Towards Embedded System Hardware Security Design and Analysis

Yanping Gong
University of Connecticut - Storrs, yanping.gong@uconn.edu

Follow this and additional works at: https://opencommons.uconn.edu/dissertations

Recommended Citation
https://opencommons.uconn.edu/dissertations/2354
Towards Embedded System Hardware Security Design and Analysis

Yanping Gong, Ph.D.
University of Connecticut, 2019

ABSTRACT

Security in embedded system design, which has long been a critical problem for ensuring the confidentiality, data integrity and system reliability for embedded system designers and users, is now facing a new dimension of threat from the attacks on hardware. As the IC design reaches sub-micron regime, increased sensitivity of device under environmental condition has made some new types of attacks possible, while the analysis and detection for design vulnerabilities against these attacks are harder on the much more complicated designs nowadays. In the meanwhile, more efficient and diverse attack methodologies are developed by attackers as the technology advances. On the other hand, embedded system has limitations on the hardware resources and power consumption which can be allocated for preventive or defensive countermeasures. The future trends of system development, including cloud computing, distributed network and internet-of-things (IoT) are also pushing the edge of such limitations on embedded system designs. Low cost, high efficiency, and flexible hardware security design methodologies are needed for the current IC production flow as well as the future application scenarios. In this thesis, we’re presenting several efforts made towards low cost and high efficiency embedded hardware security design and
analysis. First, the finite state machine based circuit vulnerability analysis framework is proposed. Second, we demonstrated a secure scan architecture design which utilizes novel property of memristor devices. Lastly, a side channel resilience design methodology is presented for FPGA bitstream protection.
Towards Embedded System Hardware Security Design and Analysis

Yanping Gong

B.S., Xi’an Jiaotong University, 2012

A Dissertation
Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy at the University of Connecticut

2019
Towards Embedded System Hardware Security Design and Analysis

Presented by
Yanping Gong, B.S.

Major Advisor
__________________________
Lei Wang

Associate Advisor
__________________________
John Chandy

Associate Advisor
__________________________
Zhijie Shi

University of Connecticut
2019
ACKNOWLEDGMENTS

First of all, I would like to give my deepest gratitude to my advisor, Dr. Lei Wang for his guidance through the whole time. He helped me from finding the field of interest, developing theories, establishing experiments to my every academic writing. This thesis wouldn’t be completed without his immense knowledge in the Electrical Engineering field and beyond, extraordinary patience and continuous support. His insights and timely feedback often became the turning point when I encounter difficulties. Conducting research which involves broad subjects and in depth analysis couldn’t be as simple as he made this one for me. Besides, I found my interests on the study of embedded system security development, and has been motivated to carry on this topic as a long-time career all thank to his advises. His enthusiasm and curiosity to the open questions always inspire me to realize and understand the beauty and importance of Electrical Engineering.

Secondly, I would like to express my great appreciation to the rest of my committee members: Dr. John Chandy, Dr. Zhijie (Jerry) Shi, and committee witness members: Dr. Faquir Jain and Dr. Abhishek Dutta. Your insightful comments, feedback and suggestions were invaluable to the completion of this thesis. Andd also, I want to thank you for making every communication to be a very pleasant learning process. It’s my greatest pleasure to have such a respectable, professional and considerate committee.

My thanks should also go to my dearest friend, husband and lab mate Fengyu
Qian, who has always been there for me for the past ten years. He’s the only one who is helping on my work, studies and daily life. As a friend and family, he is always encouraging and influencing me with his enthusiasm and positive attitude. As a co-worker, he is professional, and pleasant to work with. He also greatly contributed on this study with his undoubted talent, solid knowledge and insightful ideas. To work on this thesis without his support would be unimaginable.

Last but not the least, I would like to thank everyone in my family, who has been incredibly supportive during this whole time from 12 time-zones away. We are separated by the longest distance on Earth, however, you managed to make me feel like you’re by my side. This would not been possible without your deepest caring and the most generous love. Those countless airport send-offs and evening video calls are the solid backing for me to overcome any difficulties, and all of my happiness and achievements should be shared by you all.
# Contents

1 Introduction

1.1 Embedded system security background ........................................ 2
   1.1.1 Security primitives ................................................. 2
   1.1.2 Multi-layered security system ....................................... 4

1.2 Scope of this thesis .......................................................... 6

1.3 Hardware attack models ...................................................... 6
   1.3.1 Invasive attack ......................................................... 6
   1.3.2 Non-invasive attack .................................................... 7

1.4 Embedded system hardware security design state-of-art .................. 11
   1.4.1 Security methodology in modern IC design flow .................... 11
   1.4.2 Design challenge and research state-of-art ......................... 13
   1.4.3 Physical-Unclonable-Function (PUF) and device entropy .......... 16
   1.4.4 Obfuscation and masking ............................................ 17

1.5 Security in programmable device .......................................... 19

1.6 Outline of the Dissertation ................................................ 20

1.7 Publications ....................................................................... 21

2 Probabilistic Evaluation of Hardware Security Vulnerabilities ......... 24

2.1 Introduction ....................................................................... 24

2.2 Related Work .................................................................... 26

2.3 Motivation and Attack Analysis ............................................ 29
   2.3.1 FSM Properties ......................................................... 29
   2.3.2 Attack Model ............................................................. 30

2.4 Analysis of Inner State Transitions ....................................... 34
   2.4.1 State Transitions ......................................................... 34
   2.4.2 State Transitions under Fault Injection Attacks .................... 35
3 Design for Test and Hardware Security Utilizing Retention Loss of Memristors

3.1 Introduction
3.2 Countermeasures for Scan-based Attacks
3.3 memristor devices and drifting effect
   3.3.1 Memristor devices
   3.3.2 Drifting effect
3.4 Memristor-based Scan Security Enhancement
   3.4.1 Overall architecture
   3.4.2 Scan obfuscation
   3.4.3 Security, endurance and reliability
3.5 Scan recovery
3.6 Circuit Implementation
3.7 Simulation Results
   3.7.1 Performance analysis
   3.7.2 Overhead analysis
   3.7.3 Security analysis
3.8 Conclusion

4 Masked FPGA bitstream encryption via partial reconfiguration

4.1 Introduction
4.2 Preliminaries
4.3 Partial reconfiguration based FPGA bitstream decryption
   4.3.1 Motivation
   4.3.2 The Proposed Design
4.4 Implementation and Results
   4.4.1 Circuit Implementation
   4.4.2 Resource Utilization
   4.4.3 Performance Analysis
Chapter 1

Introduction

The security of embedded system has been drawing a lot of attention especially as the form of the systems become more and more diverse and ubiquitous. Ranging from cell phones, computers, network routers, to smart dishwashers, networked sensors, smart vehicles and wearable devices, the current embedded systems are providing critical functionalities which are prone to be sabotaged and to cause serious harm. These systems are used to capture, store, process and transfer sensitive data, and hence pose unique challenges to the security designers. In the past decades, numerous efforts have been made to define the concepts and address the challenges of “Cyber Security”, “Information and Privacy”, “Critical Infrastructure”, “Networks” and so forth. Comprehensive security technologies including cryptotographics, authentication and security protocols, anti-virus mechanism and firewalls have been well established and understood. However, the never-stopping development of embedded system is always adding new security challenges. On the other hand, while the major security concerns are solved from software and protocol perspectives, recently there are more
attacks reported by taking advantage of hardware design defects and vulnerabilities. The hardware security of embedded system has rapidly become a new and critical design dimension where research effort is required.

1.1 Embedded system security background

The security concern in an embedded system varies between different entities and different user case scenarios. For instance, the end-users are often concerned about the privacy of personal data stored on their device and sent to the server, or whether the downloaded content is from trusted source; While the content or service provider might be more concerned with the secrecy of the proprietary and copyrights. The same entity can play multiple roles depending on the different user cases. The security requirement for embedded system design is the combination of all these requirements that apply.

1.1.1 Security primitives

With a variety of tools and technologies, attackers can break into embedded systems from hardware or software, and cause undesired consequences such as system malfunctioning, information leakage, privilege abuse and denial of service. The target of reported attacks often includes the cryptographic key, firmware, data storage, system configuration and identification. These are the security assets which need to be protected by security designers. For example, the encryption/decryption key of a cryptographic system is often stored on-chip in non-volatile memory, such as Read-Only Flash. However, the content of the key can be revealed by analyzing
the algorithm, eavesdropping data transmission or reverse-engineering. Once the key is revealed, the entire authentication mechanism will be bypassed. And as for the firmware, which includes low level instructions and configurations that need to be protected both by its content and the updating process. The Fig. 1.1 presents the typical components of a modern embedded system and where the targeted security assets are located. Many critical assets, such as cryptographic key or user data are subject to attacks at many different locations in the system when different attack model is applied.
1.1.2 Multi-layered security system

In order to meet the security requirements from different aspects, modern embedded system needs multi-layered protection mechanism. First, the Root-of-Trust (RoT) needs to be established which ensures the booting process starts from a trusted source. The RoT can be built on the strategies such as by using one-time or non writable memory, or by defining trusted memory zone. For example, the Intel Boot Guard technology utilize cryptographic to verify the initial boot block, and the manufacturer is required to generate key bits for the initial boot verification. Second, secure key protection and management is critical to the security of a system. Popular mechanisms, such as Trusted Platform Module (TPM) is used to store, protect and manage keys used for encryption or authentication purposes. But the key management still remains as an open problem when it comes to more complicated scenarios, such as in Internet-of-Things (IoT), or in distributed systems.

The third layer of the system security is to protect the processor and memory space, as well as the data storage. Technologies should be applied to prevent malfunctions injected by malicious users, such as stack overrun or buffer overflow. Some common security mechanisms are designed at this layer, such as Address Layout Randomization, where the virtual memory space layout is randomized in order to prevent attackers to understand the memory and hence deliberately force calls to cause buffer-overflow; And also the memory space can be marked with different executable configurations where certain area is reserved only for verified programs. On the other hand, the data storage and transmission need to be protected by cryptography modules and other security mechanisms. Designs with different security requirements should be specified with different levels of compliance. For instance, the FIPS 140-2(Federal
Information Processing Standard) specifies four levels of compliance:

- Level 1, basic security mechanism with only cryptographic modules and no physical security mechanism required;
- Level 2, improved from level 1 and requires features which shows evidence of tampering, such as using tamper-evident coatings;
- Level 3, tamper detection and response design is required which attempts to prevent intruder from gaining access to critical security parameters;
- Level 4, highest level of security. Tampering from any direction is protected at high probability. And besides, the system is protected from failures due to rigorous environment condition or environmental fluctuations.

Lastly, the embedded systems need to be protected on the physical layer, which is also the main concern of this thesis. As modern embedded system has more ubiquitous forms than before, it’s becoming easier for malicious parties to gain access to the physical devices. For example, internet of things (IoT) devices in many scenarios are placed remotely without any safeguards. Once the on-premises device is comprised, the chain effect could be spread through network. In the meanwhile, physical possession of the device allows attackers to bypass the security mechanism built on the software and protocol level, or provides side information which makes it much easier to break the system. The attacks reported based on the hardware can be mainly categorized as invasive or non-invasive, while security mechanisms are required against all possible attack scenarios.
1.2 Scope of this thesis

In this thesis we focus on addressing some challenges in secure embedded system designs, and propose several studies aiming at low cost and low redundancy security technologies. Before that, it is important to provide some background on the attack models and discuss the design challenges which concern the development of modern embedded system security.

1.3 Hardware attack models

Hardware has long been considered as a trusted party. It’s viewed as a platform runs the instruction passed from software and obeys any security rules built on the software layer. Given that assumption, the Integrated Circuit (IC) supply chain has also been considered as a safe environment and nowadays the supply chain is divided into many entities including IP designer, foundry, SOC integrator and consumer, so that the complex modern SOC design and manufacturing can be managed. Given this fact, the complexity of the supply chain also introduce more risks when the hardware security becomes a concern. As mentioned before, the reported attacks on hardware layer can be invasive or non-invasive.

1.3.1 Invasive attack

The invasive attacks are launched directly on the device and the security primitives are accessed by means such as micro probing or reverse engineering. These attacks are common on embedded systems on both circuit boards and SOCs, although SOCs
do require a little more sophisticated techniques and equipment to reverse engineer. A typical invasive attack consists of the following steps:

- **Depackaging.** Dissolve the resin covering the silicon and remove the chip package;

- **Layout Reconstruction.** Reconstruct the system layout by removing internal layers and using equipment such as a microscope. The layout can be reconstructed at different granularity. Initial attempts reveal the architectural structure such as location of memory and processor, as well as data buses; With more effort, details of lower structure can also be extracted.

- **Microprobing.** Microprobing or e-beam microscopy can be used to observe the values on the buses and interfaces on the component boundaries.

### 1.3.2 Non-invasive attack

While the invasive attacks need to be prevented at the silicon manufacturing level, the circuit and system designers should be more aware of non-invasive ones. In the meantime, it’s worth noting that the initial steps of invasive reverse engineering can be utilized to start a non-invasive attack. For instance, reconstructed layout is useful when attacker want to monitor the electromagnetic radiation as side-channel of the chip activity. The invasive attack models that’s most seen in the reports can be classified into following categories:
**Trojan injection**

Hardware Trojans are circuit modifications or extra logic added in the design by malicious users which can be triggered after the Trojans infected circuit is implemented in the system. They’re designed in a way that they can stay inactive during regular circuit functioning until the designated triggering condition is detected. Therefore, hardware Trojans are hard to detect during regular testing since the functional test and random stimulus are unlikely to trigger the Trojan circuit. The hardware Trojan has become a serious threat to the IC designs security, since the third party resources in hardware circuit design are prevailingly used in the modern IC supply chain. In order to minimize the time-to-market of complex SOC designs, the roles of chip manufacturer, designer and integrator, which once was the same party, are now more and more separated and spread around the globe, exchanging designs in forms of soft/hard IPs. The designers and consumers are vulnerable against Trojan insertion from untrusted IP providers and manufacturers.

Previous studies discussed the taxonomy of hardware Trojans and classified them based on their *physical characteristics*, *activation characteristics* and *action characteristics* [1]. The physical characteristics reflect the impact of circuit modification in terms of size or number of components, physical layout structure, timing and power traces. The activation characteristic specifies the type of activation criteria and whether it’s externally or internally activated. For example, a hardware Trojans can be connected to the temperature sensing logic so that it will be triggered when the device is operated at a certain temperature range. Or a Trojan can be triggered internally by inputting a particular sequence. The disruptive behavior of hardware Trojans can also be classified by different action characteristics. By targeting differ-

---

[1]: Previous study reference
ent security primitives as we discussed in the previous section, hardware Trojans may aim at modifying the functionality, specification or the transmission on the targeted device. To fully understand hardware Trojans’ physical, activation and action characteristics is essential to establish a comprehensive defense mechanism where Trojan insertion can be detected, disabled and prevented.

**Fault injection**

As the modern ICs are scaling down to the sub-micron regime, the implemented circuits are becoming more and more sensitive to environmental conditions. These conditions include temperature change, power supply fluctuation, electromagnetic radiation and intensive light illumination. Hence, by manipulating the conditions where the device is operating, attacker can introduce faults intentionally and alter the functional behavior. With different fault injection techniques, the injected faults could be in forms of time delay, stuck-at and single event upset (SEU) or multiple events upset (MEU). Many fault injection attacks focus on altering the device outputs in order to malfunction the system, or apply differential analysis on the output data so that hidden information can be retrieved. On the other hand, some other attacks inject faults on the control path aiming at altering the control sequence such as skipping a particular instruction. This is a more powerful attack model since the attack complexity can be greatly reduced if the control logic is comprised. In addition, the attacks on the control logic may allow attacker to breach the privilege settings which help them to bypass the security mechanism entirely.

To precisely control the process of fault injection, expensive equipment is often required. For example, the experimental setup used in [2] introduces the minimum
requirements for setting up fault injection using electromagnetic wave disturbance. A high power pulse generator is used and paired with a magnetic antenna to trigger faults on the target device. The system is mounted on the high precision motorized stage for controlling the targeted area. However, we’ll discuss in this thesis that how the characteristic of the circuit design can determine the impact of injected faults, so that randomly introduced faults can also be used to breach the security. Depending on the technology being used, some fault injection attacks require invasive steps (ie. depackage the chip and apply high intensity light illumination), while some are non-invasive such as manipulating the power or clock.

**Side channel analysis**

Side channel analysis (SCA) is a group of non-invasive attack methods where design specifications are revealed by physical characteristic of device operation, such as power or timing traces, or through test/debug channels. The most common victims of SCA are cryptographic modules. Modern cryptographic algorithms are very well designed and understood so that it’s almost impossible to break at the algorithm level. However, when the algorithm runs on the hardware, the device activities are dependent on the critical information hidden in the device which can be observed through side channel links. The nature of vulnerability of cryptographic algorithms against SCA is that the iterative computation, foundation of data confusion and diffusion, can be broken down to separate steps with side channel observations. For example, the testing infrastructures such as scan chain or JTAG boundary scan interface are widely reported as means of side channel attack on a variety of cipher designs [3] [4] [5]. In [5] a typical scan-based SCA attack is presented against AES device. The access
to the scan mode allows attacker to take a screenshot of the computation results of intermediate round stored in the state registers and shift it out from scan output. This means the attackers can isolate the algorithm in the first round, manipulate input and output until resolving the respective round-key, and then move forward with the known previous round as input to resolve for the next round. Following this manner, the complexity of breaking the cryptographic system, who’s algorithms are well understood by the public, is reduced to the minimum.

