A SECURE SOC PLATFORM FOR SECURITY ASSESSMENTS IN FPGAS

by

Geraldine Shirley Nicholas

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2023

ABSTRACT

GERALDINE SHIRLEY NICHOLAS. A SECURE SOC PLATFORM FOR SECURITY ASSESSMENTS IN FPGAS. (Under the direction of DR. FAREENA SAQIB)

With the rapid increase in connected devices and SoC design architecture being used in diverse platforms, they become potential targets to gain unauthorized access for data and privacy invasion. Therefore, heterogeneous SoC architecture raises security concerns in addition to the benefits they offer with improved throughput. They are susceptible to side-channel attacks where secure information is extracted through communication channels. Crypto algorithms implemented for secure authentication tend to leak sensitive information jeopardizing system security. Memory corruption vulnerabilities, code injection, buffer overflow attacks and other software-based attacks through untrusted channels tend to control the flow of the application with malicious data. Though traditional defense mechanisms have been implemented they are all still vulnerable to side-channel attacks.

Secure measures to protect the interfaces and data propagation through different channels are critical and building a resilient model consists of the on-chip security factors. In this work, a platform based SoC model is implemented to meet the security objectives using the RISC-V architecture in which an information flow tracking module tracks the flow of data for the system's integrity along with crypto engines and a secure boot mechanism for secure device authentication providing encrypted data transfers. For bitstream resilient SoC models the work extends a logic obfuscation module with runtime security leading to a secure assessment framework. This work explores the microarchitectural vulnerabilities with Machine Learning models and proposes a Transfer Learning technique based counterfeit detection scheme for supply chain vulnerability.

DEDICATION

This dissertation is dedicated to God and my family, for without their love and support, none of this would be made possible. To my father who has always believed in me and has encouraged me to further myself. To my mother, for the love and care to keep me going. To my brother, who has always taught me to work hard for the things I aspire to achieve. This work is also dedicated to my husband Binesh Kumar, who has been a constant source of support and encouragement during the challenges of graduate school and life. He has also been an inspiration throughout this journey of mine and therefore plays a crucial part in my success.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Fareena Saqib, for the continuous support and encouragement throughout my research. For her patience, motivation, valuable comments, suggestions, and immense knowledge to make the research meaningful.

Also, I would like to thank and acknowledge my committee members: Dr. Arindam Mukherjee, Dr. Ahmed Arafa and Dr. Gabor Hetyei for providing encouraging and constructive feedback in developing my research.

Finally, I would like to thank all my friends who contributed to the research: Chaitanya Mukund Bhure, Dhruvakumar Aklekar and Vineet Chadalavada.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
AMD	Advanced Micro Devices
API	Application Program Interface
ARM	Advanced RISC Machine
ASIC	Application-Specific Integrated Circuits
AXI	Application Extensible Interface
BBRAM	Battery Backup Random Access Memory
CEP	Common Evaluation Platform
CHISEL	Constructing Hardware in a Scala Embedded Language
CNN	Convolutional Neural Network
CRAS	Cryptographic Return Address Stack
CSR	Control and Status Register
DIFT	Dynamic Information Flow Tracking
DSLR	Digital Single-Lens Reflex
EDA	Electronic Design Automation
EMF	Electro Magnetic Field
FIFO	First In First Out
FIRRTL	Flexible Intermediate Representation for RTL
FPGA	Field Programmable Gate Array

FSBL	First Stage Boot Loader
GLIFT	Gate Level Information Flow Tracking
GNU	GNU Not Unix
GPIO	General Purpose Input Output
HDFI	Hardware Assisted Data-Flow Isolation
HDL	Hardware Description Language
HELPPUF	Hardware Embedded Delay PUF
IC	Integrated Chip
ICAP	Internal Configuration Access Port
IFT	Information Flow Tracking
IoT	Internet of Things
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
LRN	Local Response Normalization
LUT	LookUp Table
MIT	Mixed-mode Information Flow Tracking
ML	Machine Learning
OS	Operating System
PCAP	Processor Configuration Access Port
PL	Programmable Logic

PRM	Processor Reserved Memory
PS	Processing System
PSA	Platform Security Architectures
PUF	Physical Unclonable Function
RISC-V	Reduced Instruction Set Architecture
RSA	Rivest Shamir Adleman
RTL	Register Transfer Level
SAT	Boolean Satisfiability
SCA	Side Channel Analysis
SFLL	Secure Function Logic Locking
SGX	Software Guard Extensions
SoC	System-on-a-Chip
SVM	Support Vector Machine
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
TZASC	TrustZone Address Space Controller
TZMA	TrustZone Memory Adapter
VGG	Visual Geometry Group

CHAPTER 1: INTRODUCTION

1.1 Motivation

Modern heterogeneous computing systems, which incorporate IoT devices and networks, have the potential to deliver high-speed, optimized performance with enhanced capabilities. However, due to the increased number of components required to achieve such results, these systems are susceptible to various security threats. When an active device is connected to a network, it becomes vulnerable to firmware and hardware attacks, where the former involves exploiting vulnerabilities in different system components to gain control over the entire system, and the latter requires safeguarding the System-on-Chip (SoC) design against unauthorized access and data leakage [1]. The development of frameworks for protecting hardware security attacks is primarily dependent on the Intellectual Property (IP) and system architecture.

To develop a secure SoC platform, it is crucial to identify the security vulnerabilities of the system, especially in platforms with different levels of abstraction. The key factors for ensuring the security of an SoC platform are the Root of Trust [2], Secure Boot, and Execution Level Security. The Root of Trust employs a set of modules with diverse security features to monitor system functionality and provide secure authentication for each component in the system. Secure Boot establishes the integrity of the firmware and builds trust between the system and firmware before the execution of an application through authentication and validation. Execution Level Security provides access control and information flow tracking from various channels, along with a Trusted Execution Environment (TEE) to isolate security-critical components in the system.

Protecting interfaces and data propagation across various channels is critical, and building a resilient model requires on-chip security measures. Although traditional architectures provide a secure environment to some extent, they fail to ensure different levels of security within a system. Thus, a platform based SoC model can serve as a foundation for evaluation tools and techniques to achieve security objectives. To build a robust side-channel analysis framework, RISC-V architecture is utilized due to its ability to provide a platform for the custom implementation of security extensions compared to other traditional architectures. Additionally, RISC-V is an open-source Instruction Set Architecture (ISA) which further adds to its benefits. This research focuses on addressing the significant threats in an SoC design architectural platform and presenting novel security enhancements using RISC-V architecture.

1.2 Contributions

This research has the following contributions towards major challenges in an SoC Design and Architectural Platform.

- Identifies security threats and presents threat models that affect an SoC Design and Architectural Platform. Existing work and their shortcoming are also discussed.

- Proposes a novel device authentication scheme for runtime bitstream security. This framework employs logic-locking mechanism and extents its applications of secure boot process for FPGAs along with ARM TrustZone for secure configuration.

- Design of an extensible hardware cryptographic accelerator compatible with the RISC-V model in which an AES cryptographic engine with encryption function is used during critical data transfers.

- Proposes a RISC-V design of a coarse-grained hardware-based Information Flow Tracking framework with a tagged mechanism at the architectural level which detects security violations at runtime. This model uses a one-bit tag to track the data flow from untrusted channels with a separate tag cache module for minimal overhead.

- Proposes a RISC-V design of a fine-grained Gate level-based Information Flow Tracking framework for security critical datapath using optimized shadow logic for leakage model. This scheme focuses on a specific security critical module and the datapath of the module to be executed to overcome the performance and area overhead from existing schemes. Translation from the instruction level to the data level is based on the module instantiation with security-critical data.

- Presents microarchitectural vulnerabilities in the SoC design along with different machine learning models for SoC security validation and verification. This work proposes a transfer learning based counterfeit detection scheme for supply chain vulnerability.

1.3 Organization

This document is organized as follows:

Chapter 2 describes the background information on the topics involved in this research along with the overview of the existing works related to SoC Design Security and Architectural Platforms.

Chapter 3 presents a security-aware design flow scheme with secure boot features and logic locking applications for bitstream security with ARM TrustZone enabled isolation.

Chapter 4 details the contribution of the proposed extensible hardware cryptographic accelerator for the RISC-V model.

Chapter 5 proposes the coarse-grained hardware-based Information Flow Tracking framework with the performance analysis.

Chapter 6 proposes the fine-grained gate-level-based Information Flow Tracking framework with optimized shadow logic for security critical data modules.

Chapter 7 extends the research to microarchitectural vulnerabilities with Machine Learning Models and Transfer Learning Technique based counterfeit detection model. Lastly, the conclusions of the research are presented in Chapter 8.

CHAPTER 2: BACKGROUND STUDY

2.1 System-on-Chip Security

Due to its utilization in cyber-physical systems, embedded computing systems, and the Internet of Things (IoT), security has become a critical component of SoC design. With the SoC market projected to exceed 207 billion dollars by the end of 2023, the model faces new vulnerabilities and increased cyberattacks, posing reliability issues [3]. As computing devices are employed in various applications, they are susceptible to a vast amount of sensitive data and critical information that must be safeguarded against malicious access. The analysis and evaluation of resilient mechanisms against different attack models are significant aspects of SoC design. Four major types of attack scenarios must be considered when examining reference models for security and challenges in the SoC Design and Architectural Platform:

- Insertion of Malware/Unwanted Application gaining access: By using system software privilege levels, an adversary can insert a hidden functionality that tracks critical data and information or triggers disruptive outcomes in a connected network. In addition, malicious modification to the circuit can be done by bypassing the security fence of the system.

- Side-channel attacks: One of the most common methods of security exploitation involves obtaining information on crypto engines through communication channels within a system. This method enables an adversary to reverse engineer system functions by monitoring power consumption or electromagnetic fields associated with the hardware to gain access to the network. Additionally, memory access patterns and timing information at run time are targeted to exploit vulnerabilities in the system.

- Supply Chain Attacks: The globalized Integrated Chip (IC) supply chain has the potential to be vulnerable to malicious design modifications or IP theft through reverse engineering [6]. On the other hand, in software supply chain attacks, the focus is on an unsecured network or infrastructure, which can be targeted by malicious code to compromise build tools.

- Network Attacks: Denial of Service in a distributed network results in bridging through the system and gaining access to it. This type of attack can cause significant disruption to the availability of the system and the services it provides. Figure 2.1 illustrates the different security challenges in a connected network.

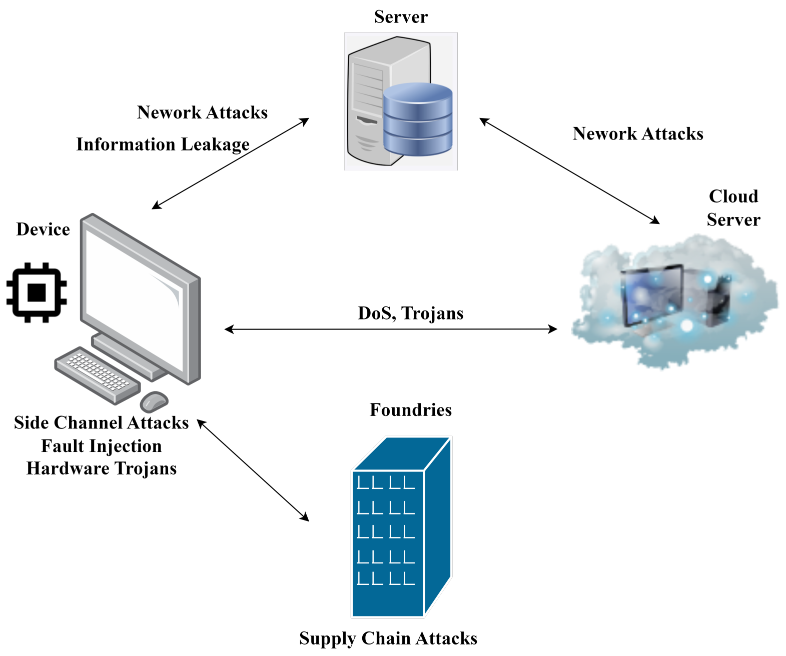


Figure 2.1: Security Challenges in a Connected Network

To ensure a secure architecture, it's essential to protect the application from untrusted sources and various vulnerabilities that may compromise the system's security. These threats include malicious IPs in hardware, vulnerable firmware, and side-channel vulnerabilities through different communication channels. Secured SoC models must implement robust security policies to prevent unauthorized access to the devices, including authentication, confidentiality, integrity, and access control to the system. There are three different groups in which security assurance can be classified, owing to the complex roots of security assurance.

2.1.1 Hardware security

Hardware platforms and ICs utilized in diverse applications are susceptible to threats that may arise throughout different phases of the component's lifecycle. Hardware security pertains to the security concerns resulting from underlying hardware and architectural vulnerabilities that affect the design, implementation, and validation of security operations in such models. Malicious alterations of ICs, IP piracy, reverse engineering, and unauthorized access to privileged resources via debugging channels are some common hardware attacks. Current approaches predominantly prioritize safeguarding against hardware supply chain attacks, which include Hardware Trojan attacks, counterfeit IPs, and implementation-dependent vulnerabilities in cryptographic modules leading to the exposure of sensitive information [5].

2.1.2 System or platform security

The platform security architecture refers to vulnerabilities that may arise from functional and performance aspects of the system, which malicious third parties may exploit during runtime. These include the leakage of security-critical data and side-channel attacks that can alter the system's behavior. Protecting sensitive information stored in hardware from untrusted software and networks is critical, and this is achieved through various levels of isolation. Access control and information flow policies with validation are implemented to ensure a robust and secure system [6]. Authentication and security controls serve as methods of protecting information and mitigating unauthorized access.

2.1.3 Cloud security

Providing resource sharing and on-demand services, cloud computing offers a distributed work environment with extensive functionality and data storage. While advantageous over traditional systems, there are privacy and security concerns that allow adversaries to gain unauthorized control over stored data. Cloud security involves vulnerabilities and security issues that arise from communication channels in a client-server model, where critical data is transmitted through the network cloud [7]. Cloud-based attacks include data breaches, denial of service, and corruption of collected data integrity through backdoor channels leading to remote access. To develop a secure platform for an SoC with various abstraction levels, it is essential to identify the system's security vulnerabilities. The key factors of SoC Platform Security are as follows.

2.2 Key Factors of SoC Platform Security

2.2.1 Root of Trust

The Root of Trust is a crucial system component that ensures data integrity and verifies the functionality and design of the system. It can be either hardware or software-based, with hardware Root of Trust being the most secure option, as it establishes a high level of trust by verifying its own integrity. The most secure hardware Root of Trust is a stand-alone security module or one that is implemented within a processor or System on Chip, ensuring isolation of resources, keys, and security assets, along with a side channel resistant model and multiple layers of defense mechanisms [8]. The Trusted Platform Module (TPM) [9] is an example of a tamper-evident hardware module that provides Root of Trust measurement, remote attestation, and cryptographic functions for the protection of both keys and sensitive data. The Physical Unclonable Function (PUF) [10] based security is another hardware Root of Trust that enables unique IDs with true random number generation and secure key storage, with an anti-tampering design. To establish a trusted supply chain platform, methodological approaches such as PUF technology and obfuscation techniques for SoC design can be employed to support Root of Trust features [11].

2.2.2 Secure boot

The programmable logic in SoC FPGAs supports both hardwired microprocessors and soft-IP-based ones. These devices are programmed through bitstreams, and an attack on the bitstream can jeopardize the entire system operation while it is deployed in the field. FPGA bitstream reverse engineering is a major concern since reverse engineering and other fault injection-based attacks can manipulate the cryptographic components, compromising the system's confidentiality and data integrity. To address this problem, various solutions such as authentication of bitstreams and encryption models have been proposed [12]. For instance, Xilinx provides a secure boot mechanism with authenticated bitstreams to safeguard against such vulnerabilities [13]. Additionally, secure root of trust architecture with TPM drivers and over-the-air updates is implemented to detect malicious modifications in configuration files [14]. Multilayered secure boot is another approach that updates the LookUp Table (LUT) frames using remote attestation and PUF-based mutual authentication during runtime [15]. Self-authentication secure boot mechanism uses PUF-based authentication to protect the secure boot process, and any modification made to unencrypted bitstreams results in key regeneration failure of the PUF [16]. For securing open-source architectures, a lightweight RISC-V-based secure boot framework with PUF and different encryption standards with secure remote key attestation is implemented [17]. Figure 2.2 shows the key factors that impact SoC Platform Security.

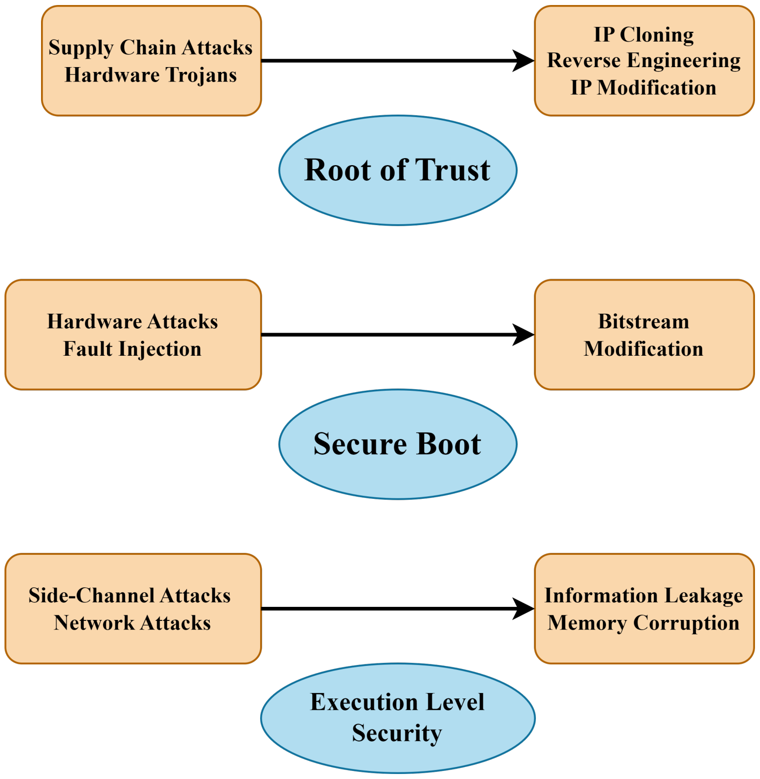


Figure 2.2: Key factors affecting the SoC Platform Security

2.2.3 Execution Level Security

To protect system assets from unauthorized access, SoC security is essential. The execution level security measures the access control mechanisms, information flow control, and isolation of the programs that are necessary to protect confidential information stored in the computing system. The access to the assets can be dependent on the state of execution, and hence, run time vulnerabilities need to be considered while implementing security measures. System-level policy classes for risk assessment are focused on execution level security. Information leakage models are secured by memory protection using programmable hardware monitors [18]. During the design phase, static techniques such as binary code checks [19] and verification tools [20] for program validation are implemented. Hardware-based techniques detect memory-based vulnerabilities like buffer overflow and format strings during runtime [21]. Hardware-based technologies have better processing speed and smaller resource overhead, making them more efficient than software-based methods [22].

2.3 Access Control Mechanism

Communication between devices connected to a network for data transfer can expose them to untrusted nodes that can exploit software vulnerabilities, making them susceptible to attacks at the application level as well as the firmware level. Therefore, it is necessary to provide attestation of the firmware to ensure that the device proves that it is attested with a trusted remote entity [23]. In addition, authentication is a process that verifies the identity of the user to access the system, while authorization is an access control mechanism that determines access rights based on the security operations performed on the system. To ensure authorized access and bitstream protection for reconfigurable devices, an authentication and encryption mechanism with access control functionality is required. Access control involves a set of class policies to access hardware and software components during execution [24]. Secured communication channels with multi-level security using PUFs for remote key updates provide access management and bitstream verification [25], thereby safeguarding the data and the system from malicious modifications.

2.4 Information Flow Tracking

Heterogeneous System on Chips and IoT devices connected to communication channels face the risk of information leakage and code injection due to their vulnerability to untrusted systems. Information Flow Tracking (IFT) technique is a promising analysis technique for security applications that enables the detection of information leakage and malicious data in real-time. Different IFT approaches, based on static verification during the design phase or dynamic checking at runtime, have been implemented. The precision of the IFT logic and the granularity of the building blocks determine the levels of abstraction. IFT implementations include both hardware and software-based approaches and depend on the explicit and implicit flow of data to design conditional behaviors for the model. In this section, existing IFT implementations based on precision granularity models will be discussed.

2.4.1 Coarse Grained IFT Models

At the architectural level, Information Flow Tracking implementations offer a security mechanism for metadata propagation and security policies by providing a coarse-grained precision logic with a dedicated tag mechanism. These implementations can either be hardware-based, depending on the architectural features and datapath of the ISA model, or coarse-grained models that track control-sensitive information along with program variables and independent labels. The access and information flow with tags can be achieved through a dedicated co-processor or a modified ISA with tag modules. Buffer overflow attacks and memory corruption can be prevented by identifying malicious data, and IFT models protect the system from these attacks [26][27].

Several IFT models have been developed to improve security. For instance, Flexi-Taint[28] is an IFT model that supports an accelerator with tainted security policies and extends the processor's datapath for tag propagation. Dynamic Information Flow Tracking (DIFT) [29] is a hardware-based approach that uses an ARM coprocessor to track and debug traces using static analysis. Hardware Assisted Data-Flow Isolation (HDFI) [30] and HyperFlow [31] are hardware-based RISC-V implementations that use a tagged mechanism and security policies for information flow control. Exploitable Buffer Overflow Detection by Information Flow Tracking (BOFT) [32] is an automated framework that integrates formal verification with IFT and leverages symbolic execution to provide explicit and implicit IFT with extensive instrumentation for taint propagation.

Mixed-mode Information Flow Tracking (MIT) [33] provides byte granularity by using taint semantics at compile time that are dependent on runtime logs, thus decoupling the tracking logic from program execution. FineDIFT [34] is a hardware-based mechanism that generates data flow graphs of a running process with a coprocessor to provide flexibility, but the metadata storage requirements result in several limitations of software instrumentation. Kejun Chen et all summarizes the current DIFT solutions referring to the over tainting problem leading to high false positive rates [35]. These issues can be addressed by adopting parallel tag propagation schemes and customizing security policies in IFT models.

Indirect Flow Propagation System (MITOS) [36] uses an analytical algorithm for indirect flow propagation with an arbitrary number of tag types for flexibility. However, a tag-based model results in architectural overhead as the memory is modified to incorporate tag bits, or if complex lattice structures are built to secure the data. An efficient model should limit hardware design overheads and provide IFT capabilities in hardware design. In such models, the tagging mechanism assigns a tag bit to untrusted data and tracks the propagation of the data using the tag bits. For example, incoming data from an untrusted source (input) is assigned a tag bit, and the IFT module tracks the data using the tag bits value to indicate if the data is spurious or safe. All incoming data is assigned a tag bit with the value 1 indicating that it is unsafe, and 0 for trusted data. If the data is copied to string1, it is identified as an untrusted source, and the IFT module tracks the data. Figure 2.3 shows an illustration of this process.

