Trusted SoC Realization for Remote Dynamic IP Integration

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS (Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von

M. Sc. Nadir Muhammad Khan
geboren in Peshawar (Pakistan)

Tag der mündlichen Prüfung: 30.11.2021
Hauptreferent: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent: Prof. Dr.-Ing. Jörg Henkel

Abstract

Nowadays, field-programmable gate arrays (FPGAs) offer enormous computational power and flexibility. Furthermore, they are often integrated on a single chip with embedded multi-core processors, DSP engines, and memory controllers. This makes them suitable for large and complex applications. Simultaneously, the progress made in the field of high-level synthesis and availability of standardized interfaces (such as Advanced eXtensible Interface 4) led to the development of specialized and novel functionalities by design houses. All this created a need for outsourcing or licensing FPGA intellectual properties (IPs). A pay-per-use IP licensing model where these IPs are protected from all the market participants will benefit the developers of the IPs. Also, FPGA system developers are usually small to medium enterprises that can benefit from it in terms of time-to-market and per-unit cost.

In academia and industry, several IP licensing models and protection solutions are available that can be deployed; however, they are prone to multiple security challenges. In some cases, the proposed security measures caused unnecessary resource overhead and restrictions for the system developers, i.e., they are restricted from using the essential features of their device. Furthermore, they do not address two functional challenges: the floorplanning of the IP on the programmable logic (PL) and the generation of IP's end-product (bitstream) independent of the overall design.

In this work, a pay-per-use licensing scheme is proposed and realized using a *security framework (SFW)* to address all these challenges. The scheme is pragmatic, less restrictive for the system developers, and offers security against IP theft. Furthermore, measures are taken to protect the system from an IP that has malicious circuitry in it. The SFW comprises a trusted operating system (OS), a rich OS, several supporting components (e.g., TrustZone logic, side-channel attack (SCA) resistant decryption engine), and software components, e.g., bitstream analysis. A device running the SFW can be considered a trusted device that can directly communicate with a repository or an IP core developer to acquire the IP in an encrypted form. The decryption and authentication

of the IP happen on the device, which reduces the attack surface and makes them less prone to IP theft attacks. Also, the plaintext IP is stored in a protected memory of the trusted OS. The plaintext IP is then analyzed and only configured on the PL if it is authentic and has no malicious circuitry. The bitstream analysis functionality and the SFW subcomponents make it possible to partition the PL resources into secure and non-secure ones, i.e., extending the trusted execution environment (TEE) concept to the PL. This is the first work so far that has extended the TEE to the PL.

The aforementioned SCA-resistant decryption engine is an advanced encryption algorithm's implementation that is modified to resist electromagnetic and power consumption leakages. The protected design has two countermeasures where the first one supports implementation diversity and moving target defense, while the second one only supports implementation diversity. These countermeasures are scalable even at run-time. The evaluation of these countermeasures also includes scalability's effect on the area overhead and security strength.

In addition, the earlier mentioned functional challenge of *floorplanning IPs* is addressed by proposing mixed-integer linear programming based fine-grained *Automatic Floorplanner*, which targets recent, larger, and complex FPGA devices. The floorplanner maps a set of IPs on the FPGA by creating precise reconfigurable regions. This maximizes the remaining available resources for the overall design. The second functional challenge is that existing tools do not provide a flow to generate IPs in a standalone environment. The challenge is addressed by proposing an independent IP generation flow. This flow can be used by the market participants to generate IPs of a design independent of the overall design without compromising IPs' compatibility with the overall design.

Zusammenfassung

Heutzutage bieten field-programmable gate arrays (FPGAs) enorme Rechenleistung und Flexibilität. Zudem sind sie oft auf einem einzigen Chip mit eingebetteten Multicore-Prozessoren, DSP-Engines und Speicher-Controllern integriert. Dadurch sind sie für große und komplexe Anwendungen geeignet. Gleichzeitig führten die Fortschritte auf dem Gebiet der High-Level-Synthese und die Verfügbarkeit standardisierter Schnittstellen (wie etwa das Advanced eXtensible Interface 4) zur Entwicklung spezialisierter und neuartiger Funktionalitäten durch Designhäuser. All dies schuf einen Bedarf für ein Outsourcing der Entwicklung oder die Lizenzierung von FPGA-IPs (Intellectual Property). Ein Pay-per-Use IP-Lizenzierungsmodell, bei dem diese IPs vor allen Marktteilnehmern geschützt sind, kommt den Entwicklern der IPs zugute. Außerdem handelt es sich bei den Entwicklern von FPGA-Systemen in der Regel um kleine bis mittlere Unternehmen, die in Bezug auf die Markteinführungszeit und die Kosten pro Einheit von einem solchen Lizenzierungsmodell profitieren können.

Im akademischen Bereich und in der Industrie gibt es mehrere IP-Lizenzierungsmodelle und Schutzlösungen, die eingesetzt werden können, die jedoch mit zahlreichen Sicherheitsproblemen behaftet sind. In einigen Fällen verursachen die vorgeschlagenen Sicherheitsmaßnahmen einen unnötigen Ressourcenaufwand und Einschränkungen für die Systementwickler, d. h., sie können wesentliche Funktionen ihres Geräts nicht nutzen. Darüber hinaus lassen sie zwei funktionale Herausforderungen außer Acht: das Floorplanning der IP auf der programmierbaren Logik (PL) und die Generierung des Endprodukts der IP (Bitstream) unabhängig vom Gesamtdesign.

In dieser Arbeit wird ein Pay-per-Use-Lizenzierungsschema vorgeschlagen und unter Verwendung eines *security framework (SFW)* realisiert, um all diese Herausforderungen anzugehen. Das vorgestellte Schema ist pragmatisch, weniger restriktiv für Systementwickler und bietet Sicherheit gegen IP-Diebstahl. Darüber hinaus werden Maßnahmen ergriffen, um das System vor einem IP zu schützen, das bösartige Schaltkreise enthält. Das „Secure Framework“ umfasst ein vertrauenswürdiges Betriebssystem, ein

reichhaltiges Betriebssystem, mehrere unterstützende Komponenten (z. B. TrustZone-Logik, gegen Seitenkanalangriffe (SCA) resistente Entschlüsselungsschaltungen) und Softwarekomponenten, z. B. für die Bitstromanalyse. Ein Gerät, auf dem das SFW läuft, kann als vertrauenswürdiges Gerät betrachtet werden, das direkt mit einem Repository oder einem IP-Core-Entwickler kommunizieren kann, um IPs in verschlüsselter Form zu erwerben. Die Entschlüsselung und Authentifizierung des IPs erfolgt auf dem Gerät, was die Angriffsfläche verringert und es weniger anfällig für IP-Diebstahl macht. Außerdem werden Klartext-IPs in einem geschützten Speicher des vertrauenswürdigen Betriebssystems abgelegt. Das Klartext-IP wird dann analysiert und nur dann auf der programmierbaren Logik konfiguriert, wenn es authentisch ist und keine bösartigen Schaltungen enthält. Die Bitstrom-Analysefunktionalität und die SFW-Unterkomponenten ermöglichen die Partitionierung der PL-Ressourcen in sichere und unsichere Ressourcen, d. h. die Erweiterung des Konzepts der vertrauenswürdigen Ausführungsumgebung (TEE) auf die PL. Dies ist die erste Arbeit, die das TEE-Konzept auf die programmierbare Logik ausweitet.

Bei der oben erwähnten SCA-resistenten Entschlüsselungsschaltung handelt es sich um die Implementierung des Advanced Encryption Standard, der so modifiziert wurde, dass er gegen elektromagnetische und stromverbrauchsbedingte Leckagen resistent ist. Das geschützte Design verfügt über zwei Gegenmaßnahmen, wobei die erste auf einer Vielzahl unterschiedler Implementierungsvarianten und veränderlichen Zielpositionen bei der Konfiguration basiert, während die zweite nur unterschiedliche Implementierungsvarianten verwendet. Diese Gegenmaßnahmen sind auch während der Laufzeit skalierbar. Bei der Bewertung werden auch die Auswirkungen der Skalierbarkeit auf den Flächenbedarf und die Sicherheitsstärke berücksichtigt.

Darüber hinaus wird die zuvor erwähnte funktionale Herausforderung des IP Floorplanning durch den Vorschlag eines feinkörnigen *Automatic Floorplanners* angegangen, der auf gemischt-ganzzahliger linearer Programmierung basiert und aktuelle FPGA-Generationen mit größeren und komplexen Bausteine unterstützt. Der Floorplanner bildet eine Reihe von IPs auf dem FPGA ab, indem er präzise rekonfigurierbare Regionen schafft. Dadurch werden die verbleibenden verfügbaren Ressourcen für das Gesamtdesign maximiert. Die zweite funktionale Herausforderung besteht darin, dass die vorhandenen Tools keine native Funktionalität zur Erzeugung von IPs in einer eigenständigen Umgebung bieten. Diese Herausforderung wird durch den Vorschlag eines unabhängigen IP-Generierungsansatzes angegangen. Dieser Ansatz kann von den Marktteilnehmern verwendet werden, um IPs eines Entwurfs unabhängig vom Gesamtentwurf zu generieren, ohne die Kompatibilität der IPs mit dem Gesamtentwurf zu beeinträchtigen.

Preface

The presented work has been accomplished while I was a research associate at the FZI Research Center for Information Technology, from February 2017 to September 2021. During this period, I was also a Ph.D. student at the Institute for Information Processing Technologies (ITIV) at the Karlsruhe Institute of Technology (KIT).

From earlier on, my interests were to gain knowledge and work in digital design and security. I was fortunate enough to work as a digital designer in the industry for a couple of years. The opportunity of researching the security of embedded systems showed up at the right time, and I am happy to have availed it. I am thankful to Prof. Dr-Ing. Dr. h. c. Jürgen Becker for offering this opportunity and believing in my capabilities. He also deserves sincere gratitude for his guidance and support that enabled me to pursue the research work at his institute. In his supervision, the most appreciatable traits were the freedom of exploring the research topics and the trust in the direction of the research I wanted to follow.

I also thank Prof. Dr.-Ing. Jörg Henkel from the Chair for Embedded Systems (CES) at KIT for the discussions, questions, feedback, and accepting the invitation to be one of my reviewers. I also owe my gratitude to Dr. Lars Bauer from CES for reviewing the thesis and offering sound advice throughout the work. I am pleased to have worked with them on several topics.

Furthermore, I thank the colleagues at FZI, ITIV, and other institutes for their friendship and support. I am privileged to have worked with these competent people and appreciate the exciting discussions, questions, ideas, and feedback. I want to mention some of them who made this journey pleasant and interesting (in no particular order): Arthur Silitonga, Bo Liu, Brian Pachideh, Carsten Tradowsky, Jorge Castro-Godínez, Manuel Haerdle, Thomas Bruckschloegl, Sven Nitzsche, Stefan Schatz, and Victor Pazmino. While working at FZI, I had the opportunity to supervise several students. Their hard work and dedication also deserve special thanks as they helped achieve the

goals I had set.

I want to thank my parents, Nisar Muhammad Khan and Parveen Nisar, for always supporting my decisions and motivating me to pursue the doctorate. I also want to thank my wife, Noreen Wali, for her support and care. She stood with me through thick and thin and was always there to celebrate all our achievements. Last but not least, I want to mention my two years old son Ibrahim Khan, who brought joy to our lives.
...
Karlsruhe, den December 15, 2021
M. Sc. Nadir Muhammad Khan

Contents

1 Introduction

Field programmable gate arrays (FPGAs) are general-purpose semiconductor devices that can be (re-)programmed after manufacturing. Among the programmable logic devices, they offer the highest configurable resources and can be used to realize more complex designs. They evolved from having fewer simple configurable logic blocks (CLBs), and programmable interconnects to complex chips combining heterogeneous configurable resources such as CLBs, blocks of random access memory (BRAMs), digital signal processing (DSP) blocks, etc. Furthermore, their variants are available where they are integrated with embedded multi-core processors, DSP engines, and memory controllers, commonly known as programmable system-on-chip (SoC) FPGAs. Few examples of their usage are accelerating algorithms, digital signal/image processing, network infrastructures, ASIC prototyping, aerospace, defense, consumer electronics, industrial motor control, scientific instruments, and security systems [137].

Since these devices offer enormous computational power and flexibility, they are used to implement large and complex systems. Also, their support of the dynamic partial reconfiguration (DPR) feature enables them even to host larger designs. Using DPR, a library of functionalities can be provided to the device via a non-volatile memory (NVM) and programmed dynamically over an existing functionality that is no longer in use. However, the development of such systems or applications often requires significant effort and competencies in several disciplines, e.g., machine learning, cryptography, computer vision. To overcome these challenges, system developers can outsource their system's sub-functionalities to specialized third parties. Similarly, FPGA design of novel algorithms or efficient implementation of existing ones can be licensed from specialized third parties. This need to outsource or license designs/sub-functionalities created an FPGA intellectual property (IP) market ecosystem. Figure 1.1 shows an SoC FPGA, where IPs of specialized applications are integrated. For example, the developer of this system can license FPGA IPs of automotive or computer vision applications.

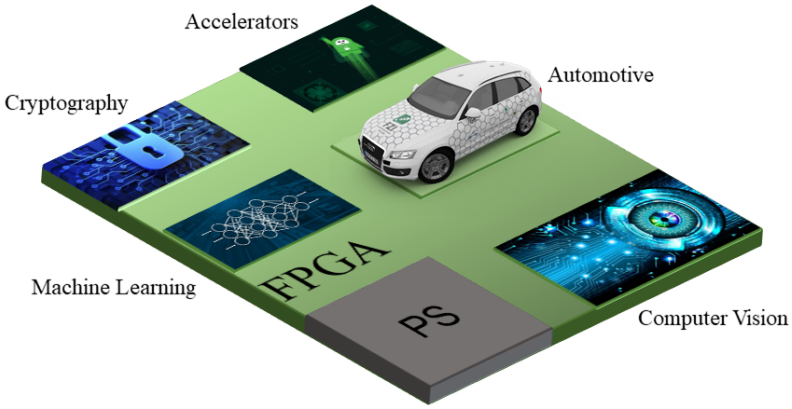

Figure 1.1: IPs of specialized applications configured on an SoC FPGA

Furthermore, these designs can be implemented at higher abstraction levels using high-level synthesis tools [92] as they have improved significantly over the years. The availability of standardized IP interfaces (such as Advanced eXtensible Interface 4 (AXI4) [125]) and DPR-like features also benefit the FPGA market by making the development and integration of FPGA designs easier.

An essential aspect of the FPGA IP market is considering the security of these devices, which can be viewed from two perspectives. Firstly, these devices are increasingly used in applications that need security against adversaries interested in bypassing the security of such applications. Secondly, the devices are used to implement sophisticated and complex systems that require significant investment, which must be protected against piracy [88], reverse engineering [11, 95], or tampering. To address the security threats from both viewpoints, the device vendors provide several protection mechanisms. However, for the second case, the focus is mainly on protecting the system developer's IPs. This makes the devices vulnerable in cases where system developers are licensing IPs from other parties. In terms of security, this work is focused on highlighting these vulnerabilities and offer solutions to protect third party IPs. In addition, security threats to the system caused by these IPs are also considered and countered.

Like this work, others have developed tools, flows, and proposed licensing models to protect the assets of individual participants of the FPGA IP market. However, they have not addressed several functional and security challenges. To explain them and their solutions, it is important first to present the participants of the FPGA IP market, types of IP licensing models, and delivery format of the FPGA-based IPs. All these topics are

presented below in the same order as their mention. Then, challenges are outlined, and the chapter concludes with the contributions of the thesis.

1.1 Participants of the FPGA IP Market

The participants of the FPGA IP market are introduced here, and their names are kept the same as are presented in other related work [25, 174, 176, 82, 157].

- **Hardware Manufacturer:** It is a semiconductor foundry that manufactures integrated circuits (ICs). There are companies, such as Intel, that do both design and manufacturing of ICs. However, the complexity and cost aspect of manufacturing led to the existence of companies that only do the design. They are known as fab-less semiconductor companies, e.g., Xilinx. Similarly, there are Pure Play foundries that only do the manufacturing of ICs, e.g., Taiwan Semiconductor Manufacturing Company.
- **FPGA Vendor (FV):** They design and sell FPGA and SoC chips. They offer families of products with varying sizes of programmable logic (PL) and hard-wired functionalities for security and flexibility.
- **IP Core Vendor (CV):** These are design houses that provide IP cores. Their specialty can be an efficient implementation of algorithms or a novel algorithm for solving a problem.
- **System Developer (SD):** These participants are the consumers of the IP cores designed by CVs. SDs provide a complete solution or product, where parts can be outsourced to IP core vendors (CVs).
- **Trusted Third Party (TTP):** It is a role that can be played by a third party, which supports the process of IP licensing as a neutral and trustworthy entity. Their service can include managing encryption keys as well as IP or device registration.

1.2 Types of IP Licensing Models

The IP licensing model can be either perpetual (one-time) or non-perpetual. The non-perpetual licensing can be per-use or periodic, or a combination of both. In software, the discussion of perpetual versus pay-per-use licensing started quite earlier, which suggests that the latter will lead to a lower cost without a large up-front payment [27]. Furthermore, it leads to a more significant investment in product development under most conditions, which results in higher software quality [22]. These findings are

also applicable to FPGA-based IPs because they are similar to software IPs from a delivery and utilization perspective. For example, both can be delivered digitally via a communication channel as their implementation's end-product is in digital form.

Both types of IPs are implemented using programming languages and are processed by tools to generate the end-product that can function on a device. The register transfer level (RTL) description of an FPGA IP can be implemented using hardware description languages (HDLs) such as Verilog, VHDL, or System-Verilog. Then, the implementation is processed by the FPGA vendor (FV) tools to generate a device-specific bitstream (FPGA configuration data). Similarly, a software IP can be implemented using programming languages (e.g., C/C++ or Java). The implementation is then processed by several tools (compiler, assembler, linker, etc.) to generate an application executable.

A perpetual IP licensing model for FPGA-based IPs will lead to a large up-front payment. Usually, FPGA applications are low volume, and paying higher fees for IPs will result in a higher per-unit cost, discouraging SDs from licensing IPs under such models. On the other hand, a pay-per-use licensing model will result in lower per-unit costs. Furthermore, the continued revenue from the licensed IP to the CV creates an incentive to invest more into the quality of their IP. In conclusion, the pay-per-use model will benefit the FPGA IP market and its participants. However, it is essential that the model is simple, feasible, and provides security assurances to the CVs that their IP will not be overused [88], reverse engineered [11, 95], or sold to another party by the system developer (SD). Also, SDs would require assurances that the licensed IP is authentic and free of malicious behavior.

1.3 Delivery Format

For FPGAs, an IP can be delivered as design files (RTL description) or configuration data, commonly known as bitstream. The RTL description is implemented using HDLs and then processed by the FPGA vendor (FV) tool to generate a device-specific bitstream. Both forms need to be encrypted before delivery to avoid IP theft attacks [88], e.g., reverse engineering and cloning. However, encrypted RTL-based IPs would require support from the FV tools to decrypt, integrate, and process them to generate their bitstreams. In addition, the generated bitstreams need to be in an encrypted form for protection. All this indicates a substantial effort in developing these features in the FV tools, which means IP core vendors (CVs) would require to share their revenue with the FV and trust them with their IPs. Xilinx's SignOnce IP Licensing [136] is an example of one such approach. Alternatively, open-source or tampered FV tools can be

used to avoid these shortcomings; however, they are often hard to use (lack of support) and require continued maintenance. Also, companies, specifically CVs, would avoid using tampered and/or non-certified tools because they might have a backdoor that can be used to access the plain-text IPs. Another major drawback of using encrypted RTL-based IPs is that they are processed with FV tools on a workstation, which leads to a large attack surface that includes but is not limited to the FV tool, operating system, and memories.

Alternatively, encrypted bitstream-based IPs (IP_{EncBit}) can be decrypted on the FPGA device and do not require further processing, which reduces the attack surface and makes them less prone to IP theft attacks. Physical attacks [175, 94] on the device can be used to extract their plain-text form, but these attacks are also valid for the encrypted bitstream generated for RTL-based IPs. Therefore, like most other IP licensing schemes [25, 42, 176, 82, 157, 158], this work focuses on distributing IP_{EncBit}.