1.4 Embedded system hardware security design state-of-art

The importance of security in embedded system, as Kocher and et al [6] [7] discussed in their works, has become a new and critical design dimension to the modern embedded system designs. As we discussed in previous sections, the reported vulnerabilities and attacks against the hardware side of the system emphasized the hardware security as an extra layer in addition to the traditional system security design.

1.4.1 Security methodology in modern IC design flow

In section 1.3, we introduced a variety of hardware attack models. Given by the fact that the current SOCs are usually designed, integrated, and manufactured by separate parties, the untrusted parties who can conduct these attacks include many entities throughout the IC production flow, which includes IP vendor, designer, manufacturer, and malicious users. The development of security mechanism against hardware attack is a complex task with subjects covering from early system and circuit design
stage to IC post-market applications. In Fig 1.2 we’re presenting the modern industry standard IC design flow and the security design requirements with respect to each different stage. From the designer’s perspective, a system design should have a complete set of security protocols, authentication mechanisms and well-understood security assets depending on the system specification and application scenarios. And then, during each stage the production flow, these security elements should be properly designed, verified, integrated, fabricated and tested. The efforts are required from aspects which includes, but are not limited to, the development of new countermeasure schemes, such as masking, obfuscation, and temper resistance design; devel-
development of novel CAD flow to evaluate the security assets and design vulnerabilities at early design stage; Trojan detection software during integration and built-in-self-test (BIST) design for later application security; Verification and formal verification mechanisms with security primitives taken into account; Obfuscation, circuit hardening and temper resistance design during fabrication; DFT security design and testing for security.

1.4.2 Design challenge and research state-of-art

Embedded system designer always need to struggle with the computational demand of security processing and the limited computation capability. Introducing security elements to design, which already has constraints on cost, power and resource utilization, will only lead to more trade-off between security versus cost, or security versus performance. Moreover, recent trends in computation and communication technology are leading to more resource constrained and diverse embedded system hardware. For example, cloud computing and future communication technology such 5G are aiming at bringing more computation tasks to the server. In this case, the end device’s role will be shifted towards data collector, display, transmitter and receiver which has remote connection to computation unit and data storage in the cloud. In the meanwhile, the connection between devices are more decentralized and the placement of devices are ubiquitous. This means security designers need to deal with more extreme cases where very low power and compact device which can be publicly accessed can still require high level of security coverage.

Therefore, it’s important to develop embedded system security solutions which are low cost, low redundancy and high efficient. To achieve this goal, studies and design
efforts are needed from several aspects. First, security design should be planned at an early stage as possible. For example, if Trojan insertion can be identified by effective flow before integration, any potential damages are prevented. Besides usually the cost of countermeasures implemented at the design stage is much lower than it is in the later fabrication. Second, development of novel devices and technologies are needed to satisfy the increasing security and performance requirements. This includes utilization of new device characteristics, novel architecture and protocol design, development and implementation of new algorithms, and so forth. Third, flexibility needs to be considered at every stage of defense mechanism. With strict restrictions on the cost and performance, the flexibility of security countermeasure used in modern embedded system has become an important factor so that their implementation can be adjusted to different application scenarios and fit for the diversity of the device forms.

CAD for security

CAD tools play essential roles throughout the IC production flow. In order to ensure the system security, well developed CAD tools and flows are required during each stage of IC production. Vulnerability definition and assessment should be carried out following aspects. First of all, a comprehensive security rule check mechanism needs to be established to ensure and enhance security at early design phase. In Nahiyan’s work [8], primitives of developing the framework for security rule check has been discussed in detail. Security rules and assets, as a new dimension of the design checking metrics, should be defined by understanding the different aspects of security requirements and potential attacks. This can include understanding of the attack targets and models, the potential adversaries, the source of vulnerabilities and
more. With these primitives as guidelines, research work is needed in terms of security metric definition, design vulnerability analysis, and security strategy development.

One important part of CAD design for hardware security is pre-silicon Trojan detection. We have discussed before that hardware Trojans can be categorized by their characteristics in terms of physical appearance, activation and action. And accordingly, the detection methods can be applied at pre or post silicon phase, on physical device or high level abstraction model. At pre-silicon phase, Trojan detection mainly rely on analyzing the physical layout. This is because that Trojan insertions and their activities leading to abnormal chip behavior, which is hard to test functionally, can be revealed by side-channel traces such as power and timing, and hence side-channel based analysis is the most common methodology for detecting the existence and activation of hardware Trojans. In Salmani’s study [9], a layout level Trojan injection vulnerability analysis is presented. Physical design inevitably leave empty regions on the chip (white space), which is often utilized by malicious party for injecting Trojans so that the Trojan’s footprint is minimized. Therefore, white space based analysis is an essential step to evaluate the chip’s vulnerability against Trojan injection. On the other hand, the injected Trojans can also reside amid the normal logic. This requires the activation logic of the Trojan to be connected on nets with very low transition probabilities (for power based Trojans), or uncritical path with small path delays (for delay based Trojans). In another word, these low transition probability nets and non-critical paths can be considered as design vulnerabilities against Trojan injection, hence, should be included as security evaluation metric during layout analysis.

The vulnerability analysis also includes some formal methods at the early design stage. Finite state machine (FSM), which is the most common model used for circuit and system level behavior, has been widely used for formal verification of circuit func-
tionality correctness. The theory can be extended for security assets verification. For instance, the work in [10] presented a case study of model checking based verification to estimate system vulnerability against device errors. On the other hand, the implementation of FSM based control modules also reveal potential system vulnerability against Trojan injection, fault injection, and failures as discussed in [11] [12].

1.4.3 Physical-Unclonable-Function (PUF) and device entropy

PUF is group of devices where the variation of fabrication process is leveraged to generate real random numbers or device entropy. A PUF function takes a challenge and outputs a device specific and unique response accordingly. Since the process variation is truly random and uncontrollable, every device will have it’s unique physical feature that can not be cloned. Hence such features can be viewed as the fingerprint of the device, and used for cryptographic system and device identification at low implementation cost. One common type of randomness leveraged by PUF design is the variation on time delay. At sub-micron regime, two identically designed gates will be manufactured with slightly different doping concentration, size and threshold voltage, which is significant enough to have different gate delays. Hence two functionally equivalent device will have different time delay traces. Special circuit components, such as Ring Oscillator [13] or arbiter [14] can be used to amplify the delay differences and make them detectable. Therefore, it extracts the process variation ad translate them into unique device ID. Another commonly leveraged process variation is the threshold voltage mismatch. For instance, the six-transistor symmetric structure of SRAM cell determines that the manufactured cells will have unbalanced threshold voltage between the ON/OFF sides. This reflects to a random ”0” and ”1” sequence
at SRAM array at power-up which is unique to the device. The SRAM-based PUF then can be utilized as a low cost and secure solution for key generation and storage. Moreover, randomness features introduced by novel devices can also be exploited for PUF design. For instance, memristor device, which is a fast speed, low power and high density memory device that is actively studied as the future memory device candidate. Memristor device has some undesired feature including process variation caused data storage instability and destructive read caused drifting effect, yet are utilized to develop memristor based PUF device and random number generators \[15\] \[16\]. The "forgetting effect" of the memristor array has also been exploited for the security of neuronetwork in Yang’s work \[17\] and for secure scan infrastructure design in our previous studies \[18\].

1.4.4 Obfuscation and masking

Obfuscation and masking based countermeasures rely on introducing randomness to the design in order to protect system from reverse engineering, data eavesdropping and side channel leakage. Obfuscation can be applied on algorithms, control path, data path, physical layout, manufacturing and circuit design. At algorithm level, masking is the most common and through protection measure against side channel leakage of cryptographic systems. Typical masking schemes applied on symmetric cryptographic algorithms are presented as in \[19\] \[20\]. A random mask is introduced into each step of cryptographic algorithm, additively or multiplicatively, in order to hide the correlation between side channel traces and the key value. The difficulty of masking the cryptographic systems in embedded system implementation is due to the high circuit complexity of masking algorithm especially for the non-linear
computation steps. Adding and removing mask on non-linear computation, such as s-box, can easily double or triple the original design size, and compensate the design performance at the same time. Therefore, low cost, flexible and efficient masking designs and scheme are important for the realization of side channel resilience in embedded hardware designs.

In the mean time, other obfuscation schemes in reported works are also adding more dimensions for the overall system security protection. For example, in [21] obfuscation is applied on the data path of AES circuit, by adding remapping Look-Up-Table (LUT) to protect it from scan based side channel attack. And adding scan chain obfuscation is the most common strategy for scan security. By randomly inserting gates, or scrambling the order of deliberately divided sub scan chains, the test data is obfuscated to unauthorized party to prevent them from leveraging the test infrastructure for malicious purposes. Similarly, logic encryption based techniques are also used to insert additional XOR gates or LUTs at random circuit interconnections for obfuscation purpose [22]. Obfuscation techniques can also be beneficial during board level design. In [23], a framework of obfuscation on the PCB level is presented for the resilience against non-destructive board level reverse engineering. The proposed framework exploited the programmable components on board, such as Micro-controller unit, DSP, and FPGA to work as a permutation block and hide connection between components. With the obfuscated interconnections, the design on board is only available to users with a correct key. On fabrication level, obfuscation schemes can be developed relying on novel materials and fabrication process to improve chip’s resilience against destructive reverse engineering [24].
1.5 Security in programmable device

In general, the design of embedded system hardware tend to become more and more flexible, so as to meet the increasing demand for the variation of application scenarios and resource limitations. On the other hand, the rapid development of programmable hardware, especially FPGA device, is leading this trend towards designs at a higher complexity and a bigger scope. Abundant works have been done implementing system on programmable hardware and optimizing the resource allocation of hardware-software co-design. Given the fact that the mainstream FPGA devices (ie. Xilinx 7 series) are already implemented with 28nm technologies and support over 2 million logic gates, it’s safe to say that the future hardware will be more and more ”software like”. Therefore, to fully understand the security of programmable devices as well the programmable device based designs is critical for the next generation of embedded system.

The most common security problem in FPGA devices is the protection of bit-stream. By far, the most popular type of FPGA device is based on SRAM. Since SRAM is a type of volatile memory, the configuration bitstream needs to be externally stored in some nonvolatile boot memory, such as Flash. The safety of this bitstream file is essential to the FPGA program as it contains all design details which can be used to fully reverse engineer the design, and the FPGA system is subject to the threats such as cloning, building, tampering and spoofing if the bitstream file is insecure. The apparent vulnerable link through the bitstream transmission is from the boot memory to the FPGA configuration memory during boot process. Therefore, the bitstreams for security sensitive programs are normally secured by cryptographic systems. Data encryption is applied after generating the bitstream, and then it will be
decrypted on chip when it’s downloaded to FPGA. Major FPGA vendors, including Xilinx, Altera and Microsemi, have all proposed their bitstream protection schemes and authentication mechanisms based on symmetric ciphers. However, side channel attacks can still be conducted on FPGA devices to break the bitstream protection. Hence, the security of FPGA also requires to be considered from various aspects including side channel resilience, tamper resistance, anti-reverse engineering and more. While the unique ”software like” characteristics of FPGA and other programmable hardware devices introduce unique security design challenges as such devices are designed with general purposes and can be implemented in to any design. For instance, the Trojan injection may have different forms on FPGA which can not be identified by typical power and timing testing. Specific security design flow needs to be established for system implemented or partially implemented on programmable devices, in order to ensure the overall system trustability, confidentiality and data integrity.

1.6 Outline of the Dissertation

This dissertation is organized as follows: in Chapter 2, a finite state machine (FSM) based probabilistic analysis framework for analyzing IC design vulnerabilities is presented. The framework provides a general discussion of the potential risks related to state transition paths and state transition features under fault injection attacks. In addition, low cost mitigation scheme is demonstrated which can be used for security enhancement at early design stage. In Chapter ??, we present a novel secure scan chain architecture exploiting the true random memory retention loss effect of memristor devices. The design utilizes the unique properties of memristor devices and
achieves low overhead, true random and highly compatible obfuscation on scan data to defend embedded system design against scan based side channel attacks. And in Chapter 4, a design is presented addressing the overhead problem of masking the side channel leakage during FPGA bitstream encryption/decryption. The proposed design takes advantage of dynamic partial reconfiguration feature of modern FPGA devices, introduced a new FPGA boot-up flow, so that the masking mechanism can be dynamically included for low redundancy and flexible masking implementation. Lastly, we summarize and discuss the research impact of proposed approaches in chapter 5.

1.7 Publications

Journal papers that are accepted and published with primary authorship include:


Currently submitted journal papers with primary authorship include:

Conference papers that are accepted and published with primary authorship include [18]:


Journal papers that are accepted and published with co-authorship include [26] [27] [28]:

1. F. Qian, Y. Gong, and L. Wang, “A Memristor-Based Compressive Sampling Encoder with Dynamic Rate Control for Low-power Video Streaming,” *ACM Journal on Emerging Technologies on Computing (JETC)*


Conference papers that are accepted and published with co-authorship include [29] [30] [31] [32]:


Chapter 2

Probabilistic Evaluation of Hardware Security Vulnerabilities

2.1 Introduction

As the rapid advance of semiconductor technology scaling, integrated circuits and systems have become more and more susceptible to noise and environmental variations. Manufacture flaws, such as doping defects and material impurity, can be activated by varying environmental conditions much easier than before, which could lead to malfunctions or even failures in embedded systems. Moreover, it is becoming a security threat that malicious users can take advantage of varied environmental conditions to attack hardware devices. Reports have shown that attackers are able to break into various devices including cryptographic systems such as AES and RSA ciphers by accurately controlled fault injection followed by fault analysis \[33, 34, 35\].

A lot of conditions are critical to sub-micro level circuits, for example, power fluc-
tuation, temperature change, electromagnetic pulse, intensive light illumination [34], [33], [36], [37]. Fault injection techniques are developed initially to validate the function of the device-under-test (DUT) and detect design weakness in the occurrence of faults, such as time delay, stuck-at, and single-event-upset (SEU). The purpose of conventional fault injection experiments is to find whether a device will fail or produce errors while still working. Therefore, the influence of the faults is usually considered at the device output [38], [39].

However, when fault injection is used as a mean of hardware attacks, the attacker’s goal is not just to fail the device at the output, but also to disturb device operation in order to create side channels, steal key information, and access privileged instructions. To fully understand the influence of the induced faults on device dynamic behaviors, it is necessary to have an analytical model of inner state transitions for the device under attacks, which has been ignored in the conventional study of fault injection or high-level function validation.

Finite state machine (FSM), as a general model for digital design, is very useful to verify the device functionality at a higher level. It is a common practice to design a complex circuit function from its FSM abstraction. For example, the FSM-based formal verification is well studied to determine the correctness of circuit functionality. Also, it is possible to use FSM model checking based techniques to identify vulnerable registers that may cause erroneous outputs [10], [40]. In this paper, we propose a probabilistic evaluation approach based on the FSM model to study the security vulnerabilities in digital circuits. We will leverage the probabilistic FSM model to identify the intrinsic design vulnerabilities under fault injection attacks and develop effective techniques to mitigate them. The contributions of our work lay mainly in the following aspects:
• First, we perform a study on the influence of environmental conditions on system behaviors and treat faulty state transitions as a new design metric for hardware security;

• Second, we propose a formal algorithm that can discover the security vulnerabilities in digital circuits at the Register Transfer level (RTL) during the design stage;

• Third, we develop an effective FSM encoding scheme to enhance the security of critical states with very small hardware overhead.

The remainder of the paper is organized as follows. In Section 2, we review the related work. In Section 3, we provide the motivation of this work and discuss the existing fault injection attacks. The probabilistic faulty transition model and the re-encoding scheme for mitigating the design risk are presented in Sections 4 and 5, respectively. Simulation results are evaluated in Section 6 to demonstrate the improvement in security resilience and the induced design overhead. A case study is also discussed in this section. Section 7 gives the conclusions.

2.2 Related Work

Fault injection attacks can be classified into three categories. The first type of attacks introduce logic errors that may alter the output of a circuit. The second one is so-called differential fault analysis usually aimed at retrieving the key of crypto systems by comparing correct outputs with faulty ones. The last group modifies control logic by skipping or replacing instructions through introduced errors. In practice, most
successful fault injection attacks are a combination of the above. Depending on different fault models, accuracy of process control, and characteristics of the underlying device, various methods have been reported to successfully attack the mainstream ciphers such as AES (typical symmetric key cipher) and RSA (typical asymmetric key cipher) [33], [34], [35]. Precise control of the fault injection process, especially in timing and location, is important to the success of many fault injection attacks. In addition, multiple device restarts are often required to obtain the sufficient results in order to determine the device signature. For example, to retrieve the 1024-bit key of a RSA engine, the total number of device restarts for a successful attack is approximately \(1024 \times \ln(1024) = 3083\) [33].