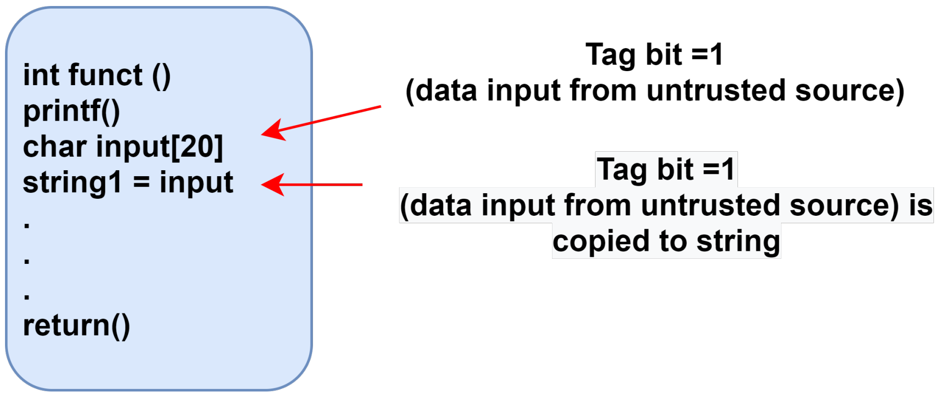

```
int funct ()
printf()
char input[20]
string1 = input
.
.
.
return()
```
Tag bit =1
(data input from untrusted source)
Tag bit =1
(data input from untrusted source) is copied to string

Figure 2.3: Tag bits assigned to untrusted sources by the IFT module

2.4.2 Fine Grained IFT Models

The implementation of IFT also includes fine-grained models with data labels associated only with data flow, instead of focusing on control flow transfers. A gate level IFT model is a fine-grained model associated with Boolean gates, where the information flow appears at the gate level netlist, providing better precision logic. GLIFT [37] is a shadow logic-based technique that adds logic to all the gates, resulting in design complexity and overhead. Several techniques have been implemented using data flow logic [38] as the backbone, including optimized labelling and enhanced encoding techniques [39] [40] [41], which help reduce complexity but affect the precision logic of the system. The asset based GLIFT [42] model provides structural checking with security properties.

Gate level-based leakage detection [43] detects the leaky path with parser and logic modules for formal verification but results in intense computation complexity for large designs. The multi-bit label tracking model [44] quantitatively detects the information leakage with area constraints, where the multi-bit labels are directly proportional to the number of gates in the circuit. A unified model for gate level propagation [45] generates synthesizable propagation logic to be used in EDA tools where the attribute labels at different levels of precision are addressed for faults with the flexibility to be used in different emulation platforms.

A gate level based IFT model consists of gates and shadow logic for all the gates, as shown in Figure 2.4 A separate shadow logic is implemented for each gate, (a and b are the inputs with an output o) where the shadow consists of the inputs along with untrusted inputs that affect the output. This scheme results in additional area overhead. Therefore, approaches that reduce the complexity of the gate level model and enable tracking of data critical modules, such as crypto engines and security accelerators, rather than tracking the whole module, enhance the overall system overhead.

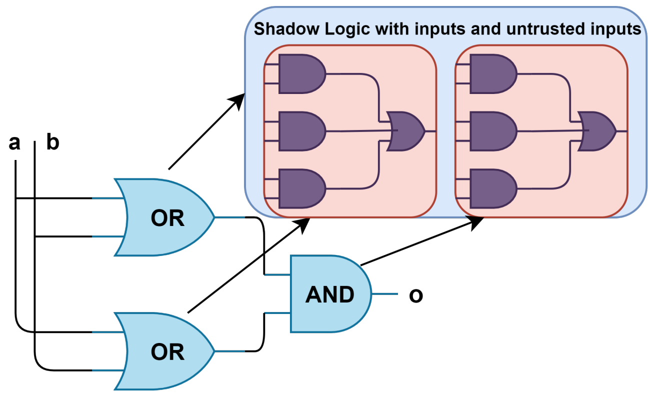


Figure 2.4: Gate level IFT with shadow logic

2.5 Trusted Execution Environment

The Trusted Execution Environment is an execution environment that provides security features through isolation of software and data [46]. TEE based on ARM TrustZone technology provides a mechanism to isolate security-critical components in a system [47], while Intel SGX enclave is used in modern processors to protect privilege levels by authorized functions [48]. Other architectures used for security-critical applications include AMD Platform Secure Processor, AMD Memory Encryption Technologies, Intel Management Engine (ME), Open Portable TEE, and various Platform Security Architectures (PSA).

In the ARM TrustZone technology, the software is partitioned into two worlds to prevent software attacks, where the secure world protects critical data and the non-secure world executes the normal operating system [49]. The secure monitor call acts as a bridge between the two worlds. Once the system boots, the processor enters the secure world of the TEE, and once all privileged operations are completed, it switches back to the normal world and yields control to the bootloader [50]. Data routed to a specific world is controlled by system operational modes and device configuration. TEE provides secure trusted services, such as authentication and remote attestation, to protect application integrity. TEE-enabled authentication from a remote device can mitigate phishing attacks [51]. TrustZone architecture uses identity authentication to provide run-time authentication and data protection mechanisms to verify private data and ensure data access security using Application Program Interface (API) calls [52]. The Programmable Logic (PL) master dynamically configures data using the Application Extensible Interface (AXI) interconnect signals, and a secure authentication scheme can be achieved using the secure slave key transaction with the master [53].

2.6 Logic Obfuscation

Logic obfuscation is a technique that enhances the security of a design by inserting key gates to the original design, thus locking the netlist. This makes the functionality of the design hidden and only allows the correct key combination to unlock the design functionality. By inserting key gates, the locked design produces corrupted outputs if an adversary tries to access or modify the design. Figure 2.5 illustrates a modified logic locked equivalent circuit for the c17 circuit, with key gate insertion. An input sequence of 10011 with the correct key combination of K0 and K1=11 produces the corresponding outputs X and Y as 01. However, if the key combination is wrong (e.g., 00), the output of the circuit is modified, and the design becomes erroneous. Nonetheless, attacks such as the Boolean satisfiability attack can eliminate the wrong key combinations using the distinguished inputs, effectively breaking the logic-locking techniques.

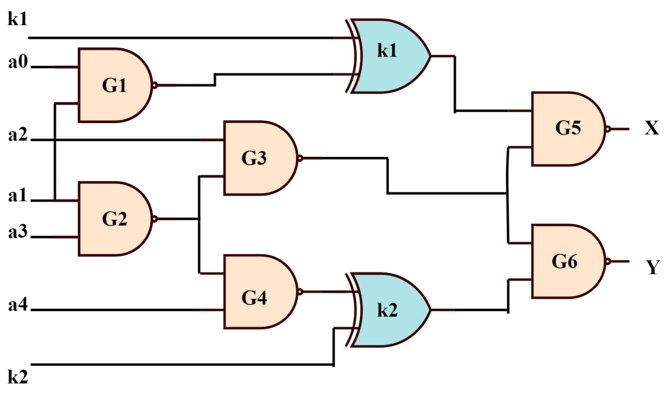


Figure 2.5: Logic-locked circuit with two new key gates added in C17 circuit

To prevent such attacks, SAT resilient techniques such as SARLock and TT-Lock make the attack iterations grow exponentially with increasing key size [54][55], whereas Anti-SAT provides tuning flexibility for the key gate configurations [56]. Furthermore, schemes such as SFLL remove a functional logic block and restore the original logic using Hamming distance [57], while fault-based logic encryption leverages EDA tools to insert key gates with fault impact metrics [58]. These techniques are useful for intellectual property protection and can also be extended to applications of secure boot.

2.7 Security Models

Most of the traditional architectures have limited capability to implement security features necessary to secure devices from different types of attacks. The main reason for this is the lack of flexibility as the developers of proprietary architectures do not offer security enhancements due to the associated performance tradeoffs. RISC-V architecture, on the other hand, provides this flexibility to customize the design of a system with reconfigurability and system security. The following section describes the security models, features and vulnerabilities in the existing traditional architectures and provides a RISC-V compatible countermeasures by taking into consideration the possible threats in the SoC design and Architecture Platform.

2.7.1 ARM Trustzone

The hardware-assisted TrustZone feature in the ARM-centric processing system provides a secure implementation, while the FPGA fabric holds the programmable logic and uses the AXI bus to communicate with the processing system. The intellectual property cores are partitioned into secure and non-secure worlds, providing isolation. The ARM TrustZone includes the TrustZone Memory Adapter (TZMA) and TrustZone Address Space Controller (TZASC), which establish partitions between memory and peripheral units for both worlds. However, modern designs with large stacks of libraries and optimized functions are vulnerable to cache-based side-channel attacks. Key extraction from secure crypto engines is possible by compromising the non-secure world Operating System (OS) or by tracking power or Electro Magnetic Field (EMF) signals during key exchanges [59]. The ARM TrustZone is susceptible to hardware attacks such as malicious modification of secure IP, denial of service, resource denial, and port attacks [60]. While configuring the NS bit along the AXI bus is simple and effective, managing the structure in a multi-core environment is challenging. Incorporating additional security enforcements for optional memory controllers outside the Cortex-M TrustZone Architecture is necessary. Figure 2.6 illustrates the ARM TrustZone implementation, which includes various units and cores.

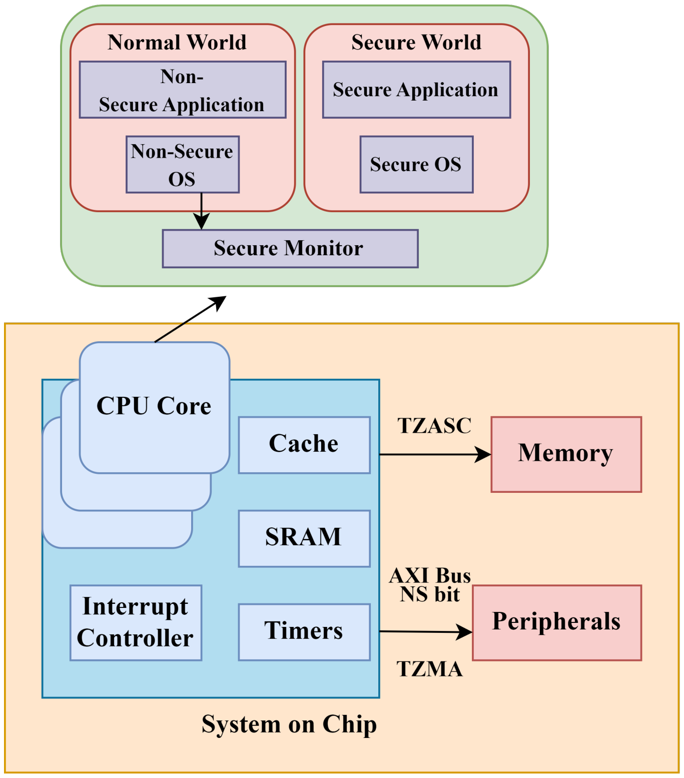


Figure 2.6: ARM TrustZone implementation

2.7.2 Intel SGX

Intel SGX allows for the protection of application address mappings and provides enclave memory access semantics [61]. The enclaves are secure regions of memory that are protected from any access or modification. These enclaves are encrypted and decrypted on the fly, providing hardware-isolated trusted environments. Figure 2.7 shows the basic operation performed in the SGX model. In this model, an untrusted application invokes a trusted function inside the enclave, which cannot be accessed by any other application. This model achieves confidentiality of the code with isolation and protects against integrity violations from software attacks. The Processor Reserved Memory (PRM) holds the enclave page cache and is protected from any non-enclave memory accesses. However, the downside of this model is that the enclave gains full access to the entire address space of the untrusted application, which makes it vulnerable to enclave malware.

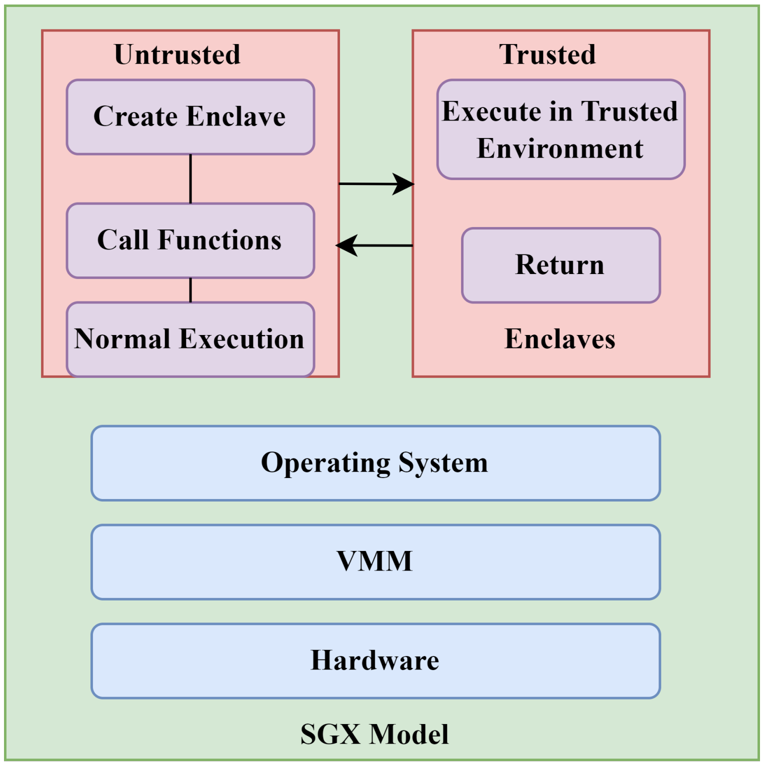


Figure 2.7: SGX Model

Furthermore, the enclave is vulnerable to cache based side-channel attacks, and software based side-channel attacks have been targeted on co-located SGX enclaves to extract Rivest Shamir Adleman (RSA) private keys [62]. Although most modern

Intel processors feature Hyperthreading, SGX does not prevent it, allowing malicious software to execute system threads. Additionally, the Intel-specific architectural and microarchitectural model with SGX's security features is not publicly available, which limits the ability to develop a trust model by customizing TEEs without proprietary rights. Bridging the support for more than one hardware-enforced isolated domain is not possible with the traditional architectures.

2.7.3 RISC-V

RISC-V is an open-source architecture that provides a highly flexible and customizable platform for implementing different levels of security in a system. Unlike SoC with 3PIP proprietary architecture, RISC-V's extensible ISA with minimal instruction set allows for the implementation of a customized processor that can support several security applications. For instance, the Common Evaluation Platform (CEF) can be used to identify the security properties of a RISC-V system [63]. The RISC-V security committee has proposed an abstraction-augmented aISA that extends a bridge between hardware and software beyond the traditional ISA for control [64]. RISC-V Multizone Security by Hex Five provides a hardware-enforced software defined separation with multiple TEEs [65]. With the benefits of RISC-V being open-source, especially in the security context, different modules can be implemented to secure the system from different kinds of attacks [66]. The physical memory protection provided by this architecture is used for authenticating the execution of trusted nodes. Integration of hardware cryptographic accelerators, key management, and security extensions are made simple using available open-source frameworks. Multi-threaded enclaves with memory-mapped resource protection are achieved by different RISC-V security implementations modules.

Additionally, RISC-V supports various extensions like Multiplication(M), Atomic (A), Single precision (F) and Double precision (D) floating point, which are collected into (G) extension that provides a general-purpose scalar instruction set. As RISC-V based systems continue to grow rapidly and new security vulnerabilities emerge, it is important to add some security features to RISC-V processors. Figure 2.8 illustrates the block diagram of Freedom E31 core with RISC-V ISA Specification, which supports 32, 64, and 128-bits instruction widths and four different instruction formats: R-type, I-type, S-type, and U-type. The processor is written in Chisel HDL (based on Scala language). The flexibility of the RISC-V architecture also supports RISC-V extensions to support the IFT model, which provides a strong timing sensitive security behavior that is otherwise not possible with traditional ISAs.

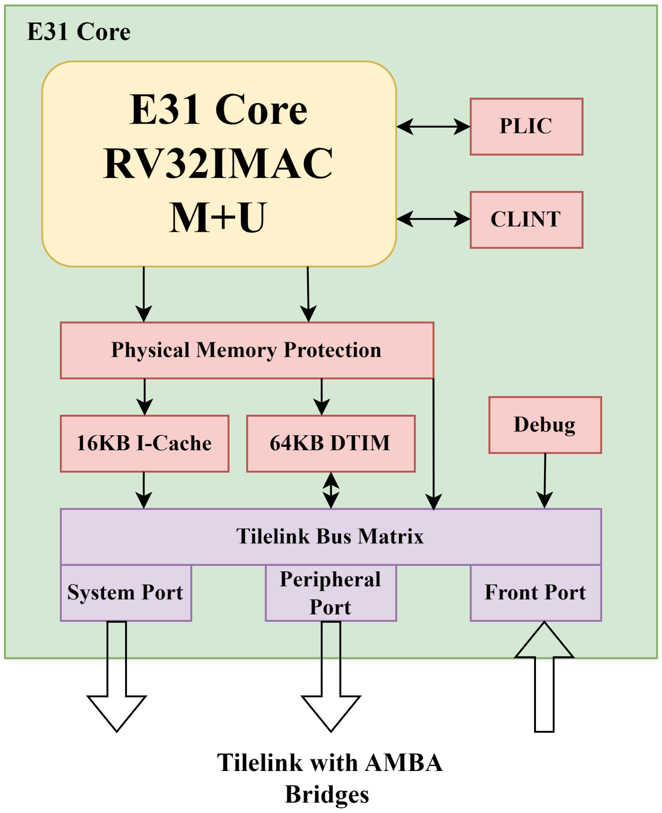


Figure 2.8: Freedom E31 core with RISC-V ISA Specification

Some of the existing information flow tracking models on RISC-V are tag based bare metal and PULPino based model to protect the system from memory attacks [67]. Liu et al [68] enforced a hardware extension of RISC-V SoC with static analysis to generate control flow graphs. Samuel et al [69] present an augmented tag isolation mechanism to improve the dynamic reuse of untrusted memory across security boundaries by sharing execution stacks with real-time constraints. Isadora [70] framework uses a minimal testbench to run an automated trace generator for information tracking limiting access to the full RISC-V toolchain. Table 2.7.3 provides RISC-V compatible countermeasures by taking into consideration the possible threats in the Trusted Execution Environment, and the SOC design and Architecture Platform.

Table 2.1: RISC-V compatible countermeasures

Threat Models	RISC-V compatible countermeasure models
Cache-Timing Attacks	Transparent Hardware-Protection Layers with memory access leakage protection [71][85]
Side-Channel Attacks	Core hardened resilient models with hardware Accelerators and virtual TEEs [71]
Denial of Service and Memory Attacks	Information flow tracking models tracking the flow of the data to protect memory corruption and mitigate DoS attacks by attestation models [86][87]
Malware Insertion	Secure boot for SoCs along with data tracking models [71]
Supply Chain Attacks	Logic Obfuscation with SAT attack resilient model for the SoC platform[71][102]

CHAPTER 3: SECURITY-AWARE DESIGN FLOW FOR BITSTREAM SECURITY

3.1 Introduction

Reconfigurable devices offer flexibility in dynamic reconfiguration features, but they are susceptible to bitstream modifications that can compromise the security of the device. An attacker can tamper with or insert hardware trojans into the bitstream during boot time or runtime if they have physical access to the configuration bitstream. To protect the bitstream during boot time, a secure boot with authentication can be implemented, which provides a root of trust. However, this can be bypassed by tampering with the device boot process using the Processor Configuration Access Port (PCAP) or Internal Configuration Access Port (ICAP) ports. Existing solutions like PUF modules [15] [16] [72], over-the-air updates, and TPM-based key management systems provide bitstream protection and authentication, but they may not be enough to prevent malicious modifications in configuration files [73][74].

Though these models provide bitstream security during boot time the PL can be replaced to perform different operations by replacing it with malicious bitstreams. To protect the application bitstream during runtime, various designs for trust techniques have been developed [75][76][77]. For instance, logic locking schemes are used to secure gate level IPs from bitstream reverse engineering, while ARM TrustZone provides isolation to secure critical applications by enforcing secure transactions during device authentication. This chapter presents the threat model for boot process and run time security and proposes a multi-layered security framework to mitigate malicious code modification during boot-up and runtime.

3.2 Threat model for Boot Process and Runtime Security of a Reconfigurable Device in the Field

Considering a reconfigurable device in the field in a client-server model that communicates with the server with critical data transactions. Authorized access control must be provided for secure communication between the client and the server module. Thus, the section represents the threat model during boot-up process and the runtime vulnerabilities.

3.2.1 Boot-Time bitstream modification

An adversary can redirect the normal execution flow of code to an unauthorized piece of code by hijacking the boot process. The Processing System (PS) controls the PL bitstream during boot up where modifications to the bitstream can be made via field configuration or other communication ports. Hardware trojans present in the system can be triggered to replace the PL logic to perform different tasks. In bitstream spoofing with relay and replay attacks, an adversary acts as an authenticated client to replace the original bitstream with a malicious one. Unauthorized memory access through hardware cores leaks critical data which can break the boot process on FPGA SoCs [78]. Authentication keys stored in the memory can be easily accessed through malicious hardware cores leading to system compromise by modifying the boot loader.

3.2.2 Runtime attacks

At runtime once the bitstream has been loaded an attacker may target dynamic reconfigurable partitions or may want to target certain portions of the configuration. Bitstreams must be encrypted to prevent IP theft and cloning and authenticated to eliminate tampering and trojan insertion. Extraction of the bitstream encryption keys leads to several runtime attacks such as:

- Side-channel bitstream key extraction: Extraction of the keys through different communication channels.

- Architectural model attacks: Hardware cores targeting the configuration files to bypass authentication.

- Bitstream Tampering: Manipulation of critical functions and targeting pr-configuration files to modify the bitstream.

3.3 Secure Design Flow

This work focuses on bitstream security to mitigate malicious code modification during the boot-up process and runtime. It provides a multi featured secure design where when the device powers on, a secure boot authentication is done using the PUF mechanism which generates a unique challenge-response pair in a client-server environment. Once the authentication is done a logic-locked mechanism based bitstream obfuscation is done to the application bitstream and programmed in the PL which provides secure IP isolation using the ARM TrustZone configurations. In addition, to eliminate tampering attacks, the key to unlock the logic-locked bitstream is securely stored in the TPM and through secure communication, it is used to unlock the bitstream.

3.3.1 Secure Boot Mechanism

During the boot process, the initial stage provides authentication for the bitstream with a unique key response generated by the PUF, followed by applying a logic-locking mechanism to the application logic. The PUF generates the per-device unique responses using input challenges, and during the enrollment phase, the verifier and the prover generate challenge-response pairs which are encrypted using AES core to prevent unauthorized access.

The client enrolls with the server in a trusted environment to receive the authentication bitstream, which is loaded during the boot process. The Hardware Embedded Delay PUF (HELPPUF) [79], which is based on path delay variation, generates the challenge-response pairs for authentication. The server sends the input challenges to the client, and the responses gathered in the server produce a unique key for authentication. The first stage bootloader loads the authentication bitstream on the PL fabric, and the PUF generates a response using the unique challenge and response pairs provided by the TPM.

In the reconstruction phase, authentication is done using the unique per-device key, and the PUF response is processed to generate a secret key for decrypting the encrypted application bitstream. The First Stage Boot Loader (FSBL) overwrites the authentication bitstream with the application bitstream, establishing secure communication between a client-server model. Custom device drivers are used to enable communication with TPM during the secure boot process to provide secure extensions for key storage [15], and TPM securely stores the keys generated during runtime to mitigate malicious intrusion. The overall key exchange mechanism for authentication is depicted in Figure 3.1.