1.4 FPGA IP Market Challenges

As argued in Section 1.2, non-perpetual or pay-per-use licensing is the approach that benefits the FPGA IP market in terms of IP quality, a steady stream of revenue for CVs, and affordability for SMEs. It is also established, in Section 1.3, that delivering IP_{EncBit} offers the highest level of security against IP theft or overuse. Of course, only encrypting IP will not solve all the security issues, but it is the starting point. Since these are the obvious choices, a large number of proposed solutions are based on them.

The primary goal of IP licensing models is securing IPs, and therefore the existing solutions are focused on it. However, they have ignored other challenges such as generation of IPs independent of the overall FPGA design, floorplanning of the IPs on the FPGA, or detecting malicious behavior in an IP to protect SD's design. In addition, some security and feasibility challenges are also not addressed by these solutions.

For an overview, all these challenges are presented below.

1. **Side-Channel Attacks:** In almost all the IP licensing models, decryption engines on the device decrypt the IP_{EncBit}. These engines implement cryptographic algorithms that are, in theory, secure against mathematical attacks. However, their implementations suffer from physical attacks that are commonly known as side-channel attacks (SCAs). Using SCAs, an adversary can extract the secret key used in the decryption process and can, therefore, access the IP in plaintext form ($IP_{PlainBit}$) (see Section 4).

2. **Manual Floorplanning:** As mentioned in Section 1.3, licensed IPs must be delivered as encrypted bitstreams. The first step in generating an IP's bitstream would be its floorplanning on the PL. However, existing FPGA vendor tools only offer manual floorplanning of IPs on the PL. Manually placing IPs on the PL can lead to inefficient utilization of the PL resources because modern FPGAs, unlike initial ones, have an irregular distribution of heterogeneous resources such as CLBs, BRAMs, DSPs, etc. Additionally, designs (IPs) are often realized using more than one type of resource.

3. **Standalone Generation of Design's Bitstream:** DPR feature, supported by major FVs (See Section 2.1.4), can generate a bitstream of a sub-design specific to a PL region, which can be afterward configured on that PL region using configuration interfaces. However, the DPR flow follows an incremental design methodology, which means IPs can only be realized after the static design (see Figure 2.2) is implemented. The static design is like a stencil, where the missing pieces are IPs. Without it, an IP cannot be implemented, and therefore its end-product cannot be generated. Using the DPR feature means that CVs would require access to the SD's proprietary design, which in itself is a security threat to SD's design. This is the second functional challenge that is not addressed by any of the existing IP licensing models.

4. **Readback Attack:** The next challenge to the FPGA IP market is the Readback attack from which all existing solutions suffer. Readback is a debug feature that can read out the configuration data even if the device's security features are enabled. This way, an adversary can access the plaintext IP ($IP_{PlainBit}$). The attack is explained in detail in Section 3.1.4.

5. **Malicious IP:** Another critical issue is the existence of malicious functionalities in IPs. Such malicious IPs can impact the system by changing its functionality, degrade performance or cause leakage of secret information. The insertion of malicious functionality by man-in-the-middle can be avoided by delivering IPs in an authenticated encrypted form. However, it cannot be ensured that the CV has not inserted such functionality. A malicious functionality can be a Trojan or an attempt to tamper other PL regions that are not reserved for the IP (see Section 3.3).

6. **Relying on TTP:** Some of the mentioned challenges can be mitigated by introducing a trusted third party (TTP) that mediates between CVs and SDs by managing secret keys or confidential data. Most of the existing licensing models rely on a TTP; however, the degree to which each party can rely on the TTP can also cause

security problems. In most cases, they have or can easily have IP's bitstream in plaintext form.

7. **Restricting Access of SD's to their Device:** Some of the existing solutions reduce the degree of trust on the TTP by proposing countermeasures like core installation modules (CIMs) [25, 42, 54, 82, 157, 158] or restricting SD's access to the processing system (PS) [39]. However, this leads to the blockage of PL or PS resources used by their owner (SDs), which is also not a favorable outcome.

8. **Hardware Modification:** Some older licensing models [41, 42, 120] require modification of FPGA devices to carry out their schemes. Such modifications make a scheme inapplicable for available products and require the development of new devices. In addition, they prevent the adoption of schemes by the industry.

1.5 Contributions and Outline

The main contributions of this thesis are listed below. They address all the challenges presented in Section 1.4.

- The first challenge to the FPGA IP market is *side-channel attacks* that are countered by proposing two countermeasures. They are based on the moving target and implementation diversity concepts. Individual contributions specific to these countermeasures are:
  - A complete implementation of the countermeasures is done and presented. An automated flow is implemented that generates different scaled variants of the countermeasures. Furthermore, the implementation diversity part of the countermeasures can be scaled up/down dynamically (see Section 4.3.7). This makes the design easily adaptable to different scenarios and is a significant improvement over other similar work.
  - The implementation is realized and evaluated on the Xilinx Zynq UltraScale+ MPSoC ZCU102.
  - Scaled variants of both countermeasures are evaluated individually and combined to create a more secure system. Furthermore, the effect of scalability on the resource overhead and security strength is presented.

  Details of the proposed countermeasures, their implementation, and evaluation are presented in Chapter 4.

- The second challenge, *manual floorplanning*, is addressed by implementing mixed-integer linear programming (MILP) based fine-grained Automatic Floorplanner,

which targets recent, larger, and complex FPGA devices, e.g., Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC. The floorplanner maps a set of IPs on the FPGA by creating an optimized floorplan of reconfigurable regions (RRs). The floorplanner considers the distribution of heterogeneous resources on FPGA's layout and the resources utilized by the IPs. The objective of the floorplanner is to have minimum resource waste, which leads to higher resources for the static design (see Figure 2.2) as the static design's logic cannot be placed in the RRs of the resulted floorplan.

The details of the floorplanner are presented in Chapter 5, including related work, problem definition, device representation, MILP modeling, experimental results, and evaluation.

- The following contribution of the thesis is solving the problem of generating IPs independently, i.e., the third challenge to the FPGA IP market. The work explains the reasons why existing FV tools do not support independent IP generation. Afterward, several requirements are presented and argued that if FV tools fulfill them, they can generate compatible bitstream-based designs independently. This makes the work general and can be adopted to any FV tools or devices. In the next step, one of the requirements is analyzed that causes the lack of support. Then, several third party tools are presented that can overcome the lack of support. Since these tools also have limitations, a flow is presented that uses a trusted third party to solve this problem. The added advantage is that it matches the target use case, i.e., IP licensing schemes. Most IP licensing schemes utilize a third party who is responsible for security-specific tasks.

  The details of this contribution are presented in Chapter 6.

- Challenges from 4 to 8, presented in Section 1.4, are addressed with the thesis's main contribution: a pay-per-use IP licensing scheme and its realization using a security framework (SFW). The scheme is practical, less restrictive for SDs to use their device, offers security against IP theft, and protects the system from malicious IPs. The SFW consists of a trusted execution environment (TEE), a rich execution environment (REE), software modules for authentication, decryption, and analysis of bitstreams. Furthermore, it also includes several hardware components that are configured on the programmable logic (PL). Details of this contribution are presented below.
  - The TEE has access to the encrypted IPs, where they are decrypted, authenticated, and analyzed for malicious behavior using trusted applications (TAs).
  - The bitstream analysis makes sure that the IP is targeted to the location of the PL resources assigned to it. This feature allows the partitioning of the

PL into a secure and non-secure region, i.e., extending the TEE concept to the PL.

- The secure region contains components supporting the SFW, such as TrustZone logic, a configuration controller, and interconnects. It also has RRs for all licensed IPs.
- The non-secure region is for SDs, where they can configure their custom designs (IPs).
- The configuration and readback of the secure PL region can only be issued by the TEE. (Re-)configuration and readback of the non-secure PL region are still possible from the non-secure master (REE) via an API implemented in the TEE. This way, security is extended to the PL without affecting the available features like DPR and readback. Here, the term non-secure only means that the resources are non-secure for the unprotected licensed IPs because SD has full access to them.
- The REE is for system developers (SDs) to run their applications. Furthermore, SDs can also configure their IPs (self-developed) on the non-secure PL region.
- Validation of the scheme is done by implementing it on a Xilinx Zynq UltraScale+ MPSoC ZCU102.

The details of the IP licensing model are presented in Chapter 7, and the implementation issues are discussed in Chapter 8.

Holistically, the work tries to solve the practicality and security challenges of the FPGA IP market. The work is the first usage of a trusted execution environment (TEE) for IP protection and TEE extension to the programmable logic. The proposed security framework isolates some device assets to provide a TEE while keeping most of the device resources and features available to the system developer. The processing of IPs (e.g., decryption, analysis, and configuration) happens in a trusted environment, significantly reducing the attack surface. In addition, physical attacks on the decryption engines and memory are investigated, and measures are implemented to counter these attacks.

2 Background

The work described in this chapter was published in [179] and is joint work with co-authors Sven Nitzsche, Asier Garciandia López, and Jürgen Becker. More details on contributions is found in Section 1.5.

This chapter explains the relevant background knowledge required to understand the contributions of the thesis, which is presented in two parts. The first one is about reconfigurable devices, where their structure, types, and features (e.g., DPR) are briefly discussed. These details will help understand automatic floorplanning and standalone generation of IPs presented in Chapters 5 and 6, respectively.

The second part of the chapter presents the background knowledge in security. Information assurance and its properties (e.g., confidentiality, integrity, authentication) are introduced in this part. Since IPs are processed by the devices in the IP licensing scheme use case, establishing these properties on the device will ensure that processed IPs are secure. A system or device can have these properties using several security mechanisms, e.g., cryptographic algorithms, trusted execution environment (TEE), and secure boot. These sub-topics are also presented in the second part of the chapter.

2.1 Reconfigurable Devices

Reconfigurable or programmable logic devices (PLDs) are integrated circuits that can be configured with any digital circuit or even reconfigure to update existing ones after manufacturing. Programmable logic arrays, generic array logic, complex programmable logic devices, and field-programmable gate arrays (FPGAs) are some examples of PLDs. The rest of the section focuses on FPGAs, where a discussion is presented on the basic components, topology, and programming technology. Afterward, the DPR feature is discussed.

2.1.1 Basic Components

The basic components of an FPGA are logic-, input/output- (I/O), wiring blocks, a clock network, a configuration/scan chain, and a test circuit [53]. A generalized discussion is provided below.

- **Logic Block**: A logic block can be implemented as a lookup table (LUT), a multiplexer, or a product term logic. A product term logic means an AND-OR array structure. These implementation methods have a programmable part, which can be updated to realize any logic circuit [53].
- **Input/Output Block**: These are the blocks that connect I/O pins and the wiring blocks. In addition, they have flip-flops to hold values and control circuits such as the pull-up, pull-down, I/O directions, slew rate, and open drain.
- **Wiring Block**: Wiring channels, connection- and switch blocks can be collectively called wiring blocks. They provide connection between logic blocks and between logic and I/O blocks.
- **Others**: The logical functionality or connectivity of all the blocks discussed above is determined by the value written in their respective configuration memory. A configuration chain exists on FPGAs, which can be used to write the configuration data bits to all configuration memories serially. Besides this, FPGAs have clock networks, scan chains, and testing circuits.

The logic and routing resources of the device do not represent a specific functionality and need to be programmed (configured) to realize one. Logic blocks, after programming, are set to represent a sub-functionality, while the wiring/routing resources are programmed to realize the desired connectivity between the logic blocks. Collectively along with IO and other blocks, the device provides the desired functionality.
The storage of configuration information in the resources can be done using static random access memory (SRAM), anti-fuse, or flash memory. Each type of memory has its pros and cons that are discussed in Section. 2.1.3.

2.1.2 Topology

The FPGA layout can be realized in several ways based on the arrangement of logic and interconnect resources. In [127], authors classified them into five categories: island style, row-based, sea-of-gates, hierarchical, and one-dimensional structures. The details can be found in [127]. An island-style architecture of the FPGA, along with the basic components, is shown in Figure 2.1.

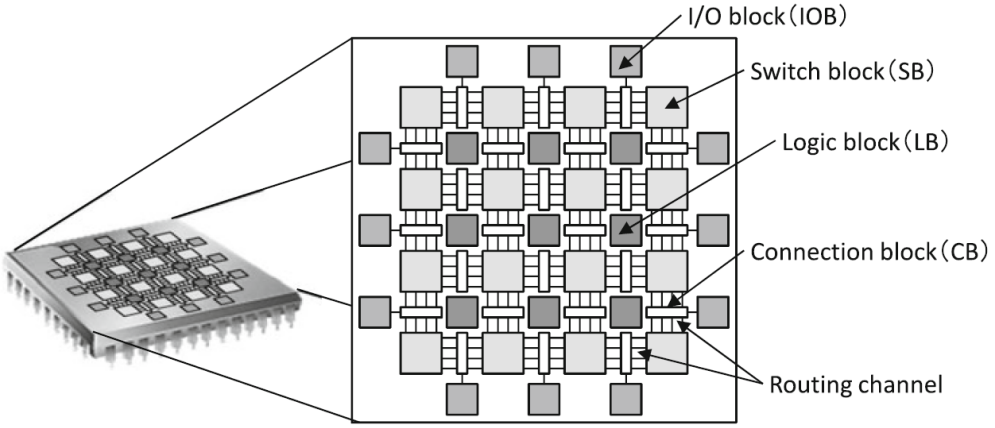

Figure 2.1: Island-style FPGA structure [53]

2.1.3 Programming Technology

FPGAs are roughly categorized based on the semiconductor technologies used for the manufacturing of the configuration memory. The technologies considered so far are erasable programmable read-only memory (EPROM), electrical EPROM, flash, anti-fuse, and SRAM. Among them, anti-fuse, flash, and SRAM are common and commercially successful. A discussion on them is presented below.

Anti-fuse

Anti-fuse, as the name suggests, performs the opposite function to a fuse. This type of memory provides volatile storage that is initially in an open state, having an impedance in the order of a few giga-ohms [127]. The application of a high voltage changes it permanently to the conducting state representing a digital circuit. Actel's programmable logic interconnect circuit element and QuickLogic's ViaLink are some examples that use this technology, whose structure and features are discussed in [4, pp. 28-29].

The pros of a programmable cell-based on Anti-fuse are as follows:

- Small size in comparison to SRAM and Flash memory cells;
- Nonvolatile, therefore, does not require external storage for configuration data on power down.
- Highly resistant to Reverse engineering;

- Robust against soft errors.

The cons are as follows:

- Cannot be re-programmed;
- Requires a special programmer;
- Programming takes longer;
- The programming yield is less than 100

Flash Memory

This type of memory uses non-volatile storage offered by EPROM, EEPROM, and flash memory technologies. A typical transistor based on these technologies has two gates instead of one, a control and a floating one. When current flows through the transistor, electrons are trapped on the floating gate because it is isolated. The charge on the floating gate is non-volatile and can be removed by exposing it to ultraviolet light in EPROMs and applying an electrical field in the case of flash and EEPROMs [127]. More details about the flash memory, its structure, types, and functionality can be found in [4, pp. 25-26].

The pros of a programmable cell-based on Flash memory are as follows [4, pp. 30]:

- Non-volatile;
- Has lower size than that of SRAM;
- Can immediately operate on power-up;
- Can be re-programmed;
- Strong against soft errors.

The cons are as follows:

- Requires a high voltage for re-configuring in comparison to SRAM;
- State-of-the-Art CMOS process cannot be used;
- Restriction on the number of re-configurations.

SRAM

SRAM is a random access memory that retains data as long as power is being supplied. They are usually composed of a positive feedback loop (two CMOS inverters) and two pass transistors (PT). Information is stored in the feedback loop, which can be overwritten using the PT [4]. As mentioned in Section 2.1.1, a configurable logic block can be implemented using LUTs and MUXs. LUTs act as a memory that stores the truth table of a logical expression. Both LUTs and MUXs can be implemented using static memory. FPGAs using this type of memory are called SRAM-based FPGAs, and they are currently the mainstream devices.

The pros of a programmable cell-based on Static memory or SRAM are as follows [4, pp. 31]

- State-of-the-Art CMOS process can be used;
- Run-time partial reconfiguration is supported;
- No limitation on the number of reconfigurations.

The cons are as follows:

- Volatile;
- Memory size is larger;
- Harder to secure the configuration data;
- Higher sensitivity to soft errors;
- Higher on-resistance and load capacity.

Target Programmable Technology

As mentioned in Section 1.3, the delivery format chosen for this work is FPGA's configuration data (bitstream) instead of RTL description. Furthermore, the section discourages integration of IPs on a workstation as that would increase the attack surface. Instead, features like DPR (see Section 2.1.4) should be used to generate IP's end-product and deliver its encrypted form to the device, i.e., to avoid integrating IPs with static design on a workstation. Therefore, anti-fuse FPGAs cannot be used for the proposed work as they can only be configured one-time and in a monolithic form, i.e., would require tools on a workstation for integration. Also, flash memory FPGAs are not suitable for this work because they are limited by the number of allowed

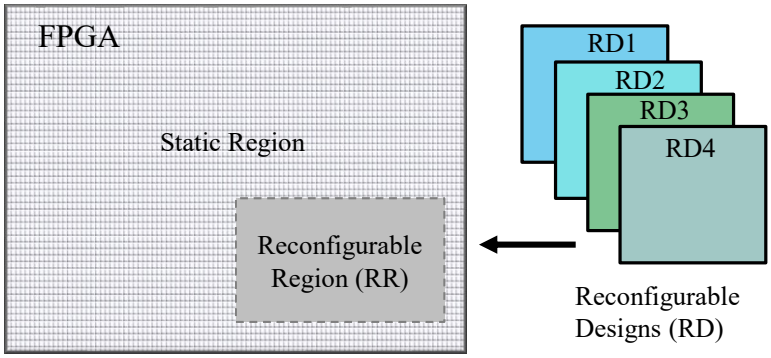

Figure 2.2: Dynamic partial reconfiguration flow

reconfigurations (about 10,000 times), and they do not offer DPR.

On the other hand, SRAM-based FPGAs can be reconfigured any number of times and support DPR. Furthermore, advanced CMOS process technologies can be applied for their manufacturing, leading to higher integration and performance. Even though their non-volatile nature makes them vulnerable to additional attacks such as probing and readback in comparison to the other types; however, they are the only one that supports configuration of encrypted IPs on the device. Hence, the focus of this work is only on SRAM-based FPGAs.

2.1.4 Dynamic Partial Reconfiguration (DPR)

Partial reconfiguration is a feature where parts of the PL can be reconfigured instead of configuring the entire PL. Dynamic reconfiguration means that the configuration can be done in run-time, i.e., while other designs are functioning. Combining these two results in DPR, which means a region of the FPGA can be reconfigured while designs on other regions can still stay active. Figure 2.2 shows this flow. The feature was introduced almost two decades ago, and today almost all available SRAM-based FPGAs support it.

Usually, different configurations (designs) are available to the device using a non-volatile memory and are configured onto the hardware on demand. This dynamic upgrading of the hardware provides new possibilities. Therefore, this technique is currently the subject of intense research in academia. More details about the feature can be found in [164].

More details of the DPR flow can be found in [147].

2.2 Information Assurance

The protection and risk management of information during its processing, storage, and transmission is called information assurance. It has five pillars [135], also called fundamental properties. An information system must have these properties to ensure that its data is protected against security threats. A brief description of these properties is given below.

1. **Authentication:** It is the process of verifying the identity of a user or a system. It also means to verify that the data produced or transmitted by a user is the producer or sender of that data.

2. **Integrity:** This property refers to the accuracy and consistency of the information over its life cycle. This can be achieved by protecting it from unauthorized tampering or modifications.

3. **Availability:** As the name suggests, it refers to the availability of data to be used or modified by an authorized user.

4. **Confidentiality:** This property refers to the protection of data being accessed by an unauthorized user. It is ensured with the use of cryptography.

5. **Non-repudiation:** In essence, non-repudiation is similar to authentication as it refers to the verification of the origin of data. This prevents possible denial that a data is sent by a specific user.