To conduct precisely controlled fault injection attacks, it usually requires an expensive setup, which includes device-under-test (DUT), fault injection equipment, input vector generator, fault impact checking and results collection. However, the difficulty and cost of such attacks can be reduced by taking advantage of disturbed inner state transitions. For example, to alter the internal states of a circuit with certain knowledge of the design is not necessarily expensive. One attack proposed in [41] against SNOW 3G was carried out without the precise control of fault injection timing and location. Instead, it induced random faults on the inner state registers and view the output of the cipher as a nonlinear function. After fault injection, the attacker can deduce the position of faults by comparing faulty outputs with correct ones, and solve a set of nonlinear equations to retrieve the key.

Usually, it is more effective to apply fault attacks on control flow than data flow. In [42], fault injection attacks were applied to a RSA cipher targeting the control logic instead of the algorithm itself. By introducing glitches into the clock signal, the attacker was able to skip certain instructions and recover all key bits at the same time.
The work in [2] investigated the effect of injected faults on a microcontroller. Although the observation was made at outputs, the presence of state transition errors can still be noticed as some of the output errors can only be explained by the replacement of certain instructions. These results suggest that some instructions or registers are more vulnerable to fault attacks than others.

The behavior of injected faults in digital circuits is analogous to soft errors, which are a well-studied subject with many analytical models available to describe the error propagation process as well as the statistics of resulted failures under different scenarios [43], [44]. However, the particular interest of these studies is usually the propagation of errors to the primary output, whereas the perturbed internal states during error propagation are often ignored. On the other hand, efforts have been made on evaluating the potential vulnerability of FSMs. The formal verification methodology of sequential logic is essentially a framework based on the state transition model of digital circuits [45], and to its extension, evaluation methods were proposed with FSM-based model checking to evaluate circuit error resilience [10], or to quantify the vulnerability of registers on the control path [46]. Some discussion on the security risks of FSM models can be seen in [12], [47], [48]. The work in [12] concerns coarsely grouped “normal” and “protected” states according to the specified state transition model, which only considers the high-level transition behavior of a logic, not related to any particular attack scenario or threat model. The method in [47] belongs to the conventional FSM design framework that employs parity checking to ensure state transition correctness. The work in [48] proposes an automatic test pattern generation (ATPG) based tool to extract the state transition graph from the synthesized netlist. It is able to reveal the logic resilience against fault injection from the “don’t care” states. Different from these existing work, we will study the impact of injected
faults on FSMs from hardware security perspective and propose a general evaluation framework based on probabilistic analysis methods.

2.3 Motivation and Attack Analysis

It has been shown that induced faults can be utilized to target the control logic of crypto systems, through which new types of side channels are created by skipping certain operations, and the encryption signature is revealed. Apparently, crypto systems are not the only victim. The state transitions in digital systems can be compromised by fault injection attacks in a similar way, which may result in the leak of critical information or enable unauthorized use. FSM is a generic model of many digital systems. In particular, circuit design at the RTL level can be easily understood as FSM, where a state transition represents a set of unique assignments of register values corresponding to the primary inputs. However, current hierarchical design practice can create loopholes because system designers usually have only limited knowledge on circuit implementations at the hardware level.

2.3.1 FSM Properties

The vulnerabilities of an FSM design come from various factors when it is implemented in hardware. First, an FSM can have a small number of states; however, the total number of states in a digital circuit implementation is $2^n$, where $n$ is the number of the registers. Hence, it is possible that extra states are implemented without considering any security implications. Second, a state transition is normally associated with only one or few input variables, and the access probability of each state tends
to be non-uniformly distributed. This may cause certain circuit nodes to have a high security risk. For instance, as discussed in the following sections, such states endure a high chance of being attacked during the random access attempt. Third, under certain design constraints such as performance and timing, EDA tools may automatically optimize a design in order to meet the design criteria. This further weakens the designer’s controllability during the design process.

To understand how attackers could leverage these design vulnerabilities, we need to first identify the critical states in an FMS design. Figure 2.1(a) shows an example of state transition diagram with four states, and the corresponding binary encoded circuit implementation containing two state registers is depicted in Fig. 2.1(b). It can be seen that states $C$ and $D$ are reachable from all other states, whereas states $A$ and $B$ are self-contained. Hence both states $A$ and $B$ are critical, and unauthorized transition paths (such as the one indicated by the red dash arrow) should be prevented. In practical designs, there are user specified critical states. For example, the encryption algorithms usually have repetitive computations and a final-round state that stores the final result into memory. In this case, the final-round state needs to be protected from unauthorized accesses, because any access to the final-round state during the iteration process will reveal intermediate computation results, which are frequently reported to be used to reveal the key. In the following discussion, we refer to these critical states as protected states and the other states as normal states.

### 2.3.2 Attack Model

In practice, the vulnerability of modern ICs is originated from device scaling and nano-technology fabrication. Manufacture flaws, such as doping defects or material
Figure 2.1: An example of FSM and its faulty transition: (a) state transition graph and (b) logic circuit implementation. The corresponding state transition table is shown in (c).
impurity, can be activated by environmental stresses to generate faults or failures in logic functions. These environmental stresses include over- or under-powered voltage supply, manipulated clock signals, radioactive or electromagnetic wave injection, etc. The cost of the equipment to setup fault injection attacks could be cheap or expensive. For example, manipulating supply power or clock signals is usually low-cost compared with utilizing electromagnetic waves that target the device-under-attack in a very precise manner. In this paper, we aim to develop a generic evaluation method for design vulnerabilities under fault injection attacks. The impact of injected faults at internal circuit nodes are modeled as error variables (see Section 4.2) to cover various fault injection mechanisms with different scales and magnitudes.

As shown in Fig. 2.1(b), if an error is introduced at internal node A when both registers are ‘1’ (state D), the error may propagate to register 1 and flip its output from ‘1’ to ‘0’. This makes the FSM to switch from state D to state B directly, generating an unauthorized path on the transition diagram as shown by the dashed line in Fig. 2.1(a). Thus, the FSM switches from a normal state to a protected state, which should not be allowed during the normal operation. An attacker may gain access to the critical states by manipulating environment conditions and inject faults to achieve this goal.

Exploiting similar vulnerabilities, fault injection attacks can be launched by the attacker to bypass the security safeguard and bring the system to a protected state. All of the factors discussed before will affect the probability that a protected state is maliciously accessed. It has been discussed in the previous section that fault injection attacks do not necessarily need the precise control of injection process, but instead target the intrinsic vulnerabilities through randomly induced faults. In such case, the higher probability a transition to the protected states, the more likely successful
attacks can be launched. If a state is critical, any malicious accesses to this state can cause serious consequences, such as denial of service or exposure of important information.

Although the fault injection process might be random, if the target state is noise sensitive, these attacks can be quite effective with the assistance of post-attack analysis. On the other hand, if the attacker knows the design, he can intentionally introduce faults into the circuit; for example, a hardware Trojan can be activated by aging the circuit nodes that are associated with the unused states, thereby creating a faulty transition path to the protected state \[49\]. Designers need to know the vulnerabilities in their design, such as how easily the protected state can be maliciously accessed and where the most vulnerable path is. In addition, a formal design metric is needed to quantify the unauthorized state transitions, thereby allowing designers to improve the resilience of their designs against hardware attacks.

Conventional FSM analysis aims at creating the correct function, e.g., using formal verification methods to check whether an implementation has the desired function. These methods usually rely upon a static/deterministic FSM model. However, randomly induced faults will introduce large uncertainties in FSM dynamic transitions. In this paper, we will develop a statistical basis to identify the potential vulnerabilities in FSM design. To start with a self-contained problem, it is assumed that attackers do not necessarily know the design specifications and can provide any input patterns to the system. Attackers can also change the environmental condition to introduce faults. The state machine can be initiated from any normal states, and attackers are able to check if the manipulated state transition is successful. These are some general assumptions in most FSM models and can be relaxed in practical circuit design.
2.4 Analysis of Inner State Transitions

In this section, we present the probabilistic model to study the inner transitions in FSMs for security enhancement. We first develop the model through a formal mathematical analysis and then discuss the complexity and scalability of this model in dealing with complex FSMs.

2.4.1 State Transitions

Since our target is unauthorized transitions from normal states to protected states, the probabilistic model needs to quantify the inner state transitions rather than the final output of the FSM. To begin with, we define the reachable states of a state \( u \) as the set of states that can be reached from \( u \), denoted as

\[
R(u) = \{v \subseteq V \mid v \text{ is reachable from } u\}. \tag{2.1}
\]

Similarly, each state has a set of states from which it can be reached. This set of states is called the starting states. The starting state of \( u \) is defined as

\[
S(u) = \{v \subseteq V \mid u \text{ is reachable from } v\}. \tag{2.2}
\]

It is easy to see that if \( a \subseteq R(b) \) and \( b \subseteq R(c) \), then \( a \subseteq R(c) \). A safe state machine must have its reachable state sets and starting state sets to be exactly the same as specified by the designer; otherwise, there exist unauthorized transition paths that can be exploited by the adversary. For example, when a system is stressed such as being exposed to the injected faults, a state may be accessed by the undefined
transitions caused by register flips. We refer to these transitions as unauthorized transitions.

We will develop a mathematical model to evaluate how induced faults will affect inner state transitions. The state transition function $\delta_0$ of an FSM is defined as

$$s(t + 1) = \delta_0[(i(t)), s(t)], \quad (2.3)$$

where $\delta_0$ maps the current state $s(t)$ to the next state $s(t + 1)$ by considering the input $i(t)$. If the dimensions of discrete input space $I$ and state space $S$ are $k$ and $n$, respectively, the state transition table will have $2^n$ rows and $2^k$ columns, where each entry provides the mapping information as defined in (2.3). An example of state transition table is given in Fig. 2.1 (c).

### 2.4.2 State Transitions under Fault Injection Attacks

FSMs can be designed in various ways with different hardware implementations. The vulnerability of critical states also varies in these implementations. To quantitatively assess a state transition, it is necessary to look the implementation of an FSM as a sequential circuit, as shown in Fig. 2.2. Here we only consider the combinational logic for state transitions and omit the output logic. The state variables are stored in the $n$ registers synchronized by a clock signal. We can track the fan-in gates of each state register. The total number of fan-in gates is $G \leq \sum_{i=1}^{n} g_i$, where $g_i$ denotes the number of fan-in gates of the $i^{th}$ register.

An indicator function $F_{\text{inde}}$ is derived from the transition map to show if there is
a transition path between any two states $s_i$ and $s_j$, such as

$$F_{\text{inde},ij} \left( \delta_0 \left[ (i(t)), s_i(t), s_j(t+1) \right] \right)$$

$$= \begin{cases} 
1 & \delta_0 \left[ (i(t)), s_i(t) \right] = s_j(t+1) \\
0 & \delta_0 \left[ (i(t)), s_i(t) \right] \neq s_j(t+1) 
\end{cases} \quad (2.4)$$

The probabilistic transition matrix $T$ is a $2^n \times 2^n$ matrix, in which the element $T_{ij}$ at row $i$ and column $j$ is the probability of one-step transition from state $i$ to state $j$. Since $\delta_0$ is a function of the input and current state, in a noiseless environment the transition probability depends on the input probability $P(i)$ and the transitions when this input appears, i.e.,

$$T_{ij} = \sum_{i \in I} P(i) \cdot F_{\text{inde},ij}. \quad (2.5)$$

The behavior of the injected errors at internal circuit nodes can be modeled as an error variable at the gate/register output as depicted in Fig. 2.2. We distinguish
the errors in combinational gates and state registers with different variables $\epsilon$ and $\gamma$, respectively. Note that gate errors may have a smaller chance to flip the state due to propagation attenuation [50]. To simplify the problem, only single event upsets (SEU) are considered. Even so, we are able to show the vulnerability of an FSM design and this approach can be extended to multiple event upsets. It is reasonable to treat the error variables as an extension of the original vector space. The dimension of the vector space is enlarged from $k + n$ to $k + 2n + G$, where $k$, $n$ and $G$ are the sizes of input space, state space, and fan-in gates of state registers, respectively. To show this, the vector space $U$ of the FSM and some related subspaces are defined as follows.

Subspace $I$ with dimension $k$ defines the input

$$I = \{i \in U; i = (i_1, i_2, \cdots, i_k), i_j = 0, 1\}.$$  \hspace{1cm} (2.6)

Subspace $U_1$ with dimension $G$ defines error variable $\epsilon$ at combinational gates

$$U_1 = \{\epsilon \in U; \epsilon = (\epsilon_1, \epsilon_2, \cdots, \epsilon_G), \epsilon_j = 0, 1\}.$$  \hspace{1cm} (2.7)

Similarly, $U_2$ is the subspace of error variable $\gamma$ at state registers. The dimension of $U_2$ is $n$, as

$$U_2 = \{\gamma \in U; \gamma = (\gamma_1, \gamma_2, \cdots, \gamma_n), \gamma_j = 0, 1\}.$$  \hspace{1cm} (2.8)

In combination with the state space $S$ of size $n$, the complete vector space $U$ that the FSM is defined upon is given as

$$U = \{I \cup U_1 \cup U_2 \cup S\}.$$  \hspace{1cm} (2.9)
Hence, the transition function $\delta_0$ in the presence of errors can be rewritten as

$$s(t+1) = \delta_0[(i(t), \epsilon, \gamma), s(t)].$$  \hfill (2.10)

A faulty transition occurs when the system in state $i$ at time $t$ fails to go to state $j$ at $t+1$ as it is supposed to. A faulty transition may create a non-existing path on the transition map, which can be checked as a new non-zero element in the indicator matrix. The probability of faulty state transitions is defined as

$$\zeta = P\{\delta_0[(i(t), \epsilon, \gamma), s(t)] \oplus \delta_0[(i(t), 0, 0), s(t)] = 1\}. \hfill (2.11)$$

Under the previous assumption, error variables $\epsilon$ and $\gamma$ are independent, each of which obeys the binomial distribution with a probability of $p1$ and $p2$, respectively. In other words, $p1$ and $p2$ are the probabilities of a single gate error and a register error, respectively. In this model, we assume if a gate error propagates to a register or an error takes place inside a register, the content of the register will be changed. A vector $A$ with size of $n$ can be defined, where the $i^{th}$ element $A_i$ indicates the probability of bit upsets at the $i^{th}$ register, such as

$$A_i = \left( \sum_{\epsilon \in U_1,i} P(\epsilon) \right) + P(\gamma), \hfill (2.12)$$

where $U_{1,i}$ is the subspace of the fan-in gates of the $i^{th}$ register. Therefore, $A$ represents the likelihood of erroneous register flips, which is an important factor to determine the faulty state transition probability. Note that the first term on the right hand side of (2.12) is due to the immediate fan-in gates as well as the error
propagation effect of other combinational gates. More comprehensive estimate can be done by weighting error probabilities at different combinational gate stages [44], but this topic is beyond the scope of this paper.

The probability of faulty state transition $\zeta_{ij}$ from state $i$ to state $j$ is related to the Hamming distance between the two states, which can be determined from $H(S_i, S_j) = S_i \oplus S_j$, as well as the likelihood of corresponding bit flips. We use function $\mathcal{B}$ to denote the probability of faulty transitions. A faulty transition between two states happens when the Hamming distance between the two states is overcome by bit errors. Apparently, $\mathcal{B}$ is a matrix with dimension of $2^n \times 2^n$. Each element $\mathcal{B}_{ij}$ gives the probability of faulty transitions from state $i$ to state $j$, which can be calculated as

$$B_{ij} = P(\zeta_{ij}) = \prod_{k=1}^{n} (S_i \oplus S_j)_k \circ A_k,$$

(2.13)

where $\circ$ stands for Hadamard product.