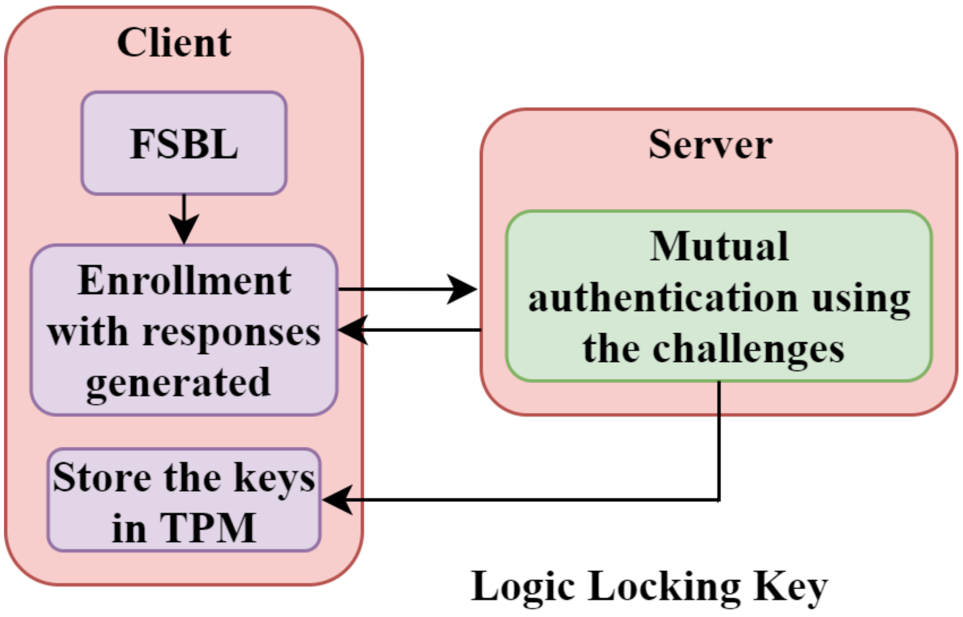

Client
FSBL
Enrollment with responses generated
Store the keys in TPM
Server
Mutual authentication using the challenges
Logic Locking Key

Figure 3.1: Secure boot authentication and key exchange process

3.3.2 Obfuscation Framework

The secure boot flow is integrated with a logic-locking automation framework, allowing key gates to be inserted at either the Register Transfer Level (RTL) or netlist level. To safeguard gate level IP, the key gates are inserted into the gate-level netlist using the insertion scheme illustrated in Figure 3.2. The flow is generated and implemented with Synopsys tools and Vivado via the automated framework, which ensures IP reuse security by integrating key gates into the design process. Utilizing solely the RTL code increases the vulnerability of the design to reverse engineering by potential adversaries. The framework is designed to switch the implementation method based on the input file type, with the ABC tool [80] being used to convert input files to Verilog. The generated test bench utilizes the input Hardware Description Language (HDL) to produce a set of vectors to verify the design.

During synthesis, the Synopsys tool transforms the RTL into a gate-level netlist. Two encryption-based logic locking schemes, SARLock and fault-based encryption, are included in the experiment to enhance resistance against SAT attacks. SARLock exponentially increases the key size to generate computational complexity, guaranteeing that a single key produces a fault for any input pattern. Fault-based encryption integrates a fault impact metric [81] to determine the highest fault impact and maximize the effectiveness of each gate inserted in the design. The key produced by logic locking is stored on the server, and any attempts to tamper with the bitstream result in a corrupted or incorrect output, rendering it impossible to clone the IP. During runtime, without the correct key provided from the server, the original application's functionality remains unknown and challenging to break. After successful device authentication, the correct key is sent from the server and stored in the device TPM.

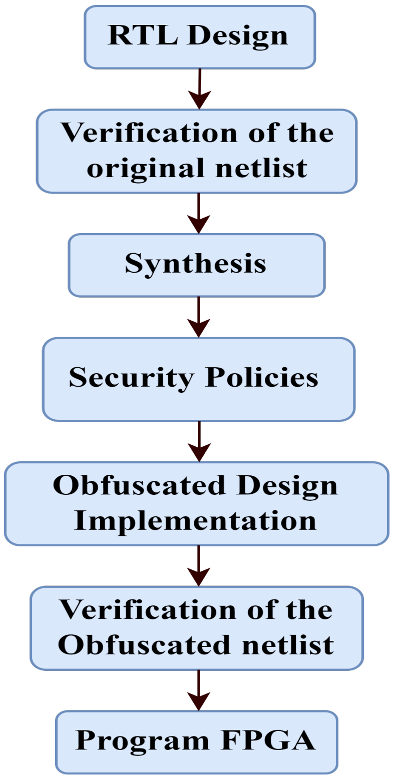

RTL Design
Verification of the original netlist
Synthesis
Security Policies
Obfuscated Design Implementation
Verification of the Obfuscated netlist
Program FPGA

Figure 3.2: Key Gate Insertion Framework

3.3.3 Secure IP Using ARM TrustZone

The ARM TrustZone configuration provides accessibility of MIO ports through the secure world. Key storage for the TPM is integrated with the FPGA via the SPI interface and the TPM driver library. To establish secure communication at the FSBL for secure boot, transfer functions are implemented. Using AXI interconnects, PS configures registers to design a secure IP in the PL, with the AXI interconnect providing the IP security status parameter. After secure boot authentication, the ARM TrustZone loads the logic-locked bitstream on the secure IP of the PL. If an unauthorized master attempts to access the secure IP, the secure IP raises an error signal, with NS bits on the AXI bus indicating the security transaction status.

The AXI bus comprises five communication signals, as depicted in Figure 3.3, to establish communication between a master and a slave. ARPROT and AWPROT signals are used to determine whether read/write access is granted to secure and non-secure IPs. Depending on the NS bits, the slave checks for security violations, and transactions are completed with read or write signals. To monitor the master-slave interface's AXI-transactions, a core wrapper is implemented. The wrapper ensures that only authorized transactions are permitted, and any configuration modification raises an exception. Secure transactions take place in the secure IP of the PL using ARM TrustZone. After device authentication, the key to unlock the bitstream stored in the TPM is securely sent to the secure IP.

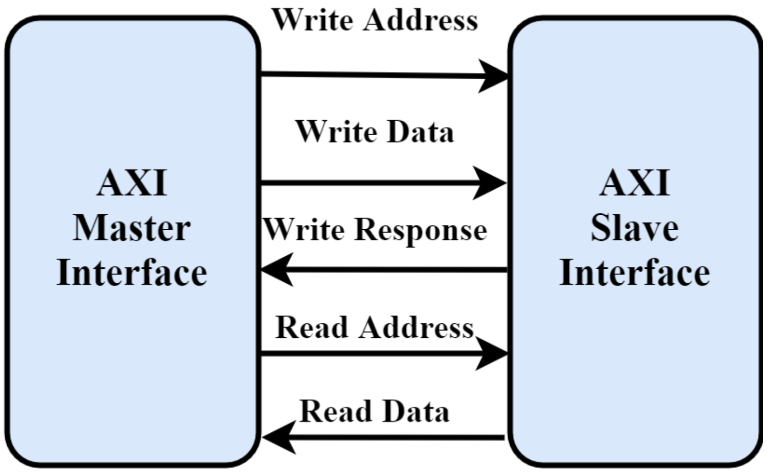


Figure 3.3: AXI communication signals

3.4 Implementation and Experimental Results

The hardware setup for the proposed framework on the Xilinx Zedboard FPGA, equipped with Zynq-7000 XC7Z020-CLG484, incorporates the ARM Cortex A9 processor and TPM SLB9670 module, as shown in Figure 3.4. To generate unique key pairs, the HELPPUF component is integrated into the existing hardware functions [79], and the device-unique encryption key is generated by the PUF. This 128-bit encryption key is then utilized by the AES cryptographic core for bitstream encryption and validation. The system block with PUF IP and AES IP are added as secure slave registers with a custom configured system wrapper, and each General-Purpose Input Output (GPIO) port is 32-bit wide. Once device authentication is completed, the authentication bitstream is programmed into the PL and verified with the PUF generated keys before the logic-locked application bitstream is sent from the server to the device.

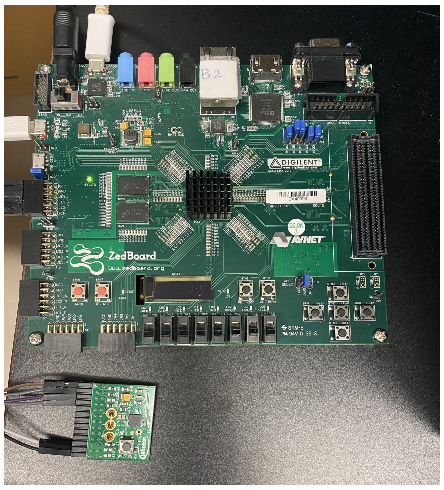


Figure 3.4: Hardware setup

The obfuscated automated framework with SARLock and Fault-based encryption is tested using circuits from benchmarks including the ISCAS-85/89 suite [82]. For fault-based encryption, the circuits are analyzed using the ABC tool [80] to generate bench and Verilog files, and the fault impact is calculated using the stuck-at fault analysis. By using the fault impact provided [83] in which using the stuck-at fault analysis the Fault impact = (#test patterns detecting sa0) x (#output bits affected by sa0) x (#test patterns detecting sa1) x (#output bits affected by sa1) is calculated. Key gates are inserted for the highest calculated fault impact, which protects the IP from reverse engineering. This design offers flexibility to control corrupt outputs and maximize design complexity for the attacker by targeting 50% Hamming distance with a smaller number of key gates, significantly reducing the area overhead. Figure 3.5 shows the system block with PUF IP and AES IP added as secure slave registers with a custom configured system wrapper.

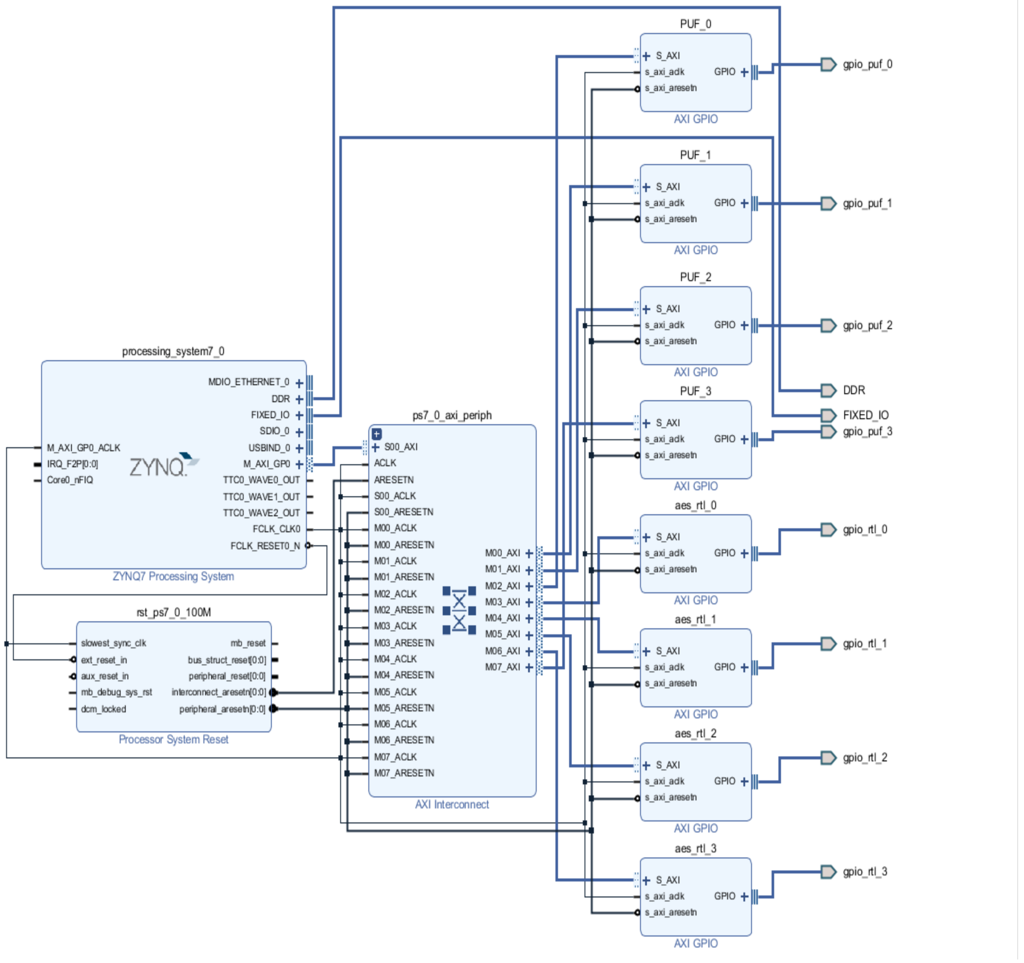


Figure 3.5: System block with secure IPs

Table 3.1 shows the fault impact summary for the ISCAS- 85/89 benchmarks with different sets of test patterns. This shows the fault coverage along with the faults detected at different gates to compute the fault impact for logic encryption. Based on the fault impact factor Table 3.2 shows the average Hamming distance calculated for the benchmarks with a range of key sizes between correct and incorrect outputs to obtain a 50% Hamming distance with a smaller number of keys to preserve the complexity of the locked design. Figure 3.6 shows the Hamming distance between the outputs of designs on applying the correct key and a random wrong key. The whole framework is automated along with functional verification for the generated netlist using input test vectors. The key generated during logic obfuscation is stored on the server and upon mutual authentication, it is sent to the secure IP and stored on the TPM. The secure IP and AXI interconnect implemented by using the TEE is used to eliminate non-secure transactions and other AXI attacks. By using an isolation design flow, the isolated secure IP block will have separate resources and ports for transactions.

Table 3.1: Fault impact summary for the ISCAS- 85/89 benchmarks

	C17	C432	C499	C7552	S510	S641	S713	S1196
No of primary inputs	5	36	41	207	19	35	35	14
No of primary outputs	2	7	32	108	7	34	23	14
No of test patterns applied faults	224	224	224	20000	224	20000	20000	20000
No of detected faults	22	472	1271	7550	561	402	475	1217
No of undetected faults	0	28	83	440	3	65	107	25
Fault coverage	100%	94.4%	93.0%	94.2%	99.5%	86.1%	81.6%	97.9%

Table 3.2: Average Hamming Distance (50%) for the benchmark circuits with different key sizes

Benchmark	C17	C432	C499	C7552	S510	S641	S713	S1196
Range of Keys Size	2-5	17-20	39-42	50-60	41-47	25-32	25-30	8-20
Hamming Distance (%)	50	50	50	50	50	50	50	50

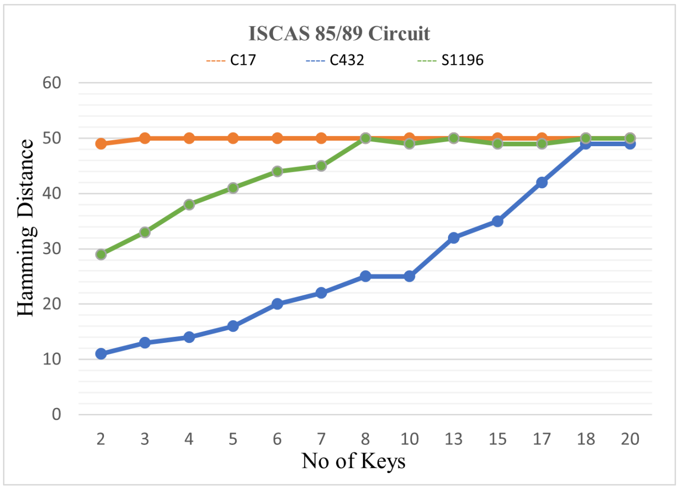
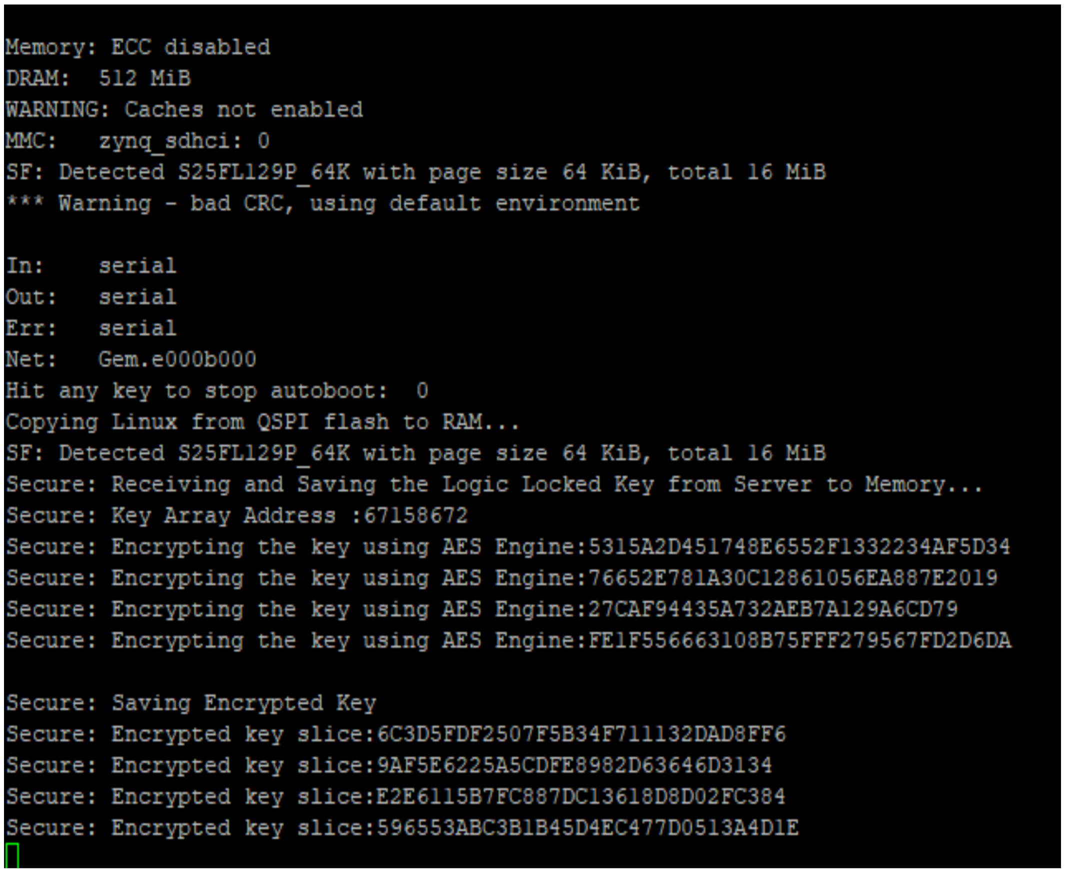


Figure 3.6: Hamming distance between the outputs of designs on applying the correct key and a random wrong key

The master-slave ports with dedicated memory space are used for memory transactions (GP AXI Ports). The keys are stored in TPM with driver functions and through authorized configuration, the application is unlocked. Thus, unauthorized transactions and readback modifications are blocked by using the isolation technique. To secure the key from non-secure IP and AXI attacks, 128- bit AES encryption is done to the logic-locked key in the Secure IP. If any non-secure IP tries to access the key, only the encrypted version of the key will be available which makes it more secure. Figure 3.7 shows the serial terminal in which the secure IP gets the key from the server for the logic-locked application and saves it in the TPM and does 128-bit AES encryption to the key to camouflage the original key. This model provides security policies such as authorization by the user to update the system once the verification is done. Transactions with the Secure IP is not possible by any Non-Secure IPs(master) which eliminates illegal memory access. The AXI wrapper with custom IP creates a bridge between the PS and the PL. Configuration registers for the AXI ports are defined for the security policies of the application.



```
Memory: ECC disabled
DRAM:  512 MiB
WARNING: Caches not enabled
MMC:   zynq_sdhci: 0
SF: Detected S25FL129P_64K with page size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
Copying Linux from QSPI flash to RAM...
SF: Detected S25FL129P_64K with page size 64 KiB, total 16 MiB
Secure: Receiving and Saving the Logic Locked Key from Server to Memory...
Secure: Key Array Address :67158672
Secure: Encrypting the key using AES Engine:5315A2D451748E6552F1332234AF5D34
Secure: Encrypting the key using AES Engine:76652E781A30C12861056EA887E2019
Secure: Encrypting the key using AES Engine:27CAF94435A732AEB7A129A6CD79
Secure: Encrypting the key using AES Engine:FE1F556663108B75FFF279567FD2D6DA

Secure: Saving Encrypted Key
Secure: Encrypted key slice:6C3D5FDF2507F5B34F711132DAD8FF6
Secure: Encrypted key slice:9AF5E6225A5CDFE8982D63646D3134
Secure: Encrypted key slice:E2E6115B7FC887DC13618D8D02FC384
Secure: Encrypted key slice:596553ABC3B1B45D4EC477D0513A4D1E
```

Figure 3.7: Design using AXI GPIO Ports for Secure IP

3.5 Security Analysis

The implementation of a secure boot with various security features for reconfigurable logic is crucial in forming a resilient model against malware attacks. The proposed framework incorporates the following security properties:

- PUF-based challenge-response pairs are used for mutual authentication, providing a unique set of key pairs for authentication between the device and the server.

- The PUF keys are employed for decrypting the logic-locked application bitstream, and the logic-locking key is shared by the server after the authentication process is completed. The keys are securely stored on the device's tamper-resistant memory, and the TPM module mitigates invasive attacks aimed at acquiring the key.

- The logic-locked bitstream produces corrupted output if the authentication fails, making it practically impossible to clone the IP. Additionally, the original functionality of the application remains unknown and difficult to break during runtime.

- Fault-based encryption analysis using the fault impact metric and Hamming distance shows that even with a smaller number of key gates, the ambiguity of the locked design functionality is preserved. Any single incorrect key changes 50% of the outputs or the functionality, thereby preserving the security of the design.

- TrustZone IP isolation and wrapper implementation for secure IPs integrating AXI signals help eliminate unauthorized transactions and readback modifications from the FPGA ICAP interface. AES encryption provides additional security by encrypting the keys and camouflaging the original key, protecting the keys from other access mediums.

3.6 Conclusion

This work investigates the threat model in the boot time process along with runtime attacks and presents a secure framework to implement logic-locking for runtime security and extend its application of secure boot process for FPGAs. The automated framework demonstrates the secure design flow to enable security functions such as RTL to secure bitstream, logic obfuscation, technology mapping, IP isolation, and support secure boot applications during runtime. The framework is tested using ISCAS 85/89, benchmark suite and is demonstrated on the Zynq 7000 family of Xilinx FPGAs for secure boot applications with authentication over the fly and secure IP transactions using TrustZone features.

CHAPTER 4: EXTENSIBLE HARDWARE CRYPTOGRAPHIC ACCELERATOR FOR RISC-V MODEL

4.1 Introduction

The importance of cryptographic operations in networked computing platforms cannot be overstated, particularly when it comes to maintaining data confidentiality and ensuring secure communication. SoC design models rely heavily on algorithms like AES for encrypted data storage and RSA/ECC for key exchange during protected communication [84]. Hardware models require security policies such as key management and information flow to support ISA extensions.

However, the efficient and secure sharing of resources from crypto accelerators presents challenges for applications that require protection against untrusted interfaces. Resource sharing complicates data isolation and poses high security risks, particularly with traditional architectures that are vulnerable to side-channel and memory-based attacks. Additionally, complex designs that result from implementation technology, microarchitectural extensions, and other hardware-based approaches make it difficult to track the flow of integrated models.

To address these challenges, RISC-V-based open-source hardware accelerators have been developed for extensible and efficient implementation, offering design insights for future RISC-V core designs and implementations. For example, the Advanced Encryption Standard (AES) cryptographic accelerator is specifically designed to perform computationally intensive encryption and decryption operations, making it an efficient and beneficial option for client and server platforms.

4.2 Threat model

To ensure secure data communication in a client-server model, authentication of the client is required. The AES engine is commonly used for remote authentication, with different key lengths being used to balance security and performance requirements. However, implementing the cipher algorithm itself can leave a system vulnerable to side channel attacks, in which an adversary can obtain keys through timing or power-based analysis. Multiple security levels in a system and parallel operations can also lead to secret key leakage, and hidden trojans can obtain unauthorized access by modifying the boot process and obtaining necessary authentication keys. Considering these risks, this work focuses on developing an independent crypto accelerator using the AES algorithm with architectural flexibility, to enable secure attestation.