The properties mentioned above are essential for the security of the information or IPs in an embedded system. The use case of the proposed work is an IP licensing scheme, where FPGA-based IPs are delivered to the devices. The IPs must be kept confidential during their transmission, use, or storage to protect them against IP misuses (see Section 3.2). Also, the receiver must be able to authenticate that the IPs are produced by the IP core vendor (CV). Also, it needs to be made sure that the IPs are not tampered with, i.e., their integrity is not compromised.

Information assurance properties can be achieved using cryptographic algorithms. For example, they can provide **Confidentiality** of the data at rest or transit using symmetric or asymmetric encryption. Similarly, **Authentication/Non-repudiation** of data can be achieved with the use of hashing functions (see Section 2.3.1) and digital signature schemes (see Section 2.3.3). Together they also offer **Integrity** of the data at rest, in use, or at transit. Integrity can also be achieved using hashing functions and message authentication codes (MACs) (see Section 2.3.2). MACs are based on symmetric cryptographic algorithms, and digital signature schemes are based on asymmetric

algorithms. All the mentioned algorithms are cryptographic and are presented in Section 2.3.

Cryptographic algorithms are essential in achieving these properties; however, they also require other security mechanisms to protect devices against security threats. One such security mechanism is Secure Boot. This feature offers the possibility to boot the device with authentic encrypted system files, e.g., boot loaders, firmware, and operating systems. In the absence of such mechanisms, an adversary can boot the device with system files that have malicious functionality to steal information, IPs, or encryption keys. A detailed discussion on the Secure Boot mechanism, specific to the target device, is present in Section 2.4, which is preceded by cryptographic algorithms for ease of understanding.

Other security mechanisms essential for security are specific to the storage or usage of the keys used by cryptographic algorithms. If these keys are not properly protected, an adversary can steal them, which will compromise the entire system's security. The hacking of the first generation gaming console from Microsoft, namely Xbox, is a good example of not protecting the key in use. In 2002, Andrew Huang developed a hardware board to intercept data transfer over the HyperTransport bus of the Xbox [51]. Since the symmetric encryption key used to protect the secure boot loader was going over the bus in plaintext form, he was able to read it. This led to the execution of malicious code on the device. Therefore, key storage techniques need to be investigated, whose details are presented in the following section.

2.2.1 Key Storage

In SoC FPGAs, cryptographic algorithms are available in three ways: the hardwired dedicated decryption engine (DDE) of the device, a custom cryptographic IP core programmed on the PL, or a software application running on the processing system (PS). Also, hardware security modules (HSMs) [84] or trusted platform modules (TPMs) [2] can be used to perform some security operations. However, if HSMs/TPMs are used for decryption, the plaintext returning from them can be read over a bus/port. The use case of this work is an IP licensing scheme where IPs need to be decrypted securely, i.e., without exposing them on a bus. Therefore, HSMs/TPMs are not used or even discussed anymore, and the rest of the discussion is focused on the first three ways of using cryptographic primitives.

DDEs are usually used by the secure boot mechanism. They can also be used for decrypting FPGA IPs. They are implemented with specialized hardware processors that are only used for security-specific operations where keys are stored in protected storage. For example, Xilinx Zynq UltraScale+ devices offer to store the key in volatile battery-backed RAM (BBRAM) or non-volatile eFUSE storage locations. Furthermore, the key stored in eFUSE can be either plaintext or obfuscated (i.e., encrypted with the device family key or with a key generated by a physically unclonable function (PUF)). Also, the key loading path to the device is write-only, and there is no physical data path to read back either key (For details, see Chapter 12 of [141]). With all these built-in security features, it can be stated that the encryption keys used by the DDEs on the target device are secure. In this work, DDE is used for the Secure Boot mechanism (see Section 2.4) and not for the decryption of IPs.

As mentioned earlier, cryptographic operations can also be performed using an FPGA IP or a software application. For these implementations, devices do not offer any security features or protected storage. A simple solution can be hard coding the cryptographic key in the software or FPGA IP implementation. However, keys can be extracted from the software code using reverse engineering. In the case of FPGA implementation, the readback attack (see Section 3.1.4) can be used to read FPGA's configuration data. Afterward, reverse engineering [96] of the configuration data will yield the key.

Among others, the secure storage-specific challenges can be solved using a trusted executed environment (TEE). TEEs are secure and isolated environments assisted by the hardware where trusted applications (TAs) run. This isolation is system-wide, where resources, applications, and even read-write operations are partitioned into secure and non-secure. Cryptographic algorithms can be implemented as a TA where secret keys are hardcoded in it. Since TAs execute inside the TEE, they will be secure even if the keys are hardcoded in them.

A limitation of the hardware-assisted isolation (TEEs) is that the existing technologies only partitions PS's resources. For the PL, only read-write transactions among PS and PL are distinguished as secure or non-secure. This means reconfiguration and readback features performed on the PL are not differentiated as secure or non-secure. Even a non-secure environment or resource can access the entire PL region. This is countered by partitioning the PL into secure and non-secure regions (see Section 7.4.2), which is one of the focuses of the proposed work. In cases where the TEE concept is extended to the PL, hard-coded keys inside the FPGA-based IP's implementation will also be secure.

Since this work uses cryptographic algorithms, secure boot mechanism, and TEE to protect the device and IPs, background information on these topics is presented in the following sections.

2.3 Cryptographic Algorithms

Cryptography is the science of converting secret messages with the intention of hiding their meaning [99]. The secret and hidden messages are commonly referred to as plaintext and ciphertext, respectively. The process of converting plaintext to ciphertext is called encryption, and the reverse of this process is called decryption. In this field, techniques and algorithms are developed to offer security in the presence of an adversary.

The main two main branches of cryptographic algorithms are symmetric- and asymmetric-key algorithms. The main difference among them is that symmetric algorithms use the same key for encryption and decryption. However, the asymmetric ones use different keys. There is another type of algorithms that does not use any key for computation called hashing functions. Here, first, the hashing functions are presented that are followed by symmetric- and asymmetric-key algorithms.

2.3.1 Hashing Functions

These cryptographic primitives are widely used in a range of protocols. However, they were created to support other cryptographic primitives to implement integrity and authenticity properties. Hash functions compute a message-digest of data, which is short and has a fixed length. Data, in this case, can be of any length. The message digest is also commonly called a hash value and can be seen as the fingerprint of the data. These functions do not require any key for computing the hash value [99, pp. 293].

The motivation behind using hash functions is to compute a fixed-length short message from it, which can be transmitted along with the data. Since computing the hash value does not require any key, the receiver can compute the hash of the received data and compare it with the received hash value. The verification will ensure that data integrity is not compromised. However, suppose data and its hash value are sent as plaintext. In that case, an adversary can modify the data and update the hash value with the new hash value computed from the modified data. Therefore, at least the hash value must be sent as ciphertext. Using hash functions to support other cryptographic primitives for data integrity and authentication are discussed in Section 2.3.3 and 2.3.2.

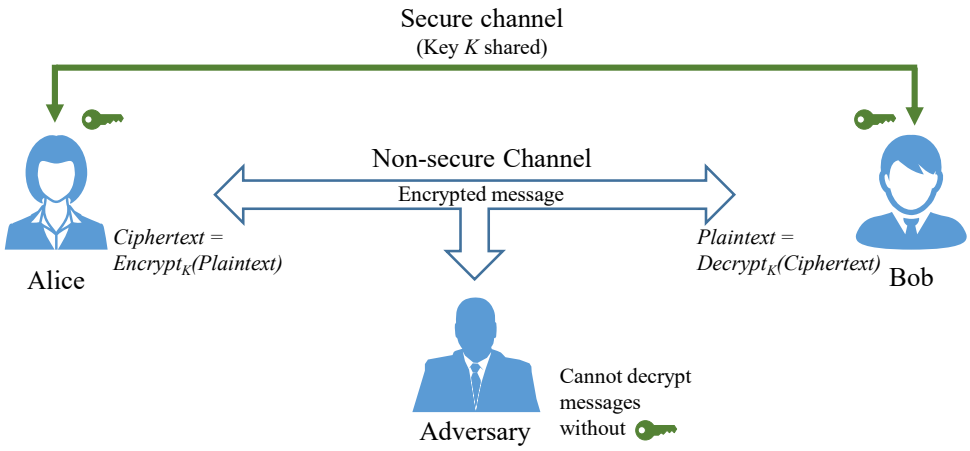

Figure 2.3: Secure communication between Alice and Bob on a non-secure channel

2.3.2 Symmetric Key Algorithms

As the name suggests, this class of algorithms relies on similar cryptographic keys for both encryption and decryption. The keys can be identical or can be easily converted into each other. This implies that the communicating entities share a common secret, which is one of the drawbacks of these algorithms because the distribution of the shared secret would require a secure channel. Figure 2.3 shows a secure communication between Alice and Bob on a non-secure channel using a shared key, which was distributed via a secure channel. Even if an adversary acquires secure messages, he can not decrypt them as he does not have the shared secret. The secure channel shown in Figure 2.3 can be implemented using asymmetric-key cryptography, which is commonly used for key distribution.

Another important fact is that the strength of these algorithms can benefit from making them public. It seems that making the algorithm secret will improve security. However, this is a disadvantage as secret algorithms cannot be tested by other cryptographers. Examples of symmetric key algorithms are data encryption standard (DES), triple-DES (3DES), and advanced encryption standard (AES). Their details can be found in [99]. Here, background information on AES is presented as one of the contributions of this work is a side-channel attack (SCA) resistant AES implementation (see Chapter 4).

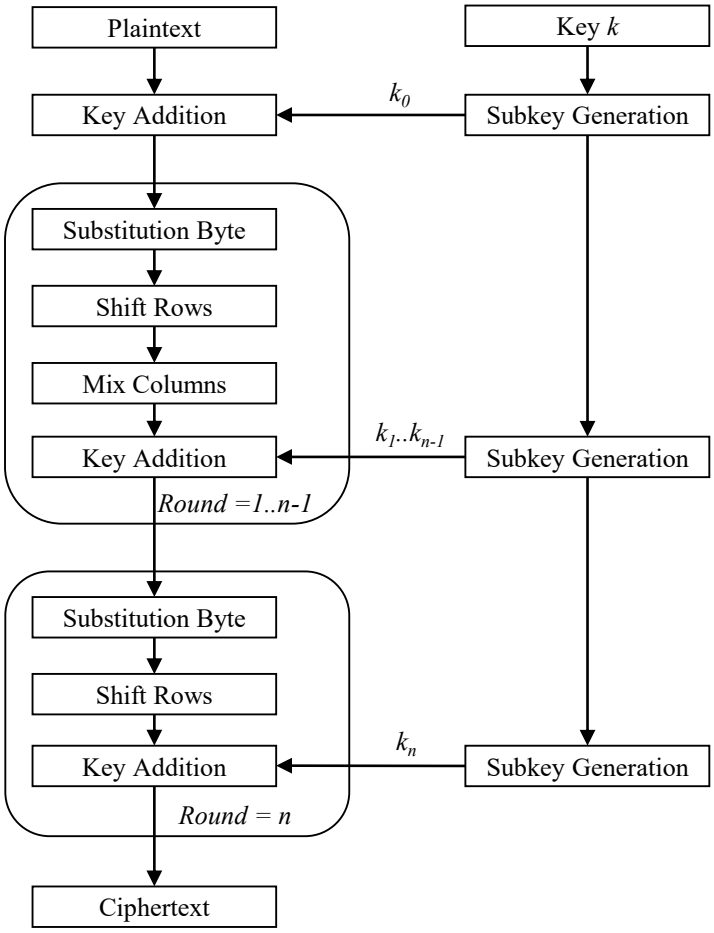

Figure 2.4: AES Encryption flow diagram

Advanced Encryption Standard (AES)

The US National Institute of Standards and Technology (NIST) called in 1997 for proposals for a new block cipher, an encryption algorithm that can be applied to a group of bits called blocks. The call was influenced by the security and implementation weaknesses found in DES and 3DES. The evaluation process of the submitted proposals was open and done by the international scientific community organized by NIST. Among the proposal, the Rijndael block cipher was proposed by two Belgian cryptographers that can have a block and key size of 128, 192, and 256 bits. In 2001, NIST selected Rijndael with a block size of 128 bits and a key size of 128, 192, and 256 bits as the new

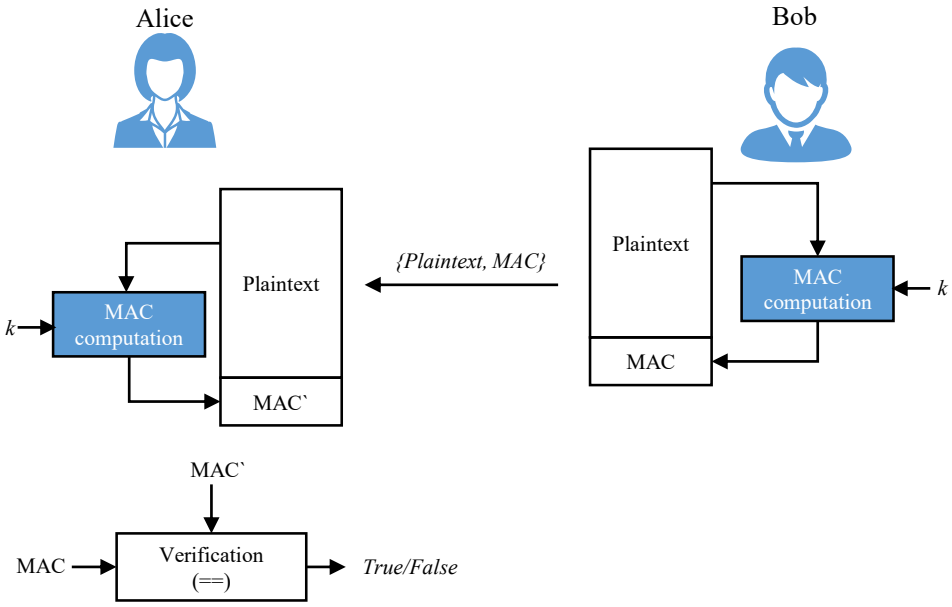

Figure 2.5: MAC computation and verification overview

block cipher, commonly known as AES.

AES performs four different operations on a block of data in several rounds. The number of rounds depends on the key size, which are 10, 12, and 14 for the key width of 128 bits, 192 bits, and 256 bits, respectively. AES operations are substitution byte (SBOX), shift rows, mix columns, and key addition. An AES round has all four operations in the order they are presented except the final round, which does not have the mix columns function. AES encryption starts with Key Addition operation followed by all the rounds as shown in Figure 2.4. More details about its operations and the decryption process can be found in [99].

Message Authentication Codes (MACs)

Message authentication codes (MACs) are widely used to provide message integrity and message authentication. These algorithms are used to compute a short fixed-length code/tag from a message, which is then appended to the message. As shown in Figure 2.5, Bob computes a MAC from the plaintext using a key, which is then appended with the plaintext. Afterward, this data is sent to Alice. After receiving the message, Alice would like to know whether the data is modified during the transmission or not. So she computes MAC value (MAC') from the plaintext and compares it with the MAC

value received from Bob. Since only Alice and Bob have the common key, only they can generate or verify a valid MAC.

MACs use the same key, which means they do not provide non-repudiation, i.e., one cannot say with certainty who created the MAC among the people who have the shared symmetric key. However, the non-repudiation limitation does not exist with digital signatures (See Section 2.3.3), which is the asymmetric equivalent of MACs. The advantage of MACs over digital signatures is that they are faster to compute. HMAC (Hashing MAC) is a type of MAC where key is appended at the start/end of the data, and its hash value is computed. Alternatively, MACs can also be computed from symmetric encryption algorithms such as AES. More details about MAC, its properties, and types can be found in [99].

2.3.3 Asymmetric Key Algorithms

As mentioned in Section 2.3.2, one of the drawbacks of symmetric cryptography is key distribution. Even if the key distribution challenge is solved, the algorithms would require a large number of keys as each pair of users would require a separate key [99]. These drawbacks can be overcome using asymmetric cryptography (also known as public-key cryptography) that was introduced to the public in 1976 by Whitfield Diffie and Martin Hellman [24].

Asymmetric encryption relies on a pair of keys per user, where messages are encrypted with one key and decrypted by the other. The basic idea is that one key of the pair is made public, called the public key, while the other is kept secret and is referred to as the private key. One of the advantages of asymmetric cryptography is that it can be used to distribute keys used in symmetric algorithms over a non-secure channel. The key distribution and secure communication over a non-secure channel are shown in Figure 2.6, where Alice sends a secret key to Bob after encrypting it with Bob's public key. After receiving the key, Bob decrypts the ciphertext with his private key. Now that both parties have a common key, they can securely communicate. Since only Bob's private key can decrypt the message, an adversary cannot eavesdrop on their communication. Asymmetric ciphers are computationally several times slower than symmetric ciphers; that is why the former is used for key distribution while the latter for encryption large data blocks. Public key cryptography also offers other security mechanisms such as non-repudiation, identification, and encryption [99, pp. 154].

The only remaining challenge with public-key cryptography is regarding the authenticity of the public key, i.e., how can one be sure that a public key is of the entity that

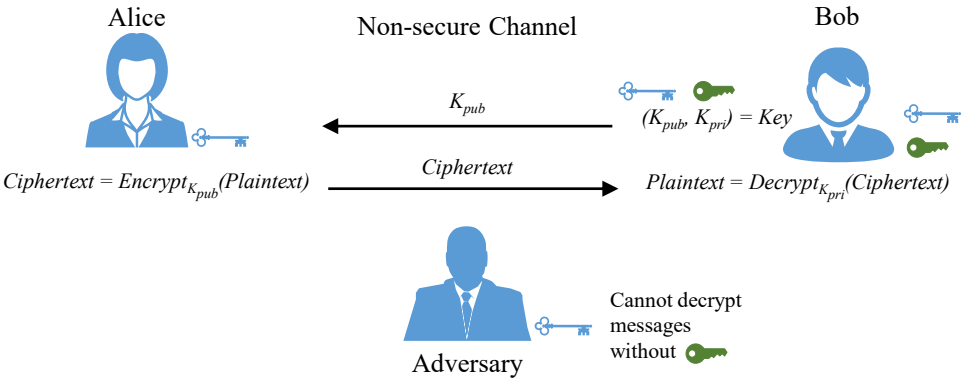

Figure 2.6: Secure communication from an entity (Alice) to another one (Bob) using Asymmetric Cryptography

claims its ownership? In practice, the issue is solved with certificates. Certificates are issued to users or organizations by trusted authorities who bind a public key to a certain identity [99, pp. 344]. Another minor issue with this branch of cryptography is that the keys are very long, which results in slower execution times. Further details on these issues can be found in [99, pp. 155].

RSA cryptosystem

Asymmetric ciphers can be implemented using one-way functions. One-way functions are the ones where it is easy to compute the function $f(x) = y$ but hard to compute its inverse $f^{-}1(y) = x$. For example, finding two large prime numbers and computing their product can be considered a one-way function, whose inverse would be finding the prime factors from the product. This is an integer factorization problem that is considered computationally infeasible for large numbers (e.g., 1024 bits or more). This principle is the basis for the Rivest–Shamir–Adleman (RSA) cryptosystem, a family of asymmetric ciphers proposed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. Other families of asymmetric ciphers are based on Discrete Logarithm Problem and Elliptic Curves.

