inputs and final outputs. Correspondingly, we also add edges to connect the source and the primary inputs as well as edges to connect the outputs and the sink.

Therefore, the goal of the state search algorithm is to, given a computation control data flow graph, find a register subset of minimal cardinality that stores all the variables of at least one complete cut. To find such state, we have applied our modified min-cut algorithm and the minimal state is shown in G' of Figure 8.2 (assume that each functional unit generates the same amount of data). More details on our min-cut algorithm are illustrated in Section 8.5.

8.5 Approach

We address our approach of solving two optimization problems in this section. The first is state selection with the motivation to minimize the amount of data that needs to be written to NVM. The second issue is variable scheduling. As we demonstrated in the system overview, only a single port is available to write to cache. Considering that each write to NVM employs a large timing overhead, thus when to write and what to write to the NVM have become an important issue.

8.5.1 State(s) Selection

Our key algorithm for state selection is through the min-cut max-flow algorithm. The insight is that we want to minimize the energy spent to write a state to the NVM. The required energy is proportional to the amount of data to write. It is a natural idea to apply the min-cut algorithm as it directly returns the state with least data. However, there still exists some modifications we need to make to the standard min-cut algorithm to suit our problem. The first problem is that standard min-cut algorithm applies cut on the edges while in our problem, we should cut on the nodes since we only need to store a single copy of the arithmetic unit result. To address this issue, our solution is to convert the original data flow graph to a new graph and then apply the standard min-cut algorithm. The other issue is regarding the number of cuts in the graph. As min-cut only returns the optimal cut across the circuit, it is not addressed in the standard algorithm about how to minimize the total flow when multiple cuts are required.

In the following parts, we first discuss our algorithm to find a single optimal min-cut state on the graph, then we extend our algorithm to select multiple states.

8.5.1.1 Single State Selection

We formally explain our formation of data flow graph. Following the same notation in Section 8.4, we create a directed graph $G' = (V', E')$ where each node $v \in V'$ represents a primary input or an arithmetic unit and each edge $e \in E'$ represents a flow of data between two nodes. Each node in the graph has an attribute *capacity* which describes the data size of the node. For a node of a primary input, the capacity represents the size of the input; for a node of an arithmetic unit, the capacity represents the data size of the unit output. Each edge (from node i to node j) also has the *capacity* attribute which represents the size of data transmitted from node i to node j. We virtually add a source node which has edges connecting to all the primary inputs and a sink node that is linked with all the program outputs. The capacity of all the edges that are directly connected to source or sink is initialized to ∞.

Our goal is to find the nodes based min-cut on graph G'. To be consistent with the standard min-cut algorithm model which only has capacity constraints for edges, we transform our data flow graph $G' = (V', E')$ to $G'' = (V'', E'')$, where each node v in G' breaks into two nodes v_1, v_2, as well as an edge from v_1 to v_2 to carry the original capacity of node v. With the above step, the node capacities are converted to edge capacities.

Figure 8.3 shows the transformation from graph G' in the motivational example to graph G''. In G'', each solid edge (black) from node i to node j represents the flow of data from unit/input i to unit/output j. The capacity of such edges is set to ∞. Each dashed edge (red) in G'' is an added edge inside each node of graph G'. The capacity of a dashed edge equals to the capacity of the node it corresponds to, which equals to the size of data the node outputs.

When applying the standard min-cut algorithm on graph G'', it naturally finds a minimal complete cut over only the dashed edges since all the solid edges have the capacity of ∞. Therefore, the corresponding nodes in G' that map to the min-cut edges in G'' are selected to form a complete state. Due to the nature of min-cut, the amount of data that is generated by such state is minimized.

One problem of the above-proposed max-flow based state selection is that the cut result can be highly biased to inputs or outputs. For example, in our graph shown in Figure 8.3, although the cut we denote in G' indeed finds an optimal state in terms of energy saving, the problem is that the state is too close to sink. To be more specific, the selected state directly

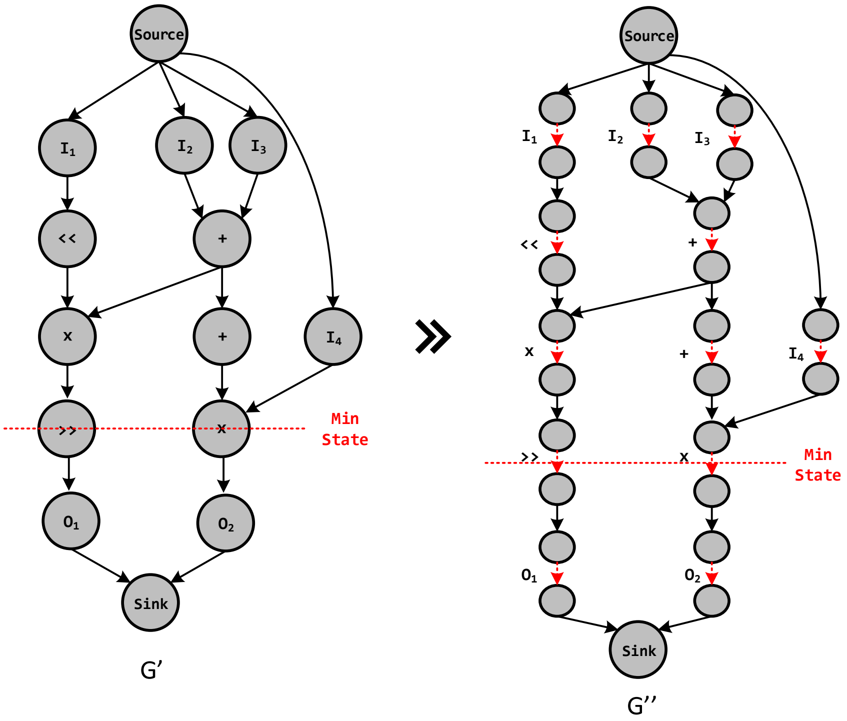


Figure 8.3: Graph transformation to find nodes based min-cut on the motivational example shown in Figure 8.2.

saves the program outputs. Consider a fault occurs during the program execution (before the program outputs are generated), our state can not be used as a checkpoint to resume the program. On the other hand, a state too close to primary inputs is also questionable. It is because in such cases when the program resumes from the state, almost all the operations in the program need to be recalculated in which case a state does not help anything.

As explained above, it is important to have a state somewhere in the middle of the program flow, although this can sacrifice the size of cut to be no longer optimal. Here we formally define the position requirement of our desired state s. The maximum distance from the source to any node in the state (d_{src-s}) has to be smaller than α, and the maximum distance from any node in the state to sink (d_{s-snk}) has to be smaller than β. α and β are constants decided by the size of the graph. Using the above notation, our objective is shown in Equation 8.1.

$$\begin{aligned} &\text{Minimize: } Capacity(s) \\ &\text{Subject to} \\ &\qquad d_{src-s} > \alpha \\ &\qquad d_{s-snk} > \beta \\ &\qquad \alpha,\ \beta \ are\ constants \end{aligned} \tag{8.1}$$

We propose Algorithm 3 to find a state that meets the above criterion. The basic idea is to iteratively find the top n min-cuts on the graph until a cut meets the position constraint. To find the top n optimal min-cuts ($n > 1$), we iterate through the edges from the 1st optimal min-cut until the $n-1$th optimal min-cut, assign the capacity of selected edges to ∞, and then continue to find the next min-cut. By assigning edges to ∞, we avoid the algorithm finding a min-cut solution the same as the top $n-1$ optimal min-cuts. Thus, the cut we achieve in the nth iteration comes as the nth optimal min-cut. The search for the top n cuts continues until a cut is found to meet the position constraint.

8.5.1.2 Multiple States Selection

In the previous discussion of state selection, we assume that there only exists a single cut in the program. However, multiple states selection is also commonly required especially for large programs. When the frequency of state selection is high, more energy needs to be spent to write states to NVM. Meanwhile, as the distance between different states decreases, the expected energy that is required to resume program after faults occur is reduced. There exists a trade-off between the write energy and the resume energy.

The first important issue in multiple states selection is to decide the desired number of states k ($k > 1$). To quantitatively decide an optimal k, we define the following two concepts.

- E_{write}: Energy spent to write the data of a state to NVM.
- E_k: Given k states, the expected energy required to recover the program from faults.

Algorithm 3 Single State Selection

Input: Graph $G'' = (V'', E'')$, constants α, β.

Output: Optimal state selection s.

```
Find min-cut $s$ on Graph $G''$.
$Set_s = \varnothing$.
Append $s$ to $Set_s$.
**While** $d_{src-s} < \alpha$ or $d_{s-snk} < \beta$:
    $Set_e = \varnothing$.
    **For** all cuts $s_i$ in $Set_s$:
        Select $e_{ij}$ from $s_i$.
    **Endfor**
    $G''_{modi} = G''$.
    **For** each combination of $e_{ij}$:
        Modify $G''_{modi}$ with capacity($e_{ij}$)=$\infty$.
        Find min-cut $c$ on $G''_{modi}$.
        Append $c$ to $Set_e$.
    **Endfor**
    $c_{min} = c_0$ in $Set_e$.
    **For** each cut $c_i$ in $Set_e$:
        **If** capacity($c_i$)¡=capacity($c_{min}$):
            $c_{min} = c_i$.
        **Endif**
    **Endfor**
    $s = c_{min}$.
    Append $s$ to $Set_s$.
**Endwhile**
**Return** $s$.
```

We should increase the number of states (k) only when $E_k - E_{k+1} > E_{write}$. As the number of states increases, the average distance between the neighbor states is reduced, thus, on average it takes less computational efforts to recover the program when faults happen. The left side of the equation $E_k - E_{k+1}$ represents the expected energy saving of program recovery by increasing the number of states from k to $k+1$. The right side of the equation E_{write} is the energy required to add a state. Only when the energy saved by increasing a state is larger than the energy overhead, we increase the number of states by 1. A program with long clock cycles but relatively "thin" states tends to have a larger k. It is because in such cases, the difference from E_k to E_{k+1} is large (long execution flow) while the E_{write} is small ("thin" states). As k increases, the difference between E_k and E_{k+1} becomes smaller and smaller, and is eventually converged to be below E_{write}. In most benchmarks, because the write to NVM is energy expensive compared to the recovery calculations in the datapath, k is only set to be a small value.