**Example 1**: Assume that state 1 and state 2 are encoded as “1101” and “1011”, respectively. A faulty transition between the two states occurs when the registers of middle two bits flip at the same time, which can be represented as $[1, 1, 0, 1] \oplus [1, 0, 1, 1] = [0, 1, 1, 0]$. The probabilities of bit errors of these registers are stored in matrix $\mathcal{A}$. Let’s say $\mathcal{A} = [1e - 4, 2e - 4, 1e - 3, 1e - 4]$. Then the probability of faulty transitions between these two states can be calculated as

$$\mathcal{B}_{1,2} = \prod_{k=1}^{4} (S_1 \oplus S_2)_k \circ \mathcal{A}_k$$

$$= (1e - 4 \times 0) \times (2e - 4 \times 1) \times (1e - 3 \times 1) \times (1e - 4 \times 0)$$

$$= 2e - 7$$

(2.14)
Note that (2.13) represents the static state transitions under errors. Now we can study the dynamic behavior of state transitions under hardware attacks. Dynamic state transitions need to be refined by considering different access patterns of the states in an FSM. The one-step transition matrix $T$, which is defined in (2.5), is determined not only by the input probability $P(i)$ but also by the error probability $P(\epsilon)$ and $P(\gamma)$. A faulty transition could happen if faults propagate to, or occur at the state registers. Hence, the probability of dynamic faulty transitions from state $i$ to state $j$ is the sum of probabilities of faulty flips to state $j$ from any states that are accessible by state $i$. This can be calculated as the matrix multiplication of $T$ (see (2.5)) and $B$

$$T'_{ij} = T_{i,:} \cdot B_{:,j}. \quad (2.15)$$

**Example 2**: Assume that state 4 is a protected state, and we need to calculate the probability of a faulty transition path from state 1 to state 4. The probability of legitimate transitions from state 1 to other states are stored in the first row of matrix $T$ (e.g., $[0.75, 0.15, 0.1, 0]$). The erroneous flipping probabilities from all other states to state 4 is stored in the fourth column of matrix $B$ (e.g., $[5e^{-6}, 7e^{-3}, 7e^{-3}, 0.99]$). Hence, the probability of the faulty transition path from state 1 to state 4 can be calculated as

$$T'_{1,4} = T_{1,:} \cdot B_{:,4}$$

$$= [0.75, 0.15, 0.1, 0] \cdot [5e^{-6}, 7e^{-3}, 7e^{-3}, 0.99] \quad (2.16)$$

$$= 0.00175$$

Combining (2.4), (2.5), (2.12), (2.13) and (2.15), we obtain the probabilistic transition matrix $T'_{ij}$ with faulty transitions as
\[ T'_{ij} = \sum_{i}^{n} \left\{ (P(i) \cdot F_{indc,ij})_{i,:} \cdot \prod_{r=1}^{n} \left( (S_{i} \oplus S_{j})_{r} \cdot \left( \sum_{\epsilon \in U_{1,i}} P(\epsilon) + P(\gamma) \right) \right)_{r,:j} \right\} \quad (2.17) \]

2.4.3 Model Complexity, Scalability and Limitations

In the above matrix \( T' \), each element \( T'_{ij} \) represents the probability of transitions from state \( i \) to state \( j \) under fault injection attacks. To determine each entry of this matrix, the most complex calculation is the multiplication of matrices of size \( NS \times NS \) with a complexity of \( O(NS^3) \) using naive algorithms, where \( NS \leq 2^n \) is the total number of states. Advanced methods such as Coppersmith-Winograd algorithm [51] can greatly reduce this complexity. Potentially, faulty transition paths exist between any two states, but those paths leading to a normal state do not pose security concerns (though there may be some function issues). This is because normal states are designed to be accessible by other states. Attention should be paid to the paths that enable malicious accesses to the protected states. Hence, if the total number of protected states is \( N_{ps} \), then the multiplication complexity is only \( O(N_{ps} \cdot NS^2) \). Note that our work targets the control functions in security operations. Usually, these control functions are a subset of the entire state machine and contain a limited number of states. Also, state-of-the-art module-based designs build large state machines from small modules with manageable sizes in a hierarchical way. For complicated FSMs extracted from synthesized and flatten netlists, measures can be taken to decompose them into sub-FSMs, thereby reducing the computational complexity of the proposed method.
Figure 2.3: FSM decomposition example: (a) original FSM graph; (b) original next state transition table; (c) decomposed state machine M1; (d) decomposed state machine M2; (e) next state transition table of M1; and (f) next state transition table of M2.
FSM decomposition is a group of methodologies that are effective in dealing with large state machines. It decomposes a large state machine into smaller sub-machines while maintaining the same functionality, so that the searching space and design complexity can be greatly reduced. Consider an example based on the method proposed in [52] as shown in Fig. 2.3. Here the original STG with six states (Fig. 2.3(a)) is transformed into two smaller STGs: sub-state machine M1 (Fig. 2.3(c)) with five states and sub-state machine M2 (Fig. 2.3(d)) with three states. Two new states $R$ and $S$ are introduced to M1 and M2 respectively in order to retain the original functionality. The new states $R$ and $S$ are only for function equivalence and do not need to be encoded. We know that the matrix $B$ (see (2.13)) will not change for the original encoding, and hence there are two different situations of computing the faulty
transition probability after decomposition: (1) the initial state does not directly transit to state \( R \) or \( S \) (such as states \( C, E, F \)), and (2) the initial state has a direct path to state \( R \) or \( S \) (such as states \( D, A, B \)). Assume state \( D \) is the protected state in this design. To take a faulty transition path from state \( F \) to \( D \) which is an example of the first situation, we can calculate the probability according to (2.12), (2.13), and (2.15) based on FSM \( M_1 \):

\[
T'_{F,D} = T^M_{F,\cdot} \cdot B^M_{\cdot,D} = 0.5(B_{C,D}) + 0(B_{D,D}) + 0.5(B_{E,D}) + 0(B_{F,D})
\]  

(2.18)

where \( T^M_{F,\cdot} \) is the row of the next state probability for \( M_1 \), which is \([0.5, 0, 0.5, 0]\), and \( B^M_{\cdot,D} \) is the portion of matrix \( B \) with respect to the states in \( M_1 \).

Similarly, an example of the second situation is shown to calculate the faulty transition probability from state \( A \) to \( D \), and a two-step procedure needs to be taken as follows:

\[
T'_{A,D} = T^M_{A,\cdot} \cdot B^M_{\cdot,D} = 0(B_{A,D}) + 0(B_{B,D}) + 1(T'_S/A,D)
\]  

(2.19)

\[
T'_S/A,D = T^M_{R,\cdot} \cdot B^M_{\cdot,D} = 0.5(B_{C,D}) + 0(B_{D,D}) + 0(B_{E,D}) + 0.5(B_{F,D})
\]  

(2.20)

In this process, an intermediate term \( T'_S/A,D \) is calculated based on sub-FSM \( M_1 \) and then substituted into sub-FSM \( M_2 \) as the faulty transition probability from state \( S \) to \( D \). The computation is divided into two steps and the complexity is bounded by the size of the largest sub-FSM. Clearly, by decomposing the original FSM into smaller sub-FSMs, the complexity of the proposed model is reduced. Nevertheless, we acknowledge that the proposed method has its limitations. For example, for users who do not possess the FSM specifications of a design, or in the cases when circuits are not designed from state transition specifications, existing tools such as FSM complier
of Xilinx tool set, state machine viewer of Altera QuatusII, or various open-source tools such as AVFSM as presented in [48] will be needed to extract the FSMs. Also, in order to manage the computational complexity, FSM decomposition methods need to be applied in dealing with large FSMs. As discussed before, after decomposing a large FSM into smaller sub-FSMS, the computational complexity will be contained by the size of sub-FSMS. Furthermore, decomposition can be applied multiple times depending on the computation needs.

2.5 Improving FSM Security Through Re-encoding

In the previous section, we modeled the behavior of FSM under random fault injection attacks. The probability of each faulty transition path, in particular the unauthorized transitions to the protected states, can be derived from the proposed statistical model. In order to improve the resilience of an FSM design, the probabilities of these faulty transitions must be minimized. In this section, we will develop a technique to achieve this goal.

2.5.1 Identification of Protected States

By searching the indicator function $F_{\text{inde}c}$, it is possible to identify all the protected states as well as the accessibility between any states. In the indicator function, a value ‘1’ at entry $F_{\text{inde}c,ij}$ means there is a transition path from state $i$ to state $j$. Whether or not a state should be protected is determined by its starting state set.

The searching procedure is shown in Fig. 2.4. It will consist of $n$ loops to search through the connectivity of all $n$ states in an FSM. In the first loop, the states with
one-step path (e.g., direct transition path) to a target state are found by checking the corresponding value in the indicator function. These states will be saved along with their connection paths in space $N$ (e.g., with direct paths to the target state), and the rest states are saved in space $M$. If space $N$ is empty after the first iteration, meaning there is no direct transition to the target state, the loop will end and the target state will be protected from all the other states. Otherwise, the loop continues to move the elements in space $M$ to space $N$ when there is an indirect transition path from a state in $M$ to any states in $N$. Hence, the searching during each round depends on the current size of $N$. Both spaces will keep updated until either $M$ becomes an empty space, which means the target state of the current iteration is a normal state (e.g., all the states can access the target state); or no more transitions can be found from $M$ to $N$, which means the target state should be protected from the states in space $M$, as there is no direct or indirect transitions from any states in $M$ to the target state. Either condition will terminate the loop and move on to the next state. Once searching is done, all states will be classified into the normal states
and the protected states. Searching results will also show that every protected state should be protected from all the normal states, which is easy to understand because a normal state is accessible from other states by definition.

### 2.5.2 Re-encoding of FSM

In Section 3, we discussed some security vulnerabilities in FSM design. An important finding is that, since the access possibility of each state is usually non-uniform, some states may see more risks than the others. To achieve the expected function, not all transition paths need to be specified. This reflects on the indicator function as many zero elements. As the size of FSM increases, more transition paths may be left un-specified.

During the circuit operation, the possibilities of state accesses are also very different. There are always some states being accessed more frequently than the others. A portion of a probabilistic FSM transition graph is shown in Fig. 2.5 as an example. In this figure, state $A$ is protected from the normal states $B$ and $C$. In a specific design, these normal states have different access patterns. For example, it can be seen that state $B$ is a “hot spot” with many path transitions and a high access probability. In contrast, the probability of accessing state $C$ is relatively small. This makes state $B$ a much more vulnerable state, because if maliciously flipped, it has a higher chance to access the protected state $A$.

It can be seen from (2.13) that the probability of faulty transitions is a function of the Hamming distance between two states. A larger Hamming distance means that more register bits need to be upset to enable a transition. In random faulty injection, flipping of each register bit is relatively independent, and thus the probability of a
faulty transition path is effectively the product of the probability of each bit being maliciously flipped. Since the chance of a register being flipped is usually very small, with a larger Hamming distance, the probability of a faulty transition path is reduced. Considering this fact, a decision must be made to keep the Hamming distance as far as possible from the protected states to those states with high access probabilities. This will greatly mitigate the chance of malicious accesses to the protected states. For example, in Fig. 2.5, state $B$ should have a large Hamming distance to state $A$ in order to avoid faulty transitions from $B$ to $A$. On the other hand, it might be acceptable if state $C$ holds a small Hamming distance to state $A$ as state $C$ is accessed less frequently.

The unused states in an FSM design can be utilized to reduce security risks. In this paper, we propose a re-encoding algorithm to enhance state assignment and adjust the Hamming distance to the protected states. The proposed scheme manages security risks according to the design properties so as to avoid risky faulty paths. After searching for the protected states as shown in Fig. 2.4, all states will be tagged as either normal or protected. In addition, designers are also able to customize state tags according to their functions and security needs. The proposed algorithm consists of two steps: state re-ordering and state re-encoding. First, a binary vector space $space_s$ is defined to store all the possible encodings of $n$ state variables; therefore the size of $space_s$ is $2^n$ with width $n$. Then, two other vector spaces, $norm_s$ and $protect_s$, are exported from the searching program to store the normal and protected states, respectively. The total size of $norm_s$ and $protect_s$ is equal to or smaller than the size of $space_s$ as there are unused states.

In algorithm 1 (state re-ordering), $space_s$ is re-ordered according to the Hamming distance of each element to the elements in $protect_s$, and the states with larger
Algorithm 1 State re-ordering

Require: Originally encoded states are stored in $space_s$ of size "$n \times 2^n\$", where $n$ is the number of the state variables

Require: Encoding of protected states are saved in space $protect_s$, normal states are saved in space $norm_s$

procedure STATE RE-ORDERING
2:  $\text{norm}\_\text{len} \leftarrow \text{length}(\text{norm}_s)$;
   $\text{protect}\_\text{len} \leftarrow \text{length}(\text{protect}_s)$;
4:  $\text{top}$:
   if $i > 2^n$ or $j > \text{protect}\_\text{len}$ then return false
6:    end if
loop1:
8:  for $i = 1$, $i++$, while $i <= 2^n$
   if $\text{Hamming}(space_s(i), \text{protect}_s(1)) < \text{Hamming}(space_s(i + 1), \text{protect}_s(1))$ then
10:     $\text{temp} \leftarrow space_s(i)$.
     $space_s(i) \leftarrow space_s(i + 1)$.
12:     $space_s(i + 1) \leftarrow \text{temp}$.
     goto loop1.
14:    close;
   end if
end loop:
16: loop:
   for $j = 1$, $j++$, while $j < \text{protect}\_\text{len}$
18:     $\text{protect}_s(j) \leftarrow space_s(space\_len - j + 1)$
  goto loop.
20: close;

goto top.
22: end procedure
Algorithm 2 Encoding normal states according to the access frequency

**Require:** Transition function for all normal states are extracted from transition function $T_{ij}$ in error-less environment and store in space $P$

1: **procedure** STATE RE-ENCODING ($P$)
2: \hspace{1em} $Space \leftarrow P$
3: \hspace{1em} $Index \leftarrow Index(norm_s)$.
4: \hspace{1em} $Sum \leftarrow Sum(Space)$.
5: \hspace{1em} $R_{space} \leftarrow Index \cup Sum$. \hspace{1em} ($\cup$ here donates concatenation of two spaces)
6: \hspace{1em} $SortRow(R_{space}, 2)$.
7: \hspace{1em} Output $Index$;
8: \hspace{1em} close;
9: **end procedure**

Hamming distance to the protected states will be brought to the top. Note that if there are multiple protected states, the designer could re-order the $space_s$ according to the priorities of these protected states. Also, we are not concerned too much about the transitions between two protected states, as it is more important to prevent the malicious access into these states from the outside. In other words, it is suitable to encode all protected states in a bundle that is difficult for all the normal states to access. Hence, the elements in $space_s$ are only compared with the top elements in $protect_s$ during re-ordering. After that, other protected states will be re-encoded by the states with the smallest Hamming distance to the first protected state. By doing so, the overall Hamming distance from the protected states to the normal states is maximized.

The re-encoding of normal states is summarized in algorithm 2. First, all the normal states need to be re-ranked according to their access probabilities. The access probability of each state can be derived from the transition function $T_{ij}$ as defined in (2.5), where the total probability for a state to be accessed is the sum of the
corresponding column. By doing so, we acquire an updated state index where a smaller index number indicates a more frequently accessed state. According to this index, the normal states can be re-assigned by the re-ordered state space from space_s. For example, if state 5 is the most frequently accessed normal state, then it is indexed by 1. Meanwhile, the space_s is re-ordered according to the Hamming distance to the protected state 15, coded as “1111”. Hence, state 5 will be re-assigned to code “0000”, as this is the first state found in the re-ordered space_s. After state re-assignment, it will ensure that normal states are moved further away in Hamming distance from the protected states. This reduces the risk of faulty transitions caused by hardware attacks.

![Average Faulty Transition Probability](image)

**Figure 2.6:** Comparison of the average faulty transition probability.
2.6 Simulation and evaluation

In this section, FSM benchmarks from LGsynth91 [53] will be evaluated using the proposed work. We built the probabilistic transition model for each benchmark and calculated the faulty transition rates to the protected states using the method developed in this paper. Quantitative results were derived for measuring the design risks in a fault injection scenario. We also applied the re-encoding algorithm to mitigate the design risks, and the new designs were re-evaluated to show the improvement. The probabilistic transition matrices of the benchmarks were derived and evaluated.
through simulations. The synthesis information of the circuit implementations with
different encoding schemes was obtained in the Synopsys EDA environment.

2.6.1 Benchmark Simulation

It is worth mentioning that in most of the LGsynth91 benchmarks, states are de-
signed to be accessible from each other due to design simplicity. To evaluate fault
injection attacks, we remove some paths from the original designs to create the fea-
ture of protected states. The removed paths need to be carefully selected in order to
keep the modified design still reflect the properties of the original design. For this
consideration, an algorithm was applied to the derived transition matrix to find the
least likely visited state, and the transition paths to this state were removed. For
the purpose of illustration, we only create one protected state in each benchmark.
Eight different benchmarks were simulated, and the information of these benchmarks
is listed in Table 2.1.

We define error variables $\epsilon$ and $\gamma$ to represent the errors at combinational logic
and registers, respectively. The probability of errors at each circuit node obeys a
binomial distribution with a probability $w$. To consider the attenuation of errors in
combinational logic, a variable $\alpha$ is applied to the probability of $\epsilon$ so that $\epsilon = \alpha \cdot w$.
Here $0 \leq \alpha \leq 1$, which indicates the probability of error propagation from the fault
site to the register input \cite{54}. For the purpose of illustration, the value of $\alpha$ is assumed
to be 0.5 in our simulation. On the other hand, $\gamma$ is equal to $w$ as there is no error
propagation effect in register errors (e.g., the input to a FSM). For each benchmark,
the average faulty transition rate from all the normal states to the protected state
is determined using the proposed model in Section 3, as shown in Fig. 5. In these
simulations, the value of $w$ was chosen to be $5e-3$ [55]. We also provide other evaluation results to compare the worst-case faulty transition path under different error rates. Clearly, after the re-encoding technique is applied, both the average faulty transition rate and the transition rate of the worst faulty path (i.e., the most vulnerable path) have been significantly reduced.