4.3 Advanced Encryption Standard

AES is a symmetric-key encryption algorithm which is a variant of the Rijndael cipher. AES has fixed block size of 128 bits with different possible key lengths such as 128, 192 and 256 bits. Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Except for the last round in each case, all other rounds are identical with four main sub processing steps. Figure 4.1 presents the overall structure of AES with a 128- bit encryption key. The AES operations are performed on a 2-dimensional array of bytes called the state. The main four steps in each round are:

- Substitute Byte: It consists of a lookup table using 16x16 matrix of byte values called an S-box. The entire lookup table is created by multiplicative inverses in GF (28) and it scrambling to eliminate bit-level correlations in each byte.

- Shift Rows: This step involves shifting the row array by the first row of the state is not altered. The second row is shifted by 1 byte to the left in a circular manner and similarly, the third row is shifted by 2 bytes and 4th row is shifted by 3 bytes to the left. The goal of this transformation is to scramble the byte order inside each 128-bit block.

- Mix columns: This step is a substitution stage which uses the arithmetic of GF (28) where each byte of a column is mapped into a new value that is a function of all four bytes in the column. Each element of the product matrix is the sum of the products of elements of the one row and one column where the multiplication operation has been reduced to a shift and XOR operation making the AES algorithm very efficient o implement.

- Add round key: In this step, the 128 bits of the state are bitwise XORed with the 128 bits of the round key. A column wise operation between the 4 bytes of a state column and one word of the round key is done which affects every bit of the state.

For the encryption process, all the four steps are carried out for all the rounds except for the last round which eliminates the mix column stage. The Key Expansion is done to generate a key schedule where it takes a 4-word key and produces a linear array of 44 words. The key expansion is designed to be resistant to known cryptanalytic attacks. The decryption process has the same key scheduler, but the entire process is the inverse of the encryption process.

4.4 Design Implementation and Experimental Results

The AES accelerator consists of different sub modules implemented on the RISC-V architecture. This work implements a software and hardware-based AES crypto engine for RSIC-V architecture for securing the critical data by leveraging the open-source benefits of the architecture.

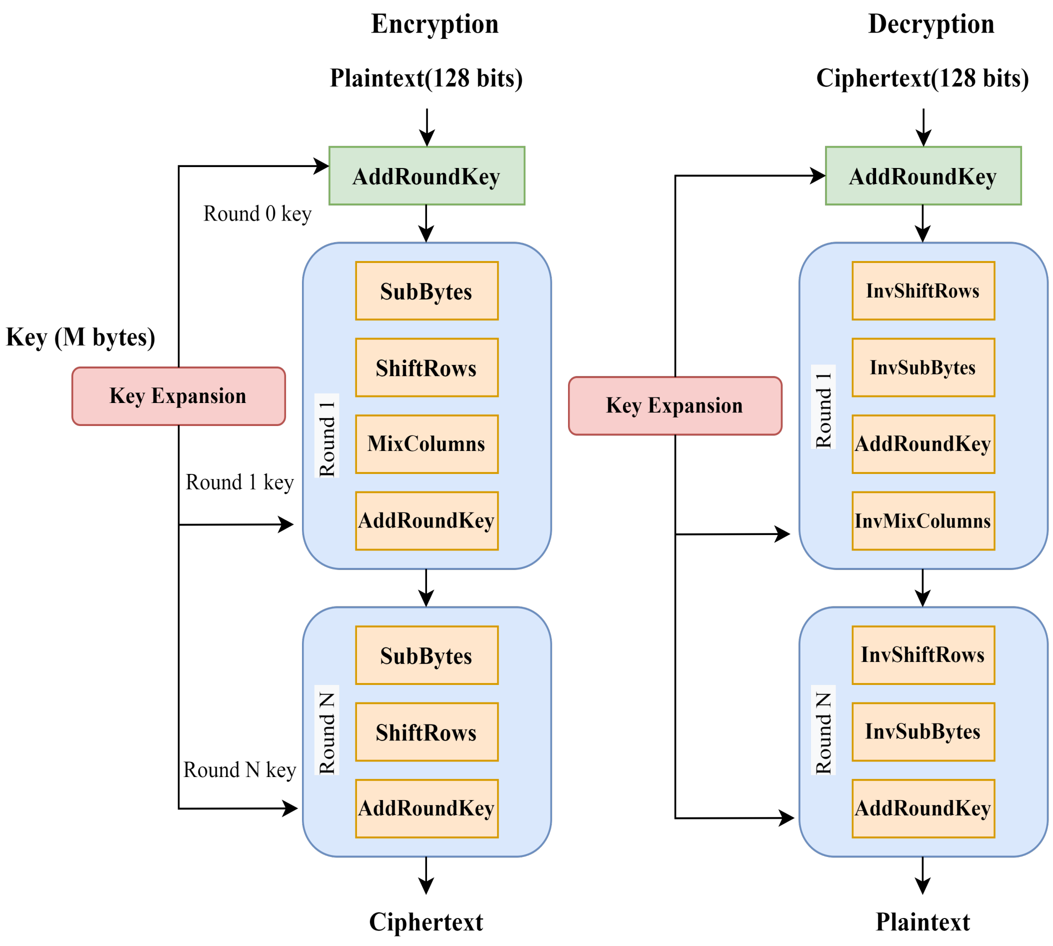


Figure 4.1: Overall structure of AES with 128- bit encryption key

4.4.1 AES Algorithm Implementation using Chisel

The AES algorithm is implemented using the Constructing Hardware in a Scala Embedded Language (Chisel) which is an open-source hardware description language that is utilized to create digital electronics and circuits at the register-transfer level. It is an open-source platform that facilitates advanced circuit generation and design reuse for both FPGA and Application-Specific Integrated Circuits (ASIC) digital logic designs. By adding hardware construction primitives to the Scala programming language, Chisel empowers designers to produce synthesizable Verilog through complex and parameterizable circuit generators. This allows for the creation of reusable components and libraries, such as the FIFO queue and arbiters in the Chisel Standard

Library, which raises the level of abstraction in design while maintaining fine-grained control. Chisel is powered by the hardware compiler framework, FIRRTL (Flexible Intermediate Representation for RTL), which optimizes Chisel-generated circuits and supports custom user-defined circuit transformations.

The AES algorithm consists of an encryption module, a decryption module, and a key expansion module. The top module consists of the class engine which holds the encryption and decryption class bundles with different key stated to perform the key expansions. Figure 4.2 shows the basic AES algorithm block diagram with the different modules in which the plain text and key are provided as the input with the cipher text output. An efficient model which is highly flexible and with less resource utilization is implemented to support multiple configurations for adaptivity.

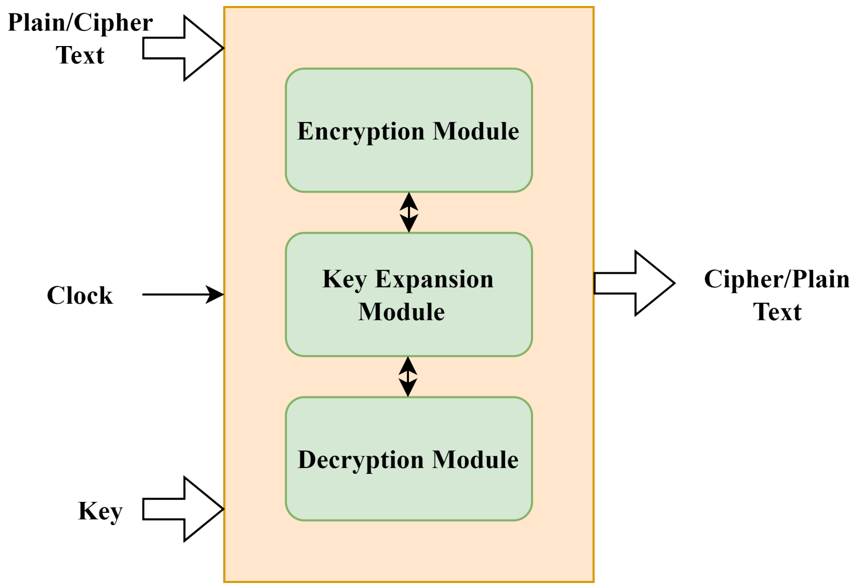


Figure 4.2: AES algorithm block diagram

Encryption Module: The encryption process consists of different transformations applied to the data blocks where each round depends on the length of the key used. An encryption class is implemented to perform the non-linear substitution using s-box and cyclic shift operations with different operations. Initially, the algorithm checks for the encryption engine's busy cycle for pipeline availability and allocates the required cycles based on the valid key size. This is done by using a state machine counter to initiate and check for the different states such as idle, ready and run and to reduce the latency. Once the key size and the data are provided it uses the different sub classes and the key expansion results to perform the different sub operations. Figure 4.3 illustrates the pseudo code for the encryption algorithm.

```
Encryption Algorithm:
    define Encrpty class()
        {
        EncryptEng := ready
        Check for valid text and key
        Specify the key size
        Get the KeyExp valid data
        Perform - >SubBytes     // AES LUT Table
                 ShiftRegister  //  AES State Array
                 MixColumns  //  AES Poly Vector Array
        }
```

Figure 4.3: Pseudo code of the encryption algorithm

Decryption Module: The decryption module is similar to the encryption module but consists of an inverse version of the sub byte, states and mix column array vectors from the AES LUT implementations to provide efficient area and energy utilization. It checks for the states to start the engine with valid data and performs the different operations with inverse mapping. Based on the key size the number of rounds is performed in an iterative way with a state check. All the transformations applied in the encryption process are inversely applied to this process. The last round values of both the data and key are first round inputs for the decryption process.

Key expansion: This module gets the key size (currently the key size is assigned to be 256-bit with 14 rounds) where the state machine initiates the state to perform the key expansion operations. The sub words are generated as a product of the previous round key, a constant that changes each round and a series of S-box lookups for each 32-bit word of the key.

The proposed algorithm was implemented on Arty A7-100T Xilinx XC7A100T

FPGA in which the RISC-V SiFive Freedom E310 was soft-cored. Figure 4.4 shows the Hardware setup with JTAG Debugger to flash the program. The top module calls the class engines to perform the encryption and decryption operations based on the key size.

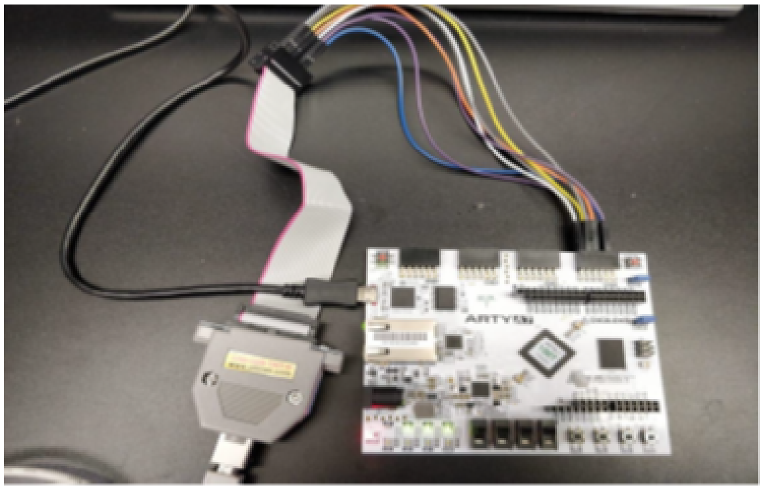
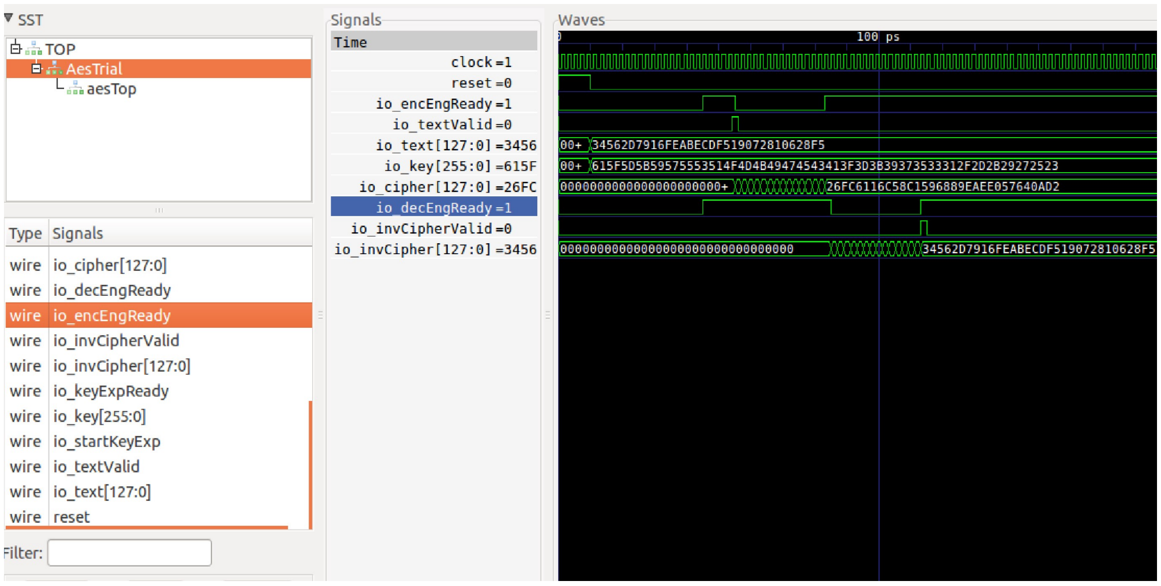


Figure 4.4: Hardware setup with JTAG Debugger

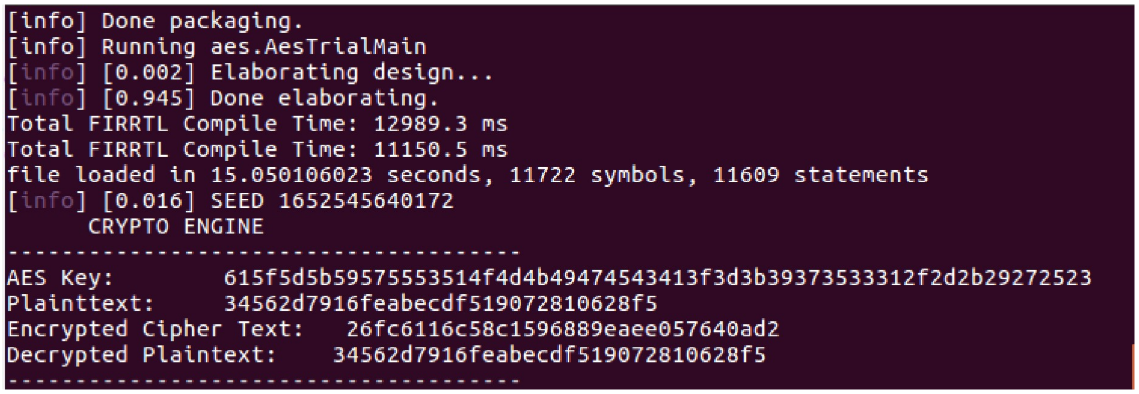
Figure 4.5 shows the simulation results consisting of the 128-bit plain text and the 256-bit key in which 14 rounds of iterative operations are carried out to encrypt the plain text and when the EngReady function signal is high the cipher text is obtained. For the decryption process when the inverse EngReady function signal is high the cipher text and key are taken as input for the decryption engine to produce the 128-bit plaintext. The AES Engine is executed on the RISC-V platform successfully to perform the encryption and encryption for the 1280-bit text, Figure 4.6 shows the AES engine running on the RISC-V core.



```
wire io_cipher[127:0]
wire io_decEngReady
wire io_encEngReady
wire io_invCipherValid
wire io_invCipher[127:0]
wire io_keyExpReady
wire io_key[255:0]
wire io_startKeyExp
wire io_textValid
wire io_text[127:0]
wire reset
```

```
Time
clock =1
reset =0
io_encEngReady =1
io_textValid =0
io_text[127:0] =3456          00+ 34562D7916FEABECDF519072810628F5
io_key[255:0] =615F           00+ 615F5D5B59575553514F4D4B49474543413F3D3B39373533312F2D2B29272523
io_cipher[127:0] =26FC        000000000000000000000000+ 26FC6116C58C1596889EAEE057640AD2
io_decEngReady =1
io_invCipherValid =0
io_invCipher[127:0] =3456     00000000000000000000000000000000 34562D7916FEABECDF519072810628F5
```

Figure 4.5: Simulation Results



```
[info] Done packaging.
[info] Running aes.AesTrialMain
[info] [0.002] Elaborating design...
[info] [0.945] Done elaborating.
Total FIRRTL Compile Time: 12989.3 ms
Total FIRRTL Compile Time: 11150.5 ms
file loaded in 15.050106023 seconds, 11722 symbols, 11609 statements
[info] [0.016] SEED 1652545640172
      CRYPTO ENGINE
--------------------------------------
AES Key:          615f5d5b59575553514f4d4b49474543413f3d3b39373533312f2d2b29272523
Plaintext:        34562d7916feabecdf519072810628f5
Encrypted Cipher Text:    26fc6116c58c1596889eaee057640ad2
Decrypted Plaintext:     34562d7916feabecdf519072810628f5
--------------------------------------
```

Figure 4.6: AES Engine

4.4.2 Hardware based AES Accelerator

The hardware based crypto accelerator consists of the AES core with 2 pipeline-based models where the first pipeline is used for the transformation of the 16-byte state and the other pipeline is used for computing the 16-byte key in each round. This is done by modifying the RSIC-V architecture for state transformation pipelines. In the first clock cycle the three stages namely: sub bytes, shift rows and mix column operations are carried out and in the second clock cycle the key expansion with the round key which is XORed is performed. Figure 4.7 shows the AES block diagram with the RISC-V core.

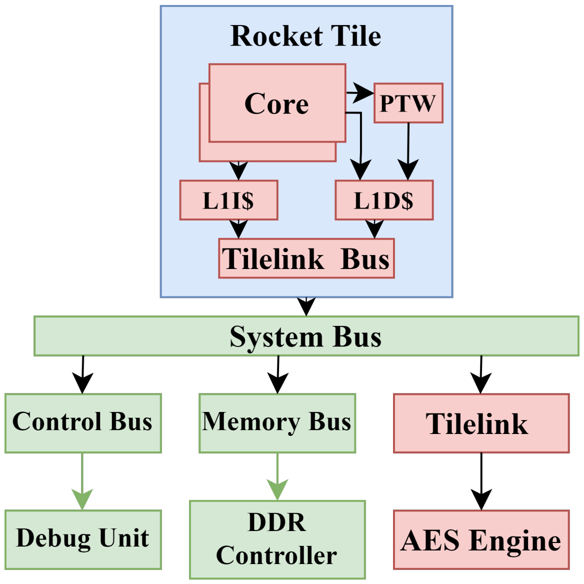


Figure 4.7: AES block diagram with the RISC-V core

The AES core bundle is instantiated to the registers where it is mapped to the addresses based on the size of the key and data. The AES control status register is created to control the valid address mapping for read and write operations. The AES core is implemented in chisel and using generators the Verilog files are generated for the core. The Tilelink module is used as the interface to interconnect the AES core with the processor core. The hardware-based model also consists of a state machine which keeps track of the state of the engine using a counter. Figure 4.8 shows the AES Algorithm instantiated with the RISC-V core.

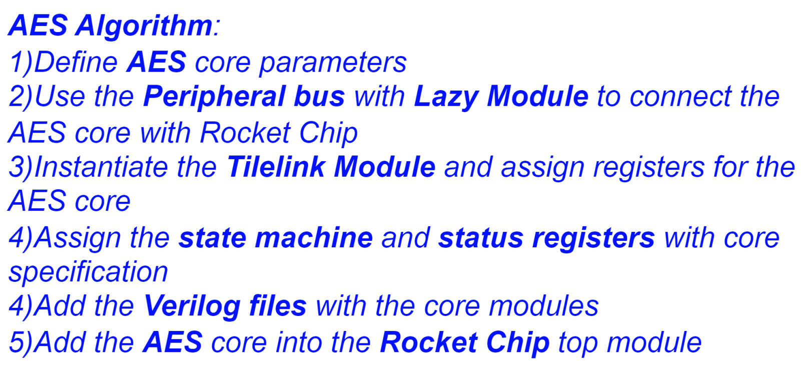

AES Algorithm:
*1)Define **AES** core parameters*
*2)Use the **Peripheral bus** with **Lazy Module** to connect the AES core with Rocket Chip*
*3)Instantiate the **Tilelink Module** and assign registers for the AES core*
*4)Assign the **state machine** and **status registers** with core specification*
*4)Add the **Verilog files** with the core modules*
*5)Add the **AES** core into the **Rocket Chip** top module*

Figure 4.8: AES Algorithm instantiated with the RISC-V core

The AES core is implemented with the most convenient tradeoff between performance and complexity: only the functions belonging to one round are implemented and used iteratively for several rounds ranging from 10 to 14 depending on the key size. The modified RISC-V architecture with the AES core was soft cored on the Arty A7-100T Xilinx XC7A100T FPGA. The SoC has been synthesized with an AES core which contains the state machine, status registers and other data and key registers. Table 4.1 presents the resource utilization of the AES core on the FPGA. Figure 4.9 and 4.10 shows the relative resource utilization percentage for AES and RISC-V core with the SoC. The results show very less area utilization as most of the computation uses the Battery Backup Random Access Memory (BBRAM) for the AES core which is an efficient model.