After k is configured, similar to the single state selection, multiple states selection imposes requirements for the data size of the cut as well as the positions of the cut. Ideally, the total amount of data in multiple states should be as small as possible under the constraint that the distance between each state is larger than some threshold γ. Note that the distance between state A and state B is defined as the average mutual distance between the nodes in A and the nodes in B. Depending on the total number of states (k), γ needs to be adjusted to make sure that a k-states solution exists. The above problem can be formulated using the following notations.

$$
\begin{aligned}
&\text{Minimize: } \sum_{i=1}^{i=k} Capacity(s_i) \\
&\text{Subject to} \\
&\qquad d_{src-s_1} > \alpha \\
&\qquad d_{s_k-snk} > \beta \\
&\qquad d_{s_i-s_{i+1}} > \gamma \ (0 < i < k) \\
&\qquad k,\ \alpha,\ \beta,\ \gamma\ are\ constants
\end{aligned}
\tag{8.2}
$$

We employ a similar algorithm as the single state selection to solve the above problem. The detailed algorithmic flow is shown in Algorithm 4. The key idea is to incrementally find top min-cuts in the data flow graph, and then search for all the k cuts combinations such that

Algorithm 4 Multiple State Selection

Input: Graph $G'' = (V'', E'')$, constants k, α, β, γ.

Output: Optimal multiple states selection s_i, $i \in \{1, 2...k\}$.

```
Find min-cut $s$ on Graph $G''$.
$Set_s = \varnothing$, $check = 1$.
Append $s$ to $Set_s$.
**While** $check$:
    $Set_e = \varnothing$
    **For** all cuts $s_i$ in $Set_s$:
        Select $e_{ij}$ from $s_i$.
    **Endfor**
    $G''_{modi} = G''$
    **For** each combination of $e_{ij}$:
        Modify $G''_{modi}$ with capacity($e_{ij}$)=$\infty$.
        Find min-cut $c$ on $G''_{modi}$, append $c$ to $Set_e$.
    **Endfor**
    Find $c_{min}$ in $Set_e$ with the smallest capacity.
    $s = c_{min}$, append $s$ to $Set_s$.
    $Set_m = \varnothing$
    **For** all $k$ cuts combinations ($c_1$ to $c_k$) in $Set_s$:
        **If** $c_1$ to $c_k$ satisfy $d_{src-c_1} > \alpha$, $d_{c_k-snk} > \beta$,
        and $d_{c_i-s_{c+1}} > \gamma$ $(0 < i < k)$:
            Append $\{c_1$ to $c_k\}$ to $Set_m$.
        **Endif**
    **Endfor**
    **If** size($Set_m$)¿0:
        $check = 0$.
        Select $c_1$ to $c_k$ in $Set_m$ with the smallest sum.
        $s_i = c_i$, $i \in \{1, 2, ...k\}$.
    **Endif**
**Endwhile**
**Return** $s_i$, $i \in \{1, 2, ...k\}$.
```

if the cuts in the combination can satisfy position constraints, the k cuts form a candidate multiple states selection and is appended to Set_m. Eventually, we traverse through all the candidates in Set_m to find the one with the smallest sum of cut capacity and use it as our final solution.

8.5.1.3 Influence of Unfolding

Many programs are designed to run multiple iterations, such as various filters in DSP applications. State selection in these programs commonly faces a major problem that each iteration of the program may only have very limited computations. Accordingly, it is not worth it to assign a new state for every iteration. For example, it is possible that $E_0 - E_1 < E_{write}$, making no state is needed per iteration. Hence, there is a need to update our state selection strategy for such programs.

Unfolding addresses the above concerns. First proposed by Parhi and Messerschmitt in 1989 [48], unfolding is a transformation technique which duplicates the functional units to increase the program throughput while preserving its functional behavior at its outputs. For an unfolding factor J, the core idea of unfolding is to duplicate all the functional units in the original program by J times and reconnect everything without altering program functionality. It produces the same outputs compared to running the original program by J iterations, but generally offers higher throughput and a smaller average iteration period.

By applying unfolding on iterative programs, we can achieve a data flow graph over multiple program iterations (J). Note that J can be adjusted to suit the need of state selection. On one hand, the graph is greatly expanded in such a way that a state can be selected in every J iterations, avoiding the situation where no state is needed for a single iteration. On the other hand, it provides a better global picture for state selection. We are able to flexibly pick the optimal number of states as well the optimal positions of states using our state(s) selection algorithms.

8.5.2 Variable Scheduling

Our design requires writing all the variables in the states to the NVM cache through a single port in a limited number of computational iterations. However, as shown in Figure 8.1, the write speed to NVM is slow, making the state writing to NVM easily becomes a bottleneck of the program execution. Note that when the system writes the data of a state from the registers to the NVM cache, the rest of the program has to halt to avoid overwriting the registers.

We leverage the above issue by using temporary register files. Instead of directly storing data from registers in the datapath to NVM, we first transfer the data temporarily to a register file, and then store it to NVM. Although the mechanism introduces extra hardware, the advantage comes in two aspects. The first is that the program no longer needs to wait for the write to NVM to finish, which employs a large timing overhead. The write from the temporary register file to NVM can be executed in parallel with the program running. The other advantage is that it completely avoids the situation of overwriting. As the temporary register file holds all the data of a state, the original registers that hold the data are free to be used by other functional units. Our mechanism will introduce small timing overhead which accounts for the write from regular registers to the temporary register file, however, compared to the huge time expense of writing to NVM, the overhead is negligible.

The variable scheduling of the above scheme can be split into two parts, respectively to schedule the write from regular registers to register files, and to schedule the write from the register files to NVM. The basic policy is generalized in the following.

- Regular registers → register files: timing order of variables.
- Register files → NVM: timing order of states.

In terms of the scheduling from the regular registers to register files, it should strictly follow the timing order of the generation of variables, so that the regular registers can be immediately freed after the variable calculation. In many cases, the data variables in a single state are generated in different clock cycles. It is likely that the calculation of next functional unit requires the use of an occupied register from the current state. Thus, as soon as a variable is generated, it should be transferred to the temporary register files.

The scheduling from register files to NVM follows the timing order of states generation. It is simply because each NVM space is expected to store a single complete state. Assume that variable v_1 belongs to state s_1, variable v_2 belongs to state s_2, and s_1 is prior to s_2. Both v_1 and v_2 are in the register file, waiting to be written to NVM. Note that we assume there exists only a single port between the register file and the NVM cache, hence, we can only choose to write either v_1 or v_2. Although it is possible that v_2 is generated before v_1, it is meaningless to first write v_2 to NVM. Because if so, when a fault occurs shortly after s_2, neither s_1 nor s_2 is available in the NVM as a complete state for recovery. Nonetheless, if v_1 is stored to NVM as a priority, there is a higher chance that by the time when a fault occurs, s_1 has been ready for recovery as a checkpoint.

8.6 Experiments

8.6.1 Energy Model

We build energy models for the program execution on datapath ($E_{original}$) as well as the energy required to write the state (E_{state}).

$E_{original}$ consists two parts of energy consumption, energy for arithmetic units, and energy for register read and write. To evaluate $E_{original}$, we have applied the energy model from circuit simulator Hspice. The main advantages of the circuit-level simulation are its accuracy and generality. It estimates the energy consumption of our program execution on datapaths regardless of technology, design, style, functionality, and architecture.

E_{state} also involves two components, the energy to write to temporary register files (E_{reg}) and the energy to write to NVM (E_{nvm}). E_{reg} can be easily modeled using circuit simulator. In terms of E_{nvm}, the type of NVM we have applied in our model is STT-RAM. Its energy and latency data used in our simulation is obtained from CACTI [49] and Chang's paper [50] as shown in Table 8.2. The high-capacity cache is a 32nm, 32MB, 16-way cache that is partitioned into 16 banks and uses 64-byte blocks. Given the same size of data, E_{nvm} is orders of magnitude larger than E_{reg}, making E_{nvm} the dominant energy in E_{state}.

Cache	Area (mm^2)	Latency (ns)	Energy $(nJ/access)$
STT-RAM(32MB)	16.39	read:3.06 write:25.45	read:0.94 write:20.25

Table 8.2: Parameters of STT-RAM (32nm).

8.6.2 Simulation Results

We apply our FT mechanism on a set of controller benchmarks as shown in Table 9.6. The first three examples are controllers for Aircraft Advanced Flight Control (VAAC) aircraft. The plane is a British VSTOL air force aircraft with vertical take-off and landing capabilities. Each of the examples corresponds to three different phases and three different speeds of 6, 86, and 122 knots respectively. We see that the effectiveness of the approach increases in the more demanding situation where faster reaction on the measurement of control signals is required. Image and video are small processing systems that work on their corresponding streams by applying a variety of nonlinear filters and transforms for noise removal and contrast enhancement. Chemical.s and chemical.l are small and larger controllers used by a chemical plant. Honda and honda.lp are the high and low speed of an automotive controller for adjusting gasoline and oxygen into the engine. Steam is a small linear controller of a steam plant engine. Finally, gps.nav is a location tracking system used for GPS-driven navigation. In our simulation, we assume that faults can happen in any arithmetic unit of the design.

In Table 8.3, column 2, 4, and 5 display the information of the program including the total number of generated variables, the number of multiplications, and the number of additions. Column 3 shows the number of variables in the selected checkpoint state. The size of the variables in each program equals to 16-bit. Column 6 and 7 respectively show $E_{original}$ of the original program, and E_{state} of the selected program state. The last column calculates the energy overhead of E_{state} compared to $E_{original}$. From the results, we have the following observations.

First of all, the percentage of energy overhead has no direct correlation with the size of the program or the size of the state. For example, chemical.s and video are two programs with the largest percentage of overhead. However, chemical.s has only 302 total variables, and video benchmark has as large as 20928 variables. A further observation suggests that the ratio between the number of state variables and the number of total variables has a dominant effect on the percentage of energy overhead. A larger ratio usually suggests a larger percentage. It is because that as the number of overall variables in the program increases, $E_{original}$ grows since more variables suggest more arithmetic units and more write operations to the regular registers. On the other hand, as the number of state variables decreases, the energy required to write data to NVM also decreases dramatically. To conclude, the most favored programs to our FT strategy are the ones with a large number of variables while the size of min-cut on the program is small.