Since FSM circuits are often designed with non-uniformly distributed transition paths, there always exist some paths that have high transition probabilities. Identifying these paths is critical at the design phase, as they are more likely to allow the adversary to gain the access to the protected states. To study this problem, we

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>No. of states</th>
<th>No. of inputs</th>
<th>No. of transitions</th>
<th>No. of removed edges</th>
<th>No. of removed transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbtas</td>
<td>6</td>
<td>2</td>
<td>24</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>dk14</td>
<td>7</td>
<td>3</td>
<td>56</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>beecount</td>
<td>7</td>
<td>3</td>
<td>28</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>bbara</td>
<td>10</td>
<td>4</td>
<td>60</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>train11</td>
<td>11</td>
<td>2</td>
<td>25</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>dk512</td>
<td>15</td>
<td>1</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ex2</td>
<td>19</td>
<td>2</td>
<td>72</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>dk16</td>
<td>27</td>
<td>2</td>
<td>108</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.1: Benchmark information

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Comb Area ($\mu m^2$)</th>
<th>Non-Comb Area ($\mu m^2$)</th>
<th>Total Area ($\mu m^2$)</th>
<th>Area Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Orig</td>
<td>Modi</td>
<td>Orig</td>
<td>Modi</td>
</tr>
<tr>
<td>bbtas</td>
<td>93</td>
<td>108</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>dk14</td>
<td>430</td>
<td>449</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>beecount</td>
<td>318</td>
<td>346</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>bbara</td>
<td>301</td>
<td>298</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>train11</td>
<td>343</td>
<td>344</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>dk512</td>
<td>261</td>
<td>273</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>ex2</td>
<td>1187</td>
<td>1180</td>
<td>73</td>
<td>73</td>
</tr>
<tr>
<td>dk16</td>
<td>998</td>
<td>1114</td>
<td>73</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 2.2: Area overhead of original vs. modified encoding schemes.
evaluated the worst-case scenario of each benchmark when error rate $w$ varies from $1e^{-5}$ to $5e^{-2}$. The results are shown in Fig. 2.7, in which the dash lines represent the transition probability of the most vulnerable path in the original designs, and the solid lines show the improved transition probability in the modified FSM using the proposed re-encoding technique. First of all, we can see that the probability of faulty transitions increases linearly with the induced errors, which is consistent with the empirical observation. Second, by re-ordering the states according to their Hamming distance to the protected state, the overall faulty transition rate is reduced. This demonstrates that the proposed technique is more robust against fault injection attacks. As shown, the transition probabilities of the most vulnerable paths can be very high in some situations. For example, benchmark *bbara* has a worst-case faulty transition rate as high as 18%. This value can be reduced almost by half after applying the proposed re-encoding technique. Actually, the risk of the most vulnerable path is reduced by more than 50% in most benchmarks, indicating that the resilience of the FSM circuits against faulty transitions is improved.

From the results above, we can also see that for some cases the improvement is not so obvious. This is largely due to the fact that the qualities of the original FSM designs are quite different. In general, the effectiveness of state re-encoding increases as the complexity of a design goes higher and the design itself becomes more complicated. This is because a larger input space means a huge number of possible input patterns. Hence it is difficult for the designer to assign transition paths for all these inputs. The resulting design will be incomplete and non-uniformly distributed. On the other hand, as the size of state space goes up, there will be more unspecified states in the circuit. The more incomplete and non-uniform a design is, the more vulnerable it could be to the induced faults. Therefore, the state re-encoding technique tends to
be more effective for a large FSM design.

To evaluate the hardware overhead of the proposed technique, the benchmarks were implemented using Verilog language in the Synopsys EDA environment with a 65nm CMOS technology. Table 2.2 compares the areas of these designs in the original linear encoding scheme and the modified implementations. For all the benchmarks being studied, the area overhead due to re-encoding is no more than 10%. It is worth mentioning that state re-encoding does not necessarily introduce area overhead as opposed to conventional security enhancement schemes where either redundant logic [47] or additional functional blocks [18] will be needed. Actually, area reduction can be seen in some cases. This is because the original encoding scheme is the standard linear binary encoding, which is not intended for area optimization. The proposed re-encoding technique aims to mitigate the security risks from state transitions without increasing the number of state registers. The area change is mainly due to different placement and routing of combinational logic implementations. Further improvement can be achieved if redundant registers are allowed. For example, in Section 4.2 we calculated the faulty transition probability between state [1,1,0,1] and [1,0,1,1] in Example 1. If one additional bit is allowed for state encoding, these two states could potentially be encoded as [1,1,0,1,0] and [1,0,1,1,1], respectively. As a result, the Hamming distance between the two codewords is increased by one. The faulty transition probability $B_{1,2}$ will be reduced to $2e - 10$ in this case. In the extreme case that expands the coding space to the maximum, the Hamming distances between any two states are all equal to $2^n - 1$. This is also known as the “one hot” encoding, where the Hamming distances between all states are maximized regardless of their transition characteristics. While the probabilities of all faulty transitions (including those to normal states) would be reduced, the hardware overhead is also significant.
2.6.2 A Case Study

We will demonstrate the application of the proposed technique through an open source IP. In practice, the FSM model of an IP implementation could be extracted from its netlist using tools such as in [48]. We studied the control specification of a RISC xr16 CPU core and extracted the FSM of its pipeline control path. The RISC xr16 CPU core is a simple 16-bit reduced instruction set processor, which is optimized for an efficient pipeline implementation. It features a classic 3-stage pipeline (IF, DC, EX) of 16-bit instructions, byte addressable memory (load/store), integrated direct memory access (DMA) engine, fast interrupt handling and \( \sim 1.4 \) cycles per instruction in a zero wait-state memory system.

The pipeline control logic of RISC xr16 is extracted as an FSM model shown in Fig. 2.8 based on the implementation in [56]. The pipeline control interacts with memory controller, datapath, decoder, registers and operand selectors. The regular
Figure 2.9: RISC x16 pipeline control flow FSM model: (top) faulty transition probability comparison on the worst case and (bottom) on average.

pipeline is scheduled in the state $IF$ until it has been interrupted by DMA, load/store (LS), interrupt (INT) or Jump and branch requests. In the RISC architecture, the control and status registers (CSRs) are used to divide the system privilege. The privilege of the current application is marked by one bit of the CSRs which is only flipped through interrupts. Since the interrupt handler always runs at the kernel mode, the privilege bit is set whenever an interrupt is handled, and then set back to user-mode when executing the $iret$ instruction. Therefore, maliciously transition
paths to state DCINT (int in DC stage) need to be protected in order to prevent unauthorized kernel accesses.

Since the transition pattern of a processor is program dependent, we took a typical condition with 36% Load instructions, 19% Store instructions, 15% DMA, 7% Branch, 2% Jump, 1% Interrupt, and 20% others, and then calculated the probability of faulty transitions to the state DCINT. We also calculated the average probability and the worst case of a regular binary state encoding and the proposed re-encoding scheme. The results are shown in Fig. [2.9] for comparison with fault injection rates from 5.0e-4 to 5.0e-2. It can be seen that 50%–85% reduction in faulty transitions can be achieved by the proposed technique.

2.7 Conclusions

In this paper, we analyzed the design vulnerabilities and potential security risks in conventional digital circuits subject to randomly injected or induced errors. The FSM-based stochastic model was developed to evaluate the probabilistic behavior of FSM transitions under hardware attacks. Although the problem of faulty transitions is not entirely eliminated but mitigated in the proposed re-encoding scheme, compared to the conventional hardware-based fault-tolerant design techniques, the proposed method avoids the duplication/redundancy in the circuit design, thereby leading to much smaller hardware overheads. Note that in this paper we only considered one protected state in each design so that the proposed idea can be presented in a simple but clear way. With a natural extension the proposed technique can cover multiple protected states. The proposed technique is best-suited for light-weight or
resource-constraint applications such as Internet of Things. Future work will focus on reducing the complexity of this technique by exploring efficient FSM characterization techniques that avoid explicit state enumeration of a given circuit.
Chapter 3

Design for Test and Hardware Security Utilizing Retention Loss of Memristors

3.1 Introduction

Modern IC design requires structural testing as chip fabrication may introduce defects. Scan chain test is the most commonly employed design-for-testability (DFT) scheme to increase the observability and controllability of the device-under-test (DUT). This scheme allows the tester to access the internal nodes of DUT to detect any manufacturing related defects. Unfortunately, a scan-based testing structure, when exploited by malicious parties, can make the circuit susceptible to different types of attacks and information leakages. Reports have shown that scan chains are widely used by malicious parties to retrieve the important information of IC chips. In fact, scan chains allow attackers to retrieve more comprehensive and accurate information of
chip internal structures than any other side-channel attacks can, and thus put IC chips in serious security threat.

Common cryptographic systems, such as Advanced Encryption Standard (AES) and Data Encryption Standard (DES), have been hacked by scan-based attacks [57], [58], [3], [5]. In [3], scan chains were utilized to retrieve the key of the DES algorithm. Because of the iterative computation of DES, different parts of the user key is carried by each round of key generation. If attackers can switch scan chains between the normal and test modes, by putting carefully prepared plaintext contents (i.e., plaintexts with only one bit difference) into the DES circuit, they are able to determine the user key from scan outputs during the internal rounds. Similarly, scan-based attacks can be carried out against AES ciphers as well [5]. It takes advantage of the basic differential properties of AES as a block cipher, where if a pair of plaintext inputs are only one bit different in the least significant bit of any byte, the possibility of output difference of first round is restricted. Moreover, only a few of these output pairs can be generated by a unique pair of S-box inputs. Hence, attackers can switch between the test and normal modes of scan chains in order to observe the first round response for input pairs with designated differences. The overall computational complexity for retrieving the user key is greatly reduced with the acquired internal state information. Furthermore, the need of switching between the scan and test modes can also be avoided once attackers figure out the mapping between the input bit to the scan cell [4].

Many secure scan chain designs have been proposed to restrict the access of unauthorized users or to confuse the scan response of the device [59], [60], [61]. However, these schemes are subject to several limitations such as deficient scan protection, complex architecture design, or compromise of chip testability. Previously, we proposed a
secure scan chain obfuscation design utilizing the inherent performance degradation of memristor cells [18]. The scan chain output is multiplied with a built-in memristor array before it is received by the tester. The authorized tester allows regular refreshes of the memristor content; otherwise, the scan output is obfuscated by the natural degradation of memristor devices. Without losing any test capability of the original scan chain architecture, the proposed scheme can provide configurable and truly randomized confusion to the scan response with minor complexity. In this paper, we extend our past work with the following contributions. First, we propose a new design that achieves scan obfuscation on analog-based memristor devices and develop an analytical scheme to recover scan responses. Second, we improve obfuscation performance by developing statistical models as a guidance for determining the key design parameters, such as memristor refresh frequency. Third, circuit implementations are presented showing the memristor-based obfuscation method with very low hardware overhead and little timing and power constraints. Furthermore, the proposed approach can be easily adopted by the existing scan chain structures without the need of re-designs or modifications.

The reminder of this paper is organized as follows. In Section II, we briefly review the characteristics and existing countermeasures for scan-based attacks. Memristor devices and their retention loss is studied in Section III. The proposed secure scan chain technique is presented in Section IV and reliable scan data recovery scheme is discussed in Section V. Details about the circuit implementation are discussed in Section VI. Section VII presents the simulation results and performance evaluation in terms of hardware cost, timing and power consumption as well as security enhancement.
3.2 Countermeasures for Scan-based Attacks

Scan-based attacks exploit the leakage of intermediate computations in iterative ciphers to reduce the complexity of key extraction. These attacks are usually performed under the following conditions: (1) the attacker has access to the scan chain, and can even switch between the normal and scan modes as needed, but the structure of the scan chain is unknown; (2) the key and round-keys are stored in the secure RAM/ROM; and (3) the cryptographic circuits under attack are known to the attacker since these are public information. The procedure usually consists of two phases: analysis in hardware on the actual circuits and analysis in software, along with additional steps for deriving the internal hardware and testing structure. As a result, hundreds of well-selected scan vectors are needed to complete the attack, which can translate to tens of thousands or more test cycles depending on the DUTs [57], [3].

Existing countermeasures to scan-based attacks either rely on restricting the access to the scan functionality only to the legitimate users [59], [60], or obfuscating scan data so that the unauthorized parities cannot interpret [62], [63]. In the industry, a common practice is to physically disconnect the scan chain after manufacture testing by burning the anti-fuses. Although this method is easy to implement, the drawback is also obvious, since no follow-up testing will be possible once the chips are sold to the market. In [60], a locking mechanism is proposed such that only the users with the matched key can have access to the scan infrastructure; otherwise both scan input and output accesses will be gated by a test wrapper. The design complexity of this method depends on the key length. A higher security level introduces large overhead.

To obfuscate the scan response in a way that only the authorized tester can interpret, simple scan chain modifications are suggested to randomly insert inverter or add
XORs between scan flip-flops in order to confuse an unauthorized tester [62]. However, these methods are ineffective against differential scan attacks [64]. The similar problem exists among the security enhancement provided by the advanced testing techniques, such as scan compaction [65]. In addition, some techniques relying on the structural modifications of scan chains are proposed to prevent unauthorized users from accessing the test data. For example, in [59], [61], scan chain scrambling schemes are designed to obfuscate scan output sequences from unauthorized testers. To do so, the scan chain is split into small segments connected to a scrambling logic such as a random number generator (RNG) or a linear feedback shift register (LFSR). The scan output is in a predictable order only when a correct test key is received; otherwise the scan chain will work in a randomly ordered manner. The problem of these designs is that they require significant scan chain modifications or complicated locking mechanisms, which can lead to large logic and routing overheads. Also, the security of the design depends on its complexity. For example, a report shows that the scrambling techniques may still leak the data dependency on the secret key through the parity information [66]. In addition, the data would not be scrambled within the same sub-scan chain. The requirement of re-designing the scan structure also limits the compatibility of these techniques on various test interfaces, protocols, and designs with third-party IPs. Attempt has been made to encrypt the scan output with additional encryption blocks [63], but on the other hand, it raises new problems of testing the corresponding scan data encryption circuit. Also, design complexity and hardware overhead make such techniques unsuitable for resource-constrained devices such as Internet-of-Things (IoT).

In this work, we propose a memristor-based secure scan technique suitable for resource-constrained implementations. By exploiting the intrinsic retention loss of
memristor devices, the scan data can be obfuscated with true randomness. Furthermore, since memristor devices feature some unique properties such as high density, high speed and low power consumption, light-weight scan wrappers can be implemented. Compared to the existing approaches, memristor devices in our design operate in the analog mode, which further reduces the hardware overhead.

3.3 memristor devices and drifting effect

3.3.1 Memristor devices

Memristors have gained significant attention in recent years as one of the promising devices for future hardware design. With the nonvolatile nature, memristors also feature good properties such as high density, fast operating speed, multiple stable states, and low power consumption. Memristor devices can be employed in digital or analog circuits, such as neuromorphic computing networks and multi-state memory cells. The fabrication of memristor devices can be done in different ways, each resulting in some unique characteristics. The most common structure is Metal-Insulator-Metal (MIM) bipolar devices, where metal on both ends are used as electrodes, and the state of the cell is determined by the growth of a conductive filament inside the insulator material. Take Titanium oxide memristor ($Pt/TiO_2/Pt$) as an example. When a positive bias voltage is applied, electron ionization occurs near the anode area and turns $Ti_{4+}$ into $Ti_{3+}$. Positively charged $Ti_{3+}$ starts to react with free $O_2$ and generates $Ti_2O_3$, which accumulates around the cathode and hence forms a conductive nanowire growing toward the anode [26]. The conductance of the cell is determined
by the length of the filament and its geometric dimensions, such that:

\[
G_{\text{mem}} = \frac{1}{R_{\text{mem}}} = \frac{A}{\rho_1 l + \rho_2 (d - l)}, \tag{3.1}
\]

where \( A \) and \( d \) are the cross section area and the total length of the cell, respectively, \( l \) is the length of the filament, and \( \rho_1 \) and \( \rho_2 \) are the resistivity of high-conductive filament and high-resistive TiO\(_2\), respectively.

The growth of conductive filaments is sensitive to many factors such as device dimensions, applied voltage and operation temperature. Another commonly reported problem is process variations. When a memristor cell is fabricated, there are tough issues such as line edge roughness (LER) and thickness fluctuations (TF). As a result, the response of a memristor cell can vary slightly under the same operation condition. The switching mechanism of resistive memristor devices has been well-understood, analytical models were developed for typical MIM devices with a resistive material layer, such as TiO\(_2\), TaO\(_x\) and ZnO, under the influence of process variations \[67\], \[68\]. Studies have shown that due to process variations, conductivity with a quasi-Gaussian distribution \[30\], \[29\] was observed when the same writing operation was applied to a memristor array. This leads to uncertainties and randomness in memory states. However, it makes various memristors suitable for introducing randomness for security purpose, such as being utilized as true random number generator (TRNG) \[15\], physical unclonable function (PUF) \[16\], and scan security enhancement discussed in this paper.
3.3.2 Drifting effect

Another significant effect of the uncertainties during state switching is that memristor devices endure retention loss because read is destructive [69]. In other words, the read operation will cause state drifting in memristor cells. Similar to other resistive memory devices, the write and read operations of memristor devices are accomplished by applying bias voltages. As discussed before, bias voltages facilitate the chemical reactions in a cell, and hence change the state of the cell in the form of increasing or decreasing the conductive filament length. Normally, a high voltage pulse with a long duration is used to write the memristor cell. On the other hand, when reading the memristor cell, a small voltage pulse is applied in order not to perturb its current state. However, experiments have shown [70] that small state drifting can still occur when a small bias voltage is applied. This may alter the memory state eventually.