Since these algorithms use large numbers, they are computationally several times slower than symmetric ciphers. Therefore, they are mainly used for encrypting small pieces of data, e.g., keys used in symmetric ciphers or signing hashes in the case of digital signatures. In this work, RSA is used for distributing symmetric cipher keys (see Section 8.5.1) used for the FPGA IPs. Furthermore, encrypted IPs are also authenticated using RSA asymmetric cipher (see Section 8.5.2). As the technical details of the algorithm

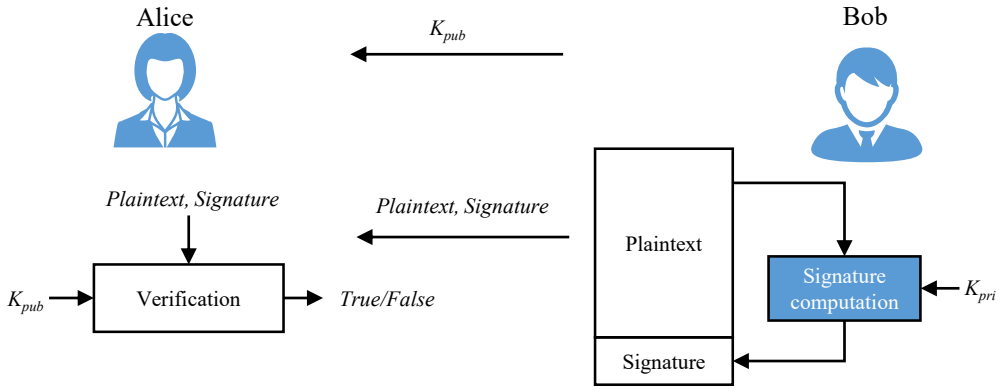

Figure 2.7: Digital Signatures

are beyond the scope of this work, they are not presented here. However, they can be found in [99, pp. 173-199].

Digital Signatures

The cryptographic algorithm discussed so far are focused on providing encryption and data integrity. These security measures are enough against an external adversary; however, there are several scenarios where legitimate users can act in an untrustworthy fashion. For example, users communicating using a shared key can agree on the price of an object, but later one of them denies sending the message. The property of information assurance dealing with this situation is called non-repudiation, which is the verification of data's origin. This can be achieved using a public key cryptographic algorithm called digital signatures.

Digital signatures can be seen as an asymmetric key equivalent of MACs, as they share some functional properties such as offering message integrity. As shown in Figure 2.7, Bob computes a signature of a plaintext using his private key (K_{pri}). Afterward, he sends both the signature and the plaintext to Alice. Since she already has access to Bob's public key (K_{pub}), she can verify the validity of the signature. Since every user is responsible for keeping their private key secret, Alice can be confident that Bob cannot, in the future, deny that he sent the message. Furthermore, any modification of the plaintext by an adversary can be detected as the verification process will show that the signature is not valid, i.e., digital signatures also offer message integrity.

2.4 Secure Boot

In FPGAs, the secure boot mechanism is traditionally implemented using a hardwired dedicated decryption engine (DDE) and NVM-based keys. An encrypted bitstream is provided to the device from an external storage medium (e.g., Flash memory), where DDE decrypts it and configures it on the PL [105]. In Xilinx Ultrascale and Ultrascale+ architecture-based FPGAs, DDE is an AES - Galois/counter mode (AES-GCM) decryption and authentication logic. The encryption keys for which can be stored in dedicated RAM or eFUSE. Even though AES-GCM is a self-authenticating algorithm with symmetric keys, these architectures also provide an alternative authentication way using RSA-2048 (For details see [134]).

Since SoCs have a hardwired PS, their secure boot mechanism involves more components. This mechanism can be used to boot authentic encrypted system files (e.g., boot loader, firmware, and OSes) on the SoC FPGAs. Zynq Ultrascale+ MPSoC is an example of SoC FPGA that utilizes dedicated state machines, a platform management unit (PMU), and a configuration security unit (CSU) to do the system boot-up process. CSU is the DDE, which contains a triple-redundant processor for controlling boot operation and a crypto interface block.

Once Zynq Ultrascale+ MPSoC is powered up, the dedicated state machine performs a series of mandatory and optional tasks. Then, it sends an immutable ROM code to the PMU, whose integrity is validated against a golden copy stored in the device using the secure hash algorithm 3 (SHA3) engine. Once these security operations are completed, PMU sends an immutable ROM code to the CSU, which is again validated against a golden copy using the SHA3 engine. At this stage, reset to the CSU is released. So far, the hardware root of trust (ROT) was established using immutable ROM codes (PMU's and CSU's) whose integrity was validated. As CSU is at the center of the secure boot process, it enforces the hardware ROT once enabled. It also maintains the device's security state by prohibiting switching between secure and non-secure states without a full power-on reset. In addition, CSU is used for the public key's validation, revocation, authentication, and decryption of the first stage boot loader (FSBL). Once all these operations are completed, CSU releases the reset to the specified processing unit. Here, a summary of the boot process is presented that is focused on the security aspects; further details on it can be found in Chapters 11 and 12 of [141].

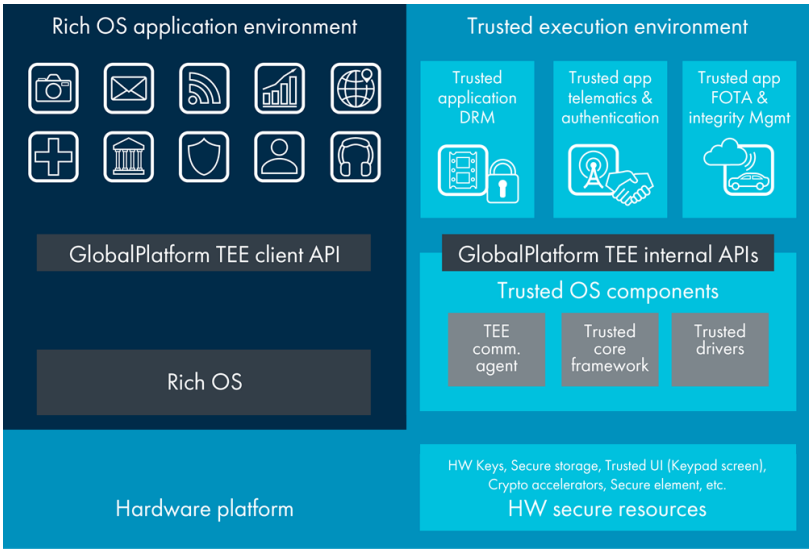

Figure 2.8: Architecture of the Trusted Execution Environment

2.5 Trusted Execution Environment (TEE)

GlobalPlatform, in TEE System Architecture version 1.2 [36], defines a TEE as "An execution environment that runs alongside but isolated from an REE (Rich Execution Environment). A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly".

Both TEE and REE have access to resources such as processing core, RAM, ROM, etc. However, at any given time, only one of them is accessing a resource. When a TEE accesses resources, they are isolated from REEs unless access is authorized. A controlling TEE considers all of its non-shared resources to be trusted, and these resources are only accessible by other trusted resources. This makes it a closed-trusted system. There may be some resources that all execution environments can access. Also, some REE-specific resources can be made accessible to a TEE, but the opposite may not be allowed. The execution environment's isolation requires support from both software and hardware. The TEE architecture is shown in Figure 2.8. Details about the architectural specification and interfaces are given in [36].

TEEs offer a secure environment for trusted applications (TAs) by protecting their execution code and data. TEEs also offer high processing speeds and a large amount of memory in comparison to other security environments [38]. Their primary purpose is the protection of device and TA assets, which is achieved by security features such as isolation from the rich OS, isolation from other TAs, application management control, binding (Secure Boot), trusted storage, trusted access to peripherals, and state of the art cryptography [38].

2.5.1 Applications

As discussed in Section 2.2.1, TEEs offer an excellent solution to store and manage keys used by the cryptographic algorithms, which is one of the reasons TEE is used for the use case of the proposed work. Furthermore, the TEE isolates both the PS and PL configuration interfaces (see Section 3.1.4), memory regions, TAs, and peripherals. These security features are requirements for an IP licensing scheme, which is the proposed work's use case.

Other TEE applications can be implementing bio-metric authentication solutions, e.g., face/fingerprint/voice recognition. User's private data used as credentials can be stored in isolated resources to ensure the Rich OS does not have access to it. For example, users' fingerprint-specific data (e.g., Hash value) must only be accessible from specific TAs inside the TEE. Also, applications like contactless payments and mobile wallets require handling user credentials that could benefit from using a TEE.

Copyrighted data such as movies, music, and books can also be protected using a TEE. They are protected during transmission via encryption; however, it is decrypted on the device for consumption. TEEs can make sure that the decrypted content can only be viewed or heard on the authorized device and can not be copied to an external storage or the memory regions of the rich OS.

2.5.2 Hardware Support

Several hardware technologies support secure and isolated environments. ARM TrustZone (TZ), Intel Software Guard Extensions [23], IBM SecureBlue++ [12] and RISC-V MultiZone^{TM} Security [30] are examples from industry. AMD Secure Processor (formerly known as Platform Security Processor) [32] and Intel Trusted Execution Technology [91] are not mentioned as an example because the former utilizes ARM TZ while the latter relies on a trusted platform module (TPM) and cryptographic techniques to provide trust on the device. In academia, several approaches [21, 28, 50, 74, 98] have

been proposed. The focus of this work is using ARM TrustZone to provide IP licensing; hence only this technology is discussed.

2.5.3 ARM TrustZone

Arm TrustZone (TZ) is a hardware-based system-wide security solution that is available on recent Arm application processors and microcontrollers. Several TZ components (e.g., the TZ Protection Controller and TZ Address Space Controller [6]) can be used to logically partition the system into a non-secure and secure world. With this partition, a single processor core can execute code from the non-secure and the secure world in a time-sliced manner. When the processor is executing code from the non-secure world (REE), it enters a non-secure state where it can only access resources of the non-secure world. In the opposite case, when the processor enters a secure state, it can access resources located in both worlds. The switching between the secure and non-secure world is managed by a security monitor that ensures that the current world's state of the processor is securely stored before it leaves the current world. Furthermore, it makes sure that the state of the world the processor is switching to is also correctly restored.

The access to secure world resources from a non-secure world is restricted by hardware logic present in the TZ-enabled advanced microcontroller bus architecture AXI buses. This hardware logic includes an extra control signal for each read/write channel on the system bus. These additional bits are called non-secure (NS) bits [6].

[3] provides more details and [104] provides a comprehensive survey on the relevant work. Even though TZ was proposed in 2002, it got more attention during the last few years from both industry and academia [5, 29, 77, 85, 112, 123].

3 Security Threats

The work described in this chapter was published in [179] and is joint work with co-authors Sven Nitzsche, Asier Garciandia López, and Jürgen Becker. More details on contributions is found in Section 1.5.

As discussed in Section 1.3, IPs can be delivered in encrypted form as RTL code or bitstream, and each format has its risks regarding IP theft. However, RTL ones are prone to more attacks, and licensing schemes using them add extra inefficiencies. For example, one of the main problems with RTL-based IPs is that they are processed with FV tools on a workstation, which leads to a large attack surface that includes but is not limited to the FV tool, operating system, memories. On the other hand, bitstream-based IPs ($IP_{PlainBit}$) are delivered in encrypted form (IP_{EncBit}), which does not require any pre-processing. IP_{EncBit} can be directly decrypted and configured on the device, which reduces the attack surface to the programmable device. Therefore, this chapter mainly focuses on the security threats specific to bitstream-based IP and the programmable device. These threats are categorized into three types and are presented below.

1. **Theft-Attacks:** The first type of attacks are the ones that try to breach the device's security to get access to the confidential data inside the device. The confidential data can be $IP_{PlainBit}$, the cryptographic key(s) used to encrypt the $IP_{PlainBit}$, the cryptographic key(s) to protect the secure boot process or communication of the device with the outside world. These attacks are named as Theft Attacks (see Section 3.1).

2. **IP Misuses:** These attacks are specific to $IPs_{PlainBit}$. At the end of a successful attack of the first type, an adversary can get $IPs_{PlainBit}$. Afterward, he can make this type of attack. Also, these attacks apply to IPs delivered in plaintext form, i.e., without encrypting them. These attacks are named IP Misuses (see Section 3.2).

3. **Malicious IPs:** The third type of attacks are the ones where a malicious functionality is inserted in the IP to harm the overall system. These attacks are named as Malicious IPs (see Section 3.3).

3.1 Theft Attacks

3.1.1 SCAs on Cryptographic Implementations

Modern key-based encryption algorithms are, in theory, considered mathematically secure; however, this assumption is not valid for their respective implementations. Attacks that can take advantage of specific implementation characteristics of cryptographic algorithms are among the most common type of side-channel attacks. They reveal secret information of the implementation's inner state, which then, in turn, can be used to reconstruct the cryptographic key in use [52]. Common side channels in the case of SCAs on cryptographic algorithm's implementations are power consumption, execution time, acoustic and electromagnetic (EM) radiation. A power analysis attack, for example, exploits the data-dependent nature of the switching activity of a cryptographic implementation. Since these attacks can be non-invasive and only use information extracted from physical observation, it is not easy to detect them. Consequently, one cannot be sure if a secret key is already compromised [52]. Using SCAs, an adversary can acquire data leaked by their implementation and use statistical methods to acquire cryptographic keys [175, 61, 90, 121]. Differential power analysis [63], and correlation power analysis [33] are examples of such statistical methods.

On SoC-based FPGA devices, there are three ways that cryptographic algorithms are realized, which are presented in the following subsections. All of these implementations are vulnerable to side-channel attacks.

Hardwired DDEs

Hardwired dedicated decryption engines (DDEs), discussed in Section 2.4, are used to provide a secure boot mechanism of the device. Since DDEs are at the center of the secure boot process, it enforces the hardware root of trust (ROT) once enabled. Moreover, they maintain the security state of the device. If they are compromised (i.e., keys are acquired using SCAs), all the subsequent security measures are compromised, and secret assets of the device can be accessed.

In addition to the secure boot mechanism, DDE is often used to provide security to bitstream-based IPs by decrypting them on the device before configuration. If an adversary acquires DDEs' keys, they will have access to the IPs in plaintext form.

FPGA IP

The second way an SoC device can offer cryptographic functionality is by offering a custom cryptographic IP core programmed on the PL. These IPs can be used for secure communication by the device. SCAs can also compromise them, which will again lead to a range of attacks, such as an adversary can pretend to be the device or an adversary can listen to the communication of the device.

Software IP

The third and final way cryptographic functionality can be implemented is to use a software application running on the processing system (PS) of the device. These implementations can also suffer from SCAs, and the result will be the same as discussed in the last section.

3.1.2 DDR Memory Attacks

Double data rate (DDR) memory or DDR synchronous dynamic random access memory (SDRAM) is a common type of memory used as random access memory (RAM) for modern processors. These memories are volatile; however, they retain their contents for several seconds after losing power. This is another side-channel that the SCAs can exploit. For example, an attacker can do a hard reset of the target machine and acquire memory contents. This attack is commonly known as a cold-boot attack [47]. The attack can target all software applications because their sensitive code and data are often placed in the DDR memory in plaintext form.

Modern DDR memories also suffer from a reliability challenge known as *Rowhammer*. This issue emerged due to an increase in DRAM cell density and a decrease in capacitor size over the past decades. Rowhammer is the bit flip in a specific row of the memory by repeatedly accessing its neighboring rows [59]. It is considered a threat to data integrity where an adversary or unprivileged user tampers the data without accessing it. However, recently another work named RAMBLEED [67] shows its effect on data confidentiality. Using that attack, an adversary can leverage rowhammer-flipped bits to read the value of neighboring bits. Attack like these are more relevant to this work, as plaintext bitstream of IPs are stored in the DDR memory (see Chapter 8).

These attacks are more severe in cases where cryptographic algorithms are implemented as software applications that keep the keys and the decrypted data in the memory. For example, if IP_{EncBit} is decrypted using a software application, the decryption key, and $IP_{PlainBit}$ will be stored in the DDR memory, which can be stolen using these attacks.

3.1.3 Probing

As the programmable logic (PL) in SRAM-based FPGAs/SoCs is volatile, the full bitstream needs to be configured on the PL every time the device is turned on. Furthermore, the partial bitstreams (IPs) generated using DPR flow are configured in run-time on the PL using programming interfaces. Both types of bitstreams need to be provided to the device using non-volatile memory (NVM). This frequent transfer from NVM to the PL makes them more vulnerable to probing attacks, which is capturing bitstream using an electrical probe [26]. Another probing attack is recently published [126], where an optical probe was used against the Dedicated Decryption Engine (DDE) of a device to capture the bitstreams in plain-text form.

3.1.4 Readback Attack

Readback is a feature provided for most FPGA families. It allows reading the FPGA's configuration data for debugging [145, pp. 176]. However, this feature can also be used to obtain secret information (e.g., keys, proprietary algorithms). Xilinx's SoC-based FPGA devices include a PS that has full access to the PL. The PS can read configuration data using its processor configuration access port (PCAP). In Xilinx devices (both SoC or non-Soc FPGAs), readback is also supported from the programmable logic via an internal configuration access port (ICAP). According to Xilinx's Zynq-7000 SoC manual [150, pp. 783], both processor configuration access port (PCAP) and ICAP are trusted channels, and they can be used to read out bitstream even when an encrypted bitstream is loaded onto the FPGA. We verified this behavior on a Zynq UltraScale+ device by reading out the configured bitstream via PCAP when bitstream encryption was enabled. In theory, readback can be disabled by setting the *bitstream.readback.security* property to *Level1* while generating bitstreams. *Level1* security implies that the readback feature is disabled, and *Level2* implies that both readback and reconfiguration features are disabled [140, pp. 289]. This setting is part of the full bitstream where the security bits (SBITS) field of the control register 0 of the device is set with the defined security level [145, pp. 157, 163]. However, this setting can be disabled by updating the SBITS of the Control Register 0 from PS.

Similarly, custom logic (e.g., reconfiguration controllers) on the PL can read out the configuration data using ICAP. PS does not have direct access to ICAP; however, it could have access to the reconfiguration controller that can directly access ICAP. Furthermore, a malicious user can configure a configuration controller on the PL while connecting it to the ICAP. Using these two scenarios, ICAP could be used to launch this attack.

3.2 IP Misuses

Successful theft attacks will lead to the acquisition of the IP by an adversary. Once the plaintext IP is stolen, it can suffer from multiple IP misuses that are presented in the following sections.

3.2.1 Cloning

Cloning is the unauthorized use of intellectual property that might require little or no modification. In the case of bitstream IPs, it would be creating a copy of the IP's configuration data to overuse and sell. This way, an adversary can avoid paying extra licensing fees and even sell to get revenue from the IP with minimal engineering cost. FPGAs are off-the-shelve products, and the configuration data (bitstream) is easy to duplicate, which makes FPGA-based IPs very sensitive to this misuse.

Several countermeasures can avoid this cloning attack. For example, friend or foe identification, device identifier detection, watermarking, using a physically unclonable function (PUF), and bitstream encryption. These countermeasures are explained in detail in [156].

3.2.2 Reverse Engineering

It is the process of analyzing an existing IP, in bitstream format or as design files (RTL/netlist), with the intention of learning the innovation behind it. This allows an adversary to create the IP with minimum research and development effort, causing the IP owner and their competitors an unfair disadvantage. Even though the act of reverse engineering is slightly more challenging with bitstream-based IPs compared to the design files, still the bitstream obfuscation does not provide any cryptographic security. It can be reverse-engineered to design files [96]. Furthermore, the design can be modified to gain access to the protected data or the communication by the device. This combination of reverse engineering and modification is commonly known as tampering. Several countermeasures against reverse engineering are presented in [156].

3.3 Malicious IPs

There is another type of attack that makes use of malicious circuitry in the bitstream of an FPGA IP. They are categorized into two types, namely, tampering configured designs and hardware trojans. They are presented in the following sections.

3.3.1 Tampering Configured Designs

To better explain this attack, a short description of bitstream's structure is needed. The bitstream of the target device consists of three sections, namely bus width auto-detection, sync word, and configuration data [145, pg. 154]. The configuration data is composed of commands and data. Examples of configuration commands are no operation, writing configuration registers, or writing memory frames. The configuration memory frames are the smallest addressable segments of the FPGA configuration memory space, and all operation acts upon an entire frame. Each frame is uniquely addressed, which is referred to as frame address. The command specific to writing a frame is called "Write frame address register" (opcode 30002001 in hex) [145, pp. 156].

When the bitstream of IP is generated, it contains frame addresses of the frames (PL resources) that are assigned to the IP. For example, a system developer (SD) is licensing IPs from two different core vendors (CVs) to build a system shown in Figure 3.1. From the system specification, Region IP_1 is reserved for IP_1 and the other region is reserved for IP_2. If the developer of IP_1 generates a bitstream that has "Write frame address register" commands with addresses of the region reserved for IP_2, the configuration process will lead to tampering of IP_2. Similarly, a malicious CV can tamper the static design (see Figure 3.1), which belongs to the SD.