To verify our observation, we choose four benchmark programs from MediaBench to apply our FT scheme. The four programs are intentionally chosen in such a way that they require long execution cycles while the selected state is relatively “small”. Therefore, all four programs employ high ratio between the total number of variables and the state variables. As shown in Table 8.4, the average energy overhead is reduced to only 14.9%.

Secondly, the number of multiplications in the program is also important. Multiplication is among the most expensive operations in a program; an N-bit multiplier consumes around one magnitude more energy compared to an N-bit adder. Thus, if multiplications take a large percentage of the total arithmetic operations, $E_{original}$ will be increased a lot assume the total number of operations is fixed. However, this has no influence on the result of E_{state}, hence the ratio between E_{state} and $E_{original}$ will decrease.

Design	Number of overall variables	Number of state variables	Number of multiplications	Number of additions	$E_{original}$ (nJ)	E_{state} (nJ)	Percentage of Energy overhead
VAAC.6	223	15	118	90	0.883	0.480	54.4%
VAAC.86	431	17	214	200	1.66	0.544	32.7%
VAAC.122	482	19	241	221	1.87	0.608	32.6%
chemical.s	302	24	152	126	1.17	0.768	65.8%
chemical.l	870	36	438	396	3.38	1.15	34.1%
image	2323	64	966	1303	8.36	2.05	24.5%
steam	112	7	56	49	0.433	0.224	51.8%
honda	361	9	192	160	1.44	0.288	19.9%
honda.lp	376	8	0	368	0.822	0.256	31.1%
sound	269	19	126	124	1.01	0.608	60.3%
video	20928	1536	9728	9664	78.2	49.2	62.9%
gps.nav	489	16	226	247	1.83	0.512	27.9%
						Avg.	41.5%

Table 8.3: Application of the fault tolerance to a set of controller benchmarks for the estimation of energy overhead.

Design	Number of overall variables	Number of state variables	Number of multiplications	Number of additions	$E_{original}$ (nJ)	E_{state} (nJ)	Percentage of Energy overhead
D/A converter	426	4	222	200	1.69	0.128	7.6%
PGP	1940	34	468	1438	5.84	1.09	18.6%
JPEG.enc	1026	18	504	504	3.96	0.576	14.5%
MPEG2.dec	864	25	527	312	3.68	0.8	21.8%
GSM.enc.dec	1400	22	852	526	5.96	0.704	11.8%
						Avg.	14.9%

Table 8.4: Application of the fault tolerance to a set of benchmarks from MediaBench for the estimation of energy overhead.

Finally, as shown in the last row of Table 8.3, without any intentional design selection, the average energy overhead of our FT mechanism is around 41.5%. Although our algorithm targets to reduce the energy overhead, it is still not a negligible ratio for IoT applications. For those IoT devices that are extremely sensitive to energy overhead, our mechanism is not the best option. Alternatively, our FT strategy works best for the IoT systems that require real-time updates, high accuracy calculations, as well as a moderate energy restriction.

8.7 Summary

We have proposed an NVM based fault tolerance mechanism for IoT applications. By storing intermediate states of a program to NVM, the program can be recovered from the states when faults occur, e.g., the loss of external power. By applying our modified min-cut max-flow algorithm on the data flow graph, we optimize the size of states in such a way that the energy overhead to store the state data to NVM is minimized. Moreover, we apply the temporary register files to hold the state data before storing them to NVM, reducing the timing overhead to be almost negligible. Our test results on a variety of controller benchmarks indicate that the average energy overhead introduced by our FT scheme is 41.5%. Moreover, when intentionally choose programs that favor the appliance of our FT mechanism, the average overhead can be reduced to 14.9%.

CHAPTER 9

Combine Pipelining with Dual-supply Voltages

9.1 Motivation and Problem Formulation

Low energy has emerged to become one of the most important design metrics in the last 25 years. It is especially important for IoT applications due to their highly constrained resources in many scenarios. Battery technology improvements have not kept pace with the rapid device scaling that has continued to follow Moore's Law. The platforms such as mobile phones and tablets are highly restricted by the amount of energy that can be stored in the battery and, therefore, energy efficiency has become a premier design requirement. On the other hand, energy consumption on the hardware level directly impacts the circuit temperature, which leads to the creation of hotspots. Frequent and significant fluctuations in temperature can have ramifications on the circuit reliability.

Many efforts have been made to achieve power/delay optimization, two techniques among them are well recognized, respectively pipelining, and dual-V_{dd} assignment. Pipelining is a circuit optimization technique which is frequently used to speed up the execution of computations in a circuit by dividing the original circuit into n consecutive sub-computation units and overlapping their executions. By adding flip-flops between the sub-computation units to temporarily store the intermediate results, theoretically, an n-stage pipeline circuit should yield a factor of n times delay improvement over the non-pipelined circuit. Dual-V_{dd} is another circuit design technique in which two supply voltages (high V_{dd} and low V_{dd}) are applied to a circuit in order to save power compared to only applying a single supply voltage (medium V_{dd}). The appliance of dual-V_{dd} optimization should not sacrifice either the throughput or the delay of the original circuit.

It is a natural idea to combine pipelining with dual-V_{dd} to jointly optimize circuit, expecting to achieve both low delay and low power. However, the optimal way to combine the two techniques is not yet a solved problem, which is due to the following two facts. The first is that there exist multiple pipeline schemes with the same circuit delay. For example, assume that a circuit is to be transformed into a 2-stage pipelining. Since the delay of a pipeline circuit is decided by the maximum delay of each single stage of the circuit, then it is important to guarantee that the delay of the original critical path d_c is evenly divided in such a way that each stage has delay $d_c/2$. However, the non-critical path logic of the circuit can be assigned to either stage as long as it does not create a new critical path longer than $d_c/2$. Consequently, multiple pipeline schemes with the same delay exist. The second fact is a technical constraint of dual-V_{dd} assignment. Only the gates on high V_{dd} can directly drive gates on low V_{dd} to prevent direct current due to incomplete positive-channel metal-oxide semiconductor (PMOS) cut-off [51]. The only exception is that a level converting flip-flop allows the logic after low V_{dd} gates to switch back to high V_{dd}. To summarize, only high V_{dd} gates can drive low V_{dd} gates unless converting flip-flops are used.

Combining the above two observed facts, we claim that different pipeline schemes even though with the same delay can have very different power consumption after dual-V_{dd} optimization. Figure 9.1 shows a motivational example of the effect of dual-V_{dd} assignment on a circuit with different pipelining. The original circuit contains 8 gates, from $G1$ to $G8$. Assume that all the gates have the same delay d_G on the original V_{dd}. Both pipeline designs in (a) and (b) introduce 4 extra flip-flops. We denote that the delay of flip-flops equals to d_{FF} on the original voltage. Then for both pipeline designs in (a) and (b) before dual V_{dd} optimization, the circuit has the same delay $2 * d_G + d_{FF}$. Now we apply dual-V_{dd} optimization on both pipeline designs. The gates and flip-flops on high V_{dd} are colored red (solid line) and their delays are denoted as d_G^h and d_{FF}^h. Correspondingly, the gates and flip-flops on low V_{dd} are colored blue (dashed line) and the delays are denoted as d_G^l and d_{FF}^l. Thus, for both pipeline designs in (a) and (b) after dual-V_{dd}, the critical path delay equals to $d_G^l + d_G^h + d_{FF}^h$. Note that in our dual V_{dd} optimization, the critical path delay

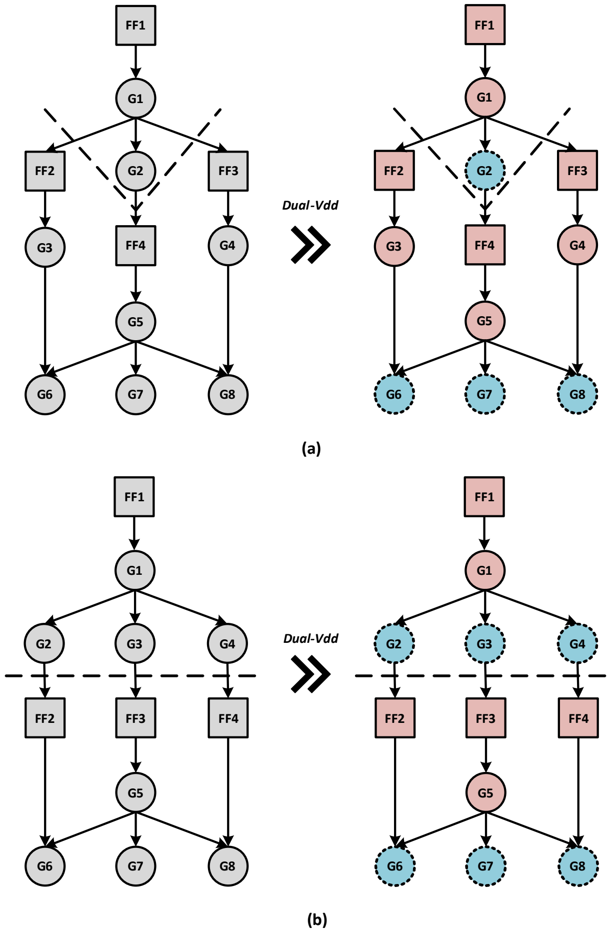


Figure 9.1: An example of dual-V_{dd} assignment on (a) non-optimized pipelining, (b) optimized pipelining. The flip-flops in the circuit are represented with rectangle, and the gates are represented with circle. The gates/flip-flops that are on original V_{dd} are colored grey (middle V_{dd}), the ones on low V_{dd} are colored blue (dashed line), and the ones on high V_{dd} are colored red (solid line).

before and after the optimization should stay the same. However, the key observation is that the number of gates assigned to high V_{dd} in (a) is 4 while the number of high V_{dd} gates in (b) equals to 2. It suggests that the circuit power consumption after dual V_{dd} assignment in (b) is less than the power consumption in (a) although both circuit designs enable the same delay.