In order to prevent memory states from being perturbed during the read operation, some works [69] utilized a zero-net-flux read pattern as shown in Fig. 3.1. The read pattern is composed of positive and negative pulses with equal width so that it brings the memristor back to its original state after each read operation. However, since the pulse generation process also bears a certain tolerance level, the uneven positive and negative pulses can still cause shifting on memristor states. For example, assuming there are variations on the generated pulse width as in Fig. 3.1, when the initial state of a cell is logic zero, an excessive flux injection $\Delta \phi$ due to the difference $\Delta T$ between pulse widths $T_n$ and $T_p$ will gradually shift the state of the cell and may eventually flip the bit. The similar problem could happen to memristors with logic one as well [70]. Therefore, the width of reading pulses must be carefully controlled to ensure the internal state in the safe margin.
The flux injection required to change the memristor state from $w_0$ to $w$ is expressed as \[69\]:

$$\Delta \phi = \frac{\phi_D}{R_{off}^2} \left\{ (R(w_0))^2 - (R(w))^2 \right\}, \quad (3.2)$$

where $\phi_D$ is the amount of the flux to switch on or off of a cell, and $R_{off}$ is the resistance of a cell when it is at the off state. As the number of cell accesses increases, more unbalanced flux injections would occur, which bring the state of the cell to an uncertain state. In addition, process variations also cause cells to respond to the injected flux in a less predictable way. Therefore, in memristor-based memory circuits, regular refreshes or some limitations on read access are required to maintain cell stability. In general, the retention loss process shares a lot of similarity with the memory loss of biological systems. It can be described as a stretched-exponential function \[71\], also known as the Kohlrausch’s law given below:

$$\omega(t) = \omega_0 \cdot \exp \left[ -\left(\frac{t}{\tau}\right)^\beta \right], \quad (3.3)$$

where $\tau$ is the characteristic relaxation time and $\beta$ is the stretch parameter ranging.
between 0 and 1. In memristor circuits, time $t$ can be interpreted as the increment of clock cycles.

3.4 Memristor-based Scan Security Enhancement

Exploiting the properties of memristor devices, we propose a new scan chain scheme to address scan attacks and enhance the security of testing. The basic idea is to store a matrix only known to the legitimate tester in a memristor array and use it to modulate scan outputs. Due to process variations and destructive reads, memristor cells lose their values and thus randomize the scan response. A configurable refresh scheme is designed that is only accessible by the authorized test engineers in order to obtain the uncontaminated scan data. For unauthorized parties, the received scan output is not only altered by an unknown matrix but also contaminated by the random retention loss, thereby achieving thorough scan obfuscation. The proposed scheme does not introduce much complexity as the memristor crossbar structure is a
natural fit for such task. Furthermore, it can operate with little power and timing overhead when compared to the regular testing process.

### 3.4.1 Overall architecture

The proposed secure scan readout architecture is shown in Fig. 3.2. A $n \times k$ memristor array is implemented for the obfuscation purpose. A key comparator allows authorized users with the test key to enable the memory refresh mechanism. The refresh controller reloads the memristor array with the preset values regularly so that the authorized tester is able to receive the correct test response. For the unauthorized user, refresh is disabled due to the absence of the key, and after a certain period the testing output will become erroneous due to memristor retention loss. Matrix multiplication is a costly operation in digital circuits; however, it can be efficiently implemented in the memristor crossbar structure as illustrated in Fig 3.3 [26, 72]. Since the binary scan response vector $S$ can act as the switch for reading each row, the output vector $Y$ is effectively the multiplication of $S$ and the conductance of memristor cells at the present state. Therefore the size of the array depends on the buffered size $n$ of scan outputs. It is worth noting that the sneak-path problem of the crossbar memristor array can be coped with using diffusive type devices [73], and transistor/diode gated solutions [74]. Initially a binary matrix $\Phi$ should be stored in the array as the base of computation. Overall, this architecture can provide a double-layer protection. First, the scan output is obfuscated by multiplying with the matrix $\Phi$. Second, even if the attacker figures out the content of $\Phi$, without the key he cannot acquire enough correct scan outputs to effectively launch scan chain attacks.

Initially, we can expect the same conductance for memristor cells with the same
value ("0" or "1") once the memristor array is refreshed to the matrix $\Phi$. Then, memristor cells begin to loss their values as soon as the multiplication starts. Therefore, the multiplication implemented on the memristor array should be considered as a non-ideal analog process. Testing process requires high reliability and accuracy. Thus the memristor array needs to be regularly refreshed to avoid testing errors. The critical issue in this design is to ensure a reliable recovery of digital scan response $S$, and based on that, to determine the frequency of reloading the matrix $\Phi$ to the memristor array.

### 3.4.2 Scan obfuscation

As discussed above, by implementing matrix multiplications in the crossbar memristor array, scan outputs can be obfuscated through a simple mechanism. As depicted in Fig 3.3, the chip scan response $S$ is shifted out and stored in a register with the size of $n$. By multiplying this vector with the $n \times k$ matrix $\Phi$ stored in the memristor array, vector $Y$ of size $k$ will obtained, i.e.,

$$Y = \Phi \times S. \quad \text{(3.4)}$$

As long as the matrix $\Phi$ has full rank, based on the linear system theories \[75\], if $n = k$, $S$ can be easily recovered from $Y$ by

$$S = \Phi^{-1} \times Y. \quad \text{(3.5)}$$

Or if $n < k$, $S$ also has unique solution by solving the linear equations if both $Y$ and $\Phi$ are given. The number of columns $k$ needs to be equal to or larger than
n to ensure reliable recovery of the scan response $S$ from the received output $Y$ (see Section V).

![Figure 3.3: Matrix multiplications implemented on a memristor crossbar array.](image)

Note that to complete (3.4), read operations are performed on every memristor cell, which will perturb the preset memory states. Due to process variations and drifting effects discussed before, random state shifting will occur in the memristor array. In our previous work [18], we have discussed the feasibility of using a digitalized memristor array for scan obfuscation. After certain clock cycles, the accumulated effect will cause bit flips, which introduce errors in the memristor array and make the scan response $S$ unrecoverable. The introduced errors in the digitalized array follow a Bernoulli distribution [18]. Simulation has shown that once errors start to occur, the correlation between the original and obfuscated scan outputs will drop and eventually approach zero, which indicates a good data obfuscation. However, the disadvantage of this design is the need of comparator circuits at each memristor cell, which introduces large hardware overhead. To address this issue, the obfuscation
design in this paper is based on the analog memristor array where comparator circuits are not needed. The scan response vector \( S \) is multiplied with the memristor cells having continuously changing resistance/conductance values. This can be expressed by adopting the conductance matrix \( G \), of which each element corresponds to the conductance of a cell in the memristor array. The initial state of \( G \) is determined by the binary matrix \( \Phi \) as:

\[
G_{ij} = \{G_{on} \text{ if } \Phi_{ij} = 1 \text{ or } G_{off} \text{ if } \Phi_{ij} = 0 \}, \tag{3.6}
\]

and the received scan response \( Y \) will become:

\[
Y = G \times S. \tag{3.7}
\]

The random memory shifting at each read cycle can be modeled as an error matrix \( \Theta \) with the same dimension as \( G \). After \( m \) read cycles, the conductance matrix \( G \) will be contaminated to \( \hat{G} \), which can be expressed as:

\[
\hat{G}_{ij} = G_{ij} + \sum_{k=1}^{m} \Theta_{ij}^k. \tag{3.8}
\]

Also, the conductance of a memristor cell is bounded by the conductance of the on and off states, \( G_{on} \) and \( G_{off} \), respectively, as below:

\[
\hat{G}_{ij} = \begin{cases} 
G_{on} & \text{if } \hat{G}_{ij} \geq G_{on} \\
G_{ij} + \sum_{k=1}^{m} \Theta_{ij}^k, & \text{others} \\
G_{off} & \text{if } \hat{G}_{ij} \leq G_{off}
\end{cases} \tag{3.9}
\]
Reports have shown that due to manufacturing and environmental variations, such as variations on dimensions, temperature and time of measurement, the conductivity changes of memristor devices under the same given switching condition demonstrate a quasi-Gaussian distribution [30]. The mismatches of the positive and negative read pulses also have a normal distribution. Thus, the errors at each read cycle can be considered as independent Gaussian noise as below:

\[ f(\Theta_{ij}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\Theta_{ij}^2}{2\sigma^2} \right) \], \quad (3.10)\]

where the variance \(\sigma^2\) can be determined during manufacture testing. Figure 3.4 shows the simulated memristor degradation process as discussed above. It is the visualization of a 32×32 memristor array before (left figure) and after (right figure) being read for 1000 cycles. The on and off states are normalized as “1” (white blocks) and “0” (black blocks), respectively. Due to the repeated read operations, shifting-induced errors occur and result in intermediate conductance values in the right figure (gray blocks). Here we choose the variance of errors \(\sigma^2\) to be 1e-3 for the purpose of illustration. Due to the retention loss, the scan output becomes

\[ \hat{Y} = \hat{G} \times S. \] \quad (3.11)\]

With the repeated read operations, \(Y\) and \(\hat{Y}\) will be less and less correlated, which means the received scan output will become more and more erroneous over time. Eventually one would not be able to recover the scan vector \(S\) from \(\hat{Y}\) even with the knowledge of the matrix \(\Phi\).
3.4.3 Security, endurance and reliability

The security of the proposed architecture is ensured by two mechanisms. First, the matrix content \( \Phi \) is used to obfuscate the scan output data. This is similar to the classic Hill cipher [76], which utilizes linear transformations on the message space by multiplying the message vector with a key matrix to achieve the encryption goal. As linear transformations are vulnerable to the known plaintext attack, we can restrict the multiplication function to taking place only after the scan chain captures the circuit response. Second, the degradation of the memristor array introduces random scan data obfuscation, and ensures the security even if the matrix \( \Phi \) is leaked. Since read operations induce state drifting in memristors, as long as the refresh is not enabled, the memristor array will lose its values in an unpredictable way.

As discussed before, the scan based attacks need to collect a large amount of test data to succeed. Memristor degradation will contaminate the scan output far before that. On the other hand, testing reliability needs to be ensured during an authorized test. The endurance of memristor devices can support this requirement. In general,
the resistive memristor devices can endure $10^9$ write cycles \cite{68}, which means if the memristor content is refreshed every 1000 clock cycles as discussed in the proposed design, a total of $10^{12}$ test cycles can be supported. Since the memory refresh rate needs to be determined to ensure reliable testing, we develop an analytical scheme to model the scan obfuscation process. This scheme can be used to determine the memory refresh frequency as well as to recover the scan data reliably. The details of reliable scan data recovery will be discussed in the next section.

### 3.5 Scan recovery

With regular memory refreshes, authorized test engineers are able to access the scan chains. However, the conductance of memory cells is sensitive to read operations, and scan outputs can be tainted in a very short time. Since testing requires a high reliability, the critical requirement is to ensure that scan vectors can always be obtained correctly. Thus, the memory refresh frequency must be carefully determined. In this paper, we utilize the Linear Mean Square Estimator (LMSE) to recover the vector $S$ in the presence of memristor non-idealities. The benefit of this method is that we can quantitatively find the confidence level of such estimate to determine the refresh frequency.

The mathematical formula of LMSE is described in \cite{77}. The specific problem in this paper can be considered as the scan response $S$ going through a linear system $\Phi$ with noise $\Theta$. Hence, the vector $\hat{Y}$ is the system output with $k$ observations. In this case, the LMSE $\hat{S}$ of the original scan response can be expressed as:
where $G$ is the expected conductivity matrix of the memristor array determined by the pre-defined matrix $\Phi$. Note that $G$ needs to be a full-rank matrix. The vector $\hat{Y}$ is the collected current value at the output, which will be digitized by ADCs. In this way, analog computation is fulfilled in a digital environment simply by utilizing the digital scan vector $S$ as the enabling signals for the memristor read operations. One way to improve the estimation quality is to increase the number of observations $k$. We define the size ratio between the column and the row of $\Phi$ as $r = \frac{k}{n}$. The higher the ratio $r$, the more accurate the estimator becomes. As for the test engineers, two steps are needed once the vector $\hat{Y}$ is received: (i) to find the $\tilde{S}$ using LMSE according to (3.12), and (ii) to derive the scan response $S$ from its estimator $\tilde{S}$.

Since the scan response is a binary sequence, test engineers can recover $S_i$ based on its estimator $\tilde{S}_i$ as below:

$$
S'_i = \begin{cases} 
1 & \text{if } \tilde{S}_i > 0.5, \\
0 & \text{if } \tilde{S}_i \leq 0.5.
\end{cases}
$$

(3.13)

where $S'$ is the recovered scan vector. Since $S'$ is derived from the estimator $\tilde{S}$, it is important to understand the quality of this estimate in order to ensure reliable testing. It is known that the confidence level of the estimator $\tilde{S}$ can be quantified using the covariance matrix as $(G^T G)^{-1}\sigma^2$ [78]. This gives the estimate of the variance of each $\tilde{S}_i$ as [78],

$$
\text{var}(\tilde{S}_i) = r_i^2 \sigma^2,
$$

(3.14)
where $\tau_i^2$ is the $i$th element on the diagonal of square matrix $(G^T G)^{-1}$. The estimator of the variance of the accumulated errors from previous cycles is denoted as $\tilde{\sigma}^2$. As discussed before, $\sigma^2$ is the variance of error matrix $\Theta$, which is added to the conductance matrix $G$ at every read cycle. Hence, the received errors after the array multiplication can be written as:

$$\hat{\Theta} = \Theta \times S.$$  \hfill (3.15)

The estimator of its variance, for $\kappa$ clock cycles, can be expressed as:

$$\tilde{\sigma}^2 = S^T \times \kappa \cdot \sigma^2 \times S.$$  \hfill (3.16)

The confidence interval for each estimator $\tilde{S}_i$ can be obtained as:

$$S_i' \in [\tilde{S}_i - c \cdot \sqrt{\text{var}(\tilde{S}_i)}, \tilde{S}_i + c \cdot \sqrt{\text{var}(\tilde{S}_i)}],$$  \hfill (3.17)

where $c$ is a constant chosen by the confidence level. For example, a 95% confidence level ($c = 1.96$) means 95% of the chance that the actual value of $S_i$ will fall into this interval [79]. Since testing requires high accuracy, $c$ needs to be chosen for a high confidence level. As we can see, the range of the confidence is not only determined by the confidence level but also depends on the variance of each estimator $\tilde{S}_i$. The distribution of the estimators and their confidence intervals can be illustrated in Fig. 3.5. With an appropriate refresh frequency, we can expect errors to be small and the estimated values not far off from “0” or “1”. As a result, the recovery scheme in (3.13) is reliable. The confidence interval, represented by the dashed circles, can have different radius for each estimator $\tilde{S}_i$. Typically, the higher the noise, the larger the
confidence interval becomes (due to high variance). Considering the recovery scheme utilized in (3.13), when the confidence interval is too large, there are chances that the original “0” is interpreted as “1” or vice versa. Hence, in order to ensure reliable scan data recovery, confidence intervals of all estimators should be bounded to not exceeding 0.5. This is a relatively loose bound showing when the LMSE recovered scan vector starts to be erroneous and memory refresh becomes necessary. Therefore, it helps us to establish a statistical model that determines the memory refresh frequency at the early design stage.

3.6 Circuit Implementation

A detailed implementation of the memristor array is depicted in Fig. 3.6. As shown, the multiplication operation in (3.11) is realized with a $n \times k$ memristor array. The scan vector $S$, which is a binary sequence, is used to enable the read pulse generator of the corresponding row. The circuit of the pulse generator is shown in Fig. 3.6(c).
Figure 3.6: Overall architecture of the proposed secure scan chain scheme.

When $P_{\text{sel}}$ line is high, a positive voltage is generated; and when $P_{\text{sel}}$ line is low, the generated voltage is reversed to negative. This circuit is used for both read and write pulse generation. The multiplication product is essentially the accumulated current $Y_1, Y_2, \ldots, Y_k$ collected from each column, which are analog signals. Therefore, ADCs are needed to convert $Y$ into digital signals.

To reduce the hardware overhead of ADCs, a multiplexed scheme is employed by using a $n$-bit Read Buffer as shown in Fig. 3.6(a). The scan output from the
circuit under test is acquired and shifted into a shift register at every test clock cycle. However, the matrix multiplication takes $n$ bits scan output as the multiplicand, so it only needs to be done once every $n$ clock cycles. Hence, the Read Buffer holds the scan output for $n$ cycles while the next $n$ bits are being shifted into the shift register. In the meanwhile, the ADC conversion can be done in sequential between columns. For example, if the data is sampled at the same rate as the test clock, only two ADCs will be needed when the size ratio $r \leq 2$. In this case, when the Read Buffer is updated, both ADCs start to convert the signal $Y_1$ and $Y_n$, and then switch to $Y_2$ and $Y_{n+1}$ at the next cycle and so on.

From the discussion in Section V, a memory array refresh frequency $f_m$ can be determined from the scan clock frequency $f$. During each refresh, the memristor array needs to be written to its pre-defined value. The entire memory refresh can be done row-by-row by using a Re-load Buffer and a Row Selector signal, which switches from the 1st row to the $n^{th}$ row once the write mode is enabled. The number of pulse generators needed for $V_{\text{write}}$ will be reduced to $k$, and one complete memory refresh will take $n$ clock cycles to finish. For write operations, a positive voltage pulse writes memristor cells to "1", and a negative voltage pulse writes memristor cells to "0". The buffered data can be used as the $P_{\text{sel}}$ signal to control the pulse generator.