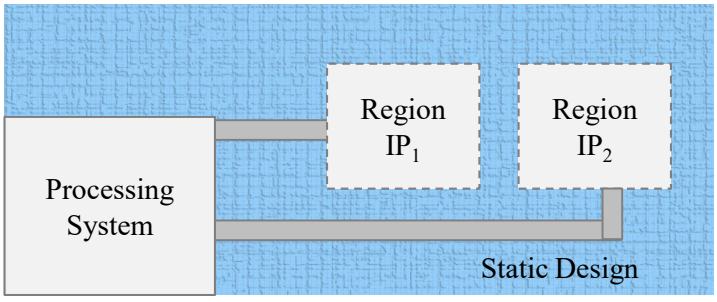

Figure 3.1: Prcoessing system with two IPs on the PL

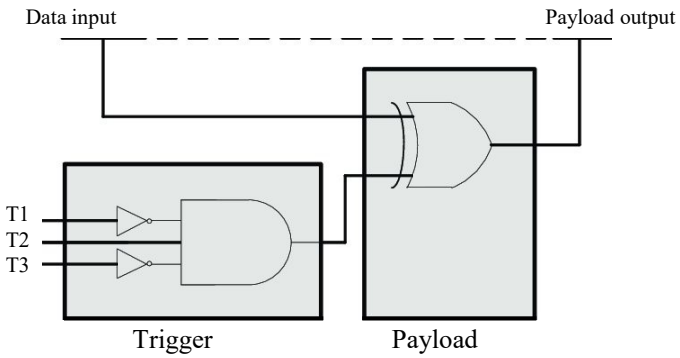

Figure 3.2: Example Hardware Trojan [156]

3.3.2 Hardware Trojans

Here, only those hardware Trojans are considered that are inserted in the bitstream of an IP. They usually have a trigger and a payload circuit, and they are activated under a rare condition making their detection harder by randomizing the input vector, as shown in Figure 3.2 [156]. They can be used to change the functionality, degradation of performance, or information leakage (see [117] for more details). Even though IPs are delivered in authentic encrypted form, which ensures that a man-in-the-middle cannot insert a Trojan in the IP. However, a malicious IP core vendor (CV) can insert Trojans in their IP to launch a software-based power side-channel attacks [161]. These attacks do not require physical access to the device, and with them, an adversary can compromise the security of the device remotely.

Furthermore, Trojans can create logical or electrical conflicts to cause malfunction or damage to the device. This attack was first introduced by Hazdic et al. [162] in 1999. Similarly, FPGA power-hammering attacks can be used to damage the device. They use short and self-oscillating circuits to increase power consumption. The work of La et al. [68] demonstrates that malicious circuits with just 3% of logic on an Ultra96 FPGA board can consume the power that is usually consumed by the entire FPGA.

4 Countermeasures against SCAs

The work described in this chapter was published in [178] and is joint work with co-authors Benjamin Hettwer, and Jürgen Becker. More details on contributions is found in Section 1.5.

This chapter addresses side-channel attacks (SCAs), the first challenge to the FPGA IP market among the ones outlined in Section 1.4. These attacks are a significant threat to the modern cryptographic algorithms that are, in theory, considered mathematically secure. However, the same cannot be stated about their implementations because they often leak secret information about their internal state via side-channels, e.g., power consumption [83], electromagnetic (EM) emission [34], and execution time [62]. Once side-channel information is collected, an adversary can run statistical methods to extract the cryptographic keys. The statistical analysis method can be differential power analysis [64], and correlation power analysis [16].

Over the years, researchers and industry developed several methods to counter SCAs. For instance, *masking* aims to randomize intermediate values that are internally processed by the cryptographic device to break the connection between the secret (respectively some intermediate value that depends on the secret) and its power footprint [83]. In contrast, *hiding* countermeasures are different from masking because their goal is to change the power characteristics directly and thus reduce the signal-to-noise ratio (SNR). There is a particular class of hiding techniques that use DPR available on modern FPGAs to increase the SCA resistance of cryptographic systems [87].

These methods aim for randomizing the structure and execution process of a cryptographic hardware implementation while maintaining functional correctness. DPR-based SCA countermeasures often create only moderate overhead because the employed reconfiguration infrastructure can be reused for other purposes as well, e.g., to build a fault-tolerant system [93, 109].

This chapter presents two countermeasures against SCAs. The first, Target Function Relocation (TFR), relocates the target function using DPR. Furthermore, the target function is implemented using different algorithms with varying power consumption. The selection of the target function's implementation, its location on the FPGA, and the time interval between DPR operations are made using the outputs of random number generators (RNGs). The moving target approach serves as a countermeasure against EM-based measurement setups, while the implementation diversity approach counters power analysis-based attacks. The second countermeasure is a Noise Generator (NG), where different noise modules are implemented at an algorithmic level and are configured on a single location. Here, the selection process is also random.

The proposed countermeasures are scalable, which means the number of locations and implementations can be increased or decreased depending on the use case. Furthermore, several versions of each countermeasure are implemented and evaluated to show the relationship between resource overhead and achieved security.

The main contributions presented in this chapter are:

- DPR is used in several use cases (e.g., accelerators, communication systems), and its support requires additional logic such as a reconfiguration controller. This work proposes to utilize these additional resources to strengthen the security of these systems.
- A complete implementation of both countermeasures is done and presented. An automated flow is implemented that generates different scaled variants of the countermeasures. Furthermore, the implementation diversity part of the countermeasures can be scaled up/down dynamically (See 4.3.7). This makes the design easily adaptable to different scenarios and is a major improvement compared to other similar work.
- The implementation is realized and evaluated on the Xilinx Zynq UltraScale+ MPSoC ZCU102, a state-of-the-art platform for advanced automotive and Internet of Things (IoT) applications based on 16 nm production technology.
- Scaled variants of both countermeasures are evaluated individually and combined to create a more secure system. Furthermore, the effect of scalability on the resource overhead and security strength is presented.

The rest of the chapter is organized as follows: Section 4.1 provides state-of-the-art countermeasures against SCAs. The proposed countermeasures and their implementations are discussed in Sections 4.2 and 4.3, respectively. Afterward, the evaluation is presented in Section 4.4.

4.1 Related Work

The first DPR-based countermeasure against physical attacks has been presented by Mentens et al. [86]. DPR was used to introduce temporal jitter by adding or removing registers between subfunctions of an implementation of the AES. Additionally, spatial jitter could be generated by relocating the subfunctions to four different positions on the chip. However, the number of different configurations that were achieved in that specific approach was comparably low (maximum ten) and thus increased the complexity for a skilled adversary only marginally. Furthermore, the evaluation was only done on paper without performing any power or EM measurements.

In 2011, Güneysu and Moradi exploited the dynamic reconfigurability of selected FPGA components to build a set of generic SCA countermeasures [43]. First, Gaussian noise was generated using lookup tables (LUTs) in shift register mode. Second, a Substitute Byte (SBOX) scrambling scheme was suggested using the dual-port feature of block RAM (BRAM). Third, several Digital Clock Managers were stacked together to create a randomized clock for the cryptographic core. The proposed countermeasures can be easily combined with arbitrary cryptographic implementations.

Sasdrich et al. employed fine-grained reconfiguration of LUTs (i.e., CFGLUTs) to randomize the SBOX of a PRESENT implementation [113]. This was achieved by splitting up the masked-SBOX into two parts with a register in between during runtime. Additionally, the register was pre-charged with the content of dummy encryption to avoid a hamming distance leakage. However, the scheme is only practical for cryptographic schemes with smaller (e.g., 4x4) SBOXs, but not for ciphers with a larger SBOX such as AES.

Hettwer et al. proposed implementation diversity in combination with DPR to protect cryptographic circuits [49]. From a single netlist of an AES implementation, a number of functionally-invariant physically-different circuits were created by randomizing the placement and routing process. The generated configurations are dynamically exchanged during runtime using DPR. However, since the complete AES was reconfigured, the resource and memory overhead is quite large while achieving only an increased SCA resistance of factor three.

The work proposed in [15] is similar to this; however, they did not implement most of their claims. For example, like the proposed work, they claimed to use DPR for the configuration of SBOX regions and randomized the selection of location for the target function, its implementation, and the time interval between DPR operations

(i.e., between consecutive relocations). Instead, they used twelve static versions of the AES engine, i.e., did not implement and evaluate the stated claims. Furthermore, they discussed the usage of relocatable partial bitstreams, which means a single SBOX's partial bitstream can be relocated to any location by changing the frame addresses of the bitstreams. However, it was mentioned that this feature is only part of their future work. In order to simulate the effect of DPR, they mixed up traces from different configurations. The results indicate that it can improve CPA resistance by more than two orders of magnitude. However, the authors stress that a fully operational system is required to determine the actual number of traces needed to break the system, which is not available up to now.

4.2 Proposed Countermeasures

In this chapter, two DPR based countermeasures are proposed that utilizes the concepts of moving target (relocating target function) and implementation diversity. Sections 4.2.1 and 4.2.2 introduce these countermeasures, respectively, while Section 4.3 discusses their implementation in detail.

4.2.1 Target Function Relocation (TFR)

The relocation part of the TFR countermeasure is achieved by having multiple reconfigurable regions (RRs) for the target function of an encryption algorithm. This serves as a countermeasure against location-based SCAs such as EM attacks. The term target function refers to the operation(s) of a cryptographic implementation, which an adversary typically attacks (e.g., the SBOX of an AES in our case). The target function is the main cryptographic operation in the power leakage model of the attacked device (see Equation (4.2)), which is assumed to have a measurable influence on the deterministic part of power or EM traces [83]. Localized EM-based attacks are usually conducted with small probes having a diameter smaller than one millimeter. The probe is moved over the chip using a spot size smaller than the probe diameter, and a certain number of traces are acquired from every position. An attack is then performed for all positions to find the most suitable probe position. Moving the target function randomly to different positions on the chip using DPR makes EM attacks substantially harder because the traces for a single location are mixed up with EM radiations from different logical elements of the cryptographic circuit. In general, if there are l different locations for the target function, it can be expected that at least l times more EM traces are needed to recover the secret key [49].

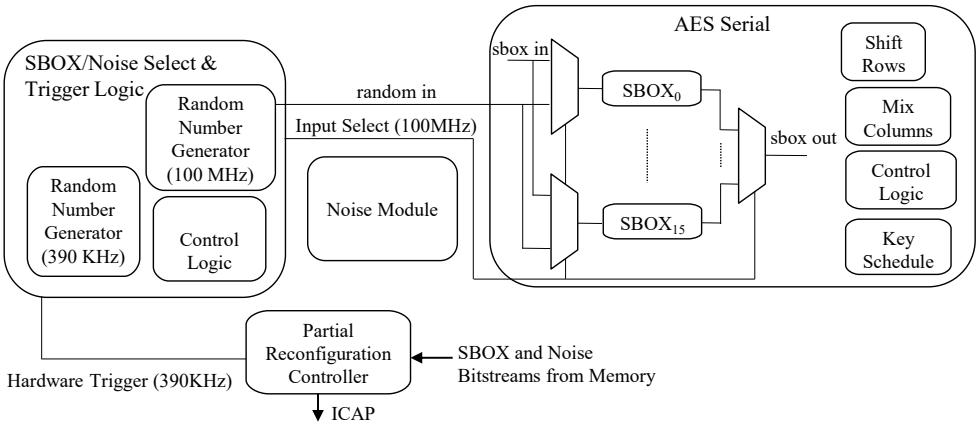

Figure 4.1: Block diagram of the SCA-resistant AES.

In order to achieve implementation diversity, the target function is implemented in several ways, where each implementation generates a different power footprint. The underlying motivation is that the distinct physical layout (placement and routing) of each target function's implementation induces a varying charging capacitance when the implementation is continuously replaced using DPR. This affects the dynamic power consumption of the target function and connected resources (e.g., registers) and counters power measurement-based SCAs. These implementations will be referred to as reconfigurable modules (RMs). RNGs are used to select location (i.e., RR), target function's implementation (i.e., RM), and the time interval between DPR operations. These issues are discussed in detail in Section 4.3.

4.2.2 Noise Generation (NG)

For this countermeasure, the noise generation module can be implemented in several ways and is configured on an RR. In this work, hardware description language (HDL) implementations of a Gaussian noise generator [80], three pseudo-RNGs [19, 70, 106] and two sine wave generators [119, 130] are used as basic noise sources, which are taken from open sources. All noise modules are scaled by cascading several of their instances in series so that they have a significant influence on power consumption. Afterward, all possible combinations of the six basic noises are generated that resulted in 63 different noises, which can be calculated using Equation (4.1):

$$C = \sum_{r=1}^{n} \frac{n!}{k!(n-k)!} \tag{4.1}$$

C represents the total possible combinations, and n is the number of noises among which k noises are chosen for a specific combination.

Like the TFR countermeasure, the selection of a noise RM for configuration and the time interval between DPR operations are made randomly. Both countermeasures are general and can be implemented for any encryption algorithm. Implementation differences of the RMs for the target function and noise modules can be at an algorithm-, synthesis- and/or circuit-level. Algorithm-level differences among RMs mean that the RMs are implemented using different algorithms, e.g., an AES SBOX implemented using LUTs and combinational logic. Synthesis-level differences would be changes in the logic structure, placement, and routing. Circuit-level would be fine-grained changes to specific paths such as clock inputs of flip-flops. The two latter cases are discussed in detail by Bow et al. in [15]. In this work, RMs differ on the algorithmic level that creates variants with a more distinct power footprint than only changing placement and routing. The number of RMs can be easily increased by combining algorithmic, synthesis- and circuit-level methods.

4.3 Implementation

The baseline design used for implementing the countermeasures is an 8-bit serial AES-128 encryption core, which requires 264 LUTs and 253 registers. The target board used in the measurement setup is a Xilinx ZYNQ UltraScale+ system-on-chip (SoC) evaluation board. Since the processing system (PS) is used to deliver plain-text data and key to the core, it is interfaced with the PS using Advanced eXtensible Interface 4 (AXI4). It is also assumed that the target system has a partial reconfiguration controller (PRC) module, which can be used for the configuration of other logic. Therefore, the AXI-interfaced AES and PRC, along with AXI interconnect blocks, are considered the reference design for this work, which requires 3317 LUTs, 4014 registers and 922 CLBs.

The implementation takes 210 clock cycles for single encryption, where each round of AES is computed in 21 cycles. For TFR, SBOX is selected as the target function, as discussed in Section 4.2.1. The reference AES design uses only one SBOX for computing substitute byte functionality for each byte in all the rounds and key scheduling. The block diagram of the SCA-resistant AES is shown in Figure 4.1. The functionality of the sub-modules is presented in the following sections.

4.3.1 AES Serial

AES Serial is a modified version of the reference design. It has 16 SBOXs instead of one, and for each SBOX, an RR is created so that they can be placed on different locations of the programmable logic (PL). However, at any clock cycle, only one of the 16 SBOXs will be used during the operation of an AES encryption. Therefore, multiplexers are added at the inputs and outputs (IOs) of the SBOXs, as shown in Figure 4.1. More details about the selection of an SBOX and its IOs are discussed in Section 4.3.4.

Four different HDL implementations of the AES SBOX are used in this work: a LUT-based implementation, an inverse LUT-based, inverse on Gallois field implementation taken from opencores.org [131] and Canright's implementation of a compact SBOX [18]. Partial bitstreams are generated for each implementation targeting all the RRs. Since there are 16 RRs and four RMs, we have a total of 64 partial bitstreams.

4.3.2 Noise Module

The purpose of this module is the contribution of power consumption noise to the overall design. As mentioned earlier in Section 4.2.2, several HDL-based implementations are taken from open sources [19, 70, 80, 106, 119, 130] and by scaling and combining them, 63 different versions are generated. Here, scaling a noise module means using several instances of the module to create its scaled-up version. An extra RM for the noise module is generated that does not have any design, making 64 RMs. An RR is created for the noise module, and partial bitstreams specific to that RR for all 64 versions are generated.

4.3.3 Partial Reconfiguration Controller (PRC)

Xilinx's PRC is used for the configuration process, which is activated by a hardware trigger. The configuration is done using the internal configuration access port (ICAP). Further details about the usage of dynamic partial reconfiguration (DPR) and PRC can be found in [144] and [147], respectively.

4.3.4 SBOX Noise Select and Trigger Logic (SNTL)

SNTL is the central controller, which applies countermeasures to the AES module. It chooses values for all the parameters of the countermeasures, and afterward, sends necessary control signals to other modules, e.g., hardware trigger to the PRC. This is done using a control logic and two RNGs, namely RNG_{390KHz} and RNG_{100MHz}. Three random values from the RNG_{390KHz}, namely *SBOXImpl*, *TargetLocation*, and *TriggerTime*, are used by the control logic for the TFR countermeasure. The control logic uses

SBOXImpl to select an SBOX bitstream (RM) and configures it on the location indicated by *TargetLocation* value. The configuration process is triggered when a counter inside the control logic reaches the $TriggerTime_{TRF}$ value. For the NG countermeasure, two random values (*NoiseImpl* and $TriggerTime_{NG}$) are taken from RNG_{390KHz}. The control logic uses *NoiseImpl* value to select a noise implementation and $TriggerTime_{NG}$ to generate a hardware trigger to start the (re)-configuration.

Since the configuration process runs in parallel with AES encryption, SNTL is also equipped with the functionality to select an SBOX that should be used by the AES design during the encryption process. As stated earlier, an AES encryption takes 210 clock cycles. For each clock cycle, a different SBOX is selected and is provided with the correct input value (i.e., sbox_in in Figure 4.1). The rest of the SBOXs are given a random value to further reduce the SNR of the correct SBOX calculation. These random values ($SBOXInput_J$) are taken from RNG_{100MHz}. The SBOX selection is also made using a random value (*SBOXSelect*) from RNG_{100MHz}. Since the AES design is running at 100 MHz frequency, an RNG running at the same frequency is used for the SBOX's IO selection and delivering random values to the inputs of all other SBOXes. Also, the PRC may be currently reconfiguring the selected SBOX. In that case, a complement of the *SBOXSelect* value is used for the selection process because the output of the selected SBOX will be invalid.

The SBOX's IO selection and providing random values at the input of non-selected SBOXes happen in a single clock and for each of the 210 clock cycles. In other words, the AES design, including the countermeasures, has the same latency and takes the same number of clock cycles as the reference design.

4.3.5 Bare-Metal Application

A bare-metal application running on the PS moves all the partial bitstreams (RMs) from the non-volatile memory (NVM) to the RAM. Afterward, PRC's registers are updated with the memory location and size of the RMs. AES Serial is then continuously provided with plain-text and key values and reading the ciphertext once encryption is completed. A self-test has been implemented to ensure that the AES core works correctly.

4.3.6 Configuration Times

Since all RRs of the TFR countermeasure are equal in size, the bitstreams generated for them will have the same size, which is 179.5 KB. As mentioned earlier, PRC does the configuration of the bitstreams using ICAP. The bandwidth of ICAP is 3.2 Gb/s at 100 MHz [147], which means PRC will reconfigure SBOX in 0.488 ms. For the NG

countermeasure, there are 64 RMs generated for the same RR, where each is 203.7 KB in size. Consequently, the configuration time for the noise module is 0.509 msec.

4.3.7 Scalability — Variants Generation and Deployment

Both the countermeasures can be implemented in several configurations to generate different variants. Variants can differ based on the number of reconfigurable regions (#RRs), reconfigurable modules (#RMs), or a combination of both that are used in the design. Variants that differ in terms of #RRs require changes to the design flow. This is mainly because for each RR, a dummy design with interface definition needs to be instantiated in the system while connecting it with other sub-designs. Furthermore, a constraint file is required, which provides information about the physical location of the RR. This file can be generated by doing floorplanning of the RRs manually. For further details about these steps or Xilinx's DPR flow, please see [147].