Motivated by the above example, we propose a pipeline design with the use of an objective function in which the number of gates on high V_{dd} is minimized under the constraint of keeping the delay of the standard optimal pipelining. We only introduce level converting flip-flops between the stages of pipelining, which suggests that within each pipeline stage only high V_{dd} gates can drive low V_{dd} gates. Therefore, it is important that the width of the circuit is small at the positions immediately after the flip-flops so that fewer gates need to be placed at high V_{dd} while satisfying the delay constraints. To distinguish our paper from the previous related work of pipelining and dual voltage assignment, we summarize our contributions as following.

- A novel approach to combine the pipelining and the dual-V_{dd} assignment to jointly optimize energy consumption.
- Modification to the current pipelining algorithm to enable a more energy efficient dual-V_{dd} assignment.

9.2 Preliminaries

9.2.1 Pipelining

The task of pipelining is described in standard textbooks such as [52] and [53]. Many researches have been focused on the automatic generation of high efficient pipelining designs. Kroening et al. proposed automated pipelining which transforms a sequential machine into a pipelined machine by adding forwarding and interlock logic [54]. Cong et al. presented an architecture-level synthesis solution to support automatic pipelining of on-chip interconnections [55]. The work from Calceran-Oms et al. discussed an automatic pipelining method for micro-architectural systems with loops [56]. Most of the conventional pipelining methods target to maximize the processing performance by reducing the clock period [57][58].

9.2.2 Dual V_{dd} Optimization

Usami et al. first proposed to use multiple supply voltages as a way to reduce energy [59][60]. Salil and Sarrafzadeh applied multiple supply voltages at the behavior level for energy minimization [61]. Dynamic programming techniques for solving the dual-V_{dd} scheduling problem in both non-pipelined and functionally pipelined datapaths were proposed by Chang et al. [62]. A low power implementation on the FPGA platform using dual-V_{dd}/dual-V_{th} fabrics was proposed by He [63][64]. Ishihara proposed the level converter required for dual-V_{dd} systems [65]. Srivastava has introduced technology that minimizes both switching and static power using simultaneous V_{dd} assignment [66]. Lee et al. studied the use of dual voltages by considering the requirement for power-network planning [67].

To the best of our knowledge, we report the first approach that combines pipelining and dual voltage assignment for the reduction of circuit power. Our dual-V_{dd} assignment method specifically does not require voltage converters inside each pipeline stage. Under this assumption, our algorithm guarantees significant power minimization using dual-V_{dd} assignment. The most important observation is that pipelining is used both for delay minimization as well as to improve the performance of dual-V_{dd} assignment. The effectiveness of our approach in this generalized technology can be further combined with unfolding and other sequential and combinational transformations.

9.2.3 Power and Delay Model

We build our gate level simulation model by using the delay and power models from Markovic et al. [68]. The delay model is shown in Equation (9.1), where k_{tp} is the delay-fitting parameter, C_L is load capacitance, V_{dd} is supply voltage, n is substreshold slope, μ is mobility, C_{ox} is oxide capacitance, W is gate width, L is effective channel length, $\phi_t = kT/q$ is thermal voltage, k_{fit} is a model-fitting parameter, σ is the drain induced barrier lowering (DIBL) factor, and V_{th} is threshold voltage. Load capacitance C_L is defined in Equation (9.2), where γ is the logical effort of the gate and W_{fanout} is the sum of the widths of the load gates.

$$D = \frac{k_{tp} \cdot C_L \cdot V_{dd}}{2 \cdot n \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot (\frac{kT}{q})^2} \cdot \frac{k_{fit}}{(\ln(e^{\frac{(1+\sigma)V_{dd}-V_{th}}{2 \cdot n \ \ cdot(kT/q)}}))^2} \tag{9.1}$$

$$C_L = C_{ox} \cdot L \cdot (\gamma \cdot W + W_{fanout}) \tag{9.2}$$

We have also considered two sources of power dissipation in an IC. One is from gate switching, in which the ICs dissipate power by charging the load capacitances of wires and gates. The other source is leakage power. Even if the gates do not switch, they dissipate power due to the leakage current. Equation (9.3) describes the leakage power model, and Equation (9.4) describes the gate-level switching power model, where α is the activity factor and f is the frequency.

$$P_{leakage} = 2 \cdot n \cdot \mu \cdot C_{ox} \cdot \frac{W}{L} \cdot (\frac{kT}{q})^2 \cdot V_{dd} \cdot e^{\frac{\sigma \cdot V_{dd} - V_{th}}{n \cdot (kT/q)}} \tag{9.3}$$

$$P_{switching} = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f \tag{9.4}$$

9.3 Optimization Flow

Our flow of optimization is shown in Figure 9.2. In the first stage, we apply gate level simulation to the benchmark circuits based on Markovic's delay and power model. Based on the achieved circuit model, we then apply pipelining to find out where to split the pipeline stages. Our pipelining has the following properties. (1) It does not compromise the performance of standard optimal pipelining. (2) It facilitates the dual-V_{dd} assignment. In the simulation, our pipelining is tested on the 2-stage situation, however, it is not limited to such, as it can be easily extended to an n-stage pipelining ($n > 2$). The last step in the flow is to assign dual-V_{dd} to the circuit which operates at the gate-level. Our cell V_{dd} selection strategy is by efficiently identifying gates to place at high voltage without violating the basic rules that only high V_{dd} gates can drive low V_{dd} gates. In the dual-V_{dd} optimization, we set a target delay and then assign dual voltages to meet the delay while optimizing the overall circuit power. Our algorithm returns both the optimal high/low voltages to apply as well their coverage on the circuit.

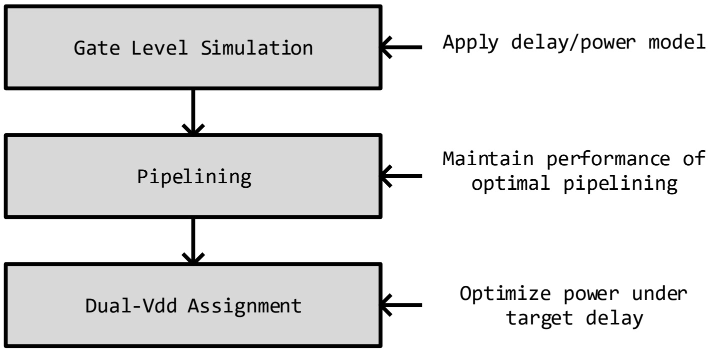

Gate Level Simulation
Pipelining
Dual-Vdd Assignment
Apply delay/power model
Maintain performance of optimal pipelining
Optimize power under target delay

Figure 9.2: Flow of optimization.

9.4 Pipelining

We discuss our pipelining design in this section. We have focused on the 2-stage pipelining on combinational circuits. Our whole process has two major steps, respectively "critical path identification" and "splitting pipeline stages".

The first step is to identify all the critical paths in the original circuit under the single V_{dd}. This can be easily achieved through the standard critical path identification method, such as dynamic programming proposed by Kirkpatrick [69]. The clock frequency of pipeline circuit is decided by the larger delay of the pipeline stages. Therefore, in order to achieve the optimal delay after pipelining, the gates on the critical paths should be evenly split into 2 groups in terms of delay and then distributed to the two pipeline stages. Assume that the original critical path delay is d_c, then the optimal delay can be achieved after pipelining is $d_c/2 + d_{FF}$, where d_{FF} is the delay of flip-flops.

However, the key remaining question is which pipeline stage should the non-critical path logic be assigned to. Figure 9.3 depicts an example circuit with critical paths represented by solid black lines. The optimal cuts on the critical paths are represented by the dashed lines in the first figure of 9.3(a) and 9.3(b), with which the original critical path delay is split into approximately half to half. Then two pipeline strategies are presented in the second figure of 9.3(a) and 9.3(b), where the key difference is the assignment of non-critical path logic. Note that we assume in both pipeline cases, the assignment on non-critical path logic does not introduce new critical paths. In other words, the delay in each pipeline stage stays to be approximate $d_c/2 + d_{FF}$.

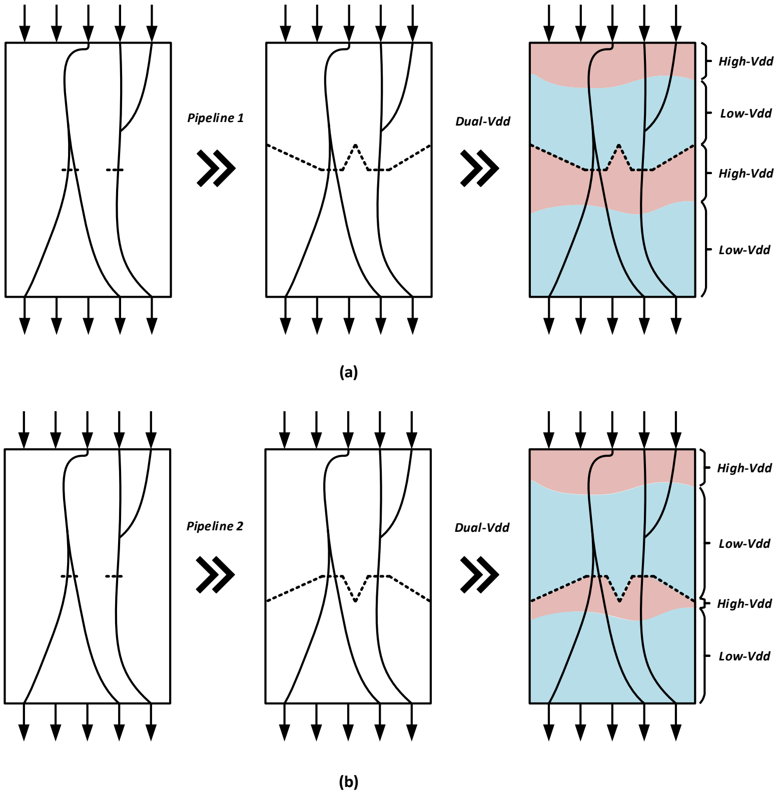


Figure 9.3: Comparison of dual-V_{dd} assignment on pipeline method 1 and pipeline method 2.