In order to secure the memristor matrix content in case of test key leakage, we only allows the tester to receive multiplied scan data after the scan chain captures the circuit response. In other words, if the scan chain remains at the Shifting Mode, the Scan Output will not go through the memristor array. This prevents the attacker from learning the memristor content by controlling the matrix inputs when the test key is leaked. The logic diagram of this function is shown in Fig. 3.7. An Output Select ($O_{\text{sel}}$) signal is generated to determine whether the multiplication should be
Figure 3.7: Control logic bypasses matrix multiplication when circuit response is not captured.

bypassed. The control logic monitors the mode of the testing, which is indicated by the Test Mode Select (TMS) signal. When TMS = 1, the scan chain is in the *Shifting Mode*, and when TMS = 0, the scan chain is in the *Function Mode*. Only after the scan chain captures the circuit response in the *Function Mode*, the shifted scan response will be obfuscated by the memristor array; otherwise, the matrix multiplication will be bypassed. A counter (Cnt) counts the number of the shifted data. Once a full scan is completed, the control resets to the *Bypass* mode until the next full scan starts.

### 3.7 Simulation Results

In this section, the performance of the proposed technique will be evaluated. Note that different memristor array sizes will affect the security enhancement. We choose $n = 32$ as an example to illustrate the proposed design, and also provide the results for different memristor size ratio $r$. Employing the confidence intervals, we define a
Recovery Index (RI), which can accurately predict the degradation of scan data due to the retention loss. The refresh frequency can be safely chosen by placing an upper bound based on the RI.

![Figure 3.8: Exponential decay of scan data recovery during destructive reads.](image)

### 3.7.1 Performance analysis

As discussed before, the memristor array will be loaded with a $n \times k$ matrix $\Phi$ initially. This matrix can be randomly generated as long as it is full-rank. The uncontaminated matrix $G$ can be determined from (3.6). We also normalize the $\hat{G}$ matrix by mapping its values to the range of 0 to 1. Hence, the variance $\sigma^2$ in (3.10) describes the amount of possible conductance shifting resulted from each read operation.

The quality of scan data recovery can be evaluated by a reversed Hamming (RH) index $RH(S, S')$ defined as:

$$RH(S, S') = 1 - \frac{\sum_1^n \text{Hamming}(S, S')}{n},$$

(3.18)
which is calculated from the bit-wise Hamming distance between the original scan response \( S \) and the recovered \( S' \) using the proposed scheme. Thus, the value of the RH index ranges from 0.5 to 1. When \( RH(S, S') = 1 \), the recovery is accurate; while when \( RH(S, S') \) approaches 0.5, the recovered \( S' \) is vastly different from the original scan response \( S \). In Fig. 3.8, the change of \( RH(S, S') \) during the scan process without applying memristor refresh is simulated. The memristor array size ratio \( r \) is 1 and the variance \( \sigma^2 \) in (3.10) is set to \( 1 e^{-3} \), meaning the variance of conductance shifting caused by the flux mismatch is 0.1% during each read operation. As it can be seen, when memory refresh is off, \( RH(S, S') \) quickly drops and approaches 0.5 eventually. This trend fits the Kohlrausch model as described in (3.3), which is a stretched-exponential function. In addition, when the size ratio \( r = 1 \), the scan data recovered from the LMSE method become erroneous as soon as the scan starts. Hence, a larger size ratio is needed for a reliable recovery.

The change of \( RH(S, S') \) with the increased size ratios is shown in Fig. 3.9 as the dotted lines. Since statistically scan responses can be considered as random binary sequences, all \( S \) vectors are randomly generated in these simulations. We simulated four cases where \( r = 1.25, 1.5, 1.75 \) and 2, respectively. Each result shown here is the average of 1000 Monte Carlo simulations. As \( r \) increases, the time period during which the scan data can be reliably recovered increases as well. After that, the data become too erroneous to be correctly recovered by the LMSE method. Here we define this time duration as the maximum refresh time (MRT). The trend of the dotted lines in Fig. 3.9 confirms that the accuracy of the LMSE method improves with more memristor columns because more data are sampled. Therefore, the parameter \( r = \frac{k}{n} \) determines the memory refresh frequency. At the design stage, a loose bound of the MRT can be found by exploiting the confidence interval of the LMSE as expressed in
Figure 3.9: Reduction of reverse-Hamming index $RH(S, S')$ with respect to read cycles for different memristor array size ratio $r$. 
As discussed in Section V, to reliably recover the scan data, the confidence interval of the estimator should be small (e.g., < 0.5). Based on this fact, we assume if one estimator bit $\tilde{S}_i$ has a confidence interval larger than 0.5, the recovered bit is likely to be erroneous. Hence, a normalized Recovery Index (RI), which describes the ratio of the estimator bits having a confidence interval no larger than 0.5, is defined as below:

$$RI = \frac{\text{Number of } \{ \tilde{S}_i \mid if \; c \cdot \text{var}(\tilde{S}_i) \leq 0.5 \}}{n}.$$

(3.19)

Apparently, $RI = 1$ indicates all the estimated $S$ can be trusted, while $RI = 0$ means none of them can be trusted. The variance of the estimator bit $\tilde{S}_i$, $\text{var}(\tilde{S}_i)$, can be computed from (3.14).

The calculation of RI as a function of clock cycles for the cases where $r = 1.25, 1.5, 1.75, 2$ are presented in Fig. 3.9 as the solid lines. By comparing the $RH(S, S')$ and RI lines, we can see that RI can be used as an accurate model to predict the MRT for different $r$ configurations. The predicted MRT from a given RI should act as the loose bound for determining the memory refresh frequency. For example, in these cases, the memristor array can be refreshed every 1000, 2000, 3000 and 4000 cycles, respectively. The impact of different memristor array sizes is shown in Fig. 3.10. Under the same error rate, since the memristor array with a smaller size is read more frequently, the memory content degrades faster. Moreover, a larger $r$ ratio reduces the memory refresh need with longer MRT. However, it can be seen in Fig. 3.9 that a larger $r$ also increases the time to obfuscate the scan data. In addition, different $r$ ratios will affect the hardware overhead as well as the timing and power performance of the proposed design. This will be discussed in the following subsection.
Figure 3.10: Impact of different memristor array sizes and dimension ratios on MRT.

<table>
<thead>
<tr>
<th>$r=k/n$</th>
<th>Memristor ($\mu m^2$)</th>
<th>Read Control ($\mu m^2$)</th>
<th>Refresh Control ($\mu m^2$)</th>
<th>Total ($\mu m^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.25</td>
<td>87</td>
<td>628</td>
<td>441</td>
<td>1156</td>
</tr>
<tr>
<td>1.5</td>
<td>104</td>
<td>628</td>
<td>528</td>
<td>1260</td>
</tr>
<tr>
<td>1.75</td>
<td>122</td>
<td>628</td>
<td>614</td>
<td>1364</td>
</tr>
<tr>
<td>2</td>
<td>139</td>
<td>628</td>
<td>701</td>
<td>1468</td>
</tr>
</tbody>
</table>

Table 3.1: Circuit implementation area with a 32-row memristor array under different $\frac{k}{n}$ ratios

3.7.2 Overhead analysis

The proposed design for scan obfuscation is suitable for resource-constrained systems because memristor arrays have high density, low power consumption and fast switching speed. The size of the memristor array can vary resulting in different refresh frequencies $f_m$, which also have different implications on the security of the proposed design. A large array increases the overhead and slows down the memory degradation as it takes more scan cycles to read through the array. On the other hand, a small
array may not provide sufficient obfuscation. Here we discuss the implementation of a 32-row memristor array. Based on the discussion above, the scan response can be safely recovered with memory refreshes every 1000, 2000, 3000, 4000 clock cycles for $\frac{k}{n}$ of 1.25, 1.5, 1.75 and 2, respectively. We implemented the proposed scan wrapper and estimated the hardware cost for each part of the design as presented in Table I, where a larger $\frac{k}{n}$ ratio will increase the size of the memristor array and the memory refresh circuit. The compatibility of the ADC is also essential to this design due to the potential overhead. Fortunately, state-of-the-art ADCs allow high resolution conversion with a very compact circuit. In [80], a 10-bit SAR ADC is reported with only $36 \times 36$ $\mu m^2$ in 65nm technology. Alternatively, the column level ADCs [81] widely used in image sensor arrays are well-suited for the proposed design. ADCs are integrated into the $\sim 10\mu m$ pitch columns and share the common logic so that they can covert data in parallel at a high sampling rate and high resolution. In Table II, we present the normalized hardware overhead of several designs from the ISCAS’89 Benchmarks [82] and a pipelined AES-256 encryption core [83] for $\frac{k}{n} = 1.25$. The area of the two ADCs is estimated based on the column level ADC design in [81]. As shown, the hardware cost of the proposed design is very small, i.e., about 10%. For encryption designs such as AES-256, the overhead of the proposed scan wrapper is only 0.6%.

In Table II, the test lengths generated by an automatic test pattern generator are also shown for each benchmark design. To achieve the required fault coverage, the test length will grow dramatically as the design complexity increases. Considering the size of the test patterns for modern ICs, the time for inserting and checking the test key at the beginning of the test does not cost a lot of time. It only needs to be done once and after that tests can start normally. Based on the scan operation described
<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of Flip-Flops</th>
<th># of Gate Equivalents</th>
<th>Length of Test</th>
<th>Area ($um^2$)</th>
<th>%Overhead (with ADC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s9234</td>
<td>228</td>
<td>5597</td>
<td>108774</td>
<td>20683</td>
<td>11.63</td>
</tr>
<tr>
<td>s13207</td>
<td>669</td>
<td>7951</td>
<td>329639</td>
<td>37781</td>
<td>6.37</td>
</tr>
<tr>
<td>s35932</td>
<td>1728</td>
<td>16065</td>
<td>143506</td>
<td>85493</td>
<td>2.81</td>
</tr>
<tr>
<td>s15850</td>
<td>597</td>
<td>9772</td>
<td>309763</td>
<td>40953</td>
<td>5.87</td>
</tr>
<tr>
<td>s38417</td>
<td>1636</td>
<td>22179</td>
<td>1800699</td>
<td>99786</td>
<td>2.41</td>
</tr>
<tr>
<td>s38584</td>
<td>1452</td>
<td>19253</td>
<td>1110091</td>
<td>87396</td>
<td>2.75</td>
</tr>
<tr>
<td>AES-256</td>
<td>83</td>
<td>2700</td>
<td>125196</td>
<td>310952</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 3.2: Hardware overhead comparison with benchmark circuits

<table>
<thead>
<tr>
<th>Memristor ($uW$)</th>
<th>ADC ($uW$)</th>
<th>Shift register ($uW$)</th>
<th>Data buffer ($uW$)</th>
<th>Decoder ($uW$)</th>
<th>Pulse generator ($uW$)</th>
<th>Total ($uW$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.352</td>
<td>76.54</td>
<td>44.42</td>
<td>3.46</td>
<td>17.35</td>
<td>0.95</td>
<td>143.07</td>
</tr>
</tbody>
</table>

Table 3.3: Power consumption of the proposed scan wrapper based on a 32 × 48 memristor array.
### Table 3.4: Overhead comparison with related works (*' stands for data not provided in the reference)

<table>
<thead>
<tr>
<th>Area overhead(%)</th>
<th>Lock &amp; Key</th>
<th>Test wrapper</th>
<th>Scan encryption</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td># of sub-chain</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>s38417</td>
<td>2.9</td>
<td>5.7</td>
<td>13.5</td>
</tr>
<tr>
<td></td>
<td>s38584</td>
<td>3.8</td>
<td>7.5</td>
<td>26.6</td>
</tr>
<tr>
<td></td>
<td>AES core</td>
<td>*</td>
<td>5.2</td>
<td>1.39</td>
</tr>
<tr>
<td>Timing overhead(%)</td>
<td>*</td>
<td>*</td>
<td>0.8</td>
<td>2.4</td>
</tr>
</tbody>
</table>

In Section VI, the overall redundant test time is mainly caused by the memory refresh which is linearly dependent upon the refresh frequency \( f_m \) as well as the time cost for each refresh. Since in this design the memristor array is refreshed row by row, the time for each refresh takes \( n \) clock cycles. On the other hand, the multiplexed read operation in Fig. 3.6(a) minimizes the time redundancy during read operations. The additional time cost is \( n \) clock cycles for shifting the first \( n \) scan data. The total scan time of the proposed design can be expressed as:

\[
T = \frac{(l_t + n + q)}{f} + \frac{l_t}{f_m} \times n,
\]  

(3.20)

where \( l_t \) is the length of the test, \( q \) is the length of the test key, \( f \) is the test clock frequency, and \( f_m, k, n \) are the frequency of memristor refresh and implemented array sizes, respectively. Apparently, the impact of \( n \) and \( q \) becomes less significant when the test length \( l \) is long. In the case of the pipelined AES-256 core, if the memristor array is implemented with 32 rows and 48 columns, it is safe to refresh the memristor.
array at the frequency of $f_m = f \times 2000$. Thus, the time overhead of this test is only 2.4%.

To evaluate the power consumption of the proposed design, we simulated each circuit component using the SPICE tool. The estimated power consumption is presented in Table III. These results were obtained at the clock frequency of $f = 10\text{MHz}$ for a 130nm technology. The read voltage $V_{\text{read}}$ is set to 0.3V and the other operations are with $V_{\text{DD}} = 1.2V$. The largest power consumption is from the ADCs because high-speed ADCs are needed to meet the test requirement for high-speed ICs [84]. From these results, even at the high frequency, the overall power consumption is in the order of 100µW.

In comparison with the prior works, our design features significant reduction in area, power with less or comparable timing overhead. A detailed comparison with some existing works are given in Table 3.4. The scan output re-ordering method [59] introduces an area overhead ranging from 3.8% to 66.8% when dividing the scan chains into 4 to 12 sub-chains on the s38584 benchmark circuit. The test time also increases as more sub-chains are used for higher security levels. The scan test wrapper [60] built upon the IEEE 1500 standard reports 20% - 50% area overhead for the similar set of benchmarks. By implementing an additional light-weight block cipher algorithm to the scan chain, small area (1.39%) and timing (0.8%) overheads are achieved [63]. However, the testability problem of the additional cipher logic remains unsolved. It is worth mentioning that, while minimizing area and timing overhead is important for designing scan infrastructures, low power is also importance, as large power consumption can negatively affect circuit reliability, or cause difficulties in performance verification. Previous studies on secure scan chain designs often overlook this issue. Hence, we compare with the encryption circuit data path obfuscation technique [21],
which introduces $12.2 - 13.6mW$ power overhead when implemented on an AES chip.

### 3.7.3 Security analysis

The proposed secure scan wrapper provides a high level of security to the scan data. As discussed before, conventionally scan security is enhanced by employing pseudo-randomness generated by circuits such as LFSR, or randomly inserted a limited number of dummy elements. In contrast, the proposed memristor-based technique introduces true randomness as it relies on the intrinsic memristor state drafting process. In general, scan-based attacks utilize the internal round computation results obtained from the scan chain to retrieve the key information of DUT (e.g., cryptographic circuits). For example, the attack method on a DES cipher in [3] mainly relies on the pattern of the S-box where each S-box output corresponds to only 4 out of 64 addresses. Hence, by knowing three S-box outputs from the scan chain one can uniquely determine the round key. However, the obfuscated scan responses greatly increase the possible results of locating the S-box input from the output. For example, if the 64-bit S-box output (coming from 16 sub S-boxes) contains one bit error, for each plaintext a total of $\binom{64}{1}$ possibilities need to be considered. Moreover, three plaintexts are needed to uniquely find the round key, which makes the total complexity become $\binom{64}{3} = 262,144$, meaning that the same method will produce 262,144 possible round keys. When the error rate increases, the avalanche effect between multiple steps quickly increases the number of possible keys that the attacker needs to consider. The attack complexity can be calculated as:

$$ATCK_{COMP} = C_n^{H^{N_{step}}}.$$  \hspace{1cm} (3.21)
where $H$ is the Hamming distance between $S$ and $S'$, derived from $RH(S, S')$, and $N_{\text{step}}$ is the number of attack steps. In Fig. 3.11, we present the attack complexity based on the simulations with an array size $n = 32$ and $k = 40$. Hamming distance $H$ is rounded to its nearest integer. The attack step is chosen to be three for the purpose of illustration. As the testing proceeds, the increased error rate drastically increases the complexity of the attack. For example, at $RH(S, S') = 0.95$, the attack complexity is already in the order of $\sim 10^8 (\sim 2^{26})$. Moreover, the described attack model is optimistic. In reality, the matrix $\Phi$ is private. Also, some preliminary attack steps, such as identifying the scan structure, will introduce additional complexities as well.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.11.png}
\caption{The complexity of breaking the proposed technique as the error rate increases.}
\end{figure}
3.8 Conclusion

Modern IC systems suffer from severe security threats. The scan chain structure has been exploited as one the most capable hardware side channels. In this paper, a secure scan chain design is developed by exploiting the retention loss of memristor devices. It distinguishes the authorized and unauthorized users by checking the test key, and obfuscates scan responses for unauthorized users by introducing true randomness from decayed memory content. The propose technique can be adopted by conventional scan designs without modifying the scan chain structure. In addition, the analog-based memristor array is exploited for more compact design and to introduce more randomness. A multiplexed read process is utilized to minimize the hardware overhead. The results show that this technique only introduces minor hardware, timing and power overhead. Future work is directed to more in depth performance analysis incorporating fabricated memristor cells, thorough theoretical analysis on the optimization of security versus error rate, and exploiting the memristor based security enhancement scheme against other type of hardware attacks.
Chapter 4

Masked FPGA bitstream encryption via partial reconfiguration

4.1 Introduction

Along with the ever-increasing demand for system design flexibility, field programmable gate array (FPGA) devices have become popular in recent years with its grown capacity and complexity. Modern FPGA devices are usually equipped with millions of logic gates, various interfaces, megabytes of memory and even processor cores to support diverse applications. With the reconfiguration capability, they also exhibit promising potentials on resource constrained end device applications where flexibility is desired. As a result, protecting the data processed and the design implemented on FPGA devices has become a critical problem. The most common FPGA device, SRAM-programmed FPGA, is vulnerable to the so-called ”cloning” attack, where
malicious users can reverse engineer the entire design by acquiring the FPGA design bitstream \[85, 86\]. Therefore, bitstream protection is widely employed in modern FPGA vendors to support end-to-end confidentiality by utilizing symmetric cryptography schemes \[87\]. For example, Xilinx FPGA starting from the Virtex-II family to the recent 7 series use Triple-DES (data encryption standard) or AES (advanced encryption standard) in the Cipher Block Chain (CBC) mode \[88, 89\]. Encryption is completed by CAD tools (e.g. Vivado or ISE), and decryption is conducted by the FPGA on-chip decryptor. The cipher key is stored at the CAD software and on-board memory symmetrically \[87, 88\].