Table 4.1: Resource Utilization of the AES core

Module	LUT	FF	BBRAM
SoC	63400	126800	135
RISC-V	54523(86%)	32296(25.4%)	164(121%)
AES	3156(4.9%)	6806(5.3%)	100(74%)

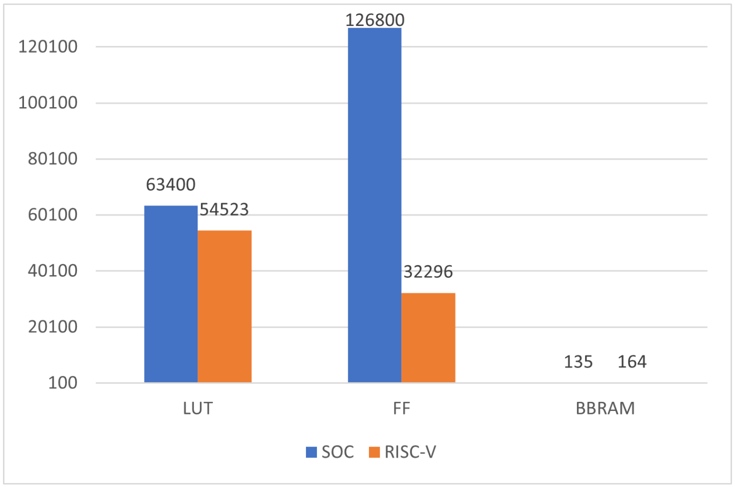


Figure 4.9: Relative resource utilization of RISC-V

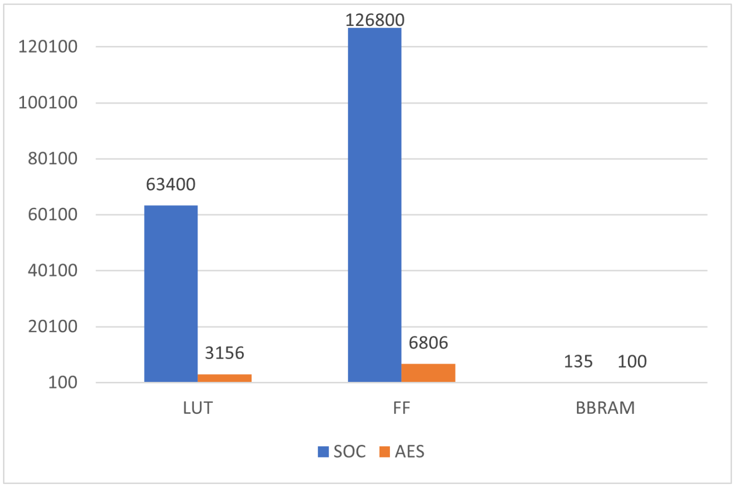
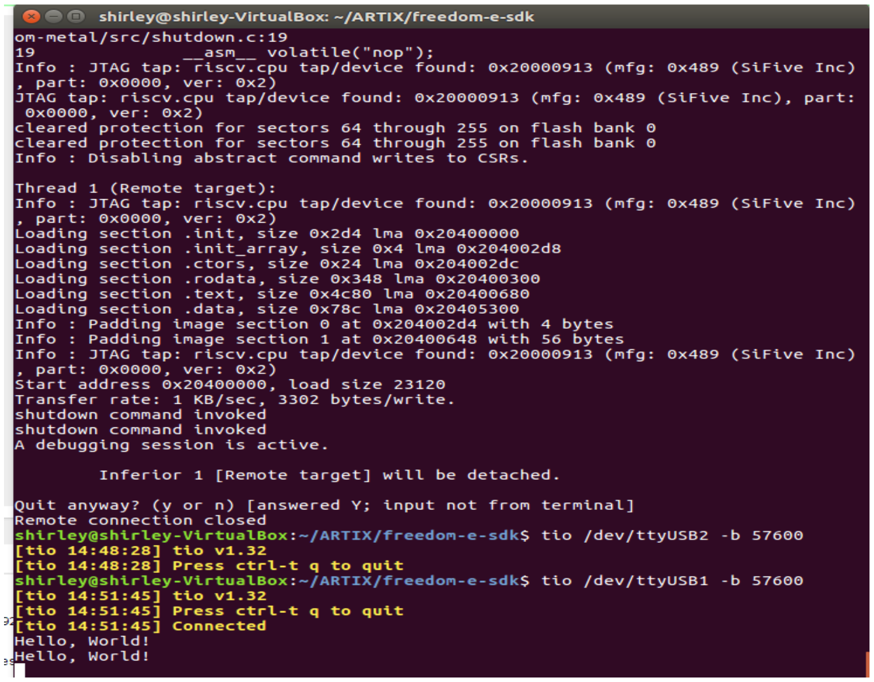


Figure 4.10: Relative resource utilization of AES

The synthesized core with the AES engine is flash programmed to the Artix board using the JTAG debugger and Figure 4.11 show the hello world application running on the modified RISC-V core. This acts as an assessment tool consisting of the crypto engine which can be extended based on the security policies thus providing a secured SoC platform for remote authentication.



```
om-metal/src/shutdown.c:19
19                  __asm__ volatile("nop");
Info : JTAG tap: riscv.cpu tap/device found: 0x20000913 (mfg: 0x489 (SiFive Inc)
, part: 0x0000, ver: 0x2)
JTAG tap: riscv.cpu tap/device found: 0x20000913 (mfg: 0x489 (SiFive Inc), part:
 0x0000, ver: 0x2)
cleared protection for sectors 64 through 255 on flash bank 0
cleared protection for sectors 64 through 255 on flash bank 0
Info : Disabling abstract command writes to CSRs.

Thread 1 (Remote target):
Info : JTAG tap: riscv.cpu tap/device found: 0x20000913 (mfg: 0x489 (SiFive Inc)
, part: 0x0000, ver: 0x2)
Loading section .init, size 0x2d4 lma 0x20400000
Loading section .init_array, size 0x4 lma 0x204002d8
Loading section .ctors, size 0x24 lma 0x204002dc
Loading section .rodata, size 0x348 lma 0x20400300
Loading section .text, size 0x4c80 lma 0x20400680
Loading section .data, size 0x78c lma 0x20405300
Info : Padding image section 0 at 0x204002d4 with 4 bytes
Info : Padding image section 1 at 0x20400648 with 56 bytes
Info : JTAG tap: riscv.cpu tap/device found: 0x20000913 (mfg: 0x489 (SiFive Inc)
, part: 0x0000, ver: 0x2)
Start address 0x20400000, load size 23120
Transfer rate: 1 KB/sec, 3302 bytes/write.
shutdown command invoked
shutdown command invoked
A debugging session is active.

        Inferior 1 [Remote target] will be detached.

Quit anyway? (y or n) [answered Y; input not from terminal]
Remote connection closed
shirley@shirley-VirtualBox:~/ARTIX/freedom-e-sdk$ tio /dev/ttyUSB2 -b 57600
[tio 14:48:28] tio v1.32
[tio 14:48:28] Press ctrl-t q to quit
shirley@shirley-VirtualBox:~/ARTIX/freedom-e-sdk$ tio /dev/ttyUSB1 -b 57600
[tio 14:51:45] tio v1.32
[tio 14:51:45] Press ctrl-t q to quit
[tio 14:51:45] Connected
Hello, World!
Hello, World!
```

Figure 4.11: Hello world application running on the modified RISC-V core.

4.5 Security Analysis

A secure hardware based crypto extension is implemented to provide secure encryption and decryption for a client server model. Chisel based implementation model highly reduces the long Verilog source-code to just a few lines limiting the memory usage with additional generator benefits. RISC-V based model provides flexibility by transplanting the cryptographic accelerators with less performance overhead compared to traditional architectures. This proposed AES algorithm uses minimal requirements of clock cycles with fewer hardware resources and can be extended to other security policies for secure transactions. With the evolving threats in the open-source, these security extensions provides a layer of security for securing critical data.

4.6 Conclusion

This research investigates the threat model in client-server networks and proposes a cryptographic AES engine on the RISC-V architecture which acts as a secure extension for authentication. This provides data encryption and decryption with limited resources. The research provides an in-depth description of the AES engine, the software and hardware implementations along with the resource utilization and encryption/decryption mechanism for a secured SoC platform.

CHAPTER 5: COARSE-GRAINED HARDWARE-BASED INFORMATION FLOW TRACKING

5.1 Introduction

The IFT technique focuses on preventing memory corruption and protects the return address from software attacks. The IFT technique is a security measure that focuses on preventing memory corruption and protecting the return address from software attacks. To achieve this, control data flow integrity is used with tagged bits to ensure that the return address matches the corresponding address after context switching, thereby preventing an adversary from hijacking the return addresses. The tag-based analysis approach is flexible in tracking data records with minimal overhead if a single bit is used as a tag. To implement this, compiler-based modifications are made to add the necessary security policies and to assist with the additional new instructions added to the architecture.

To prevent software attacks that focus on the return address, stack and data protection is achieved by tracking and detecting the tagged data. In the tagged mechanism, labels are associated with the address of data received from an untrusted source. The tag mechanism is implemented with minimal hardware overhead by using a 1-bit tag for the data address. During program execution, the tag mechanism assigns tag bits to spurious data through the tag propagation module. The tag propagation module assigns tags using new custom instructions implemented in the ISA architecture to store and check the tag bits. To reduce the dedicated memory assigned for storing tag bits, the tag bits are stored in the tag cache.

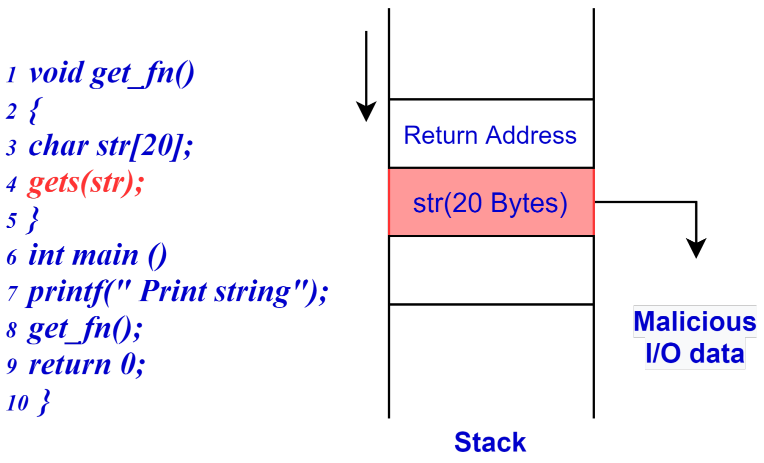
The RISC-V Rocket core is modified to incorporate security features at the ISA level to detect and eliminate buffer overflow and string format attacks. The tag module at the ISA level consists of all the tag management units, and translated compiler modifications are developed in RISC-V GNU. A simulation-based IFT model is demonstrated that translates the architecture-specific extensions to compiler-specific simulation model with toolchain extensions for RISC-V to verify the security extensions.

5.2 Threat model

An adversary can use an untrusted communication channel to exploit vulnerable codes by providing malicious inputs to the system. The application is vulnerable to permitting modification in the configuration parameters through software-based attacks. In this section, we focus on memory corruption through buffer overflow attacks and return address attacks.

5.2.1 Buffer Overflow Attacks

Buffer overflow attacks can occur when the stack, which is responsible for storing temporary variables created by a function, is overwhelmed with excess data provided by an adversary without proper bound checking. This can lead to the overwriting of important information such as the return address and base pointer with malicious data, resulting in system compromise or the execution of malicious code that can leak sensitive information. Therefore, it is important to implement measures to prevent these attacks and protect the system's integrity.



```
void get_fn()
{
char str[20];
gets(str);
}
int main ()
printf(" Print string");
get_fn();
return 0;
}
```

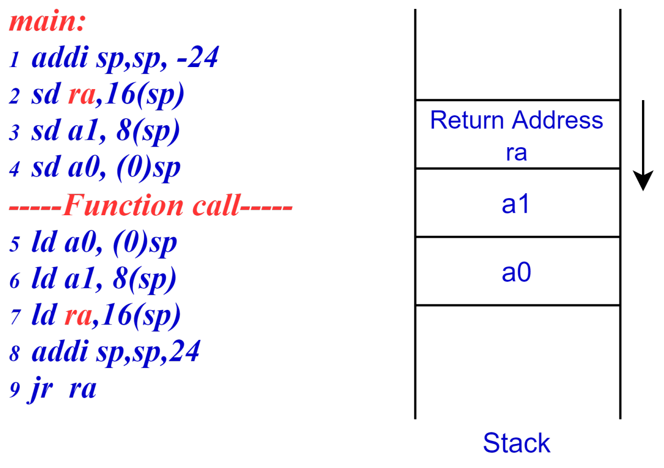
Figure 5.1: Buffer Overflow Attack

Figure 5.1 shows the buffer overflow attack caused by the sample program. The memory structure consists of the stack and heap structure along with other sections to be stored in the memory. Considering the example given, the get (str) can be passed with size longer than the buffer size causing the return address to be overwritten or the main function calls the get_fn() and the return address of the main is saved on the stack. The local variables are created for the get_fn and the specified space is allocated to the buffer. The gets(str) function which is highlighted gets an I/O string data that is received from a communication channel. If an adversary takes control of the I/O channels providing data that exceeds the limit of the buffer capacity, it results in a buffer overflow and overwrites the return address. Thus, buffer over attacks needs to be detected and eliminated before the return address is modified by using security policies which assign tag bits to the data return address.

5.2.2 Return Address Attacks

The return address attack, also known as the return-to-libc attack, is a major threat that must be addressed [85]. An attacker can modify the return address through a subroutine, injecting hidden code that can execute in the background without detection. Unlike other attacks, this method does not require an executable stack or shellcode but instead causes the main code to jump to a malicious program.

Figure 5.2 illustrates how the return address can be affected by input data from an untrusted communication channel during a procedure call. In this scenario, the stack is adjusted to make room for 3 items, including the saved values of the a1 and a0 registers and the return address. The normal load/store operations are used to implement this process, making it easy for an attacker to modify and overwrite the return address during context switching. This can result in the execution of hidden functionalities and data leakage without being noticed.



```
main:
addi sp,sp, -24
sd ra,16(sp)
sd a1, 8(sp)
sd a0, (0)sp
-----Function call-----
ld a0, (0)sp
ld a1, 8(sp)
ld ra,16(sp)
addi sp,sp,24
jr  ra
```

Figure 5.2: Return Address Attack

To protect against such attacks, most systems use address space randomization protection. However, this technique is vulnerable to side-channel attacks. To protect against return address attacks, new custom instructions can be implemented to track the flow of the address using tags in an IFT model.

5.3 RISC-V Architectural Design Approach

The proposed scheme consists of the Hardware IFT mechanism for data tracking framework with a tagged mechanism. The architectural model consists of the RISC-V platform for implementation in which by adding security extensions to the model it detects and eliminates the software attacks which corrupt the memory. Security policies are written to secure the system from different untrusted sources and by protecting the return address with an extendable tag mechanism. The tag mechanism module consists of different units which are used to assign tag bits and track the propagation of the tags during run time to detect spurious data.

5.3.1 RISC-V Core

The RISC-V core which features both 32- and 64-bit variants has fixed-length 32-bit instructions with variable length encoding for customizable applications [66]. This feature leverages the ISA to extend the instruction-set extensions. Being a load-store architecture, in which only load, and store instructions access the memory the IFT model focuses on the load and store instructions to check the integrity level of the memory contents. The loads are encoded in the I-type format and the store instructions are encoded in S-type format. Two new instructions are added in the ISA which is used in providing security checks for the 1-bit tag in the tag mechanism [86]. Based on the load and store encoding specified in the RISC-V core the new instructions are:

- In the load instruction encoding: LDTCHECK
- In the store instruction encoding: SDTCHECK

The new store instruction encoding is used to store data and assign a tag bit to the address of the data assuming the data to be spurious and the new load instruction encoding is used to load the data and check if the tag bit is 1 or 0. A new tag module is created, and an array table is assigned for the tag bits making it separate virtual access with a tag cache instead of the memory. The Control and Status Register (CSR) which offers custom address space for unused addresses is used to assign a new status for the tag values and mismatch conditions which results in an exception. The core is extended to add the tag bit along with the read and write requests to accommodate the new tag bit. The tag module is used to check the address of the load and store instructions which reduces the overhead in the architecture.

5.3.2 Tag Mechanism

The tag module consists of the tag initialization, tag propagation and tag checking modules. Usage of 1-bit tags reduces the overhead and additional memory access since all the tag mechanism is done separately in the tag cache. The tag initialization module assigns tags to all the data addresses from untrusted sources using the newly added custom instructions. The main purpose of the tag propagation unit is to track the tag bits from all the data addresses and store the details of the tags in the tag cache. Minimal information along with the path is stored in the tag cache to avoid overhead. Finally, the tag checking module is used to check the tag bits for all the data addresses after a procedure call and if there is a mismatch in the tag bit then an exception is raised. The checking unit checks all the properties associated with the security policies for the data. Based on the security policies the data is assigned as spurious and tracked completely to protect it from being modified or it affecting the memory leading to different attacks. Figure 5.3 shows the overall system design with RISC-V core Tag mechanism to protect the memory from software attacks.

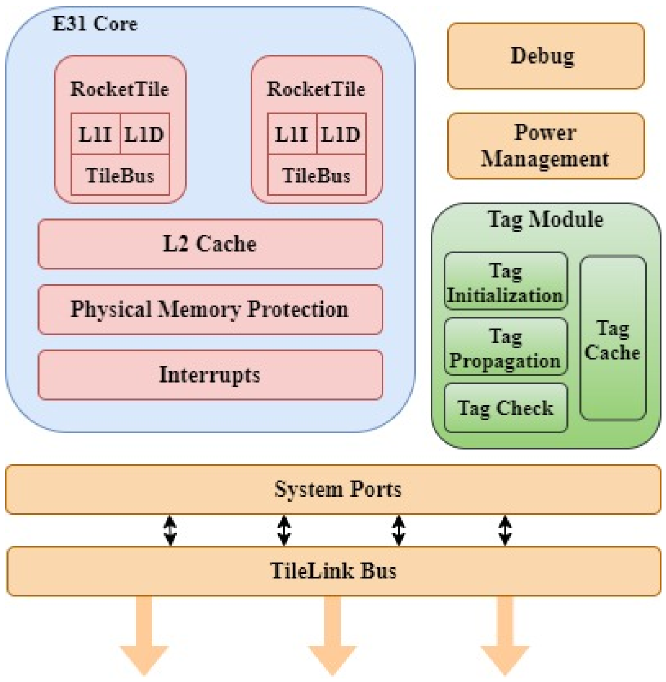


Figure 5.3: IFT design with RISC-V core

5.3.3 Security Policies

Potentially malicious I/O channels are identified by specifying the security policies for the system. Focusing on memory corruption the security policies are associated with the load/store instructions and the return addresses. All user supplied data is marked as malicious since it uses the load/store instructions with source and destination addresses. The program counter value is also tracked as the return address should be protected from modification. The tag check rules are applied to the newly created custom instructions tag bits and an exception is raised if the tag values mismatch which eliminates the attack. If a conditional branch instruction is performed the tag check module checks for the address using the tag bits and permits the execution only if the tag bits match with each other or terminates the call thus providing a coarse granularity model with better precision. The security policies implemented for protecting the return address assign checking rules for tracking all the procedure calls along with user input data.

5.4 Compiler-Specific Simulation Model

Many simulators use different techniques and technologies to achieve the required architectural specifications and one such RISC-V ISA simulator is the Spike simulator which adapts the ISA specifications and acts as a reference model for RISC-V. This paper focuses on extensibility and modifying the simulator with new security extensions with verification for runtime attacks in the RISC-V simulation model. A software simulation model that highly correlates with the hardware model with verification and runtime execution for the IFT technique.

The major goal of compiler/assembler modification is to extend the ISA for several instruction set extensions. Spike ISA simulator is used to test the modification of ISA on the software level and is divided into different modules. Each module is responsible to simulate a block of architecture and adding new blocks in the simulator yields new security extensions in the model. The initial step is to declare the instruction to its instruction format to generate matching and masking address values for the instruction. Then the new instructions corresponding to the matching address and master address are introduced by describing the instruction length, operands, and functionality like the hardware model. Figure 5.4 shows the filed format for the RISC-V architecture to add the new instructions in the RISC-V opcode structure. The highlighted red fields are modified to accommodate the new instructions in the toolchain. The check tag and store tag functions are implemented to execute the tag condition for the return address along with security policies to determine the tagged flow.

Name	Xlen	Instruction Class	Instruction Operands	Match	Mask	Match Opcode	Pinfo

Figure 5.4: Field Format to add new instructions in RISC-V

To simulate the newly added instructions on the Spike simulator the tag module is added with functions to check and store the tag bits. For the IFT simulation module, a threat model with a buffer overflow attack in which the user input data is untrusted is implemented and the IFT model identifies the return address modified by eliminating the attack. The location of the stored return address on the stack is retrieved by intentionally causing the buffer overflow attack. The toolchain support provides the flexibility to add security policies and develop test beds to carry out security analysis for RISC-V ISA with minimal design overhead and better precision logic.

5.5 Experimental Results

5.5.1 Architectural Model

In the Tag Module architecture, the tags that are labelled for the return addresses and data are processed and all the information is stored in the tag cache. Similar to the load/store instructions the newly created tag-based instructions are used for adding tags to only the return addresses and the user defined data. Figure 5.5 shows the pseudo code for the tag module with tag cache where a class is created for the tag cache with various tag parameters. Check functions are implemented for tag matching and checking alone with a counter to maintain the tag array for the tag cache and updated based on entries. Security policy functions are written to fetch the tags and match the tag bits for validation.

```
Pseudo Code for Tag Module with Tag Cache:
1 Create a class to extend the tag module with the bundle parameters
      class Tagcache extends Tagmodule
          {
              val index, matchbit, tagbit ...
          }
2 Condition check functions for tag matching and valid checking
      when tagbit == matchbit
          {
              update the tag fields
          }
3 Update the entries and the array in tag cache
      class Counter extends Tagcache
          {
              tags updated with direct mapping entries
          }
4 Security policies applied to check functions
          {
              security policy functions to fetch the values and check the
              validity (store tag and check tag)
          }
```

Figure 5.5: Pseudo code for Tag Module with Tag Cache

The Xilinx Artix-7 FPGA board is used to run the modified RISC-V rocket chip with Vivado. User defined application is modified to run a simple buffer overflow attack to test the architecture. Thus, the overall increase in the usage of LUT resources does not exceed 1% and the processor performance is not affected as the tags are processed in parallel and independently along with the instructions. Table 5.1 shows the comparison of resource utilization of the proposed design with the Cryptographic Return Address Stack (CRAS) architecture.

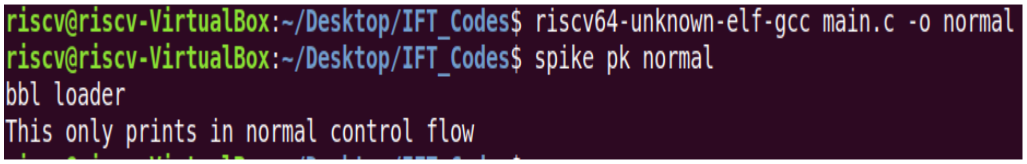
Table 5.1: Resource Utilization of IFT Module

Architecture	LUT	FF
Proposed Tag based IFT	10528	7113
CRAS	11468	6130

5.5.2 Simulation Model

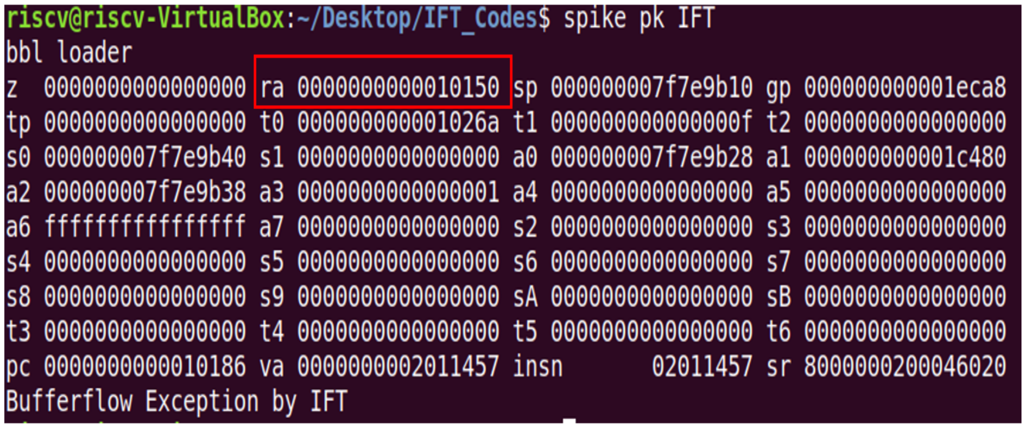
The IFT framework is tested on the Spike simulator by implementing a buffer overflow attack program. The new instructions are used with the stack operations where the return address is saved. Tag bits are assigned to the return address and the data to be stored on the stack. If there is a mismatch an exception is raised, and the program execution is stopped by eliminating the buffer overflow attack. The newly added SDTCHECK is used to assign a tag bit for the return address and the LDTCHECK is used to check the tag corresponding to the return address in the tag cache. This framework protects the stack by using customized instructions where the new return address compromised by an adversary is not loaded on the stack.

Segmentation fault occurs if the allocated memory address is exceeded but if bypassed it may access restricted locations and an adversary can gain access to the return address. This proves that even though the segmentation fault exception exits the program execution and avoids the program to access the protected memory location stored on the stack, it can still cause leakage of critically important data. When this is bypassed, the program executes on conventional RISC-V architecture by successfully executing the buffer overflow attack by modifying the return address and to eliminate such memory attack the IFT model is applied resulting in an exception to terminate the process. The model identifies the modification of the return address and stops the program execution by raising an exception to eliminate the buffer overflow attack as show in Figure 5.6 which illustrates the buffer overflow attack executed successfully without the IFT model on RISC-V and Figure 5.7 show the IFT module tracking the return address and eliminating the attack by raising an exception when the return address is modified.



```
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ riscv64-unknown-elf-gcc main.c -o normal
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ spike pk normal
bbl loader
This only prints in normal control flow
```

Figure 5.6: Buffer overflow attack in RISC-V Architecture without IFT module



```
riscv@riscv-VirtualBox:~/Desktop/IFT_Codes$ spike pk IFT
bbl loader
z  0000000000000000 ra 0000000000010150 sp 000000007f7e9b10 gp 000000000001eca8
tp 0000000000000000 t0 000000000001026a t1 000000000000000f t2 0000000000000000
s0 000000007f7e9b40 s1 0000000000000000 a0 000000007f7e9b28 a1 000000000001c480
a2 000000007f7e9b38 a3 0000000000000001 a4 0000000000000000 a5 0000000000000000
a6 ffffffffffffffff a7 0000000000000000 s2 0000000000000000 s3 0000000000000000
s4 0000000000000000 s5 0000000000000000 s6 0000000000000000 s7 0000000000000000
s8 0000000000000000 s9 0000000000000000 sA 0000000000000000 sB 0000000000000000
t3 0000000000000000 t4 0000000000000000 t5 0000000000000000 t6 0000000000000000
pc 0000000000010186 va 0000000002011457 insn       02011457 sr 8000000200046020
Bufferflow Exception by IFT
```

Figure 5.7: IFT module implemented eliminating the attack by exception

5.6 Security Analysis

The security extensions provided by the proposed framework to eliminate memory-based attacks are analyzed.