The dual-V_{dd} optimization is followed by the pipelining. We have observed a large difference in the dual-V_{dd} assignment between the pipelining in the last figure of 9.3(a) and 9.3(b). The number of gates assigned on high V_{dd} of the second stage pipeline in 9.3(a) is much larger than that of 9.3(b). This is because the critical path gates in the second stage pipeline are also driven by the non-critical path logic. Thus in order to put the critical path gates on high V_{dd}, all the non-critical path logic that provides fan-ins to them also need to be assigned to high V_{dd}. Nevertheless, to put the non-critical path logic on high V_{dd} will not further reduce the delay of pipeline stages, with the only consequence to increase the circuit power consumption.

Motivated by the above comparison, we summarize our key idea of pipeline design as following.

- Keep the optimal cut on the critical paths under the single V_{dd}.
- Include as much non-critical path logic as possible to the first pipeline stage without creating a new critical path.

With the above procedures, we guarantee that at the second pipeline stage, the gates on the critical path depend on as little non-critical path logic as possible since most of the logic has been included in the first pipeline stage. The level converters between stages allow such low V_{dd} gates to drive the high V_{dd} gates in the next stage. Consequently, when putting gates on the critical path to high V_{dd} in the second pipeline stage, the unnecessary power overhead from non-critical fan-ins is minimized.

The algorithmic process to find such pipeline design is presented in Algorithm 5. In the algorithm, we first identify all the critical paths $cp_i, (i \in \{0, 1, .., n\})$ in the original circuit, then we traverse all the gates on the critical paths with breadth-first search (BFS) to find cuts on the paths to minimize the maximum delay between the upper stage (S_1) and the lower stage (S_2) given a single V_{dd}. With the finalized cut on the critical paths, we then traverse through all the gates on the non-critical paths with BFS. As long as a gate does not create new critical paths for S_1, we incorporate it to S_1. Note that at the end of the algorithm, the delay of S_1 and the delay of S_2 should be as close as possible. And the sum of the two delays should equal to d_c plus the delay from the flip-flops.

Our pipeline design can be easily extended to the N-stage pipelining ($N > 2$). For Algorithm 5, the same idea can be applied to all the boundaries between pipeline stages. Assume pipeline stage i is more close to inputs, and its neighbor stage $i + 1$ is more close to outputs, then the goal is to push as much non-critical path logic from stage $i + 1$ to stage i as possible without creating a new critical path. So that for each pipeline stage (other than stage 1), fewer non-critical path gates need to be assigned to high voltage.

Algorithm 5 Pipelining

Input: C - original circuit.

Output: 2-stage pipelining on C. The gates in the first pipeline stage are in S_1 (close to inputs), and the gates in the second pipeline stage are in S_2 (close to outputs).

```
$S_1 = \varnothing$, $S_2 = \varnothing$, $d = 0$.
Identify all the critical paths $cp_i, (i \in \{0, 1, .., n\})$ in $C$ with delay $d_c$.
$d = d_c$.
Append all gates in $cp_i$ to $S_2$.
For all gates $g_j$ in $cp_i$ (inputs → outputs, BFS):
  If $g_j$ is directly driven by gates in $S_1$:
    If $Max(Delay(S_1 + g_j), Delay(S_2 - g_j)) < d$:
      $S_1$.append($g_j$).
      $S_2$.remove($g_j$).
      $d = Max(Delay(S_1), Delay(S_2))$.
    Else:
      Continue.
  Else:
    Continue.
$d_1 = Delay(S_1)$. $d_2 = Delay(S_2)$.
$S_2 = \{all\ gates\ in\ C - S_1\}$.
For all gates $g_j$ not in $cp_i$ (inputs → outputs, BFS):
  If $g_j$ is directly driven by gates in $S_1$:
    If $Delay(S_1 + g_j) = d_1$ and $Delay(S_2 - g_j) \geq d_2$:
      $S_1$.append($g_j$).
      $S_2$.remove($g_j$).
    Else:
      Continue.
  Else:
    Continue.
Return $S_1, S_2$.
```

9.5 Dual V_{dd} Assignment

We start this section by demonstrating the algorithmic flow of finding the optimal dual V_{dd} assignment for a single pipeline stage. Then we extend the proposed algorithm to find the optimal V_{dd} assignment on a circuit with multiple pipeline stages. In the second case, the selected voltage pair is shared among all the pipeline stages.

9.5.1 Dual V_{dd} Assignment on a Single Pipeline Stage

There exist two essential issues when applying dual-V_{dd} to circuits. The first is what voltages should be used, and the second is which part of the circuit should be assigned to high V_{dd} (or low V_{dd}). We assume that in our design, only the gates with high V_{dd} can drive the gates with low V_{dd}, thus we do not need to use the high-cost level converters inside each pipeline stage.

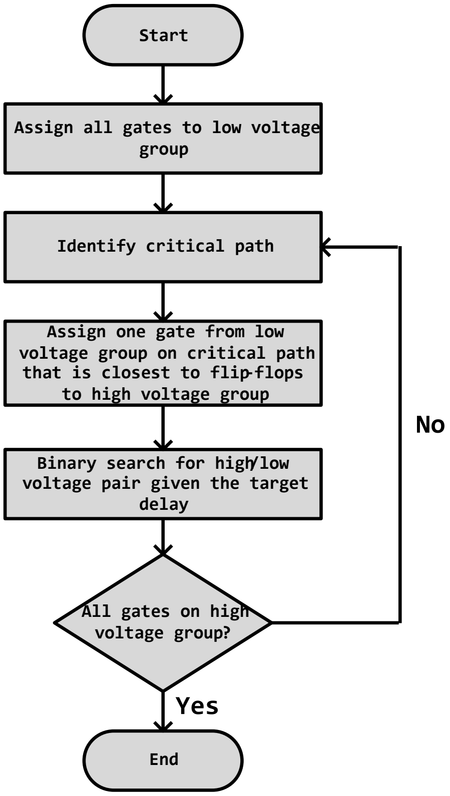

Start
Assign all gates to low voltage group
Identify critical path
Assign one gate from low voltage group on critical path that is closest to flip-flops to high voltage group
Binary search for high/low voltage pair given the target delay
All gates on high voltage group?
No
Yes
End

Figure 9.4: Algorithmic flow of dual voltage assignment.

To leverage the above two issues, we propose the algorithmic flow in Figure 9.4 to heuristically approximate the best voltage pairs and the corresponding coverage. We assume two gate sets in the circuit, respectively the high voltage set and the low voltage set. Initially, all gates in the circuit are in the low voltage set. In each iteration, we choose one gate which is on the current critical path with the shortest arrival time from the low voltage set and place it to the high voltage set. Afterwards, given the current high/low voltage set split, we use binary search to traverse the pairs of voltages that meet the given target delay and find the pair that achieves the smallest power consumption. We repeat these steps until all gates in the circuit have been placed in the high voltage group. From there, we choose the lowest point of power consumption from all the explored iterations. In practice, the minimum power is frequently achieved when only a small subset of gates are assigned to high V_{dd}.

An important point in the whole process is that the voltage for high voltage group and the voltage for low voltage group do not keep the same across iterations. Instead, they kept being changed with binary search as long as they met the following constraints. (1) Each pipeline stage must meet the target delay. (2) The voltage in high voltage group must be equal or higher than the voltage in low voltage group. That being said, the voltage used in the initial circuit will be the same as the voltage used at the end of the algorithm because all gates belong to a single voltage group.

An example flow of dual voltage assignment is shown on Figure 9.5. In each iteration, the new critical path of the circuit is circled with a dashed line. As shown in the example, one gate on the critical path of the low voltage group (blue cells) is set to high voltage group (red cells) in every iteration. The solution with the overall minimum power consumption is highlighted using a red rectangle.

Figure 9.6 depicts the performance of our algorithm on an example circuit. The tested circuit c880 has the following initial configuration: the total number of gates is 383, the initial V_{dd} is $0.7V$, the critical path delay is $3.67ps$, and the power is $798.2\mu W$. Note that this example does not involve any pipelining, the c880 circuit is considered as a whole single stage. In each iteration, one gate is switched from the low V_{dd} set to the high V_{dd} set. We

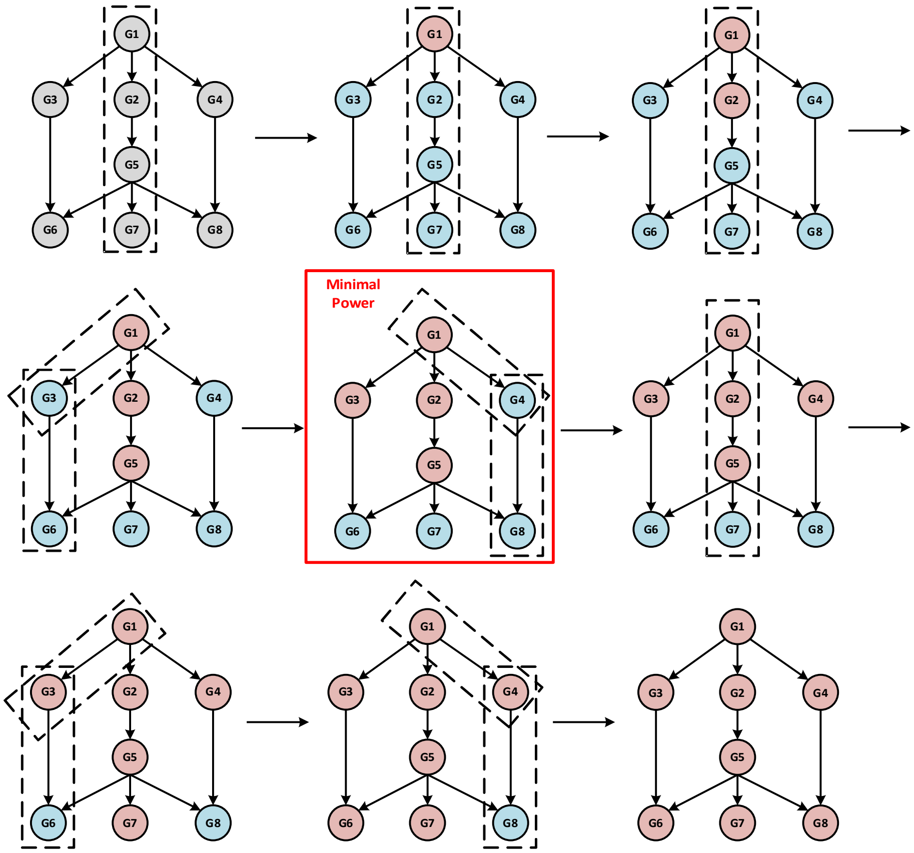


Figure 9.5: An example flow of dual voltage assignment.