The other case is where variants differ only in terms of #RMs, i.e., the number of #RRs is fixed. This scenario can fully take advantage of the proposed work's scalability feature because the design flow needs to be run only once. The deployed countermeasure can be scaled up or down in run-time, i.e., the rest of the system does not require resetting. For scaling up the countermeasure, only additional partial bitstreams for the additional RMs need to provided to the device using non-volatile memory (e.g., flash). Afterward, PRC's registers need to be updated to support more #RMs. For example, if #RRs is fixed to two and #RMs is initially set to four. The design flow will be run to generate full and partial bitstreams. For each RR, four partial bitstreams will be generated, making a total of eight. This cryptographic design, including the countermeasure, can be configured and activated on the device. In this example, we suppose that the design is scaled up from four RMs to eight RMs per RR while #RRs is kept the same. As stated earlier, only new RMs' partial bitstreams need to be generated and provided to the device. This means the additional four partial bitstreams per RR are provided to the device. Furthermore, PRC'registers need to be updated for the newer variant to take effect. Similarly, if the countermeasure needs to be down-scaled to four RMs, only PRC's registers need to be updated to support fewer #RMs.

The purpose of scaling up the countermeasures is to provide higher security, which leads to higher resource consumption. The resource overhead caused by scaling is discussed in the following section, and its impact on the security strength is presented in Section 4.4.

Table 4.1: Resource consumption overhead and memory requirement with TFR (4 RMs)

Designs	LUT	FF	CLB	Overhead (CLB)	Memory (MB)
AES-Reference	3317	4014	922	-	0
AES-TFR (2 RRs)	3626	4172	991	1.07×	1.43
AES-TFR (4 RRs)	3898	4357	1089	1.18×	2.87
AES-TFR (8 RRs)	4447	4655	1247	1.35×	5.74
AES-TFR (16 RRs)	5556	5252	1422	1.54×	11.48

4.3.8 Scalability — Resource Overhead

As mentioned in the last section, several variants of the countermeasures can be generated by using different values for $\#RRs$ and $\#RMs$. Using variable values for both parameters will lead to a high number of variants. Therefore, one parameter is set to a fixed value, while the other is kept variable to have a reasonable number of variants. This way, the effect of each parameter can be evaluated individually for both resource overhead and security.

For the first countermeasure, TFR, variants are generated by setting $\#RRs = \{2, 4, 8, 16\}$ and $\#RMs = 4$. Table 4.1 shows the resources consumed by each variant in comparison to the reference design. For the second countermeasure, NG, variants are generated by setting $\#RRs = 4$ and $\#RMs = \{8, 16, 32, 64\}$. The resource overhead for each variant for the latter case is shown in Table 4.2.

The presented results show that the proposed countermeasures have a different impact on resource consumption. For example, only 7% of extra resources are required by the TFR countermeasure with 2 RRs, and as the $\#RRs$ grow, the resource overhead also grows linearly. In the case of NG, the resource overhead is 18% for 8 RMs case. However, the NG countermeasure overhead does not grow linearly with the number of noise modules, e.g., 25% for 64 RMs. The results show that the number of $\#RMs$ has a comparatively smaller effect on the resource overhead; however, the same is not valid in the case of memory consumption (see Table 4.2).

Both countermeasures can be easily combined, and their resource overhead will be the sum of their respective overheads. For example, the look-up tables (LUTs), flip-flops (FFs), and configurable logic blocks (CLBs) consumed by combining AES-TFR (16 RRs, 4 RMs) and AES-NG (1 RR, 64 RMs) are 7042, 7889, and 1657, respectively. This is an 80% increase of CLB resources as compared to the reference design.

Table 4.2: Resource consumption overhead and memory requirement with NG (1 RR)

Designs	LUT	FF	CLB	Overhead (CLB)	Memory (MB)
AES-Reference	3317	4014	922	-	0
AES-NG (8 RMs)	4597	6337	1089	1.18×	1.63
AES-NG (16 RMs)	4662	6398	1098	1.19×	3.26
AES-NG (32 RMs)	4680	6482	1113	1.21×	6.51
AES-NG (64 RMs)	4803	6651	1157	1.25×	13.03

As stated earlier, all the partial bitstreams generated for the TFR countermeasure have the same size, which is 179.5 KB. As the countermeasure is scaled, more partial bitstreams will be required, which will lead to higher memory. Similarly, all the partial bitstream generated for the NG countermeasure has the same size of 203.7 KB. The memory requirements for all the variants of TFR and NG countermeasures are shown in Table 4.1 and 4.2, respectively.

4.3.9 Throughput Overhead

As mentioned in Section 4.3.4, AES with TFR countermeasure has 16 SBOXs, and for each of the 210 clock cycles of the AES encryption, the output value of a randomly selected SBOX is used for the computation. It is also ensured that if an SBOX is configured, it is not chosen as its output will be invalid. This whole selection process happens in a single clock without any latency. Furthermore, PRC's configuration process and the selection process run parallel to the AES encryption, which means that the TFR countermeasure does not affect the throughput of the AES encryption.

Similarly, in the case of NG countermeasure, the selection and configuration of noise's partial bitstream happen in parallel to the encryption process. Therefore, the NG countermeasure also does not have any throughput overhead. Finally, the combination of both countermeasures also does not affect the throughput of the encryption process.

4.4 Evaluation

For the evaluation of the security gain of TFR and NG countermeasures, EM measurements from a Xilinx ZYNQ Ultrascale+ MPSoC have been acquired. The EM traces have been collected by measuring the current through a decoupling capacitor of the FPGA's power supply as initially proposed by O'Flynn and Chen [97]. The signal was

captured using an ICR HV500-75 near-field probe by Langer EMV and a Picoscope 6404D oscilloscope with an anti-aliasing low-pass filter. Also, external amplifiers are not used except the one integrated into the probe. The sampling rate of the oscilloscope has been set to 1250 MS/s using a bandwidth of 500 MHz. 100,000 traces have been recorded for several configurations of the design. In order to increase the SNR, an averaging factor of 250 has been applied to each trace. This means that encryption of a certain plaintext has been repeated 250 times, and the average trace of all these encryptions has been saved for analysis. Averaging is a common technique to lower the effect of noise-generating countermeasures such as TFR and NG proposed. After pre-processing, we calculated the Pearson correlation coefficient between the traces and a register transition (hamming distance power leakage) that occurs during the first round in the serialized AES calculation. The hypothetical (power) leakage l model can be represented as:

$$l = HW(SBOX(PT[0] \oplus K[0]) \oplus SBOX(PT[4] \oplus K[4])) \tag{4.2}$$

where HW, PT, K are the abbreviations for Hamming weight, plain-text, and AES key, respectively. $SBOX$ represents the sub bytes operation in the AES calculation, which is chosen as the target function due to its non-linearity property. This is beneficial for SCAs as it introduces many bit-flips in the AES state and significantly affects the circuit's overall power consumption. The registers which hold the AES state are placed directly after the SBOX in both the reference and the protected designs. State transitions of registers (respectively flip-flops) cause the highest instantaneous power leakage in a cryptographic hardware implementation. The leakage of the combinatorial logics (e.g., *AddRoundKey*) is much lower. From an attacker's perspective, it would also be possible to target the mix columns operation. However, this would require guessing 64 bits of the key instead of 16. Such an attack would take much more time (up to several months) and traces than an attack against the SBOX output, even in an unprotected setting. That is why the SBOX is the main target in SCAs, which is protected by the TFR countermeasure in the proposed approach.

Having the results of the correlation analysis, the required number of traces to break the design n_A can be derived by:

$$n_A \approx \frac{28}{\rho^2}. \tag{4.3}$$

Table 4.3: Attack results for different configurations

Design	Security (g_{sec})
AES-Reference	1.0
AES-TFR (2 RRs)	4.53
AES-TFR (4 RRs)	11.45
AES-TFR (8 RRs)	14.26
AES-TFR (16 RRs)	17.95
AES-NG (8 RMs)	6.73
AES-NG (16 RMs)	7.25
AES-NG (32 RMs)	12.11
AES-NG (64 RMs)	40.02
AES-TFR (16 RRs) + NG (64 RMs)	94.9

where ρ is the maximum correlation value for the correct leakage hypothesis [83]. Using this number, security gain g_{sec} is defined as the ratio between the number of traces needed to break a protected design and the number of traces needed to break the unprotected reference design:

$$g_{sec} = \frac{\#\,traces\ protected}{\#\,traces\ unprotected} \tag{4.4}$$

The results for the corresponding designs can be found in Table 4.3. From there, several observations can be made. The protection level increases with the number of configurations used for both countermeasures. For instance, for AES-TFR with two RRs, 4.53 times more traces are needed to break the design compared to the reference AES, while almost 18 times more traces are needed when 16 placement locations (RRs) are used for the target function. The NG countermeasure also benefits from a higher number of configurations, although all partial bitstreams are placed in the same RR. When all 64 available noise RMs are used, the number of traces needed to extract the correct key (byte) is almost 40 times the traces required for the reference design and approximately 36 times more than that required for the design with 8 RMs. As discussed earlier, the resource overhead (in terms of CLBs) only increases by seven percent when scaling from eight RMs to 64 RMs. Combining the fully-fledged versions of TFR and NG (i.e., with 16 RRs and 64 noise modules) gives a security gain of factor 95. This number is much higher than for all other configurations of the proposed design. A reason for this security gain is that the reconfiguration of the target function and NG module is done

in parallel with the encryption, which again is a source of noise and makes SCAs harder.

Our measurement setup requires 120 traces to recover the encryption key from the reference design. However, when both the countermeasures are added, this number increases to 11 200. Although this boundary is still low, the countermeasures can be easily scaled to a higher number of configurations (e.g., by exploiting implementation diversity as proposed in [49]). Since the reference design is broken with few traces, the measurement setup used can be considered a high-quality one, and the presented evaluation a worst-case scenario.

The TFR scheme can also be applied to masked S-Box designs to achieve a higher level of protection. Suppose a first-order masking scheme would give resistance against one million power traces. Then, the TFR and NG countermeasures would require the adversary to acquire probably more than 100 million traces. We expect that such a system composed of masking and hiding would also be resistant against second-order attacks, as it has been shown in related work [114], especially with a higher number of reconfigurable modules.

The resource overhead of the proposed design compared to related work is illustrated in Table 4.4. In contrast to the work of Hettwer et al. [49], the proposed countermeasures achieve a much higher security gain (i.e., 3 vs. 95) while having the same number of configurations. Furthermore, memory and resource overhead is lower, and there is no throughput penalty. Other works based on partial reconfiguration have either no experimental results [86] or are not fully implemented [15]. The masking approaches presented in [43] and [113] can have, theoretically, an unlimited number of configurations (depending on the entropy of the RNG), thus achieving a high level of randomization and protection. However, as mentioned before, scaling improves the security strength of the countermeasures.

Potential future work can be generating more implementations for the target function and noise modules using synthesis- and circuit-level methods [15, 49]. This will increase the amount of memory required for keeping the implementation (partial bitstreams). The overhead can be reduced by implementing a bitstream modification functionality, which will modify the partial bitstream specific to one RR for all the target RRs. However, there are two limitations to this approach. The first is that the source and target RR should have the same number of heterogeneous resources. Secondly, the resources must be organized exactly. If these are done, then modifying the frame addresses in the source bitstream with the target RR's frame addresses will be enough for relocating the partial bitstream. Suppose the static design is using the routing

Table 4.4: Comparison of TFR and NG countermeasure with related work

	# Configurations	Resources[1]			Throughput[1]	Memory[1]
		LUT	FF	BRAM		
[86]	10	*N/A*	*N/A*	0	6.66×	91 kB
[43][2]	*N/A*	1706	1169	8	0	*N/A*
[113]	*N/A*	1236	388	0	3.03×	*N/A*
[49]	128	5270	3986	0	*N/A*	788.48 MB
[15]	12	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Proposed[3]	128	3725	3875	0	0	24.51 MB

[1] Overhead compared to unprotected design

[2] S-Box Scrambling (i.e., Masking)

[3] AES-TFR (16 RRs) + NG (64 RMs)

resources of the target RRs. In that case, a reconfiguration will overwrite the routing of static design inside the target RR. This is the second limitation, which can be solved by using third party tools (GoAhead [10]) that can block routing resources of the RRs. This will restrict the routing of the static design to go through the RRs.

5 Automatic Floorplanning of IPs

The work described in this chapter was published in [177] and is joint work with co-authors Jorge Castro-Godínez, Shixiang Xue, Jörg Henkel, and Jürgen Becker. More details on contributions is found in Section 1.5.

The next challenge for the FPGA IP market is manual floorplanning of IPs on the programmable logic (PL) (see Section 1.4), which is addressed in this chapter. In the FPGA domain, floorplanning of IPs can be defined as the geometrical placement of IPs on the FPGA's PL where corresponding IPs will be implemented. The purpose of floorplanning is to achieve higher density and/or performance. Dynamic partial reconfiguration (DPR), introduced in Section 2.1.4, can be used for the generation of compatible IPs and the static design. However, the flow and tools provided by the FPGA vendors (FVs) do not provide automatic floorplanning of the IPs or RRs. This chapter presents an automatic floorplanning approach that is based on mixed-integer linear programming (MILP). The following section presents the state-of-the-art of floorplanning FPGA-based IPs. Afterward, the floorplanning problem is defined in Section 5.2. The inputs to the problem are resources consumed by the IPs and the target device's layout (see Section 5.3). The problem is solved using an automatic floorplanner whose MILP model is presented in Section 5.4. Finally, the chapter concludes with experimental results and evaluation presented in Section 5.5.

5.1 State of the Art

Floorplanning of IPs on the FPGA fabric is an optimization problem for which reducing the resource wastage and the communication costs between different partial modules are the objective functions [107, 108, 116]. The complexity of this problem has increased over the years. Initial FPGAs presented a grid-like structure of homogeneous CLBs, but modern FPGAs present an irregular distribution of heterogeneous resources considering BRAMs and DSPs beside the CLBs. In addition, designs are also implemented with heterogeneous resources. This increases the complexity of finding a solution where

the location selected for each design satisfies resource requirements while reducing the overall resource usage.

Several academic tools offer floorplanning of IPs on heterogeneous [107, 108, 116], or homogeneous [89, 153] FPGAs. Since modern FPGAs have heterogeneous resource distribution, only the tools that consider heterogeneity can provide a floorplan where resource waste of any type can be kept minimum. Among them, [107] and [116] can provide optimal solutions when the number of IPs is small. However, as this number grows, the solution space grows exponentially. For the latter case, these approaches can only provide sub-optimal solutions.

The floorplanning challenge has been addressed by performing optimization based on MILP [107, 116], simulated annealing [89, 153] and genetic algorithms [108]. The rest of the section is focused on MILP-based approaches as the proposed work is also MILP-based.

In [107], the authors propose two algorithms based on MILP. The first, heuristic optimal algorithm (HOA), proposes to improve over solutions achieved by other approaches based on heuristics [13, 132]. HOA considers a sequence pair representation of the floorplan to fix the geometrical relations between the reconfigurable regions (RRs). The second, optimal algorithm (OA), does not rely on fixed sequence pairs but promises no overlapping among RRs. For OA, this work claims to provide optimal solutions for small IPs performing a full design exploration for the placing problem. When looking for placement of large IPs, OA uses an initial solution provided by HOA. In comparison, OA takes more time than HOA, differing in the definition of non-overlapping constraints: OA is harder to solve than HOA, but it can explore a much bigger search space and find better solutions.

The granularity of reconfiguration defined by the approaches proposed in [107] is a tile, which is the smallest allowed RR. In their work, a tile is a clock region high with the width of a single CLB. The floorplanning was done for Xilinx Virtex-5, which has 20 CLBs (4 BRAMs or 8 DSPs) in a column per clock region. The FPGA fabric is partitioned in portions, which contain tiles with the same type of resources. Regarding the objective functions of the placement optimization, HOA and OA aim to reduce global wire length by decreasing the distance between RRs that communicate with each other and resource waste.

Similar to [107], a recent work, FLORA [116], proposes a MILP approach with a fine-grained modeling strategy of the FPGA resources. Abiding by partial reconfigu-

ration (PR) constraints, defined by the FPGA manufacturer, this work reports better performance than the optimal approach OA presented in [107]. However, it is not clear how the implementation of FLORA surpasses the results provided by OA [107]. FLORA states that it is faster because the optimization problem is better formulated, and better solutions are found because of their fine-grained FPGA model. However, the granularity of 1 CLB width and one clock domain height is presented as the one used throughout the paper, which is the same as used in [107].

On the other hand, authors in [108] presented improvements over their MILP work [107]. This work addresses the floorplanning in two phases: In the first phase, an exhaustive exploration is performed to generate the design space, and a conflict graph is used to define the feasible FPGA sections that can be used for the placement of the PR blocks. These sections are described in terms of the actual physical position of the FPGA resources in the entire FPGA floorplan. In the second phase, the design space is explored using a genetic algorithm (GA). The results provided by the GA are then enhanced with a local search function based on the steepest descent heuristic, achieving local optimal solutions for a given input solution space. This work claims to find global optima by using this mixture of GA and local search.

The proposed floorplanner is an improved version of the OA floorplanner presented in [107]. Major improvements are related to complex fine granularity and supporting recent FPGAs, such as Xilinx UltraScale+ architecture.

5.2 Problem Definition

Finding the best geometrical placement for a set of RRs, $R = \{R_1, ..., R_n\}$, within the total available area on the FPGA fabric is an optimization problem where the objective is to reduce resource waste. Each RR is characterized by a vector $c_{i,t}$, where $c_{i,t}$ denotes the number of resources required by R_i for each type $t \in \{CLB, BRAM, DSP\}$.

The inputs for this problem are:

1. a representation of the FPGA fabric. In other words, a description of the resource distribution on the FPGA area, i.e., its layout and the pattern in which the resources are distributed (See Section 5.3);
2. resource utilization report of the partial modules, which can provide the necessary information regarding resource requirement for each partial areas: $c_{i,t}$;

Regarding required resources, the resource utilization report of each IP provides information about how many LUTs, flip flops (FFs), CARRY blocks, DSPs, and BRAMs

are needed. For BRAMs and DSPs, values from the report can be used directly. However, the minimum number of CLBs for an IP needs to calculated using Equation (5.1) because each SLICE of the UltraScale+ FPGA contains 8 LUTs, 16 FFs, and 1 CARRY8 Block.

$$n_{CLB} = ceiling(max(n_{LUTs}/8, n_{FF}/16), n_{carry}/1)) \tag{5.1}$$

The solution to the problem is a floorplan that has the left-most, bottom-most coordinates: x_i, y_i and the width, height: w_i, h_i for each RR R_i. This floorplan can specify a selection of block shapes and overlap-free placements of physical blocks.

5.3 Device Representation

This section presents the target device's layout and defines its representation while focusing on the trade-off between resource wastage and ease in finding an optimal floorplan solution. The target device is a Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC. This platform integrates a feature-rich 64-bit quad-core (or dual-core) ARM Cortex-A53 and dual-core ARM Cortex-R based processing system (PS) and Xilinx PL UltraScale+ architecture in a single device.