- The RISC-V model is vulnerable to leaks such as buffer overflows, return address attacks, fault injections etc. An adversary can modify the bit flips or the return address of the application leading to compromised data. The data flow of the security critical data is analyzed for accuracy by the tag module providing runtime protection for data integrity.

- This scheme provides the flexibility to add new security policies and develop test codes and software programs to carry out security analysis for the custom ISA.

- Provides coarse grain granularity by precisely tracking the implicit and explicit data flow and control flow paths with less area overhead.

- The simulation model with toolchain extensions supporting custom ISA is flexible to add new security policies for security analysis. It supports the architectural design and verification of security extensions to the RISC-V processor. This model acts as a security standard for accelerating the security verification from an architectural aspect to compiler specific verification.

5.7 Conclusion

This work investigates the threat model with untrusted channels leading to memory attacks and proposes a hardware-based information flow tracking mechanism to track the data form untrusted channels by modifying the RISC-V core. The hardware architecture specific extensions are translated to compiler specific simulation model with minimal design overhead and verified with an attack model. Hardware design uses 1-bit tags with separate tag cache for tag storage with less than 1% of area overhead and better precision logic. A secure RISC-V extension has been implemented by considering the security objectives for a SoC Platform assessment model.

CHAPTER 6: MULTI GRANULAR LEVEL BASED INFORMATION FLOW TRACKING

6.1 Introduction

Gate level IFT tracks information at fine granularity by targeting the gates [38]. This approach is used to detect the leakage of sensitive data and integrity violations of untrusted inputs. In GLIFT, for a given set of inputs, tainted bits are assigned, and the flow tracking is determined by tracking the outputs of all the tainted input bits. If the output is affected by any changes in the tainted inputs, then the output is marked as spurious. This approach is precise but does not ensure if the untrusted data is spread across the whole datapath and leads to architecture inflexibility. Thus, GLIFT model can be used to implement an architecture that can track the data for certain data critical modules leading to an efficient precise logic with minimal overhead. The work focuses on the gate level design to target on a specific security critical module and the datapath of the module to be executed is tracked. The design aims to detect the tainted triggering inputs that affect the output. The proposed scheme reduces the complexity of the gate level model and enables to track the data critical modules such as crypto engines and security accelerators rather than tracking the whole module. This proposed technique integrates the tagged mechanism and provides coarse and fine granularity in tracking the data.

6.2 Threat model

The threat model consists of two important scenarios. Firstly, it focuses on any data from untrusted communication channels which can trigger hidden functionality to modify the return address or corrupt the memory region during run time execution and secondly it targets hardware accelerators and crypto engines that are used for specific functions. These modules use encryption/decryption logic to secure the critical functions and may leak the keys during execution. Tracking the inputs and outputs of these modules is an essential aspect to identify the data transfer flow and abnormalities in the system. RISC-V extensions and Toolchain modifications make this approach more efficient than other hardware-based approaches. Tracking the data flow of inputs and outputs for a specific module rather than an entire design enhances the scaling flexibility of the gate level approach.

6.2.1 Data Level Leakage

The hardware design is not only vulnerable to untrusted communication channels but is also potentially vulnerable to leakage of sensitive information from security critical modules such as crypto engines and accelerators along with activated trojans which can trigger the payload condition. Therefore, the analysis or tracking of data requires access to gate level netlist to check the inputs which influences the outputs. Modelling the integrity properties of the data at the gate level is done by gate level IFT using shadow logics.

Figure 6.1 shows an example of trojan insertion in a gate level circuit consisting of logic gates. The inputs(a,b,c,d) are given to OR gates and the result is XORed which in turn is given as input to the NAND gate. A trojan input(T) is fed as another input to the NAND gate which acts as a hidden trigger. When the trigger input is 1 the output is affected by the input resulting in an activated signal which triggers the hidden function to be enabled. The secret keys are leaked or replaced by the function executed by an adversary. Thus, the violation of key integrity is tracked by information flow properties which are implemented using gate level IFT models providing fine grain precision logic.

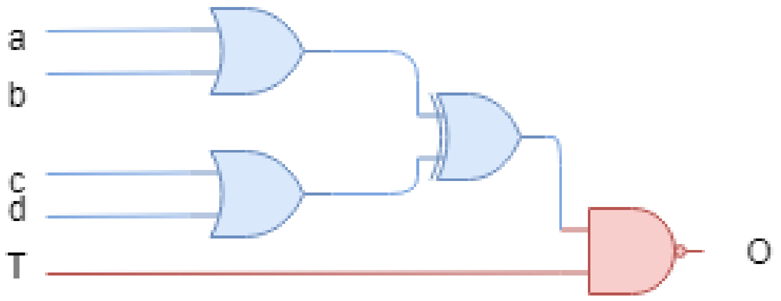


Figure 6.1: Trojan insertion in Gate Level Circuit

6.3 RISC-V Architectural Design Approach

The proposed scheme consists of the Hardware IFT mechanism for data tracking and an integrated gate-level IFT module for tracking security critical datapath modules for multi granularity. Considering only the gate level IFT module, it tracks the information flow through the logic blocks by performing a composition of augmented logic blocks in which the implicit, explicit, and covert information flow is tracked with fine granularity at the data level is done in gate level IFT models. The gate level design focuses on a specific security critical module and the datapath of the module to be executed to overcome the performance and area overhead from existing models. Special critical modules such as crypto engines and accelerators that share secret keys or security critical information are considered as the logic blocks for which gate level IFT is applied.

Without any hardware design modification, this approach is integrated with the RISC-V core as a separate module to perform information policies for user instantiated critical modules. Each input and output of the entire module is tainted and a shadow logic library for the tainted gates is implemented. Security policies are matched with the tainted information for the module under test. The tainted inputs for the gates are determined by detecting the arbitrary changes in the inputs which affect the output leading to data leakage and faults triggering hidden functionalities. Figure 6.2 shows the overall system design with RISC-V core Tag mechanism and Gate level IFT with shadow logic in which individual units are used to protect the memory from software attacks and data leakage.

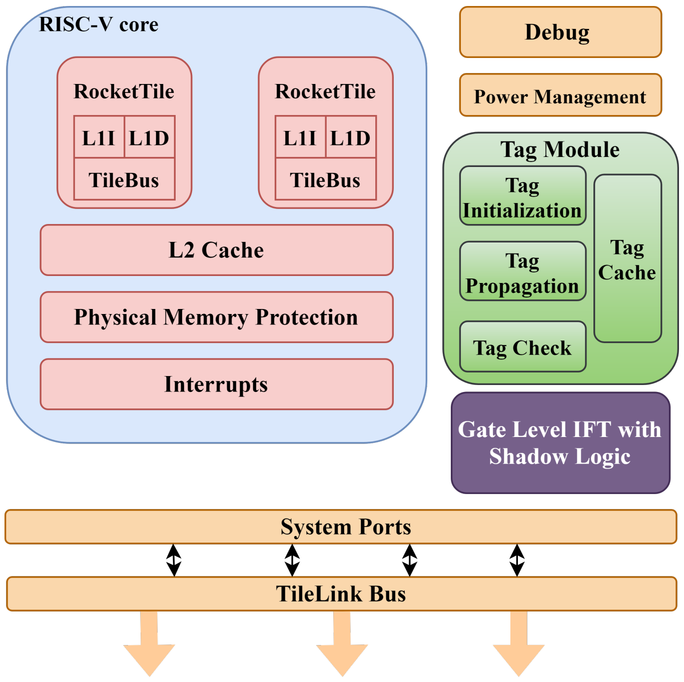


Figure 6.2: RISC-V core Tag mechanism and Gate Level IFT

6.3.1 Shadow Logic for tainted gates

All the data bit propagating from the input to the output is marked as tainted if the tainted input influences the output. Considering the example in Figure 2.4, we have two OR gates and one AND gate with a and b as inputs and o as the output. In an OR gate with reference to the truth table for the 2 inputs additionally tainted inputs are added and when the inputs are toggled and output which is affected by the tainted input is malicious and marked to be tracked through the whole information flow. Based on the minterms the shadow logic is formed for the OR gate and similarly it is done for the AND gate.

The shadow gate library is formed for all the basic gates and is compared with the main logic to mark the tainted output bits. The information flow policy checking module is used to track the flow based on the security policies implemented. The precision logic is indicated based on the tainted information. If it's true, then the output flowing from the input is spurious with information leakage.

6.4 Information Flow Policy Checking

The security policies for the logic gates and the associated tainted bits are implemented for all the gates in comparison with the shadow logic library [87]. The flow of the gate level model is described in Figure 6.3 Initially, the module to be tracked is selected by the user and verified. The module or the application is then converted to gate level netlist to track the information flow of individual data and they are separated into sub modules. The gate level IFT module gets the information and security policies from the shadow logic and the IF policy checking rules unit to determine the tainted inputs and the exact location of data leakage and spurious inputs are identified. The gate level IFT unit tracks these untrusted information paths and raises an exception if there is a change in the output and information flow resulting in malicious data detection.

This approach is efficient with no hardware modification and integrated along with architecture level IFT which makes it a model with accuracy and performance. Depending on the application or the micro-architectures added to the system, the Gate level IFT is performed on any critically sensitive modules. For example, in crypto engines, the keys in the AES encryption module can be tracked using gate level IFT to check the integrity and data leakage from the system. Figure 6.4 shows the simplified shadow logic for a 2-input OR gate. In this example, input b is considered untrusted, and the shadow logic truth table is highlighted to show the tainted input affecting the output. If a$=0$ and b is untrusted then the output(o) $= 0/1$ and when a$= 1$ and b is untrusted then the output(o) $= 1/0$ which proves that the output is affected by the input b. Similar to this the shadow logic library holds the truth table for all the gates and the tainted data is tracked using this logic and the exact location is found that leads to data leakage.

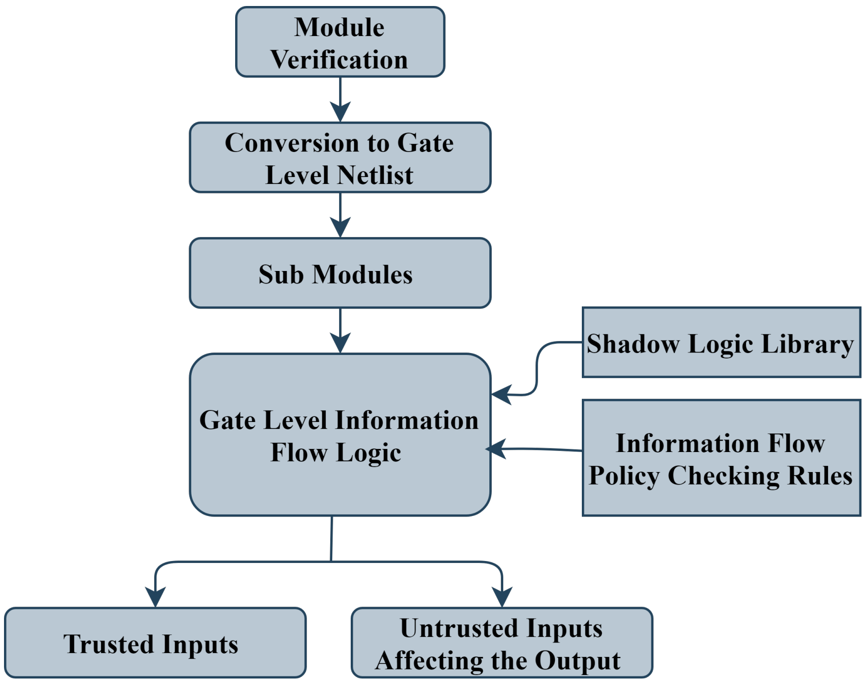
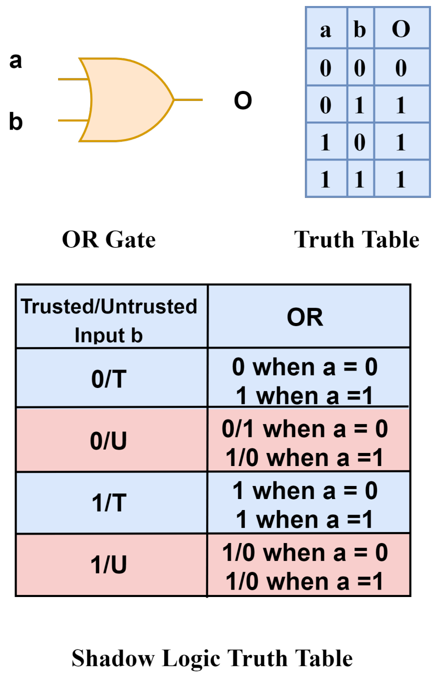

Module Verification
Conversion to Gate Level Netlist
Sub Modules
Shadow Logic Library
Gate Level Information Flow Logic
Information Flow Policy Checking Rules
Trusted Inputs
Untrusted Inputs Affecting the Output

Figure 6.3: Gate Level IFT Model Flow



a	b	O
0	0	0
0	1	1
1	0	1
1	1	1

Trusted/Untrusted Input b	OR
0/T	0 when a = 0 1 when a =1
0/U	0/1 when a = 0 1/0 when a =1
1/T	1 when a = 0 1 when a =1
1/U	1/0 when a = 0 1/0 when a =1

Figure 6.4: OR Gate Shadow Logic Truth Table

6.5 Experimental Results

To analyze the security properties of the gate-level IFT implemented and to verify the security properties for the critical datapath the ISCAS-85/89 and EPFL Benchmark circuits are used.

6.5.1 Netlist and submodules

The Xilinx PYNQ FPGA board is used to test the gate level IFT model with the c432 ISCAS-85 Benchmark circuits. As the RISC-V specification is highly flexible different soft processor variants can be used as a tool to study the threat model. Building a flexible Inter and Intra-ISA variations for comparison with the RISC-V family helps in analyzing the security extensions. RISC-V IP is implemented on the by using Vivado for the PYNQ board using Overlay. The overlay acts as a bridge between the designed IP and the FPGA. Figure 6.5 shows the PYNQ flow with the RISC-V IP and AXI Interface designed for the PYNQ board using Overlay. The GLIFT is integrated with the IP to track the datapath of the security critical modules. The combinational circuit c432 interrupt controller is considered an example to illustrate the gate level IFT implemented in the hardware. The circuit consists of 36 inputs and 7 outputs with a total of 160 gates in which based on the bus requests each channel is enabled on priority. The initial circuit is converted to gate level netlist using the implemented algorithm and is verified as the module to be tracked. The module is classified into submodules based on the information and critical data. Finally, the gate level IFT logic is used to compare the submodule to be tracked with the shadow logic library and the security policies are applied to track the gates in which the output is affected by the input. This gives a set of trusted input gates and untrusted input gates that needs to be tracked.

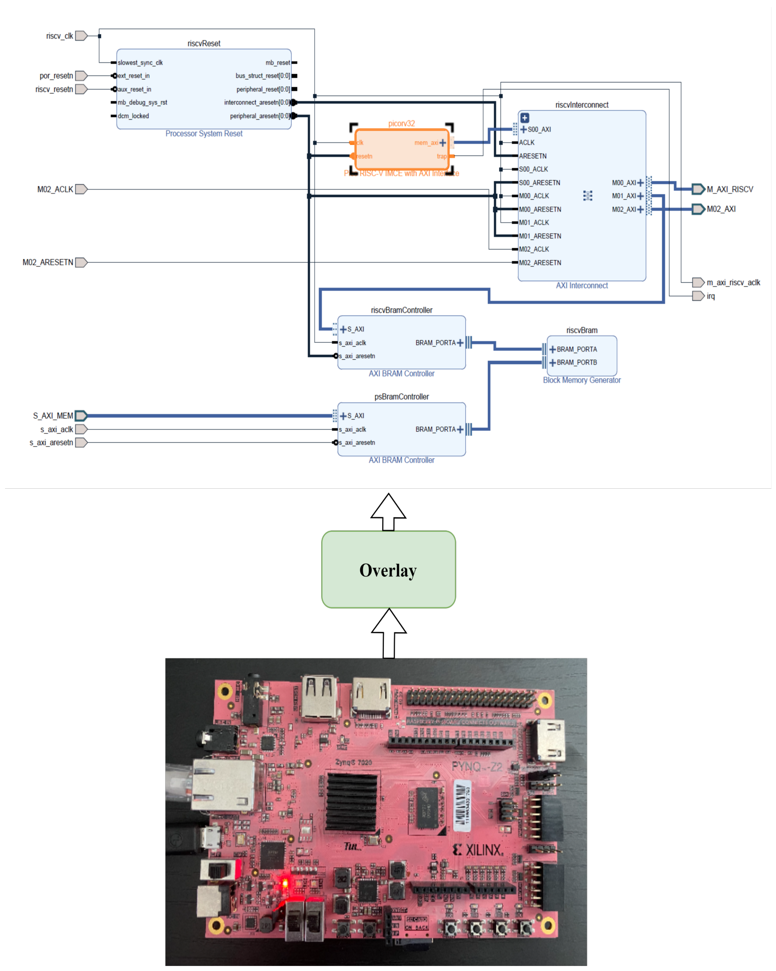
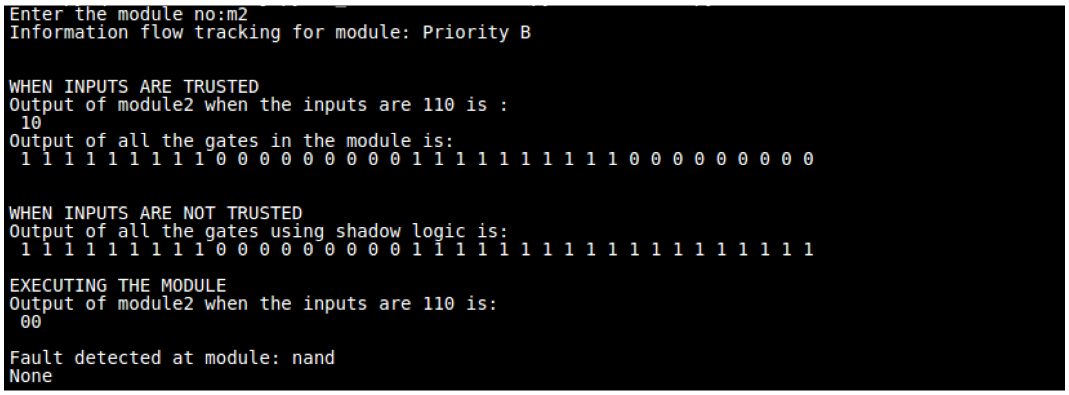


Figure 6.5: PYNQ flow with the RISC-V IP and AXI Interface designed for the PYNQ board using Overlay

The gate level IFT model then tracks the untrusted gates and detects information leakage by analyzing the data of all untrusted gate inputs. A variation in the data flow is observed and raises an interrupt indicating a subtle information flow that is hidden in the ISA abstraction. This model provides a fine precision logic with the accurate prediction of untrusted data. Implemented as a separate automated model this provides less area overhead when compared to other existing GLIFT models which increases the area by 70% by doubling the gates for shadow logic.

Figure 6.6 shows the gate level IFT detecting the untrusted gate in the submodule m2 for the c432 circuit. The circuit is divided into submodules and module 2 which is the priority encoder B is selected to be tracked with untrusted inputs and the model detects the fault at the NAND gate based on the shadow logic libraries and the sequence of inputs. It classifies the gates based on the shadow gate libraries and points out the untrusted gate. Similarly, with different sets of input, the model detects the untrusted gates and tracks the gates further during runtime to detect leakage of critical data and modification of data inputs and raises a security function call if there is a detection of malicious data inputs.



```
Enter the module no:m2
Information flow tracking for module: Priority B


WHEN INPUTS ARE TRUSTED
Output of module2 when the inputs are 110 is :
 10
Output of all the gates in the module is:
 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0


WHEN INPUTS ARE NOT TRUSTED
Output of all the gates using shadow logic is:
 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

EXECUTING THE MODULE
Output of module2 when the inputs are 110 is:
 00

Fault detected at module: nand
None
```

Figure 6.6: Gate Level IFT detecting the untrusted gate (NAND) for the submodule Priority Encoder B in c432 circuit

Table 6.1 shows the execution of the shadow logic to detect the untrusted gates in the circuit of different benchmarks. Individual security critical modules which are vulnerable to data leakage through input data path can be tracked to identify the location of untrusted inputs based on the proposed model with lesser execution time leading to a precise model without design modification or scaling.Table 6.2 shows the resource utilization of the RISC-V IP with the GLIFT module. This framework uses minimal area resulting in an efficient fine grain model. Figure 6.7 shows the relative utilization percentage graph with the SoC where only 8.5% of the LUT is utilized for the RISC-V IP.

Table 6.1: Execution Time Of ISCAS 85,89 And EPFL Benchmarks

Benchmark	No. of Inputs	No. of Outputs	No. of Gates	Shadow Logic Execution Time
C432	36	7	160	186s
S398	3	6	119	177s
Adder	256	129	2162	556s
Alu-crtl	7	26	306	373s
Memort-crtl(submodule)	1204	1231	8956	984s

Table 6.2: Resource Utilization of Proposed RISC-V IP

Module	LUT	FF	LUTRAM	BBRAM
SoC	53200	106400	17400	140
Proposed RISC-V Implementation	4506	4753	188	16

6.6 Security Analysis

The control flow hijack of the security critical modules leaking data or modifying the IP cores is an important security verification property which is handled by the gate level IFT with minimal overhead. It provides accuracy with all information flow along all timing channels. Different implementations either focus on data flow or leakage models from different channels. This model protects the system against leakage models by using gate level logic and maintains the data flow integrity by eliminating memory corruption attacks with accuracy. The proposed design is a flexible architecture with extensibility which can support the SoC Platform security reference model to protect the system during runtime.

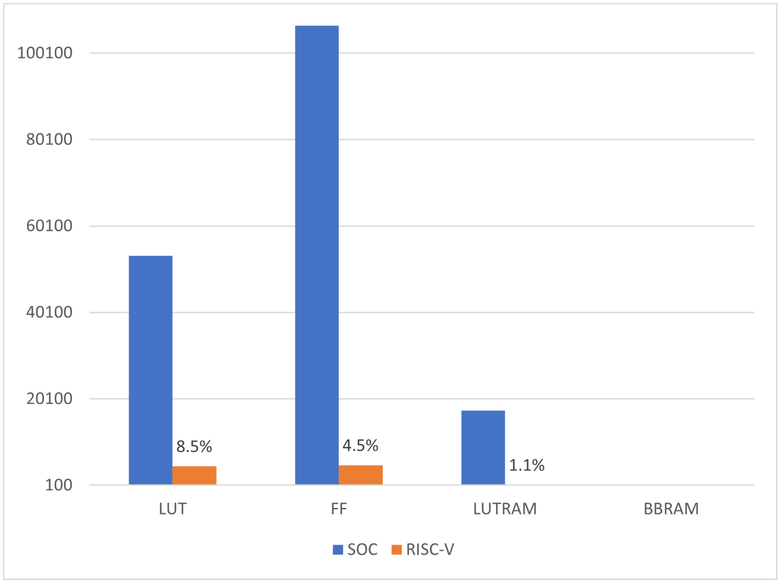


Figure 6.7: Relative Resource Utilization

6.7 Conclusion

This work investigates the data leakage threat model with architectural inflexibility and proposes a new multi-granularity based IFT model which uses a tag mechanism for data flow integrity and an optimized shadow logic for the leakage model. The modified RISC-V core provides system protection by tracking the data flow from untrusted channels as well as the data from security critical modules with accuracy during runtime.