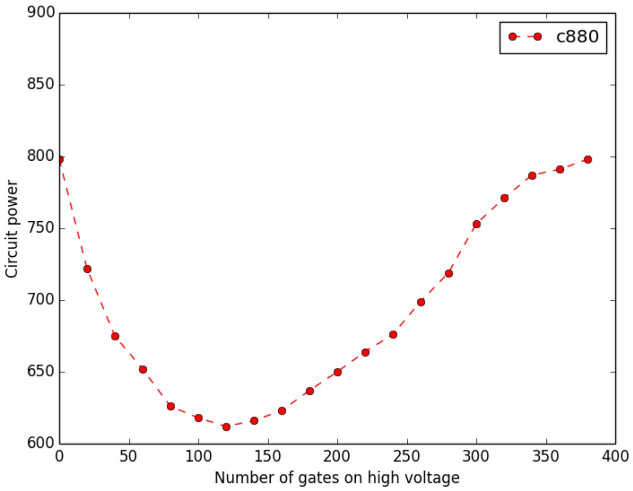


Figure 9.6: The performance of the dual-V_{dd} optimization algorithm on the c880 circuit.

observe from Figure 9.6 that the minimum power is achieved approximately at iteration 120. The circuit achieves its optimal power consumption $612\mu W$ with high V_{dd} set to $0.74V$ and low V_{dd} set to $0.62V$. The reason that the initial part of the iteration causes more energy reduction is that as more gates are assigned to high V_{dd}, the circuit becomes balanced in terms of critical paths, thus the marginal effect on energy reduction using dual-V_{dd} will decrease. Throughout the whole process, the target delay of the circuit is fixed, only the voltages are scaled to meet the target delay.

9.5.2 Dual V_{dd} Assignment on Multiple Pipeline Stages

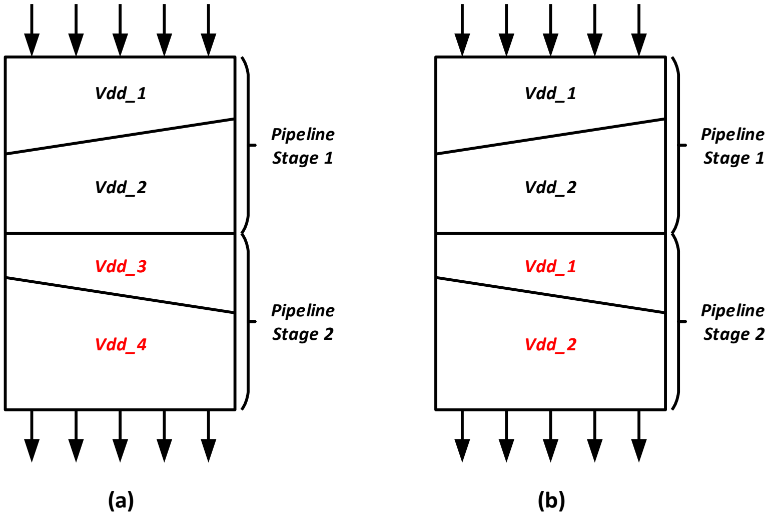


Figure 9.7: Two types of dual V_{dd} assignments on the pipeline circuit.

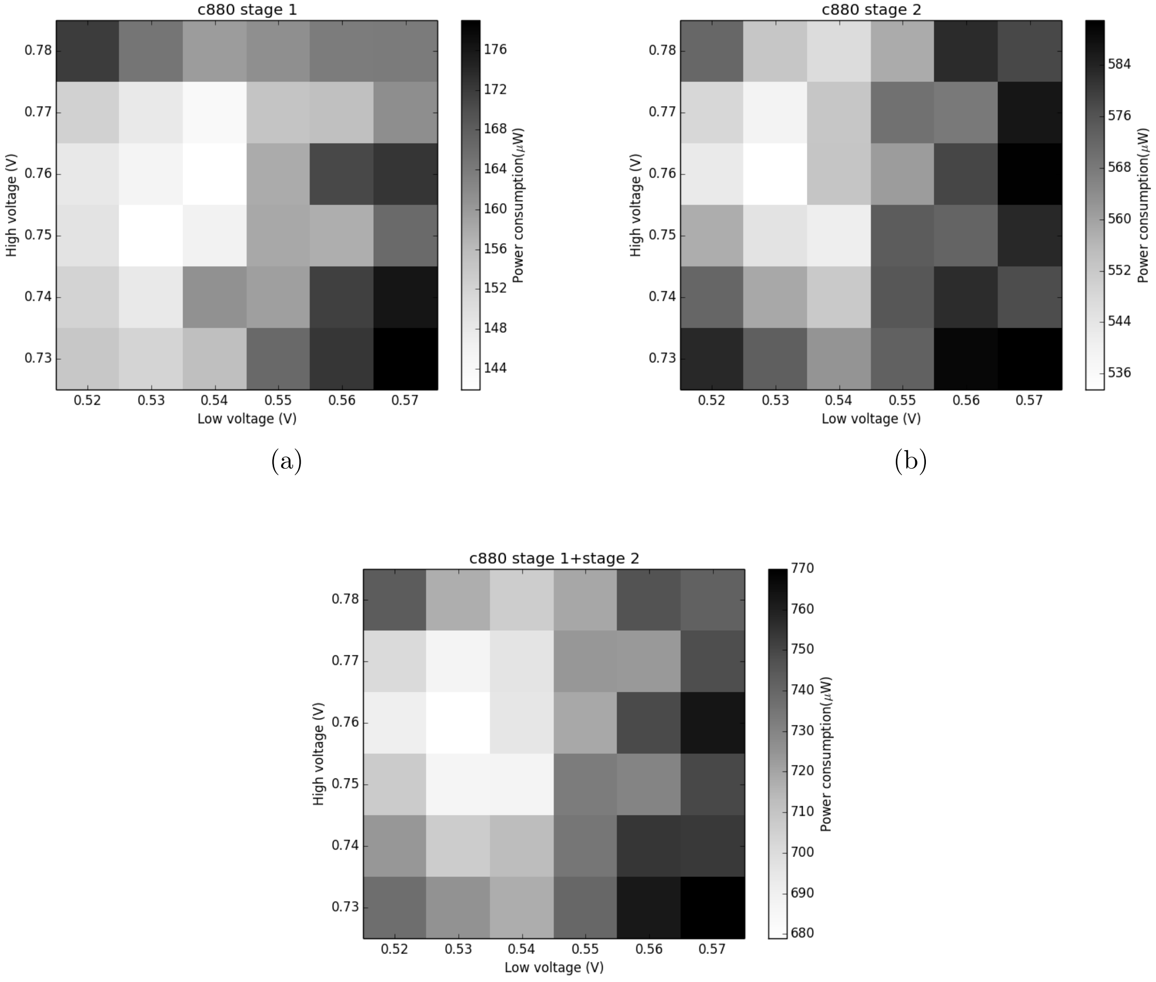
The key issue to consider in the dual-V_{dd} assignment on multiple pipeline stages is that only a single pair of high voltage and low voltage is allowed across the whole circuit (as shown in Figure 9.7(b)). However, if we simply apply the optimization flow on every single pipeline stage, then each of them will generate its optimal voltage pair as shown in Figure 9.7(a) and these pairs are not likely to equal to each other. The reason we only allow two voltages to be used is because it is always expensive in terms of design and manufacturing cost to add a new voltage to a circuit. We would show that it is not worth it to use more than two voltages because compared to using multiple pairs of voltages, a single pair of voltage only compromises a small percent of circuit performance.

To find the single optimal dual V_{dd} pair across the whole pipeline circuit, we extend the algorithmic flow presented in Figure 9.4. In the first step, we apply dual-V_{dd} assignment on each single pipeline stage, but instead of only recoding the optimal dual-V_{dd} pair for each stage, a pool of voltage pairs distinguished by certain voltage scales together with their power performance are all recorded. In the next step, an overall power evaluation is applied on each candidate voltage pair, and the pair with the least overall power is selected as the result pair.

Figure 9.8 presents the power evaluation process for different pairs of voltages applied to the c880 circuit. In Figure 9.8a, the dual-V_{dd} assignment algorithm is applied on the first stage of c880. The corresponding stage 1 power consumptions under different voltage pairs are plotted with the colormap. Similarly, the same process is applied to the stage 2 of c880, and the result is shown in Figure 9.8b. By adding the results of two stages together, we can easily generate the power consumption of the overall circuit for different voltage pairs, where the result is plotted in Figure 9.8c. We can clearly see that with the voltage pair $< 0.53V, 0.76V >$, the overall circuit employs the least power consumption.

The above approach can be extended to circuit designs with more than two pipeline stages. By applying the single stage dual-V_{dd} assignment algorithm to all n stages, and then adding up the power consumption under each voltage pair, the pair that achieves the globally minimum power consumption is easily discovered.

One question we have raised previously is that compared to using multiple pairs of voltages, how much performance compromise will be caused by applying only a single pair of voltages? For the example circuit c880, the minimum power of stage 1 is $141.9\mu W$ and is achieved at $< 0.53V, 0.75V >$ while the minimum power of stage 2 is $533.5\mu W$ and is achieved at $< 0.53V, 0.76V >$. If there is no restriction on the number of voltages we can apply on the whole circuit, then the minimum power globally is $141.9 + 533.5 = 675.4\mu W$. On the other hand, if only a single pair of voltage can be used, then the lowest power consumption is $678.9\mu W$ at voltage pair $< 0.53V, 0.76V >$, in which case, only 0.52% extra



(a)

(b)

(c)

Figure 9.8: Power consumption of (a) stage 1, (b) stage 2, (c) stage 1 plus 2 of circuit c880 on different high/low voltage pairs.

power consumption is required compared to the case of multiple voltage pairs.

9.6 Experiments

We demonstrate our experimental results on our proposed approach in this section. We first explain our experiment setup, including the cell library, power model, as well as the benchmarks we are using. Then we present the power/delay results on benchmark circuits to illustrate the effectiveness of our optimization algorithm.