However, encryption-based protection can still be broken by side channel attacks (SCA), which extract the secret information by means of analyzing physical leakage from power, timing, electromagnetic and scan chains while the cryptographic circuit is running. Recent reports have shown the security of various encryption systems is vulnerable to side channel attacks \[89, 90\]. The underlying coherence between the side channel information and secret keys or design structures can be analyzed through thousands of attacking operations. The countermeasures against SCA usually rely on masking, which removes the dependency between side channel information and secret keys via randomizing the encryption computing process.

The major concern of applying the crypto masking approach on FPGA is large hardware overhead. A variety of efforts have been made to minimize the cost of masking by exploiting the Galios Field operation \[91\], cryptographic structure \[20\], memory space optimization and FPGA resource characteristics \[92\]. But the masked implementations are still burdened with large overhead, twice as large as the original on the most compact design \[93\] and even more on others \[94\], making it impractical for resource constrained applications. At the same time, the flexibility of turning
on/off encryption is also desired for the purpose of maximizing the performance. In this paper, we propose a light-weight and flexible masking solution for FPGA bitstream encryption/decryption by exploiting the dynamic partial reconfiguration property of modern FPGA devices. The proposed design will be mainly discussed on AES systems, yet the idea can be extended to other encryption systems such as DES with similar structures.

4.2 Preliminaries

The AES algorithm consists of four steps: SubBytes, ShiftRows, MixColumns, AddRoundKey, where SubBytes (all known as s-box) is non-linear procedure and the other ones are linear operations. These four steps iterate for multiple rounds (number of rounds depends on the key length) and encrypt the plaintext with round-key during each round of computation. Among these steps, the linear computations are confusion steps which is used to complicate the plaintext, while the non-linear step is the diffusion step where real data dissipation occurs. Adding random masking on linear steps is straightforward. For example, Boolean masking of a plaintext bit \( a \) is conducted by XORing with random bit \( X \) to obtain a masked bit \( a_m = a \oplus X \), and this can be reversed by a second XOR operation \( a = a_m \oplus X \). Additive and multiplicative masking are based on the similar mechanism. However, masking the non-linear computation is complicated. In AES algorithm, the s-box is applied on

\[ a \oplus X \quad \text{Masked inversion in } GF(2^8) \quad a^{-1} \oplus X \quad \text{affine transformation} \quad b \oplus X1 \]
each byte and the computation of which is composed by two steps: 1) consider the byte an element of Galois Field \( \text{GF}(2^8) \), and find its inversion; 2) multiply the resulting byte with a given matrix and then add a given constant vector (an affine transformation). The principle of masking the s-box can be depicted as in Fig. 4.1, where \( a \) is unmasked s-box input byte, \( X \) is the corresponding random mask byte and \( X_1 \) is the affine transformation of \( X \). Since the affine transformation is also linear, the key is to implement the masked version of inversion in \( \text{GF}(2^8) \), which involves complicate algorithm modification and is the main source of the implementation overhead as well as the performance loss.

Therefore, the implementation of s-box masking is critical for masking implementation of AES. In general, this can be realized on FPGA designs by two kinds of approaches: 1) pre-compute and store the masked s-boxes; or 2) do masking computation on the flight. The former approach takes advantage of the FPGA structure, where s-box (or masked s-box) can be easily implemented as look-up-table (LUT). However, this method requires designers to pre-store one additional s-box for each possible mask value, hence they either need to compromise the flexibility of updating the mask value, or bear with large resource utilization. On the other hand, the second kind of approach is subject to high redundancy and performance loss as well due to the required algorithm modifications. Many efforts have been made in terms of simplifying the implementation of GF inversion. In [20], the authors applied the “Tower Field” algorithm which represents the \( \text{GF}(2^8) \) inversion in sub-fields \( \text{GF}(2^4) \) and \( \text{GF}(2^2) \) to minimize the implementation complexity, but still results in \( \sim 3 \) time’s longer processing time, and \( \sim 3 \) time’s resources utilization compared to un-masked design. The simplified represented is also adapted for optimized LUT structure and implemented in [94] on FPGA devices. However, the implementation reports \( \sim 6 \)
times LUTs units and $\sim 7$ times slices units and extra registers compared to un-masked reference s-box design.

### 4.3 Partial reconfiguration based FPGA bitstream decryption

#### 4.3.1 Motivation

As it’s mentioned before, the FPGA bitstream protection usually rely on symmetric ciphers such as AES. The application bistream, which is encrypted in CAD software, will be decrypted by decryptor core during chip boot-up, and SCA attack can easily compromise the encryption key during the downloading process. Existing masking
designs suffer from high resource overhead which makes it impractical for resource constrained embedded system designs. However, the modern FPGA devices are often equipped with dynamic and partial reconfiguration (DPR) features [88], [95], which supports more flexibility on FPGA based systems. Through special internal ports (such as ICAP on Xilinx devices), one or more portions of the FPGA logic can be dynamically modified while the remaining portions are operating normally. In this work, we propose to utilize the DPR capability of FPGA to design a novel bitstream protection flow, where the side channel resilience can be achieved with efficient hardware utilization and low latency design.

The masked AES decryptor in the proposed design will be implemented on FPGA as well, instead of the commercial decryption ASIC solutions. When bitstream encryption is required, the user needs to download the decryptor before boot up the system. The AES implementation on FPGA is highly modularized, where the s-box can be implemented as LUTs for resource utilization efficiency. By exploiting the DPR feature, the masking of s-box can be realized by dynamically updating the s-box module. In this way, it neither requires complicated algorithm modification nor large storage space. Besides, changing masking value can be conducted by acquiring new masked s-box from software. Hence, high flexibility and security can be achieved by proposed solution. Furthermore, in resource constrained scenarios, the decryption logic in the proposed design can be erased after the configuration of the main application bitstream, so that extra resources are released for non-secure modules.
4.3.2 The Proposed Design

The proposed design of DPR-based FPGA bitstream configuration process is depicted in Fig. 4.2 and the original non-masking design is also illustrated. In Fig. 4.2a, an application bitstream is generated and encrypted by CAD software, and then decrypted by an on-chip FPGA decryptor for the purpose of bitstream configuration. The encryption/decryption process relies on a symmetric key, which is stored in the on-chip FPGA memory powered by an external battery [96]. As discussed in the previous sections, side channel analysis is able to retrieve this secret key and therefore acquire the entire application design.

To efficiently mask the downloading process and protect the bitstream from side channel attacks, the proposed design divides the bitstream configuration process into four steps, as shown in Fig. 4.2b. At step 1, the CAD tool generates the AES bitstream and implements on the FPGA for the decryptor (denoted as DECR’) without encryption/decryption. The implemented DECR’ has two sets of s-box LUTs so that it can run at both non-masked and masked modes. The s-box LUT located at the masked data path is denoted as S’. At step 2, the CAD tool generates the partial bitstream including the pre-computed s’-box, random mask X as well as its modified values corresponding to the different AES steps. This partial bitstream will update DECR’ especially the S’ part while DECR’ is running at the non-masked mode. Step 3 is the application bitstream downloading process. Similar to the original design, the main application that contains secret designs will go through the encryption/decryption process with DECR’ running at the masked mode to prevent side channel attacks. Step 4 is more flexible in that DECR’ can be totally erased or S’ can keep on updating based on different applications. For example, in IoT applications with limited
resources, it is a good choice to erase the decryptor and allocate other normal designs on the FPGA. On the other hand, if an application will continue to configure secure designs or serve in data communications, DECR’ can be reused and S’ will be updated through the partial bitstreams.

The decryption flows of both non-masked and masked modes are illustrated in Fig. 4.2c. In the non-masked design, the data flow is from the encrypted input \( A \) to the decrypted output \( E \), and \( K_i \) represents the secret key at the \( i^{th} \) round. The data flow of the masked design is similar to the non-masked one, with each intermediate result masked with a random number, i.e., \( X \) is the random mask and \( X_1 \) to \( X_3 \) represent the modified values after each AES steps. Modified InByteSub is the inverse s-box process of decryption, which is labeled as S’ in Fig. 4.2b.

The proposed design is realized through preparing bitstreams of different parts and pre-storing them in the boot memory, as shown in Fig. 4.3. Masking based FPGA bitstream configuration is achieved through the FPGA “Boot from Memory” option, following the above four steps. In Fig. 4.3 SRAM-based FPGA is divided into two memory spaces, which are “Configuration Memory I” reserved for the main application with secret designs, and “Memory II” for decryptor allocation with masked AES protection. Additionally, the decryption logic is also separated to the static and dynamic parts, to determine which modules will be included to the partial bitstream for decryptor update. There are two sets of inverse s-box, one for the static part for non-masking use and the other served as dynamic logic. It is crucial to protect the random mask value from the attackers, otherwise the masking will be invalid. Thus, the dynamic part also needs encryption. In general, there are four blocks prepared and stored in the boot memory. They are the main application bitstream with encryption, the static decryptor parts without encryption, the encrypted dynamic
decryptor parts, and other non-secure application bitstreams that can be loaded to the FPGA without encryption for better performance.

Since partial bitstreams only have the memory mapping of the target logics, their file size is quite small compared to either main application bitstreams or decryptor static parts, resulting in negligible boot memory overhead. Compared to the non-masked designs, the resource overhead only comes from an extra set of s-box and the DPR control logic. Through exploiting the dynamic and partial reconfiguration features of modern FPGA, the proposed design only has minor overhead compared to the traditional non-masked approach. This will be verified by the implementation results in the next section.
4.4 Implementation and Results

To evaluate the proposed approach, we implemented an AES decryption circuit using the partially reprogrammable feature on Xilinx Zedboard, which is based on Xilinx Zynq-7000 all-programmable SOC and combined with a Cortex-A9 processor. The FPGA chip has 85,000 programmable logic (PL) cells, 53,200 LUTs, 220 DSP slices, up to 104MHz clock and 100MHz configuration speed. The tcl-based script is used for building the design, generating the bitstreams, and interacting with the board. The masked inverse-sbox is computed in software separately.

4.4.1 Circuit Implementation

The AES circuit is configured with both a static region and a dynamic region. The static logic consists of a complete decryption core with modifications to support the masking mode and PR control logic. The dynamic logic mainly contains the masked inverse-sbox and the registers to hold random mask values. We use ICAP ports to access the internal configuration memory through the user logic. Since our system does not use the embedded processor to control the PR process like other dynamic PR systems, the PR control is light-weight and free from process bus delays. The system implementation is shown in Fig. 4.4. In this paper, we implemented a compact AES decryptor, which supports 128bit and 256bit key lengths. The implementation is iterative with four inverse s-boxes and processes 128bit words. The maximum throughput of the decryptor is 278Mbps on Xilinx 7 series FGPAs.

As discussed before, the full bitstream is first downloaded to the FPGA and then the decryption core starts to operate in the non-masked mode. Afterwards, the PR controller sends the partial bitstream to the decryptor for decryption, and
Figure 4.4: Circuit implementation of the bitstream decryption module.
the decrypted partial bitstream is used to reprogram the dynamic module. This is required as the mask value needs to be protected. Note that if the inverse s-box is configured to be a dynamic block, the entire partial bitstream needs to be stored until it is fully decrypted. In order to avoid using extra memory and control logic, in this work we implemented a dual subByte (s-box) structure, where the unmasked subByte module is implemented as the static logic, and a separate subByte module is also implemented in the dynamic logic that contains the masked s-boxes. In this way, the masked s-box can be reprogrammed in parallel with the partial bitstream decryption. The generation of the partial bitstream is based on the masked inverse-sbox computed in software. We set the ICAP to 32 bits, hence only a 128bit buffer is needed to buffer the decrypted bitstream before sending it to the configuration memory.
4.4.2 Resource Utilization

The FPGA resource utilization of the DPR masked AES decryptor is reported and compared with the original unmasked design in Table I. We implemented a compact AES decryptor because the bitstream decryption does not require a high throughput, as the application downloading speed is also constrained by the memory configuration speed. Both the original and the masked designs implemented four inverse s-boxes to process 32bit data at once, which is also the standard data width of FPGA interface. Naturally, the same masking scheme can be extended to implementations with fewer inverse s-boxes to reduce resource utilization or more inverse s-boxes LUTs to improve throughput.

As shown in the Table, the circuit overhead mainly comes from the additional dynamic module and the DPR controller. Due to the partial reconfiguration feature and dual subByte structure, the resource utilization of the masked AES decryptor core is comparable with the original design, with minor overhead from the masking mode support circuit. Four inverse s-boxes are implemented by 590 6-input LUTs. Note that an 8-in-1 LUT can be implemented with 5 6-input LUTs, which means one s-box or inverse s-box can be implemented with 80 LUTs under circuit optimization. In addition, on Xilinx 7 series FPGAs, one slice consists of four LUTs. Thus the overhead from the dynamic masked subByte module (with four inverse s-boxes) can be contained to 80 slices. As discussed before, the compact DPR controller is possible as the DPR flow does not require processor interface and internal memory control.

To illustrate the resource improvement of the proposed design, we also compared this work with several existing designs. Since different designs are based on different AES baselines for various throughput requirements, the overheads are normalized by
their own AES designs without masking and summarized Table II. Note that on-slice resources such as LUTs and registers are merged together to calculate the overhead so that the uniform standard can be applied. Based on this comparison, the proposed design achieves the least resources overhead.

4.4.3 Performance Analysis

The portable ICAP module is used in this design for the DPR feature, which supports up to 100MHz reconfiguration speed, the fastest among all supported ports. Given the ICAP bandwidth $BW$, the reconfiguration throughput is

$$P_{icap} = BW \cdot f$$

(4.1)

In this system, the data width of the ICAP is set to be 32 bits. Hence, the maximum reconfiguration throughput is $400Mbps$. As mentioned before, the maximum throughput of AES decryptor core with four inverse s-boxes is $278Mbps$, which determines the time requirement for DPR. The partial bitstream generated in this design is $206kB$, which takes about $0.7ms$ to download. The FPGA bitstream size is proportional to the amount of the memory to be configured. Therefore, the initial setup time can also be shortened if a small configuration memory is designated for the decryption protocol. On the other hand, the resource utilization is more critical in this application since bitstream downloading only needs to be done once at power-up.
4.5 Conclusion

As the modern block cipher based FPGA bitstream protection schemes are threatened by SCA attacks, in this paper, we proposed a side-channel masked FPGA bitstream security system which is suitable for flexible security protocols and hardware resource limited implementations. The dynamic partial reconfiguration feature of the FPGA devices is exploited to overcome the high circuit redundancy caused by making the non-linear components in traditional masking designs. The designated self-reconfiguration flow requires minimal resources for control logic and internal memory storage. The prototyped system has shown that the PR based masking scheme achieves a lot of overhead decreasing compared to conventional designs, which is desired in resource constrained systems. The future work will be dedicated in developing a software-aided bitstream protection system where more flexibility in terms of choice of protocols, cipher configuration, and mask values can be supported.
Chapter 5

Conclusion

In this dissertation, the challenges in designing secure embedded hardware which suit for the increasing demand for cost, efficiency and flexibility are discussed. Comprehensive consideration is required during each step of IC production flow and various design aspects, so as to establish security countermeasures from early design stage understanding and hence with minimized redundancy and performance penalty. Based on this idea, several works are presented. First, the FSM based design vulnerability analysis framework is presented which the new probabilistic evaluation metric indicating the circuit design vulnerability against fault injection. Second, a novel compact and high security scan chain design architecture is proposed exploiting the random forgetting effect of memristor devices. Last, we discussed the feasibility and efficiency of using dynamic partial reconfiguration feature of FPGA device for implementing flexible masking countermeasure against side channel analysis on bistream encryption and decryption.
Bibliography


[32] Chen Xu, Hamed Vavadi, Jigi Chen, Mohsen Erfanzadeh, Quangqian Yuan, Yanping Gong, Hassan Salehi, Hai Li, and Quing Zhu. Toward miniature diffuse op-


117