CHAPTER 7: MICRO-ARCHITECTURAL VULNERABILITIES WITH MACHINE LEARNING MODELS AND TRANSFER LEARNING TECHNIQUE

7.1 Introduction

SoC platforms are becoming increasingly complex with a wide range of components gathered from different third-party companies in the global supply chain. This complexity leads to potential security vulnerabilities as hardware IP obtained from untrusted vendors may contain malicious implants such as Hardware Trojans or other integrity issues. In addition, recycled ICs or cloned counterfeit chips obtained through IP theft and IC tampering can decrease system performance and reliability. Reverse engineering techniques can also be used to obtain copies of the original IC design and labels, leading to critical data leakage.

As application software relies on the hardware root-of-trust for providing security guarantees, it is crucial to ensure that the underlying hardware is secure and free of vulnerabilities. Identifying and fixing these vulnerabilities is critical for ensuring the overall system security. While several research efforts have focused on hardware security verification, including techniques such as side-channel analysis, machine learning has emerged as a feasible solution for efficient detection and mitigation of hardware security vulnerabilities.

While there are a wide variety of research for hardware security verification [88], [89], machine learning has emerged as the feasible solution for efficient detection and mitigation of hardware security vulnerabilities. This work focuses on the machine learning models for microarchitectural vulnerability detection and proposes a counterfeit detection and avoidance model to ensure the authenticity of the component.

7.2 Machine Learning Models

In recent years, SoC security research has become increasingly challenging and resource-constrained. As a result, researchers have turned to Machine Learning (ML) techniques for assistance. ML algorithms are designed to create models, known as learning algorithms, from large amounts of historical training data. These trained models can be used for prediction or classification tasks. The flexibility and generalization capabilities of ML algorithms make them a promising solution for various application domains. Figure 7.1 shows a typical workflow for using ML techniques to analyze hardware vulnerabilities. The ML workflow involves four major steps: preprocessing, learning, evaluation, and analysis [90]. In the first step, datasets are collected from known hardware vulnerabilities. The second step involves training the ML model using the gathered dataset. The trained model is then evaluated in the third step. Finally, the validated model can be used to detect unknown vulnerabilities in the fourth and final step.

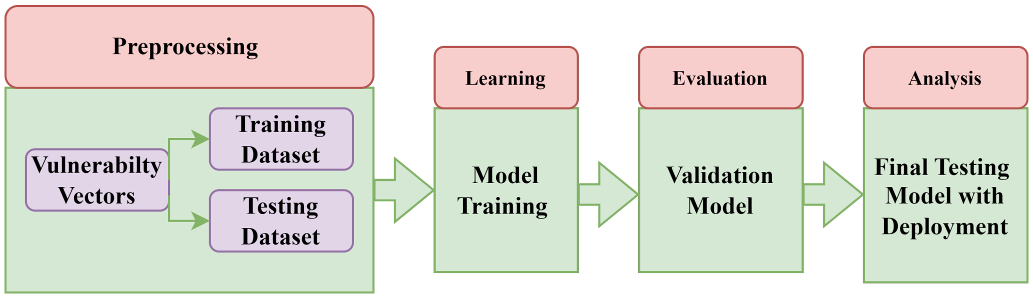


Figure 7.1: ML workflow applied for hardware vulnerability analysis

Machine Learning techniques can be broadly divided into three categories: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Supervised learning requires training labels such as Support Vector Machine (SVM) adopted in [91]. While unsupervised learning does not use labels but focus on extracting hidden features from input samples directly. The clustering algorithms that define the metric of the key for classification falls under this category [92]. Finally, reinforcement learning is a type of learning method that imitates human's learning process through continuous interaction and refinement.

ML based approach for hardware vulnerabilities is based on SoC security validation and verification where the detection models focus on a particular attack scenario. Some of the approaches that rely on trace analysis are:

- Simulation based validation: This approach focuses on the test generation to activate hardware vulnerabilities. The test vectors activates the trigger associated to the malicious trojan and a comparison of the simulation output with the expected output exposes the possible vulnerabilities in the system. Random simulation models [93], statistical test generations [94] and SVM based classifications [95] are some of the simulation-based approaches for hardware vulnerability detection.

- Side-channel analysis: SCA detection models against micro-architectural side channel attacks focuses on the time series of the data and correlates the execution time traces with the hidden information in the sequential data flow [96]. Runtime based detection models utilizes the different abstraction layers and with clustering algorithm to plot the distribution of the leakage information which distinguishes the malicious design [97].

- Formal verification: These models evaluate the functionality and security of the design using discrete mathematical models [98]. Satisfiability solving with iterations and model checking logics along with equivalence checking are some of the key steps in this approach. Extraction of the common counter examples for a given design for error patterns and graph-based models for variations lead to accurate verification with significant efficiency.

- Heuristic Analysis: This is a combinational analysis which consist of feature extraction followed by pattern recognition. Some of the countermeasures are the counterfeit IC detection based on SVM with parametric measurement analysis for feature extraction [99] along with for 2-D space mapping. Hardware Trojan detection by analyzing the structural features of the IP cores and malicious triggering modules in the gate level circuits [100].

7.3 Transfer Learning

The technique of transfer learning involves reusing a model that has already been trained on one task, in another task [101]. By using a pre-trained model with a large amount of dataset, fewer false positives/negatives can occur, resulting in a high accuracy system. To generate accurate features for prediction models, specific classification layers for the pre-trained model can be utilized. This approach reduces the amount of time the detection model needs to learn from scratch, making it self-trained and significantly improving the system's performance with limited resources.

The transfer learning method overcomes the learning paradigm of traditionally designed conventional Deep Learning and Machine Learning algorithms, and the optimized predictive model developed from this technique improves the generalization capability. Figure 7.2 illustrates the difference between traditional machine learning and transfer learning, where in the latter, knowledge learned from task 1 is extracted and transferred to the learning model of task 2, resulting in an accelerated learning process and better prediction models. Transfer learning is categorized into three sub-categories based on the type of transfer required: transductive, inductive, and unsupervised transfer learning.

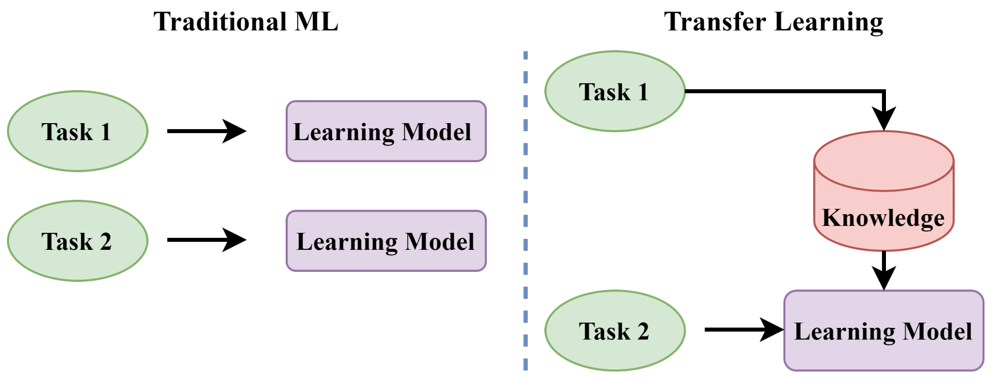


Figure 7.2: Traditional Machine Learning and Transfer Learning Model

7.4 Hardware Security Vulnerabilities

The fundamental threat to the market is posed by the supply chain vulnerabilities, which are the focus of this work. Offshore semiconductor manufacturing is now commonly outsourced by most design houses, making the supply chain vulnerable to compromise at several points. Counterfeit ICs, whether remarked or cloned, present a significant threat to various entities within the electronics supply chain. By manipulating package identifiers, adversaries can introduce remarked ICs that present used or alternative components as new and higher-grade versions of a desired IC. Cloned ICs, on the other hand, are functionally similar or identical to ICs but deviate from their authentic counterparts in implementation.

Authenticity verification of ICs is a crucial component of every entity's value proposition, and it may be subject to scrutiny if discovered by end-users. For instance, fabless design houses place substantial trust in the foundries that fabricate their designs, and they currently rely on the "golden" ICs sourced directly from the foundry. However, it is challenging for them to verify the authenticity of ICs because they must reverse-engineer the fabricated IC and compare it with their knowledge of the technology node, which is time-consuming and destroys the device in the process. Therefore, considering the time constraint issue, a transfer learning-based technique can be utilized to detect counterfeit ICs with limited data.

7.5 IC Counterfeit Detection Mechanism

To improve image classification performance while working with limited resources, a proposed scheme involves image augmentation through the creation and validation of a dataset from a pre-trained model. The model utilizes feature extractors and a CNN architecture to classify images. To achieve this, the scheme begins by implementing preprocessing layers using image transformations. The image classifier is then created by leveraging the VGG-16 model.

7.5.1 Image Augmentation with Preprocessing Layers

To enhance the diversity of the training model and improve prediction accuracy, different geometric transformations are applied to the images in the dataset containing counterfeit ICs. Training data undergoes transformations such as image rotation, height shifting, horizontal flipping, and zooming to improve model validation metrics and ensure model predictions are performed on original images rather than the augmented ones. To implement this scheme, an image class generator is first defined and instantiated to configure the image with various transformations. The transformations used include:

Rotation: Rotation augmentations are used to rotate the image right or left using degree parameters while maintaining the degree labels and applying feature standardization.

Height Shifting: This transformation technique clips off images to produce variations in the training model for generalization and avoid positional bias in the data with padding to preserve spatial dimensions.

Horizontal Flipping: As an extension to the rotation, this transformation technique helps in generalization but is not a label-preserving transformation.

Zooming: Random zooming is used to add new pixel values around the interpolated pixel values. The number of batches of samples in one epoch is specified to augment the images.

The input for image classification is the optical imaging dataset from the Florida Institute for Cybersecurity (FICS) lab, which uses two different image acquisition modalities, including the DSLR system with color normalization and Zeiss Stemi 508 Stereo Microscope. The images are preprocessed to create new ones for the training set with an optimal gradient required to update the model parameters and adapt the learning rate in the training model.

7.5.2 Pre-trained Convolution Models

The proposed approach for image classification involves the use of the VGG-16 CNN model, which was originally proposed by Karen Simonyan and Andrew Zisserman from the Visual Geometry Group Lab of Oxford University [101]. This model has achieved a remarkable accuracy of 92.7% on the ImageNet dataset, which contains 14 million images belonging to 1000 classes. The architecture of the model includes a series of convolutional layers, each with a 3x3 filter size, followed by a max-pooling layer that reduces the height and width of the image. The layers are stacked for feature mapping and equipped with rectification non-linearity, with only one Local Response Normalization (LRN) to reduce memory and computation time. The pre-trained model can be utilized by loading the model weights and architecture, and adding the weights to the respective layers using transfer learning.Figure 7.3 illustrates the VGG-16 architecture with different convolution layers and filter sizes.

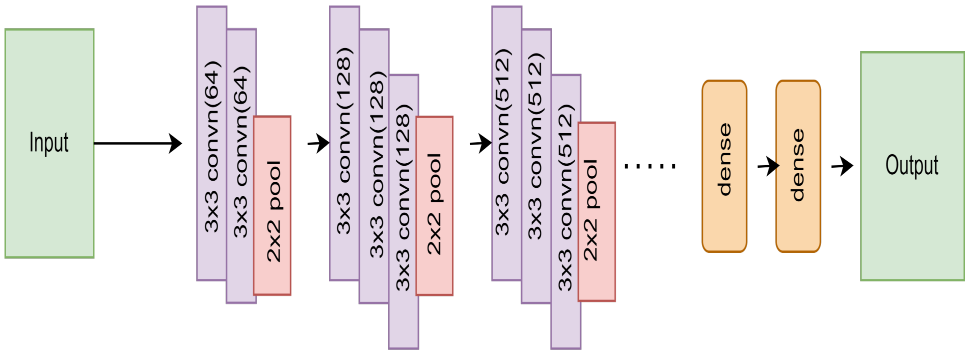


Figure 7.3: VGG-16 Architecture

To perform classification between the authentic and counterfeit ICs, the input shape along with two fully connected dense layers are specified. The model is fine-tuned with trainable parameters, and the parameters are set to false to freeze the weights of a particular layer. The model is compiled with the Adam Optimizer, which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The binary cross-entropy loss function is used for binary classification, which computes the cross-entropy between the predicted classes and the true classes. Early stopping technique is utilized to avoid overfitting by stopping the training when the model improves on a holdout validation dataset. The predict function is used to generate the classification report along with the confusion matrix, which substantially increases the depth for classification accuracy. The training data consists of high-resolution images of the front and back surfaces of different types of ICs.

7.6 Experimental Results

7.6.1 Training Data

The implementation of the detection framework on the Xilinx PYNQ FPGA board utilized the high-resolution DSLR and Zeiss Stemi 508 Stereo Microscope images, which have a resolution of 2560 x 1920 pixels. Code cells were created to train and validate the dataset, using the VGG-16 model with a split of 40% validation and 60% training data, augmented using various image techniques. Figure 7.4 represents the validated augmented images belonging to different classes. The Adam optimizer was used to minimize the loss function for 100 epochs with 3 batches per epoch for the VGG-16 model. The model comprised 19,433,793 parameters with 4 million being trainable. The VGG-16 model's deep stacked architecture accurately distinguished between features, providing reliable results.

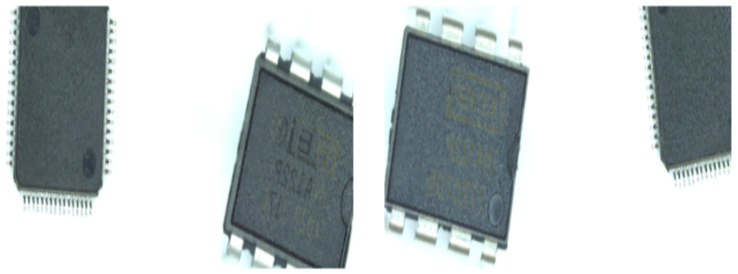


Figure 7.4: Validated Images using Image Augmentation

7.6.2 Testing Data

Using the imported dataset, predictions were based on the probability value between 0 and 1, with a binary classification of class 0 for values < 0.5 and class 1 for values greater than and equal to 0.5. For VGG-16, a learning rate of 0.0001 was assigned in the Adam optimizer. The performance metrics of the classification model included accuracy, precision, recall, and F1-score with true positive, false positive, true negative, and false negative classification. Figure 7.5 shows the Accuracy and Loss during model training and prediction for the VGG-16 model. The loss for both the training and test data is less than 1%.

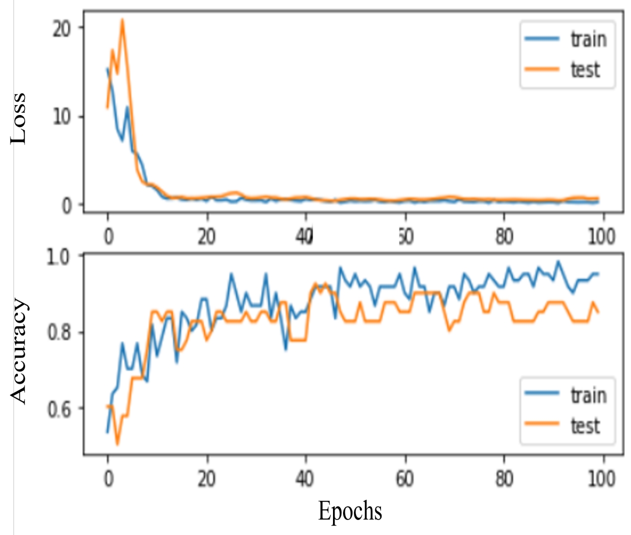


Figure 7.5: Accuracy and Loss Plot for VGG-16 model

With the trained model predictions are made for a different datasets which consists of 200 high resolution images in which 100 counterfeit and 100 authentic ICs are present for the training set and 10 counterfeit and 10 authentic ICs are present for the test holdout dataset. The model is trained with the CNN model and the classification report shows that VGG-16 performs better with 80% accuracy for the final holdout data.

The confusion matrix for the classifier performance and the VGG-16 model accuracy of 80% for 10 images are illustrated in Figure 7.6. Table 7.1 shows the classification report for the holdout dataset. The model could further enhance its accuracy with more data. The confusion matrix generated provides the true and false positives and negatives for the actual and predicted values. This model was tested using a test set to avoid retraining and demonstrated accurate classification with minimal data.

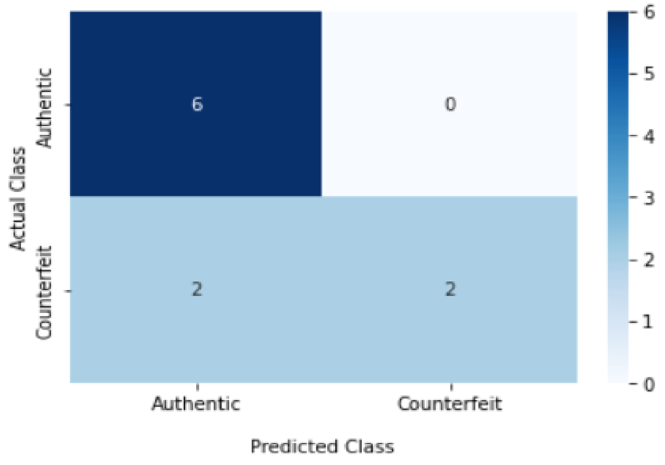


Figure 7.6: Confusion matrix for the VGG-16 model

The test images with corresponding true and mispredicted class labels are shown in Figure 7.7 where the authentic is predicted as 0 and counterfeit ICs are predicted as 1 using the binary classifier.

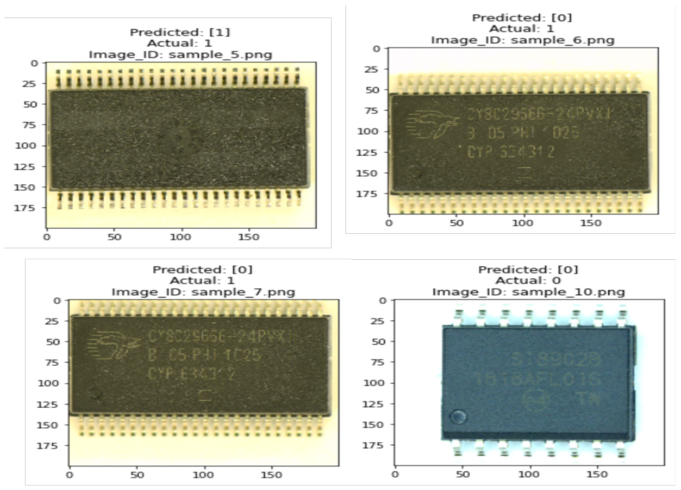


Figure 7.7: True class and predicted authentic and counterfeit images

Table 7.1: Classification Report

Model	Classes	Precision	Recall	F1score	Accuracy
VGG-16	Authentic	0.75	1.0	0.86	0.80
VGG-16	Counterfeit	1.0	0.5	0.67	0.80

7.7 Security Analysis

To identify counterfeit ICs, this study uses transfer learning technique that employs three different pre-trained models to train the prediction model. These models are pre-trained with a large set of image classification data, and their convolution layers offer optimized functions such as sparse connectivity, pooling layers, activation functions, fully connected layers, and loss functions to create a robust and generalized prediction model. The proposed system selects appropriate pre-trained models based on various factors and offers data enhancement and mapping functions to boost the performance of the model. Additionally, fine-tuning is applied at the end layers for training on a small dataset to solve the problem of insufficient data. The VGG-16 model provides 80% accuracy with limited training data, which makes it a generalized model.

The pre-trained models used in this study are all binary classification predictive models that incorporate image augmentation techniques, which increases the classifier's performance in VGG-16. The deep learning architecture model's performance is compared to the human error rate to minimize false positives when classifying images. The selected pre-trained model in this study has a lower error rate than the human error rate, making it an efficient prediction framework for IC counterfeit detection.

7.8 Conclusion

This work investigates the hardware vulnerabilities in the SoC design along with different machine learning models for SoC security validation and verification. It propsoses a IC counterfeit detection scheme based on transfer learning technique for supply chain vulnerability. This scheme contributes a counterfeit detection model with limited dataset using a pre-trained model significantly increasing the classification accuracy.

CHAPTER 8: CONCLUSION AND FUTURE WORK

The increasing use of heterogeneous SoC architectures has raised significant security concerns, particularly in relation to memory and side-channel attacks, which can compromise system security and privacy. Traditional defense mechanisms are still vulnerable to side-channel attacks, and secure measures to protect interfaces and data propagation through different channels are critical to building a resilient model consisting of on-chip security factors.

To address these concerns, a platform-based SoC model is developed using RISC-V architecture as a base, which provides a platform for custom implementation of security extensions and is an open-source ISA. The model can address different security objectives through various modules, such as Information Flow Tracking with multi granularity for better precision presented in Chapter 5 and 6 along with hardware accelerators for crypto engines which can be used for device authentication presented in Chapter 4.

Chapter 3 presents a security aware design model to protect the bitstream from modification along with multi layers of security where the bitstream is logic obfuscated during authentication by preserving the design during runtime. Isolation of the critical data provides additional security from untrusted mediums. The use of transfer learning-based IC Counterfeit detection model opens a new space to detect the vulnerabilities in supply chain by using limited dataset. Figure 8.1 shows the overall design components implemented for the SoC Secure Model.

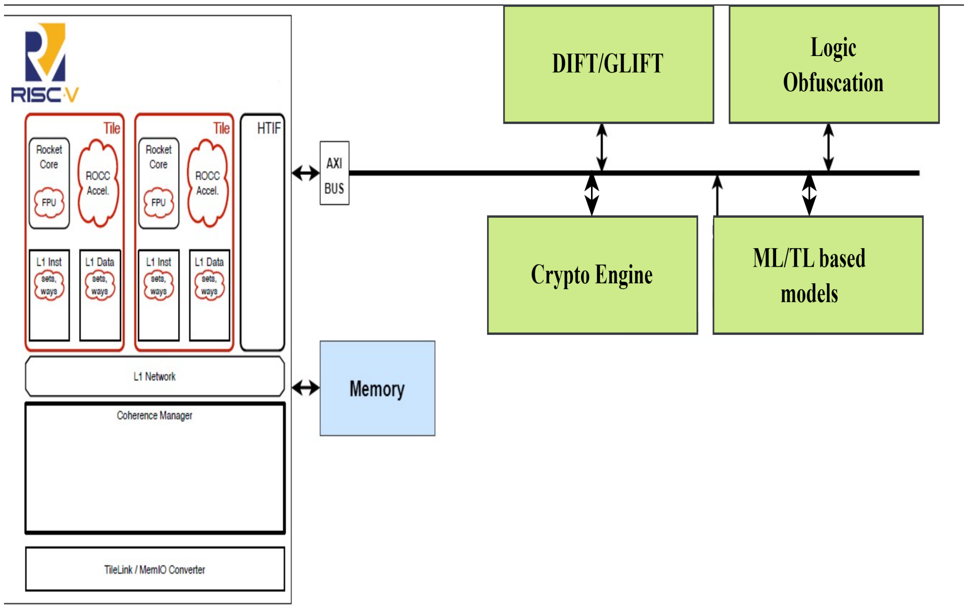


Figure 8.1: Design components implemented for the SoC Secure Model

Future work could involve further development and testing of the proposed framework using different FPGA boards and EDA tools, and exploring additional security objectives and modules to make the SoC design even more resilient to attacks. Cache-based side channel attacks which targets the hardware level to gain access to the timing information at run time obtains the memory access patterns and timing variations generated by cryptosystems. ML based detection and countermeasures for accurate detection and mitigation of these attacks can be considered as a security objective for the SoC platform assessment tool.