9.6.1 Experiment Setup

We have used the ISPD2012 standard cell library [70] and fit it accordingly to Markovic's delay and power model. We set the initial V_{dd} to 0.7V and the threshold voltage V_{th} to 0.3V. For our dual-V_{dd} optimization, we consider high and low V_{dd} within the range from 0.4V to 1.0V. We only consider the 2-stage pipelining in our experiment.

We evaluate a subset of ISCAS85 benchmarks and synthesize each netlist using Cadence Encounter to generate parasitics capacitances for all considered netlists [71]. We develop an in-house timer in C++ for flexibility and robustness in computing load and slew dependent delays. We start from the characterization using the cell library, afterwards, we use the gate-level simulation to quantify the delay, switching power, and leakage power.

In the next step, we have designed 3 circuit configurations to iteratively evaluate the performance of our algorithm. The first configuration is when the whole circuit is under the initial V_{dd}. The critical path delay of configuration 1 is recorded and the half of critical path delay is set as the target delay for each single stage in pipelining. The second configuration is to apply a baseline pipelining algorithm together with the dual-V_{dd} assignment on the circuit. Our baseline pipelining is designed to split stages at the half of circuit critical paths. The major difference compared to our pipelining design is that it randomly decides which stage the non-critical path logic belongs to. It serves as a reference to validate the effectiveness of our pipelining algorithm. The third configuration is to apply our pipelining with dual-V_{dd} assignment on the circuit. Note that for all the circuit states, the pipelining stages are optimized under the same target delay. Therefore, we use the circuit power consumption as an indicator of performance comparison.

9.6.2 Experimental Results

We test the power consumption of each benchmark circuit under the three configurations, pipelining with initial V_{dd} (C_1), baseline pipelining with dual-V_{dd} (C_2), and our pipelining with dual-V_{dd} (C_3). The results are presented in Table 9.1. Note that the low/high voltage pairs presented in the table are calculated based on the dual-V_{dd} assignment algorithm in Figure 9.4. Compared to the power consumption under initial V_{dd}, the introduce of dual-V_{dd} together with baseline pipelining can provide 3.9% to 18.5% (Avg. 11.4%) power saving. Furthermore, when comparing our pipelining with the baseline pipelining under the dual V_{dd} optimization, the percentage of power saving is from 0% to 24.0% (Avg. 10.4%). We draw the following conclusions from the results.

First of all, the power consumption of our pipeline design achieves better performance compared to the baseline pipeline design when combined with the dual-V_{dd} optimization. The power saving from C_2 to C_3 is even comparable to the power saving from C_1 to C_2. In all the tested circuits, our pipelining algorithm is at least equally well performed as the baseline standard pipelining algorithm regarding power consumption.

Secondly, our pipelining achieves more power saving when the circuit has a larger number of gates. It is because for a larger circuit, there normally exists more non-critical path logic near the half-to-half critical path cut. In our pipelining, such logic is merged to the first stage of pipelining. The gates in such logic are now switched from high V_{dd} to low V_{dd} without compromising circuit delay. As the more gates are switched, the more significant the circuit power reduction will be. From the results, we can clearly see that if many more gates are assigned to the upper stage in our pipelining than the baseline pipelining, it normally suggests that the difference of power saving between the two pipeline designs is large.

Lastly, there also exists some situation when our pipelining helps little compared to the baseline pipelining, such as circuit c499 and c880 where the power savings are 0% and 0.7%. When taking a closer look at both circuits, it is because the area of the circuit near the half-to-half critical path cut is ultra "thin", which in other words, exists almost no non-critical path logic. As a result, the stage split between the baseline pipelining and our pipelining stays almost the same, causing no further power saving.

Circuit	**c432**	**c499**	**c880**	**c1355**	**c1908**
Number of gates	160	202	383	546	880
Target delay(ns)	2.03	2.12	1.91	2.16	2.47
C_1-Power(μW)	374.8	536.9	798.2	1158.5	1816.7
C_2-Power(μW)	342.4	516.2	683.8	1067.3	1480.4
C_2-$< low\ V_{dd}, high\ V_{dd} >(V)$	$< 0.55, 0.76 >$	$< 0.52, 0.75 >$	$< 0.54, 0.74 >$	$< 0.55, 0.76 >$	$< 0.55, 0.75 >$
C_2-# of gates:<stage1, stage2>	$< 58, 102 >$	$< 80, 122 >$	$< 74, 309 >$	$< 296, 250 >$	$< 435, 445 >$
C_3-Power(μW)	282.1	516.2	678.9	955.3	1434.8
C_3-$< low\ V_{dd}, high\ V_{dd} >(V)$	$< 0.51, 0.77 >$	$< 0.52, 0.75 >$	$< 0.53, 0.76 >$	$< 0.55, 0.76 >$	$< 0.54, 0.75 >$
C_3-# of gates: <stage1, stage2>	$< 126, 34 >$	$< 80, 122 >$	$< 82, 301 >$	$< 375, 171 >$	$< 513, 367 >$
Power Save-C_1 to C_2	8.6%	3.9%	14.3%	7.9%	18.5%
Power Save-C_2 to C_3	17.6%	0%	0.7%	10.5%	3.1%

Circuit	**c2670**	**c3540**	**c5315**	**c6288**	**c7552**
Number of gates	1193	1669	2307	2416	3512
Target delay(ns)	3.09	3.54	3.38	13.58	2.71
C_1-Power(μW)	2355.8	3339.9	4582.0	5376.8	7026.1
C_2-Power(μW)	1925.2	3001.3	3857.9	5119.9	6186.2
C_2-$< low\ V_{dd}, high\ V_{dd} >(V)$	$< 0.51, 0.77 >$	$< 0.53, 0.75 >$	$< 0.57, 0.76 >$	$< 0.53, 0.75 >$	$< 0.55, 0.75 >$
C_2-# of gates:<stage1, stage2>	$< 393, 800 >$	$< 507, 1162 >$	$< 398, 1909 >$	$< 1453, 963 >$	$< 723, 2789 >$
C_3-Power(μW)	1667.6	2281.8	3234.5	4873.5	5322.9
C_3-$< low\ V_{dd}, high\ V_{dd} >(V)$	$< 0.53, 0.76 >$	$< 0.54, 0.76 >$	$< 0.55, 0.74 >$	$< 0.53, 0.75 >$	$< 0.52, 0.76 >$
C_3-# of gates: <stage1, stage2>	$< 736, 457 >$	$< 898, 771 >$	$< 1081, 1226 >$	$< 1809, 607 >$	$< 2368, 1144 >$
Power Save-C_1 to C_2	18.3%	10.1%	15.8%	4.8%	12.0%
Power Save-C_2 to C_3	13.4%	24.0%	16.2%	4.8%	14.0%

Table 9.1: Configuration 1(C_1): initial V_{dd}. Configuration 2(C_2): baseline pipelining+dual V_{dd}. Configuration 3(C_3): our pipelining+dual V_{dd}. Experimental results of ISCAS-85 benchmark circuits with 2-stage pipelining and dual-V_{dd} optimization. Stage 1 represents the pipeline stage close to inputs, and stage 2 represents the pipeline stage close to outputs.

9.7 Summary

A novel design of pipelining is proposed to facilitate dual-V_{dd} assignment to enable low power consumption in ICs. While our pipeline design maintains the optimal pipeline delay on the circuit, it allocates as much non-critical path logic as possible from high V_{dd} to low V_{dd} to reduce power consumption. Our approach is by far the first attempt to optimize circuit pipelining in conjunction with dual-V_{dd} optimization. We have tested our approach on the ISCAS-85 benchmark circuits, and the results suggest that in all benchmark circuits, our pipelining algorithm performs at least equally well as the standard baseline pipelining algorithm after dual-V_{dd} assignment. An average power saving of 10.4% is observed when comparing our pipelining to the baseline pipelining.

CHAPTER 10

Concluding Remarks

The development of IoT has dramatically changed the way people live and the trend will continue. Numerous opportunities, as well as challenges, exist in regards to the design, manufacturing, and applications of IoT. In this thesis, we sought for novel hardware solutions to specifically address the aspects of security, reliability and power consumption in IoT designs. While traditional security approaches are able to provide elegant and well-defined solutions, IoT systems have imposed a set of new requirements that can not be fully leveraged. In terms of architecture, security designs should employ ultra-low power, compact area, and resistance against side-channel attacks. In terms of applications, not only all the standard protocols such as public/private communication, authentication, but also more emerging IoT applications e.g., remote trust, low-overhead fault tolerance need to be addressed.

In this thesis, we proposed hardware-oriented solutions to meet the above requirements. We emphasized especially on the domain of hardware security primitives represented by PUFs. The first effort is to analyze and improve the analog PUFs. Our key observation is that the standard arbiter PUFs can be easily characterized and emulated. Based on the PUF characterization, we added programmable delay lines to match two arbiter PUFs, in such a way that private key communication and authentication protocols are enabled.

While realizing the problem of instability and incompatibility of analog PUFs can never be fully eliminated, we took the next effort to develop digital security primitives. We began with the design of security primitives for public key and private key communications. Both of the protocols are addressed with ultra-low energy. Then we raised the design criterion to not only digital, but also unclonable. Therefore, the concept of digital PUF is proposed along with two conceptually new architectural designs.

Reliability is equally important as security. The second domain we have worked on is to develop low-overhead solutions to solve IoT reliability. Specifically, we employed an NVM-based checkpoint technique to keep track of the program flow and store intermediate state for fast recovery.

After focusing on security and reliability, we discussed and explored energy optimization on IoT from the circuit level. We have improved and combined traditional circuit synthesis techniques: pipelining with dual-supply voltages assignment to jointly optimize circuit energy.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: a survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] "Gartner says 6.4 billion connected things will be in use in 2016, up 30 percent from 2015." `http://www.gartner.com/newsroom/id/3165317`, 2016.

[3] "IoT security needs scalable solutions." `https://techcrunch.com/2016/03/01/iot-security-needs-scalable-solutions/`, 2016.

[4] A. Juels, "RFID security and privacy: a research survey," *IEEE journal on selected areas in communications*, vol. 24, no. 2, pp. 381–394, 2006.