5.3.1 FPGA Layout

The FPGA layout of the target device is divided into 7 rows. The top 4 rows have 4 columns while the rest have 3, which makes a total of 25 rectangular clock regions, as shown in Figure 5.1. In the layout, resources such as CLBs, DSPs, BRAMs, and interconnects are distributed column-wise, i.e., each column over all the rows has a specific resource type. For instance, the clock region X2Y0 is shown in Figure 5.2, which

<table>
<tr><td>X0Y6</td><td>X1Y6</td><td>X2Y6</td><td>X3Y6</td></tr>
<tr><td>X0Y5</td><td>X1Y5</td><td>X2Y5</td><td>X3Y5</td></tr>
<tr><td>X0Y4</td><td>X1Y4</td><td>X2Y4</td><td>X3Y4</td></tr>
<tr><td>X0Y3</td><td>X1Y3</td><td>X2Y3</td><td>X3Y3</td></tr>
<tr><td rowspan="3">Processing System</td><td>X1Y2</td><td>X2Y2</td><td>X3Y2</td></tr>
<tr><td>X1Y1</td><td>X2Y1</td><td>X3Y1</td></tr>
<tr><td>X1Y0</td><td>X2Y0</td><td>X3Y0</td></tr>
</table>

Figure 5.1: Full PL layout of Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC

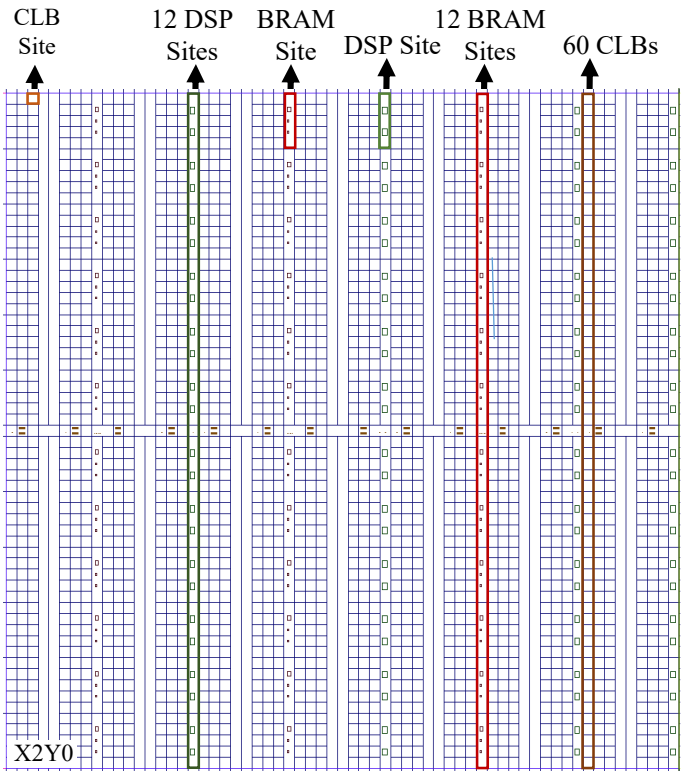

Figure 5.2: Clock Region X2Y0

has several columns of resources. Each of its columns consists of either 60 CLB, 12 DSP, or 12 BRAM sites.

5.3.2 Grid Reduction and Granularity

For the target device architecture, Xilinx Ultrascale+, the smallest allowed RR varies based on the resource type and is called programmable unit (PU). PUs for the considered resource types are described below and shown in Figure 5.3. Further details can be found in [147].

- CLB PU: Two adjacent CLBs and the shared interconnect.
- BRAM PU: One BRAM site, five adjacent CLBs, and the shared interconnects.
- DSP PU: One DSP site, five adjacent CLBs, and shared interconnects.

Since a single CLB PU can be reconfigured, the granularity for the floorplanning can be set to 1 CLB PU (1 BRAM or DSP PU). Such a fine granular floorplanner will expand

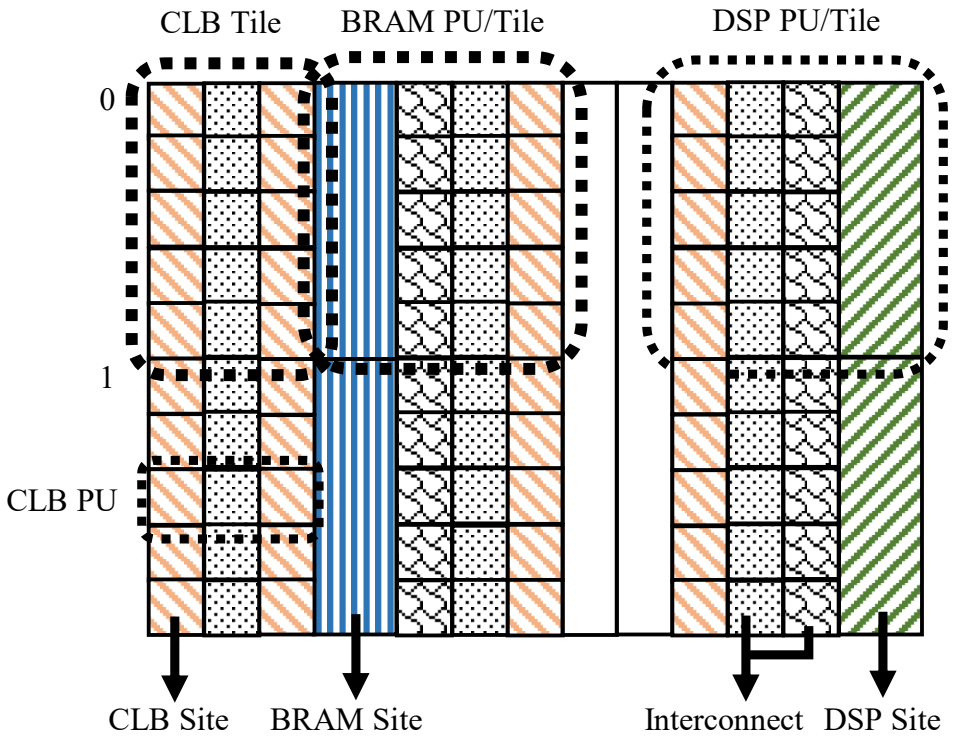

Figure 5.3: Part of the layout of Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC. Smallest PUs with dotted line.

the solution space, and finding an optimal (or a sub-optimal) solution would take too long. Therefore, 5 CLB PUs, 1 BRAM PU, or 1 DSP PU is chosen as the smallest allowed RR for the proposed floorplanner and will be referred to as a tile. Furthermore, a DSP or BRAM PU is height equivalent to that of 5 CLB PUs, as shown in Figure 5.3, which will make the FPGA layout model consistent.

This granularity can be compared to other floorplanners [107, 116], where an entire column of a clock region was considered as the smallest PU. On the target device, this would be 120 CLBs (or 60 CLBs+12 DSPs sites). Considering the worst-case example, if the resource requirement of a reconfigurable design is 121 CLBs, floorplanners [107] and [116] will create an RR of 240 CLBs, with a resource waste of 49.5%. In comparison, the proposed floorplanner will assign an RR of 130 CLBs with a resource waste of only 6.9%;

5.3.3 FPGA Partitioning

The next step in the device representation is dividing FPGA resources into portions. Each clock region has a grid-like structure similar to that of clock region X2Y0 (see Figure 5.2). Also, along the layout's height, columns inside the clock regions are of the same resource type. Figure 5.1 shows the layout where the clock regions in the first column are X0Y6, X0Y5, X0Y4, and X0Y3. Columns 4 and 6 inside these clock regions are of CLB type, and column 5 is the shared interconnect between them. These three columns (4-6) are grouped to create the first portion (*POR1*). *POR1* has a total of 240 CLB PUs that is equal to 48 CLB tiles. Similarly, *POR2* consists of columns 7 to 10, where column 7 is composed of BRAMs, column 10 of CLBs, and the middles ones are dedicated BRAM, and general interconnects. A total of 67 portions are required to characterize the PL part of the target device. The target device model is made available at [58].

5.4 MILP Modeling

This subsection presents the model of the device and the floorplanning algorithm that is based on MILP. Parameters and variables are defined first, which is followed by problem linearization, MILP constraints, and the objective function. This section ends with a comparison of results against related work.

5.4.1 Constants definition

Following are the constants of the model.

P := set of portions;
F := set of forbidden areas;
R := indexes of the rows. Each row is a clock region high, numbered from 1 to 7;
N := set of RRs;
$c_{n,t}$:= resource of type t required by n^{th} RR;
W := width of the FPGA in terms of columns;
$px1_i$:= left-most coordinate of a portion;
$px2_i$:= right-most coordinate of a portion;
$d_{p,t}$:= resources of type t in a tile within portion p;
t := resource type $\{CLBs, BRAMs, DSPs\}$;

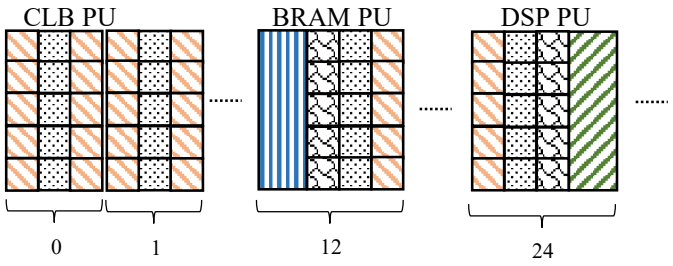

Figure 5.4: Index of PUs for POR1

5.4.2 Control Variables

Control variables are the variables that define an RR. For each RR $Ri \in R$, we define the following variables:

$a_{n,r}$:= set to 1 if region n occupies row r, binary variable;
n := index of a RR, $n \in Z+$;
x_n := left-most position of n^{th} RR, $x_n \in Z+$;
w_n := width of a n^{th} RR, $w_n \in Z+$;
$y1n$:= lowest row occupied by n^{th} RR, $y1n \in Z+$;
$y2n$:= highest row occupied by n^{th} RR, $y2n \in Z+$;

5.4.3 Problem Linearization

Floorplanning is treated as a linear problem by indexing both axes of the layout. Along the y axis, each clock region is considered as a row, and they are indexed from 1 to 7. For the x axis, the portions are composed of tiles, smallest RRs, and are indexed as depicted in Figure 5.4. Some columns do not contain any logic resources, such as input-output (IO) columns. The indexing of portions can be seen as stretching the layout horizontally consisting of smaller columns where each column is a tile (5 CLB PUs or 1 BRAM/DSP PU). This is shown in Figure 5.4. The final layout can be seen as a grid of 7 rows and 1560 columns. The resource calculation of a given width w inside a single portion p_i can be obtained by a linear function:

$$f_i(w) := w \cdot d_{p,t}. \tag{5.2}$$

5.4.4 Constraints

Constraints that enforce the correct behavior of the MILP model are listed below.

- An RR must occupy consecutive rows, i.e., RR should occupy row 2 if it has occupied row 1 and 3.
- An RR must occupy consecutive columns, i.e., RR should occupy column 2 if it has occupied columns 1 and 3 in a portion.
- An RR should stay within the horizontal limits of the overall PL layout.
- An RR should stay within the vertical limits of the overall PL layout.
- The highest row occupied by a region should be greater than or equal to the lowest row occupied by the region.

$$y2n \geq y1n \tag{5.3}$$

- Each RR must contain all the needed resources for each resource type.

$$\forall t \in T, \tag{5.4}$$

$$\delta(n,t) \geq c_{n,t} \tag{5.5}$$

- Two RRs cannot overlap each other in the same row.
- Two RRs cannot share a column within a clock region.

5.4.5 Objective Functions

Multiple objectives can be used while creating an optimal floorplan. However, here the focus is only on resource waste. Wasted Resources (WR_{cost}) is the difference between the resource required by the partial modules and the resource occupied by the corresponding RRs.

$$WR_{cost} = \sum_{n=1}^{N} \sum_{t \in CLB,BRAM,DSP} k_t \cdot (\delta_{n,t} - c_{n,t}), \tag{5.6}$$

k_t denotes the weight factor associated with wasting a resource of type t, $\delta_{i,t}$ the resource footprints inside of a R_i and c_i, t the resource requirement by the partial module.

Table 5.1: Comparisons between proposed floorplanner and the one report in [107] for Xilinx Virtex-5 XC5VLX110

# of	[107]		Proposed	
IPs	WR_{cost}	Solution Type	WR_{cost}	Solution Type
5	(1.71%)	Optimal	117 (0.97%)	Optimal
10	0	Optimal	0	Optimal
15	111 (0.91%)	Sub-Optimal	0	Optimal
20	101 (0.83%)	Sub-Optimal	0	Optimal
25	91 (0.75%)	Sub-Optimal	0	Optimal

5.5 Experimental Results and Evaluation

The proposed solution has been implemented in MathProg, a subset of the AMPL language, and solved with Gurobi Optimizer [46]. Experiments were executed on an AMD Ryzen 9 3900X 12-Core Processor (24 CPUs 3.8GHz) with 64 GB RAM. Comparisons between the proposed approach and the one reported in [107] were performed by using the same synthetic test suite used in [107], and they are available at [58]. The benchmark consists of pseudo-random circuits with several RRs in the range of {5, 10, 15, 20, 25} where each occupies 70% resources on the target device of [107], i.e., Xilinx Virtex-5 XC5VLX110. All the RRs require CLBs, 3 to 7 require BRAMs, and 1 or 2 require DSPs. The evaluation metric is the resource waste (WR_{cost}) calculated using Equation (5.6), where weight k_t for each resource type $t \in \{CLB, BRAM, DSP\}$ is set to 1. This implies that WR_{cost} is the number of unused resources in an RR. The stopping criteria of the search are either reaching the optimum or a time limit of 3600 seconds.

A comparison between the proposed floorplanner with the floorplanner from [107] for the Xilinx Virtex-5 XC5VLX110 device is presented in Table 5.1. We decided to compare our floorplanning approach against the one reported in [107], as it presented complete details for re-implementation. Another work proposed improvements over [107], in [116], but does not provide enough details for re-implementation and to perform a fair comparison. The improvement achieved by the proposed floorplanner in resource waste relies on the assumption that the Xilinx Virtex-5 device allows the smallest allowed PU/RR of 10 CLBs/DSPs/BRAMs. However, the smallest allowed PU/RR on this device is an entire column [101] for each resource type $t \in \{CLB, BRAM, DSP\}$ and the results are only presented to show the effect of fine granularity.

Table 5.2: Comparisons between proposed floorplanner and the one report in [107] for Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC, Runtime 3600 Sec.

# of IPs	Resource Waste (%)		Solution Type
	[107]	Proposed	
5	549 (4.53%)	407 (3.36%)	Sub-Optimal
10	551 (4.51%)	270 (2.22%)	Sub-Optimal
15	637 (5.18%)	134 (1.10%)	Sub-Optimal
20	483 (3.95%)	29 (0.23%)	Sub-Optimal
25	469 (3.86%)	73 (0.06%)	Sub-Optimal

For Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC, the target device for the presented floorplanner, a device model for [107] and proposed floorplanner is created. This device has more resources, complex resource distribution, and a smaller allowed RR. Therefore, an optimal solution cannot be found for any of the cases in 3600 seconds. Therefore, Gurobi Optimizer is run while setting parameter *MIPFocus* to 1. *MIPFocus* parameter, when set to 1, makes the optimizer find a good quality sub-optimal solution rather than focusing on the optimal one [45]. For finding the optimal solution, MIPFocus can be set to 2, and the solver needs to be run for a longer period.

A comparison between our proposed floorplanner and the one reported in [107] using the same benchmark is presented in Table 5.2. This shows a clear improvement in our approach over the reported work [107]. It is also important to mention that the benchmark is used without scaling it for the target device. Since the target device is approximate twice the size of Xilinx Virtex-5, the benchmarks roughly cover 30% of the PL. The proposed and reported floorplanner could not find any solution for any cases when the benchmark is scaled to occupy 70% of the device. Furthermore, the 5-IPs use case of the benchmark was run for a longer period to find an optimal solution. However, even after approximately 27 hours of run-time, the floorplanner could not find a better solution than the one found in 3600 seconds. Both of these cases can be attributed to the large solution space. For the latter one, it can be assumed that the optimal solution will have a minor improvement over the sub-optimal solution and requires more effort than the benefit. However, these challenges can be solved by adjusting the floorplanning to find a solution by only focusing on a few clock regions, which is shown for the use case presented in Section 8.3.2.

As mentioned in Section 5.3.2, the granularity of [107] is 40 CLBs, and that of the proposed floorplanner is 10 CLBs. Furthermore, each IP in the synthetic benchmarks

Table 5.3: Earlier found sub-optimal solutions by the proposed floorplanner for Xilinx Zynq UltraScale+.

# of IPs	Resource Waste (%)	Run-time (seconds)
5	3.56%	62
10	2.41%	207
15	1.30%	216
20	0.28%	266
25	0.25%	370

has a multiple of 40 resources for each IP. Since these IPs fit perfectly on the columns of the Virtex-5 device, the resource waste is lower for both the floorplanners. However, in the case of the target device, shown in Table 5.2, these IPs do not fit the column, which can be seen in the higher resource waste for [107]'s floorplanner.

Even though the proposed floorplanner was run for 3600 seconds, solutions presented in Table 5.2 were found earlier. For example, for 5-, 10-, 15-, 20-, and 25-IPs, the reported sub-optimal solutions were found in 1550, 2774, 2105, 3440, and 976 seconds. Furthermore, other good quality solutions were found even earlier with a negligibly small difference in resource waste compared to the ones found in 3600 seconds. They are presented in Table 5.3. In conclusion, the proposed floorplanner can be used to find good quality sub-optimal solutions within a few minutes.

The proposed floorplanner can also work with other devices, even with the devices from other FVs. For this, devices need to be modeled while considering the supported granularity of partial reconfiguration. Floorplanning is solved with Gurobi Optimizer [46], which is used as a black box, where both the floorplanning and the device model are inputs. Other researchers can replicate the work by implementing the floorplanner using the variables and constraints defined in Section 5.4. Device models for Xilinx Virtex-5 XC5VLX110 and Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC are made public [58], and they can be used as a reference for modeling other devices.

6 Standalone Generation of IPs

The work described in this chapter was published in [177] and is joint work with co-authors Jorge Castro-Godínez, Shixiang Xue, Jörg Henkel, and Jürgen Becker. More details on contributions is found in Section 1.5.

The third challenge, among the ones presented in Section 1.4, is the generation of IPs independent of the overall (static) design. Use cases such as IP licensing schemes and ASIC prototyping require that the participants can generate their end products (bitstreams) independent of each other's design, i.e., without sharing designs to avoid reverse engineering [11, 95]. Furthermore, these end products need to be compatible with each other. It is also equally important that the bitstreams should be generated using FV tools because companies that are consumers of the IPs (SDs) avoid using end products generated with non-certified or tampered tools. Therefore, the following work is focused on FV tools and third party tools that work along with FV tools to generate IPs.

6.1 Requirements

As mentioned earlier, FPGA vendor tools do not support generation of IPs independently. To explain the reason behind this limitation, several requirements are defined and presented below.

1. Devices and corresponding tools must support partial reconfiguration.
2. Reconfigurable designs (RDs) should not use any placement resources outside its reconfigurable region (RR).
3. RDs should not use any routing resources outside its RR.
4. Static design should not use any placement resources inside the RRs.
5. Static design should not use any routing resources inside the RRs.

It is important to mention that if a tool satisfies all these requirements, it will support the generation of IPs independent of the overall design. The requirements are general and can be applied to other tools; however, they are only discussed for the target device and corresponding FV tools in the following section.

Major FPGA vendors fulfill the first requirement. For example, PR is supported using Vivado for all Ultrascale^{TM}, Ultrascale+^{TM}, and nearly all 7 Series devices (see [147, pp. 6-7]). Requirements 2 and 3 are specific to the RD (IP) using only the resources inside its RR, and major FV tools support them (see [147, pp. 34, 75]). Even though requirement 4 is supported by FV tools, 5th one is not (see [147, pp. 34]). This means the routing of the static design is going through the RR, and an IP generated for that RR without the static design's routing information will not be compatible with the static design. In other words, if such IP is configured on the FPGA, it will tamper with the static design's routing.

To solve this problem, we investigated third party tools whose results can be imported into the FV tools. Here, the focus is blocking all the routing resources inside an RR so that these resources are not used by the static design. A discussion on third party tools is presented in the following section.

6.2 Third Party Tools

Major FV devices and tools support DPR, which can be used to implement IPs. However, these tools work on an incremental design methodology, e.g., static design needs to be implemented (where the placement and routing will be preserved) before implementing IPs. Hence, there is a need for a tool or a flow that can generate IPs with no information, or very little, about the static design.

Among the third party tools, only GoAhead [10] has been reported to assist in the generation of IPs for an IP licensing scheme [133]. GoAhead uses the notion of planning to define what belongs to the static design and what part of the design will be placed in reconfigurable containers, particularly by automating low-level design aspects of the process. This tool is more focused on the efficient utilization of a DPR flow to improve re-usability and resource utilization.

For static design, GoAhead creates all the required VHDL templates and physical implementation constraints. These constraints prevent the usage of the resource primitives inside a RR by the static design. Blocking static design from using placement resources within an RR is easy, and it is supported by FV tools, e.g., Xilinx Vivado Design

Suite and Intel Quartus Prime. However, doing the same for the routing resources is a much harder problem and is not supported by FV tools. To achieve this, GoAhead can be used to create blocker macros for all the routing resources inside the RRs. The macros show that the resources are occupied, and this way, routing of the static design can be avoided to go through the RRs. Blocking of the routing and placement resources has also been used by ReCoBus-Builder [60] and OpenPR [122]. However, all these tools are based on xilinx development language (XDL), which Xilinx no longer supports for its current FPGAs. Therefore, these external tools are incompatible with recent Xilinx devices (such as UltraScale, Ultrascale+, and beyond) [128]. Furthermore, blocking all the routing resources within all the RRs can make overall routing extremely hard.