REFERENCES

[1] Geraldine Shirley Nicholas Y. Gui and F. Saqib, "A Survey and Analysis on SoC Platform Security in ARM, Intel and RISC-V Architecture," IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), 2020, pp. 718-721, doi: 10.1109/MWSCAS48704.2020.9184573.

[2] Casper, W.D., Papa, S.M. (2011). Root of Trust. In: van Tilborg, H.C.A., Jajodia, S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-5906-5_789

[3] https://medium.com/security-risks-in-systems-on-chip-socs/an-overview-systems-on-chips-socs-and-their-security-risks

[4] https://www.edn.com/what-it-takes-to-secure-an-soc/

[5] Swarup Bhunia and Mark Tehranipoor. 2018. Hardware Security: A Hands-on Learning Approach (1st. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[6] S. Ray, E. Peeters, M. M. Tehranipoor and S. Bhunia, "System-on-Chip Platform Security Assurance: Architecture and Validation," in Proceedings of the IEEE, vol. 106, no. 1, pp. 21-37, Jan. 2018, doi: 10.1109/JPROC.2017.2714641.

[7] A. Patel, N. Shah, D. Ramoliya and A. Nayak, "A detailed review of Cloud Security: Issues, Threats Attacks," 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2020, pp. 758-764, doi: 10.1109/ICECA49313.2020.9297572.

[8] https://www.rambus.com/blogs/hardware-root-of-trust/

[9] Ezirim, Kenneth, Wai Khoo, George Koumantaris, Raymond Law and Irippuge Milinda Perera. "Trusted Platform Module A Survey" (2014).

[10] A. Halak, M. Zwolinski and M. S. Mispan, "Overview of PUF-based hardware security solutions for the internet of things," IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS), Abu Dhabi, United Arab Emirates, 2016, pp. 1-4, doi: 10.1109/MWSCAS.2016.7870046.

[11] Aniello, L., Halak, B., Chai, P. et al. "Anti-BlUFf: towards counterfeit mitigation in IC supply chains using blockchain and PUF", Int. J. Inf. Secur. 20, 445-460 (2021). https://doi.org/10.1007/s10207-020-00513-8

[12] P. Swierczynski, M. Fyrbiak, P. Koppe and C. Paar, "FPGA Trojans Through Detecting and Weakening of Cryptographic Primitives," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 8, pp. 1236-1249, Aug. 2015, doi: 10.1109/TCAD.2015.2399455.

[13] L. Sanders, "Secure boot of Zynq-7000 all-programmable SoC," Xilinx, Appl. Note XAPP 1175 (v1.0), 2013.

[14] S. M. Trimberger and J. J. Moore, "FPGA Security: Motivations, Features, and Applications," in Proceedings of the IEEE, vol. 102, no. 8, pp. 1248-1265, Aug. 2014, doi: 10.1109/JPROC.2014.2331672.

[15] Siddiqui AS, Gui Y, Saqib F, "Secure boot for reconfigurable architectures", Cryptography 4(4):26. https:// doi. org/ 10. 3390/cryptography404.0026

[16] Siddiqui AS, Geraldine SN et al. "Multilayer camouflaged secure boot for SoCs", In: 2019 20th international workshop on microprocessor/ SoC test, security and verification (MTV), pp 56-61

[17] Pocklassery G, Che W, Saqib F, Areno M, Plusquellic J, "Self-authenticating secure boot for FPGAs", In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). IEEE, pp 221-226

[18] Leila Delshadtehrani, Sadullah Canakci, Boyou Zhou, Schuyler Eldridge, Ajay Joshi, and Manuel Egele, "PHMon: a programmable hardware monitor and its security use cases", In Proceedings of the 29th USENIX Conference on Security Symposium (SEC'20). USENIX Association, USA, Article 46, 807-824.

[19] Thomas Reinbacher, Jorg Brauer, Martin Horauer, Andreas Steininger, and Stefan Kowalewski, "Runtime verification of microcontroller binary code", Sci. Comput. Program. 80 (February, 2014), 109-129. https://doi.org/10.1016/j.scico.2012.10.015

[20] Ian Kuon, Aaron Egier, and Jonathan Rose,"Design, layout and verification of an FPGA using automated tools", In Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays (FPGA '05). Association for Computing Machinery, New York, NY, USA, 215-226. https://doi.org/10.1145/1046192.1046220

[21] Wang, Weike, Xiaobing Zhang, Qiang Hao, Zhun Zhang, Bin Xu, Haifeng Dong, Tongsheng Xia, and Xiang Wang. 2019. "Hardware-Enhanced Protection for the Runtime Data Security in Embedded Systems" Electronics 8, no. 1: 52. https://doi.org/10.3390/electronics8010052

[22] A. S. Siddiqui, G. Shirley, S. Bendre, G. Bhagwat, J. Plusquellic and F. Saqib, "Secure Design Flow of FPGA Based RISC-V Implementation," 2019 IEEE 4th International Verification and Security Workshop (IVSW), Rhodes, Greece, 2019, pp. 37-42, doi: 10.1109/IVSW.2019.8854418.

[23] K. M. Abdellatif, R. Chotin-Avot and H. Mehrez, "Protecting FPGA bitstreams using authenticated encryption," 2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS), Paris, France, 2013, pp. 1-4, doi: 10.1109/NEWCAS.2013.6573635.

[24] A. S. Siddiqui, C. C. Lee, and F. Saqib,"Hardware based protection against malwares by PUF based access control mechanism", in Midwest Symposium on Circuits and Systems, vol. 2017-Augus, 2017, pp. 1312_1315.

[25] S. D. Paul, F. Zhang, P. SLPSK, A. R. Trivedi and S. Bhunia, "RIHANN: Remote IoT Hardware Authentication With Intrinsic Identifiers," in IEEE Internet of Things Journal, vol. 9, no. 24, pp. 24615-24627, 15 Dec.15, 2022, doi: 10.1109/JIOT.2022.3195546.

[26] S. Chen, J. Xu, N. Nakka, Z. Kalbarczyk and R. K. Iyer, "Defeating memory corruption attacks via pointer taintedness detection," 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005, pp. 378-387, doi: 10.1109/DSN.2005.36.

[27] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones", ACM Trans. Comput. Syst. 32, 2, Article 5 (June 2014), 29 pages. DOI:https://doi.org/10.1145/2619091

[28] G. Venkataramani, I. Doudalis, Y. Solihin and M. Prvulovic, "FlexiTaint: A programmable accelerator for dynamic taint propagation," 2008 IEEE 14th International Symposium on High Performance Computer Architecture, 2008, pp. 173-184, doi: 10.1109/HPCA.2008.4658637.

[29] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre and G. Gogniat, "ARMHEx: A hardware extension for DIFT on ARM-based SoCs," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), 2017, pp. 1-7, doi: 10.23919/FPL.2017.8056767.

[30] Song et al., "HDFI: Hardware-Assisted Data-Flow Isolation," 2016 IEEE Symposium on Security and Privacy (SP), 2016, pp. 1-17, doi: 10.1109/SP.2016.9.

[31] Ferraiuolo, A., Mark Zhao, A. Myers and G. Suh. "HyperFlow: A Processor Architecture for Nonmalleable, Timing-Safe Information Flow Security", Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (2018): n. pag.

[32] M. M. Hossain, F. Farahmandi, M. Tehranipoor and F. Rahman, "BOFT: Exploitable Buffer Overflow Detection by Information Flow Tracking," 2021 Design, Automation Test in Europe Conference Exhibition (DATE), 2021, pp. 1126-1129, doi: 10.23919/DATE51398.2021.9474045.

[33] Y. H. Hung, B. J. Jheng, H. W. Li, W. Y. Lai, S. Mallissery and Y.S. Wu, "Mixed-mode Information Flow Tracking with Compile-time Taint Semantics Extraction and Offline Replay," 2021 IEEE Conference on Dependable and Secure Computing (DSC), Aizuwakamatsu, Fukushima, Japan, 2021, pp. 1-8, doi: 10.1109/DSC49826.2021.9346239.

[34] K. Chen, O. Arias, Q. Deng, D. Oliveira, X. Guo and Y. Jin, "FineDIFT: Fine-Grained Dynamic Information Flow Tracking for Data-Flow Integrity Using Co-processor," in IEEE Transactions on Information Forensics and Security, vol. 17, pp. 559-573, 2022, doi: 10.1109/TIFS.2022.3144868.

[35] Chen K, Guo X, Deng Q, Jin Y., "Dynamic Information Flow Tracking: Taxonomy, Challenges, and Opportunities", Micromachines (Basel). 2021 Jul 29;12(8):898. doi: 10.3390/mi12080898. PMID: 34442520; PMCID: PMC8399738.

[36] N. Sapountzis, R. Sun, X. Wei, Y. Jin, J. Crandall and D. Oliveira, "MITOS: Optimal Decisioning for the Indirect Flow Propagation Dilemma in Dynamic Information Flow Tracking Systems," 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, Singapore, 2020, pp. 1090-1100, doi: 10.1109/ICDCS47774.2020.00093.

[37] Mohit Tiwari, Hassan M.G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood, "Complete information flow tracking from the gates up", SIGARCH Comput. Archit. News 37, 1 (March 2009), 109â120. DOI:https://doi.org/10.1145/2528521.1508258

[38] Hu, W., J. Oberg, A. Irturk, Mohit Tiwari, T. Sherwood, Dejun Mu and R. Kastner, "Theoretical Fundamentals of Gate Level Information Flow Tracking", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30 (2011): 1128-1140.

[39] Wei Hu, Jason Oberg, Ali Irturk, Mohit Tiwari, Timothy Sherwood, Dejun Mu, , and Ryan Kastner. 2011, "An Improved Encoding Technique for Gate Level Information Flow Tracking", In Int. Workshop on Logic Synthesis (IWLS) (San Diego, CA, USA).

[40] W. Hu, J. Oberg, D. Mu and R. Kastner, "Simultaneous information flow security and circuit redundancy in Boolean gates," 2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2012, pp. 585-590.

[41] Ryan Kastner. 2012, "Circuit primitives for monitoring information flow and enabling redundancy", In Proceedings of the 8th international conference on Hardware and Software: verification and testing (HVC'12). Springer-Verlag, Berlin, Heidelberg. DOI:https://doi.org/10.1007/978-3-642-39611-3_6

[42] T. Le, J. Di, M. Tehranipoor and L. Wang, "Tracking data flow at gate-level through structural," 2016 International Great Lakes Symposium on VLSI (GLSVLSI), 2016, pp. 185-189, doi: 10.1145/2902961.2903040.

[43] Q. Zhang, J. He, Y. Zhao and X. Guo, "A Formal Framework for Gate- Level Information Leakage Using Z3," 2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), Kolkata, India, 2020, pp. 1-6, doi: 10.1109/AsianHOST51057.2020.9358257.

[44] S. Zhang, S. Wang, J. Wang, S. Zhou and Z. Yao, "Quantitative Analysis of Information Leakage Hardware Trojans in IP Cores," 2022 9th International Conference on Dependable Systems and Their Applications (DSA), Wulumuqi, China, 2022, pp. 431-436, doi: 10.1109/DSA56465.2022.00063.

[45] Blackstone, Jeremy, Wei Hu, Alric Althoff, Armaiti Ardeshiricham, Lu Zhang and Ryan Kastner,"A Unified Model for Gate Level Propagation Analysis", ArXiv abs/2012.02791 (2020)

[46] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, 2015, pp. 57-64, doi: 10.1109/Trustcom.2015.357.

[47] TrustZone Technology for the ARMv8-M Architecture, ARM, Cambridge, U.K., 2017. [Online].

[48] M. A. Mukhtar, M. K. Bhatti and G. Gogniat, "Architectures for Security: A comparative analysis of hardware security features in Intel SGX and ARM TrustZone," 2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE), Islamabad, Pakistan, 2019, pp. 299-304, doi: 10.1109/CCODE. 2019.8680982.

[49] Ngabonziza B, Martin D, Bailey A, Cho H, Martin S (2016),"Trust- Zone explained: architectural features and use cases", In: 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC). Pittsburgh, PA, pp 445-451. https:// doi. org/ 10. 1109/ CIC. 2016. 065

[50] Pinto S, Santos N (2019),"Demystifying arm trustZone: a comprehensive survey", ACM Comput Surv 51(6):1-36. https:// doi. org/10. 1145/ 32910 47

[51] Balisane RA, Martin A (2016), "Trusted execution environment based authentication gauge (TEEBAG)", In: Proceedings of the 2016 New Security Paradigms Workshop (NSPW â16). Association for Computing Machinery. New York, NY, USA, pp 61â67. https:// doi.org/ 10. 1145/ 30118 83. 30118 92

[52] Zhao B, Xiao Y, Huang Y, Cui X (2017)," A private user data protection mechanism in trustzone architecture based on identity authentication", Tsinghua Sci Technol 22(2):218â225. https://doi. org/ 10. 23919/ TST. 2017. 78896 43

[53] Gross M et al (2019)," Breaking trustzone memory isolation through malicious hardware on a modern FPGA-SoC", Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop

[54] Yasin M, Mazumdar B, Rajendran JJ, Sinanoglu O (2016),"SARLock: Sat attack resistant logic locking", In: 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp 236â241

[55] Yasin M, Mazumdar B, Rajendran JJ, Sinanoglu O (2017)," TTLock: Tenacious and traceless logic locking", In: 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), p 166

[56] Xie Y, Srivastava A (2018),"Anti-sat: Mitigating sat attack on logic locking",IEEE Trans Comput Aided Des Integr Circuits Syst 38(2):199-207 23.

[57] Yasin M, Sengupta A, Nabeel MT, Ashraf M, Rajendran J, Sinanoglu O (2017),"Provably-secure logic locking: From theory to practice", In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp 1601-1618

[58] Rajendran J, Pino Y, Sinanoglu O, Karri R (2012),"Logic encryption:A fault analysis perspective. In: 2012 Design, Automation and Test", in Europe Conference and Exhibition (DATE), pp 953-958

[59] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Yiwei Thomas Hou. Truspy,"Cache side-channel information leakage from the secure world on arm devices",IACR Cryptology ePrint Archive, 2016:980, 2016.

[60] E. M. Benhani, C. Marchand, A. Aubert and L. Bossuet, "On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC," 2017 30th IEEE International System-on-Chip Conference (SOCC), Munich, 2017, pp. 108-113, doi: 10.1109/SOCC.2017.8226018.

[61] Intel Corporation, âSoftware Guard Extensions Programming Referenceâ, 329298-002US October 2014.

[62] Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S. (2017),"Malware Guard Extension: Using SGX to Conceal Cache Attacks", Lecture Notes in Computer Science, 3-24. doi:10.1007/978-3-319-60876-1_1

[63] Common Evaluation Platform Repository Available: https://github.com/mitll/CEP.

[64] Q. Ge, Y. Yarom, and G. Heiser, "No security without time protection: we need a new hardware-software contract", In Asia-Pacific Workshop on Systems(APSys), 2018.

[65] https://content.riscv.org/wp-content/uploads/2019/03/15.05-RISC-V-Security-Multizone-v-TrustZone-3-12-19.pdf

[66] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovia, "The RISC-V instruction set manual, volume I: User-level ISA, version 2.0," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-54, May 2014.

[67] Calvin Deutschbein, Andres Meza, Francesco Restuccia, Ryan Kastner, and Cynthia Sturton, "Isadora: Automated Information Flow Property Generation for

Hardware Designs", In Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security (ASHES '21). Association for Computing Machinery, New York, NY, USA, 5-15,2021. https://doi.org/10.1145/3474376.3487286

[68] T. Liu, G. Shi, L. Chen, F. Zhang, Y. Yang and J. Zhang, "TMDFI: Tagged Memory Assisted for Fine-Grained Data-Flow Integrity Towards Embedded Systems Against Software Exploitation," 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), New York, NY, USA, 2018, pp. 545-550, doi: 10.1109/TrustCom/BigDataSE.2018.00083.

[69] Weiser, Samuel, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard and Ahmad-Reza Sadeghi,"TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V", Proceedings 2019 Network and Distributed System Security Symposium (2019): n. pag.

[70] C. Palmiero, G. Di Guglielmo, L. Lavagno and L. P. Carloni, "Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications," 2018 IEEE High Performance extreme Computing Conference (HPEC), Waltham, MA, USA, 2018, pp. 1-7, doi: 10.1109/HPEC.2018.8547578.

[71] Geraldine Shirley Nicholas, Siddiqui, A.S., Joseph, S.R. et al.,"A Secure Boot Framework with Multi-security Features and Logic-Locking Applications for Reconfigurable Logic", J Hardw Syst Secur 5, 260â268 (2021). https://doi.org/10.1007/s41635-021-00123-3

[72] Xilinx and Inc. (2017) Using encryption and authentication to secure an ultraScale/ultraScale+ FPGA bitstream application note (XAPP1267), XAPP1267

[73] Zuo X, Liu W (2007),"TPM based key backup and recovery", Int Conf Mach Learn Cybern 2007:2164-2167. https:// doi. org/ 10.1109/ ICMLC. 2007. 43705 03

[74] Shin J, Kim Y, Park W, Park C (2012)," DFCloud: A TPM-based secure data access control method of cloud storage in mobile devices",In: 4th IEEE International Conference on Cloud Computing

[75] Rajendran J, Sam M, Sinanoglu O, Karri R (2013),"Security analysis of integrated circuit camouflaging", In: ACM/SIGSAC Conference on Computer and Communications Security, pp 709-720 3.

[76] Jarvis RW, McIntyre MG (2007),"Split manufacturing method for advanced semiconductor circuits", US Patent 7(195):931

[77] Chakraborty RS, Bhunia S (2009),"HARPOON: an obfuscationbased SoC design methodology for hardware protection", IEEE Trans Comput Aided Des Integr Circuits Syst 28(10):1493-1502

[78] Jacob, Nisha, Johann Heyszl, Andreas Zankl, Carsten Rolfes and Georg Sigl, "How to Break Secure Boot on FPGA SoCs Through Malicious Hardware", Workshop on Cryptographic Hardware and Embedded Systems (2017).

[79] Aarestad J, Ortiz P, Acharyya D, Plusquellic J (2013), "HELP:A hardware-embedded delay PUF",IEEE Des Test 30(2):17-25.https:// doi. org/ 10. 1109/ MDT. 2013. 22474 59

[80] Berkeley Logic Synthesis and Verification Group (2005) ABC: a system for sequential synthesis and verification.

[81] Hyung Ki Lee and Dong Sam Ha (1996), "HOPE: an efficient parallel fault simulator for synchronous sequential circuits", IEEE Trans Comput Aided Des Integr Circuits Syst 15(9):1048-1058. https:// doi. org/ 10. 1109/ 43. 536711

[82] Hansen MC, Yalcin H, Hayes JP (1999),"Unveiling the iscas-85 benchmarks: A case study in reverse engineering", IEEE Des Test Comput 16(3):72-80

[83] Rajendran J et al (2015),"Fault analysis-based logic encryption", IEEE Trans Comput 64(2):410â424. https:// doi. org/ 10. 1109/ TC. 2013. 193

[84] S. M. Soliman, B. Magdy and M. A. Abd El Ghany, "Efficient implementation of the AES algorithm for security applications," 2016 29th IEEE International System-on-Chip Conference (SOCC), Seattle, WA, USA, 2016, pp. 206-210, doi: 10.1109/SOCC.2016.7905466.

[85] J. Day, Z. Zhao and M. Ma, "Detecting Return-to-libc Buffer Overflow Attacks Using Network Intrusion Detection Systems," 2010 Fourth International Conference on Digital Society, 2010, pp. 172-177, doi: 10.1109/ICDS.2010.37.

[86] Geraldine Shirley Nicholas et al., "Hardware Secure Execution and Simulation Model Correlation using IFT on RISC-V", Proceedings of the 2021 on Great Lakes Symposium on VLSI. Association for Computing Machinery, New York, NY, USA, 409-414. https://doi.org/10.1145/3453688.3461517

[87] Geraldine Shirley Nicholas et al., "Multi granular level-based IFT model for RISC-V", Second iiScience International Conference 2021: Recent Advances in Photonics and Physical Sciences.https://doi.org/10.1117/12.2601036

[88] G. Sumathi, L. Srivani, D. T. Murthy, K. Madhusoodanan, and S. A. V. S. Murty, "A review on HT attacks in PLD and ASIC designs with potential defence solutions", IETE Tech. Rev., vol. 35, no. 1, pp. 64-77, 2018.

[89] J. Zhang and G. Qu, "Recent attacks and defenses on FPGA-based systems", ACM Trans. Reconfigurable Technol. Syst., vol. 12, no. 3, pp. 1-24, 2019.

[90] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware trojan attacks: Recent advances and challenges", IEEE Access, vol. 8, pp. 10796-10826, 2020

[91] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A massively parallel FPGA-based coprocessor for support vector machines", in Proc. 17th IEEE Symp. Field Program. Custom Comput. Mach., Apr. 2009, pp. 115-122.

[92] M. Xue, R. Bian, W. Liu, and J. Wang, "Defeating untrustworthy testing parties: A novel hybrid clustering ensemble based golden models-free hardware trojan detection method", IEEE Access, vol. 7, pp. 5124-5140, 2019.

[93] Y. Lyu and P. Mishra, "Automated trigger activation by repeated maximal clique sampling", in Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC), Jan. 2020, pp. 482-487.

[94] R. Chakraborty, F. Wolff, and S. Paul, "MERO: A statistical approach for hardware trojan detection", in Proc. CHES, 2009, pp. 396-410.

[95] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Hardware Trojans classification for gate-level netlists using multi-layer neural networks", in Proc. IEEE 23rd Int. Symp. On-Line Test. Robust Syst. Design (IOLTS), Jul. 2017, pp. 227-232

[96] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against microarchitectural side-channel-attacks", IACR Cryptol. ePrint Arch., Tech. Rep. 2017/564, 2017, p. 564.

[97] L. Zhang, D. Mu, W. Hu, and Y. Tai, "Machine-learning-based sidechannel leakage detection in electronic system-level synthesis", IEEE Netw., vol. 34, no. 3, pp. 44-49, May/Jun. 2020.

[98] N. Fern and K.-T. Cheng, "Pre-silicon formal verification of JTAG instruction opcodes for security", in Proc. IEEE Int. Test Conf. (ITC), Oct./Nov. 2018, pp. 1-9

[99] K. Huang, J. M. Carulli, and Y. Makris, "Parametric counterfeit IC detection via support vector machines", in Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT), Oct. 2012, pp. 7-12.

[100] E.R. Zhou, S.Q. Li, J.H. Chen, L. Ni, Z.X. Zhao, and J. Li, "A novel detection method for hardware trojan in third party IP cores", in Proc. Int. Conf. Inf. Syst. Artif. Intell. (ISAI), Jun. 2016, pp. 528-532.

[101] F. Zhuang et al., "A Comprehensive Survey on Transfer Learning," in Proceedings of the IEEE, vol. 109, no. 1, pp. 43-76, Jan. 2021, doi: 10.1109/JPROC.2020.3004555.

[102] C. M. Bhure, Geraldine Shirley Nicholas, S. Ghosh, Y. Zhong and F. Saqib, "Automated Transfer Learning Model for Counterfeit IC Detection," 2022 IEEE Physical Assurance and Inspection of Electronics (PAINE), Huntsville, AL, USA, 2022, pp. 1-7, doi: 10.1109/PAINE56030.2022.10014980.

[103] Simonyan, Karen and Zisserman, Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition." CoRR abs/1409.1556 (2014)