[5] J.-P. Vasseur and A. Dunkels, *Interconnecting smart objects with IP: The next Internet.* Morgan Kaufmann, 2010.

[6] J. Hui, D. Culler, and S. Chakrabarti, "6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture," *Internet Protocol for Smart Objects (IPSO) Alliance White Paper*, no. 3, 2009.

[7] `http://www.verayo.com/#internet-of-things`.

[8] `https://www.intrinsic-id.com/solutions/sram-puf-key-storage/hardware-ip/`.

[9] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer, "High-performance CMOS variability in the 65-nm regime and beyond," *IBM journal of research and development*, vol. 50, no. 4.5, pp. 433–449, 2006.

[10] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[11] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 148–160, ACM, 2002.

[12] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th annual Design Automation Conference*, pp. 9–14, ACM, 2007.

[13] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *International workshop on Cryptographic Hardware and Embedded Systems*, pp. 63–80, Springer, 2007.

[14] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pp. 1–6, IEEE, 2010.

[15] L. Bolotnyy and G. Robins, "Physically unclonable function-based security and privacy in RFID systems," in *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pp. 211–220, IEEE, 2007.

[16] U. Rührmair, "SIMPL systems, or: can we design cryptographic hardware without secret key information?," in *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 26–45, Springer, 2011.

[17] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pp. 176–179, IEEE, 2004.

[18] 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*, pp. 237–249, ACM, 2010.

[19] T. Hui and R. W. Mounger, "Programmable delay line," Aug. 3 1999. US Patent 5,933,039.

[20] A. Raychowdhury, S. Ghosh, and K. Roy, "A novel on-chip delay measurement hardware for efficient speed-binning," in *On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International*, pp. 287–292, IEEE, 2005.

[21] M.-C. Tsai, C.-H. Cheng, and C.-M. Yang, "An all-digital high-precision built-in delay time measurement circuit," in *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*, pp. 249–254, IEEE, 2008.

[22] M. Majzoobi, E. Dyer, A. Elnably, and F. Koushanfar, "Rapid FPGA characterization using clock synthesis and signal sparsity," in *International Test Conference (ITC)*, pp. 1–10, 2010.

[23] P. Yalla and J.-P. Kaps, "Lightweight cryptography for FPGAs," in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*, pp. 225–230, IEEE, 2009.

[24] J. Soto, "Statistical testing of random number generators," in *Proceedings of the 22nd National Information Systems Security Conference*, vol. 10, p. 12, NIST Gaithersburg, MD, 1999.

[25] E. Öksüzoglu and E. Savas, "Parametric, secure and compact implementation of RSA on FPGA," in *Reconfigurable Computing and FPGAs, 2008. ReConFig'08. International Conference on*, pp. 391–396, IEEE, 2008.

[26] N. Beckmann and M. Potkonjak, “Hardware-based public-key cryptography with public physically unclonable functions,” in *International Workshop on Information Hiding*, pp. 206–220, Springer, 2009.

[27] C. E. Shannon, “Communication theory of secrecy systems,” *Bell Labs Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[28] D. Boneh, R. A. DeMillo, and R. J. Lipton, “On the importance of checking cryptographic protocols for faults,” in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 37–51, Springer, 1997.

[29] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, “Fault injection attacks on cryptographic devices: theory, practice, and countermeasures,” *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[30] C. H. Stapper, “Modeling of integrated circuit defect sensitivities,” *IBM Journal of Research and Development*, vol. 27, no. 6, pp. 549–557, 1983.

[31] S. Mittal, J. S. Vetter, and D. Li, “A survey of architectural approaches for managing embedded DRAM and non-volatile on-chip caches,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 6, pp. 1524–1537, 2015.

[32] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “Energy reduction for STT-RAM using early write termination,” in *Proceedings of the 2009 International Conference on Computer-Aided Design*, pp. 264–268, ACM, 2009.

[33] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, “Hybrid cache architecture with disparate memory technologies,” in *ACM SIGARCH computer architecture news*, vol. 37, pp. 34–45, ACM, 2009.

[34] E. F. Moore and C. E. Shannon, “Reliable circuits using less reliable relays,” *Journal of the Franklin Institute*, vol. 262, no. 3, pp. 191–208, 1956.

[35] D. Siewiorek and R. Swarz, *Reliable Computer Systems: Design and Evaluatuion.* Digital Press, 2014.

[36] W. H. Pierce, *Failure-tolerant computer design.* Academic Press, 2014.

[37] T. Karnik and P. Hazucha, “Characterization of soft errors caused by single event upsets in CMOS processes,” *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 2, pp. 128–143, 2004.

[38] S. Kannan, A. Gavrilovska, K. Schwan, and D. Milojicic, “Optimizing checkpoints using NVM as virtual memory,” in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 29–40, IEEE, 2013.

[39] J. Ferrante, K. J. Ottenstein, and J. D. Warren, “The program dependence graph and its use in optimization,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319–349, 1987.

[40] M. Weiser, "Program slicing," in *Proceedings of the 5th international conference on Software engineering*, pp. 439–449, IEEE Press, 1981.

[41] M. Weiser, "Programmers use slices when debugging," *Communications of the ACM*, vol. 25, no. 7, pp. 446–452, 1982.

[42] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, p. 6, 2014.

[43] D. J. Sorin, M. M. Martin, M. D. Hill, and D. A. Wood, "SafetyNet: improving the availability of shared memory multiprocessors with global checkpoint/recovery," in *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pp. 123–134, IEEE, 2002.

[44] D. Kirovski, M. Potkonjak, and L. M. Guerra, "Cut-based functional debugging for programmable systems-on-chip," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 1, pp. 40–51, 2000.

[45] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows," tech. rep., DTIC Document, 1988.

[46] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity.* Courier Corporation, 1982.

[47] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1-6, pp. 5–35, 1991.

[48] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative dataflow programs via optimum unfolding," *Computers, IEEE Transactions on*, vol. 40, no. 2, pp. 178–195, 1991.

[49] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 3–14, IEEE Computer Society, 2007.

[50] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (L 3 Cs): low-leakage SRAM, low write-energy STT-RAM, and refresh-optimized eDRAM," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 143–154, IEEE, 2013.

[51] K. Usami, M. Igarashi, F. Minami, T. Ishikawa, M. Kanzawa, M. Ichida, and K. Nogami, "Automated low-power technique exploiting multiple supply voltages applied to a media processor," *Solid-State Circuits, IEEE Journal of*, vol. 33, no. 3, pp. 463–472, 1998.

[52] M. J. Flynn, "Computer Architecture Pipelined and Parallel Processor Design, 1995."

[53] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[54] D. Kroening and W. J. Paul, “Automated pipeline design,” in *Design Automation Conference, 2001. Proceedings*, pp. 810–815, IEEE, 2001.

[55] J. Cong, Y. Fan, and Z. Zhang, “Architecture-level synthesis for automatic interconnect pipelining,” in *Proceedings of the 41st annual Design Automation Conference*, pp. 602–607, ACM, 2004.

[56] M. Galceran-Oms, J. Cortadella, D. Bufistov, and M. Kishinevsky, “Automatic microarchitectural pipelining,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 961–964, European Design and Automation Association, 2010.

[57] Y. Ma, Z. Li, J. Cong, X. Hong, G. Reinman, S. Dong, and Q. Zhou, “Micro-architecture pipelining optimization with throughput-aware floorplanning,” in *Design Automation Conference, 2007. ASP-DAC'07. Asia and South Pacific*, pp. 920–925, IEEE, 2007.

[58] E. Nurvitadhi, J. C. Hoe, S.-L. L. Lu, and T. Kam, “Automatic multithreaded pipeline synthesis from transactional datapath specifications,” in *Proceedings of the 47th Design Automation Conference*, pp. 314–319, ACM, 2010.

[59] K. Usami and M. Horowitz, “Clustered voltage scaling technique for low-power design,” in *Proceedings of the 1995 international symposium on Low power design*, pp. 3–8, ACM, 1995.

[60] M. Igarashi, K. Usami, K. Nogami, F. Minami, Y. Kawasaki, T. Aoki, M. Takano, C. Mizuno, T. Ishikawa, M. Kanazawa, *et al.*, “A low-power design method using multiple supply voltages,” in *Proceedings of the 1997 international symposium on Low power electronics and design*, pp. 36–41, ACM, 1997.

[61] S. Raje and M. Sarrafzadeh, “Variable voltage scheduling,” in *Proceedings of the 1995 international symposium on Low power design*, pp. 9–14, ACM, 1995.

[62] J.-M. Chang and M. Pedram, “Energy minimization using multiple supply voltages,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 5, no. 4, pp. 436–443, 1997.

[63] F. Li, Y. Lin, L. He, and J. Cong, “Low-power FPGA using pre-defined dual-Vdd/dual-Vt fabrics,” in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pp. 42–50, ACM, 2004.

[64] Y. Lin and L. He, “Statistical dual-Vdd assignment for FPGA interconnect power reduction,” in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pp. 1–6, IEEE, 2007.

[65] F. Ishihara, F. Sheikh, and B. Nikolić, “Level conversion for dual-supply systems,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 185–195, 2004.

[66] A. Srivastava and D. Sylvester, “Minimizing total power by simultaneous Vdd/Vth assignment,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 5, pp. 665–677, 2004.

[67] W.-P. Lee, H.-Y. Liu, and Y.-W. Chang, “An ILP algorithm for post-floorplanning voltage-island generation considering power-network planning,” in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pp. 650–655, IEEE, 2007.

[68] D. Marković, C. C. Wang, L. P. Alarcon, T.-T. Liu, and J. M. Rabaey, “Ultralow-power design in near-threshold region,” *Proceedings of the IEEE*, vol. 98, no. 2, pp. 237–252, 2010.

[69] T. Kirkpatrick and N. Clark, “PERT as an aid to logic design,” *IBM Journal of Research and Development*, vol. 10, no. 2, pp. 135–141, 1966.

[70] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, “The ISPD-2012 discrete cell sizing contest and benchmark suite,” in *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, pp. 161–164, ACM, 2012.

[71] F. Brglez, “A neutral netlist of 10 combinational benchmark circuits and a target translation in FORTRAN,” in *ISCAS-85*, 1985.