Recently another tool, called RapidWright, has been proposed to work with Vivado to produce highly customized implementations [73]. RapidWright constitutes the progression of previously proposed tools RapidSmith [71, 72] and RapidSmith 2 [48]. This tool can generate pre-implemented modules, such as partial IPs, by using Vivado to synthesize, place, and route them. However, considering its documentation, RapidWright can not generate partial bitstreams independent of the static design.

Blocking of routing and placement resources can be done by ReCoBus-Builder [60], OpenPR [122] and GoAhead [10]. As mentioned earlier, these tools do not support recent Xilinx devices [128]. However, even if these tools are further developed to support recent devices, the solution will have the following limitations:

- If most of the PL resources are used by IPs, all the blocked routing resources inside RRs will cause Vivado's router to fail while routing the static design.
- The same problem will occur if multiple IPs are floorplanned close to each other. This would require iterative intervention into the floorplanning model.
- Floorplanning of the RRs can be adjusted in a way that enough routing resources are available for the static design's routing by providing gaps between the RRs. However, this will make the floorplanning problem even harder to solve and will increase resource waste.

Because of the mentioned limitations and the implied performance disadvantages, we propose a different approach where only FV tools can be used to generate IPs and SD's design independent of each other. Even though the discussion is specific to Xilinx tools, the approach can easily be used for devices from other FVs, provided they support partial reconfiguration.

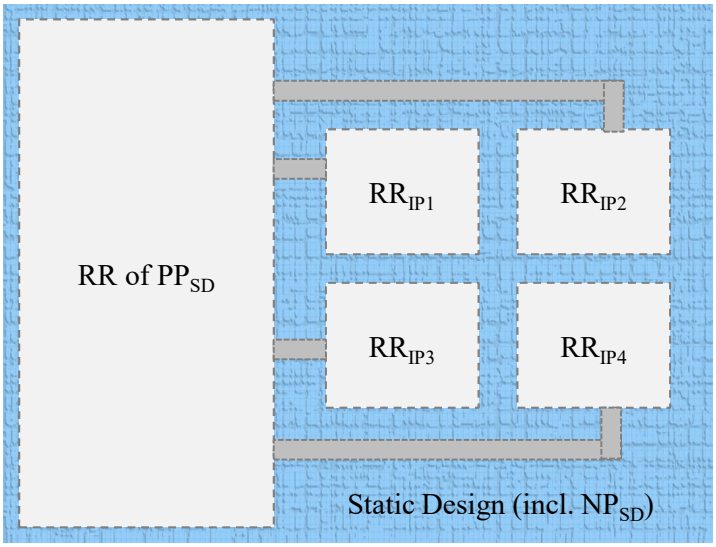

Figure 6.1: Floorplanned static design including SD non-protected design and all the RRs

6.3 Proposed Solution using FV Tools

This section presents the concept of the independent IP generation (IIPG) flow. The flow can be implemented using any FPGA vendor tools that fulfill the requirement 1, 2, and 3 presented in Section 6.1. The latest devices and tools from the two major FVs, namely Xilinx and Intel, fulfill these requirements; therefore, the flow can be implemented for their devices. Since the implementation and evaluation of this work is targeted for a device from Xilinx, their terminologies and DPR flow is used while presenting the IIPG flow. A short description of the DPR flow is presented in Section 2.1.4, and more details about it can be found in [147].

The IIPG flow is explained by showing interaction among the participants of the FPGA IP market, which was introduced Section 1.1. Using IIPG, multiple IP core vendors (CVs) and a system developer (SD) can generate bitstreams of their designs independently. Furthermore, these bitstreams (designs) will be compatible with full bitstream (overall FPGA design); and can be configured on the RRs of the already configured full bitstream. The IIPG flow is implemented using a tool command language (TCL) script that performs DPR using Vivado. The individual steps of the flow are shown in Figure 6.2, and are:

1. CVs publish the functionality, supported interfaces, and resource consumption of their IPs. The information is made available on an IP store where SDs can search for the IPs they require.

2. An SD selects IPs from the IP store and acquires their information $Info\{IP_J\}$.

3. SD partitions its design into two parts: the protected part (PP_{SD}) and the non-protected part (NP_{SD}). PP_{SD} consists of all those functionalities that SD wants to protect from getting shared with CVs. NP_{SD} part is the functionalities, which are either open source or not critical enough to be protected.

4. SD creates a static design in which wrappers with desired interfaces and port definitions are defined for all the selected IPs. A similar wrapper is created for the PP_{SD}. NP_{SD} is part of the static design; however, all the IPs and PP_{SD} are instantiated as reconfigurable designs (RDs). RDs in the DPR flow only require the interface and port definition.

5. RRs are created according to the resource requirement of all the RDs, and RDs are assigned to them. RRs of CVs' design and SD's protected design are shown in Figure 6.1

6. Floorplanning of the RRs is done using the fine-grained floorplanner, which is presented in Section 5. The static design is also shown in Figure 6.1.

7. The created static design has SD's non-protected design and all the connections among the modules. SD shares the static design with all the CVs.

8. SD generates bitstream of the static design and PP_{SD} from static design. Since PP_{SD}'s bitstream is generated from the static design, it will be compatible with the full bitstream.

9. Each CV generates bitstreams of their respective IPs using the static design. IPs' bitstream will also be compatible with the full bitstream as they are generated from the same static design.

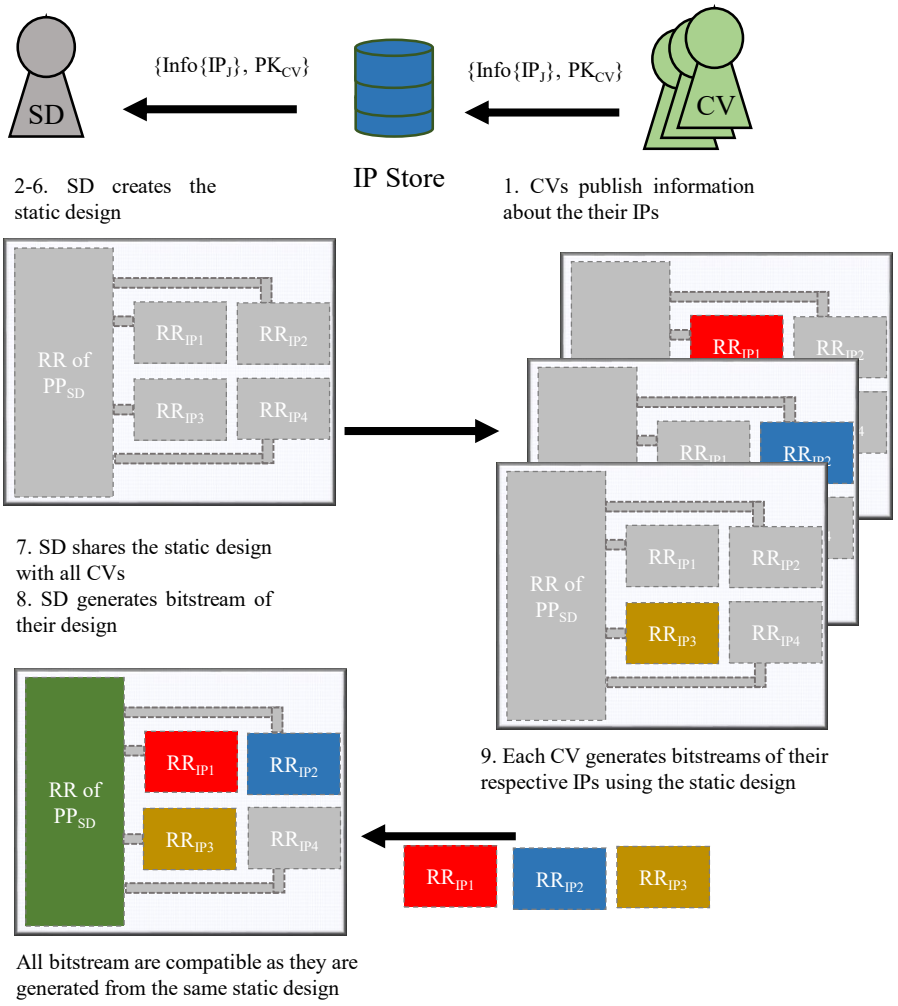

Figure 6.2: Independent IP Generation (IIPG) Flow

7 IP Licensing Schemes

The work described in this chapter was published in [179] and is joint work with co-authors Sven Nitzsche, Asier Garciandia López, and Jürgen Becker. More details on contributions is found in Section 1.5.

This chapter presents a novel, feasible and secure IP licensing scheme for FPGA devices to address the remaining challenges presented in Section 1.4. The following section evaluates existing proposals in the FPGA IP market. Furthermore, several features or limitations are defined in it, and the related schemes are summarized based on these features in Table 7.1. Afterward, basic prerequisites and assumptions are explained in Section 7.2, threat model in 7.3, and the Trusted Platform concept is discussed in Section 7.4. Finally, the proposed scheme is presented in Section 7.5.

7.1 Existing Proposals in the FPGA IP Market

The Virtual Socket Interface Alliance published a technical paper that outlines three methods of securing the virtual components called deterrence, protection, and detection. Deterrence is the use of legal systems to prevent the infringement of IPs, e.g., patents, copyrights, trade secrets, contracts, and lawsuits. In a protection approach, CVs take active measures to prevent illegitimate use of their IP, e.g., licensing agreements and encryption. In a detection approach, CVs detect the theft origin after unauthorized use of their IP has been discovered, e.g., IP watermarking [154, 163] and fingerprinting techniques [17, 20, 69]. Among them, protection methods [25, 57, 82] are more effective because they make IP infringement difficult and costly. The proposed work focuses on protection; hence the state-of-the-art discussion will concentrate on this category.

In 2001, Kean [56] proposed that a secret key in a non-volatile memory and an encryption circuitry on the FPGA can protect FPGA's configuration data (bitstream) against piracy and reverse engineering. The main attack considered for that work was monitoring

the wire that transfers bitstream from serial erasable programmable read-only memory (EPROM) to an FPGA. The encryption circuitry will encrypt the bitstream before it is stored on the serial EPROM. On startup, the encrypted bitstream will be loaded, decrypted, and then configured on the device. The following year, he proposed an IP protection scheme [57], where a secret key is stored in the FPGA that is only known to the device. Also, each device has a unique identifier. The secret key is used to generate security tokens using the device ID and a user key, which the TTP provides. Using this scheme, TTP can create tokens that allow secure communication with the FPGA once it leaves its possession. One of the major drawbacks of this scheme is the use of design files that leads to placing higher trust in the TPP. Also, it requires support/customization of the FV tools.

Simpson et al. [120] presented a software IP authentication scheme using a physically unclonable function (PUF). One of the work's weaknesses is not implementing a PUF instead assuming that a secure one exists. Secondly, assuming that encryption can be used for bitstream security without getting into details can be considered another shortcoming of the work. Similar schemes are proposed by Guajardo et al. [41, 66] while focusing on implementing reliable and secure PUFs. On SW-IP protection using PUF, Gora et al. also presented their design flow in [39]. They proposed to use an integrity kernel that validates a security kernel, which is then used to perform the decryption of SW-IP. The problem with this approach is that it will restrict the SD from using their device's PS. However, if restrictions are reduced, a malicious SD can reverse engineer the kernel to perform IP theft attacks. Furthermore, the integrity kernel's binary is stored in FPGA's configuration memory, which can be read out using the readback attack (see Section 3.1.4) and then reverse engineered to a netlist [11, 95].

There are other schemes [25, 42, 54, 82, 157, 158] where a TTP stores a unique key on the device and provides a core installation module (CIM), in encrypted form, that supports the metering process. Güneysu et al. [42] proposed it first and used it for the public-key cryptography-based key-agreement protocol between the CV and the device. An extension of the proposal for the protection of multiple cores was presented in [25]. Others [54, 158] proposed minor improvements such as reducing complexity and not requiring any hardware modification to the existing devices. Later, [82, 157] proposed more structured IP licensing schemes utilizing CIM to decrypt IP cores with symmetric cryptography. For the protection of IP cores, most of the schemes rely on a TTP, and a straightforward way would be that TTP is responsible for the programming of keys on the device and encrypting IP cores. However, this would require placing a higher trust in the TTP. The usage of CIM was mainly to reduce the degree of trust in the TTP. Even though CIM usage could cause PL resource waste but this can be

avoided by reusing these resources once CIM has served its purpose [157]. CIMs only offer an additional reverse engineering step in terms of security, as they can be read out using the readback attack. Afterward, they can be reverse engineered to a netlist [11, 95], which will lead to the extraction of keys stored in them.

Then, there are other schemes [41, 100, 120, 124, 155] that used PUFs for generating secret keys to avoid placing trust on the TTP. If reliable and secure PUFs are used, that will consume higher resources on the PL. Even though recent devices (e.g., the Xilinx Zynq UltraScale+ MPSoC ZCU102) are equipped with them; however, this would mean placing higher trust in the FV. Another IP licensing scheme [176] was proposed by the author that provides protection against IP theft without relying on any TTP. In it, a software application is used for the decryption and configuration of the IP cores. The application is protected using a hardware-assisted software protection solution that uses a tamper and side-channel attack (SCA) resistant hardware component. However, the scheme suffers from readback attacks, discussed in Section 3.1.4, restricted usage of the PS or kernel tampering attacks.

7.1.1 Features or Limitations

For SotA comparison, here, several features are presented. The first two features are based on two limitations that are not addressed by almost all the IP licensing schemes and are explained in detail in Chapter 5 and 6, respectively. The rest of the features were defined in [179]. A short description of each feature is given below, and SotA comparison using these features is presented in Table 7.1.

- Independent IP Generation (IIPG): Whether the scheme addresses or supports the generation of IPs/RRs independent of the overall design.
- Automatic Floorplanning of IPs/RRs (AF): Whether the scheme addresses or provides support for the floorplanning of IPs/RRs on the PL.
- Readback Attack Resistance (RBAR): Whether the scheme is resistant to readback attacks or not.
- Minimal Hardware Modification (MHM): Whether the scheme can use commercially available devices without any modification or not. Modifications of devices can be adding additional security components; however, they should be avoided. |They make a scheme inapplicable for available devices, and therefore, will not be adopted by the industry.
- Malicious Functionality Detection (MFD): Whether the scheme offers mechanisms to detect the presence of malicious functionalities in the IP core or not. A malicious

function can be a Trojan or an attempt to tamper with other PL regions that are not reserved for the IP.

- Side-Channel Attack Resistance (SCAR): Whether the scheme is resistant to side-channel attacks or not.
- TTP Access Prevention (TTPAP): Whether a third party can easily extract the IP cores or not (as they may have access to devices, IPs, or secret keys).

Among the defined features, the support or lack of support for the first five ones is shown with only two symbols, i.e., For IIPG, AF, RBAR, MFD, and MHM features, the ▩ symbol shows schemes' support of these features, and lack of support is indicated with the □ symbol. However, for SCAR and TTPAP features, the degree of support is shown with three symbols. Schemes that have not addressed SCAs are marked with the □ symbol. The ones marked with the ▦ have only suggested utilizing SCA-resistant cryptographic primitives without providing any implementation details. Finally, ▩ symbol highlights the schemes whose implementations are resistant to SCAs. For the TTPAP feature, □ represents the schemes where TTP has access to both keys and IPs, and ▦ shows that the TTP only has access to the keys but not the IPs. Most of the schemes, including the proposed one, are in this category. Finally, ▩ is used for the schemes where IPs and keys are not accessible to the TTP, e.g., the scheme presented in [176].

The IIPG is discussed in detail in Chapter 6, where it is mentioned that only Vliegen et al. [133] has addressed this issue. However, they only mentioned the use of a third party tool GoAhead [10] for standalone generation of IPs without providing any technical details (For details see 6.2). Still, Table 7.1 shows their scheme supporting this feature along with the proposed one. The second feature, namely automatic floorplanning, is not addressed by any IP licensing schemes except the proposed one, which is also reflected in Table 7.1.

As discussed in Section 2.4, booting of the device in a secure way is achieved by using the internal dedicated decryption engine (DDE) of the device. These engines are also used to decrypt IPs before their configuration, and several schemes rely on DDEs for this purpose. Since the keys used by the DDEs are programmed by a trusted third party (TTP), such schemes will place higher trust on the TTP that leads to lesser support of the TTPAP feature. Among the schemes, another scheme proposed by the author [176] avoids relying on any TTP, which is the only scheme shown to have full support for this feature in Table 7.1. Other schemes try to reduce the amount of trust placed on the TTP using a physical unclonable function (PUF) or core installation modules (CIMs).

Table 7.1: Comparison among the existing IP licensing schemes, where ▩ represents full support, ▦ shows partial support and □ represents a lack of support.

Year	Schemes	IIPG	AF	RBAR	MFD	MHM	SCAR	TTPAP
2002	Kean [57]	□	□	□	□	▩	□	□
2006	Bossuet [14]	□	□	□	□	▩	□	□
2006	Simpson [120]	□	□	□	□	□	□	▦
2007	Guajardo [41]	□	□	□	□	□	□	▦
2007	Güneysu [42]	□	□	□	□	□	▦	□
2008	Drimer [25]	□	□	□	□	□	▦	□
2010	Gora [39]	□	□	□	□	▩	□	▦
2012	Maes [82]	□	□	□	□	▩	▦	▦
2012	Pappala [100]	□	□	□	□	▩	□	▦
2012	Gaspar [35]	□	□	□	□	▩	▦	□
2014	Zhang, L [157]	□	□	□	□	▩	▦	▦
2015	Zhang, L [158]	□	□	□	□	▩	□	▦
2015	Sun [124]	□	□	□	□	▩	□	▦
2015	Zhang, J [155]	□	□	□	□	▩	N/A	▦
2012	Vliegen [133]	▩	□	□	□	▩	□	▦
2019	Khan [176]	□	□	□	□	▩	▦	▩
2019	Khan [174]	□	□	▩	□	▩	▦	▦
2020	Proposed Work.	▩	▩	▩	▩	▩	▩	▦

However, these solutions lead to higher resource overhead and only insert an additional reverse engineering (RE) step.

Also, all the published schemes do not support RBAR, and MFD features except the proposed scheme and [174], where [174] is an earlier concept of the proposed work, published by the author. The proposed scheme utilizes the TEE, whose SotA is presented in the following section.

In the case of the SCAR feature, schemes [14, 39, 41, 57, 100, 120, 124, 133, 158] that did not address side-channel attacks are marked with □. The rest of the schemes except the proposed one addressed the issue; therefore, they are shown in Table 7.1 as partially supporting this feature. The proposed scheme is shown in Table 7.1 to have full support for this feature because it is the only scheme that provides the concept and

implementation of an SCA resistant AES implementation that is used for the decryption of IPs.

7.2 Assumptions

7.2.1 Target Platform Features

To allow a broad range of hardware platforms to be usable, requirements on the available hardware features should be as low as possible. However, the protection of IPs can only be guaranteed if they are secured against external access at any point in the device at all times. The following components form the basis for such protection.

- **Cryptographic Primitives:** Cryptographic primitives allow for the protection of bitstreams or software using encryption and authentication. Dedicated circuits should be available to process these primitives. The required keys must be stored locally; therefore, a secure key vault should be available as well. It is assumed that these hardware implementations are secure against side-channel and other attacks. State-of-the-art is SHA-3/RSA4096 for authentication and AES256 for encryption. Other algorithms may be used as long as they are reasonably sound.
- **Secure Boot:** FPGAs offer the possibility to boot a device with authentic encrypted system files (e.g., boot loaders, firmware, and OSes) using secure boot mechanisms. These files are considered trustworthy and can be distributed and stored securely. The respective keys must be pre-configured on the device for this purpose. The internal cryptographic hardware engine is typically used for decryption and authentication. Therefore, Secure Boot is considered resistant against SCAs as well. To prevent subsequent manipulation of the keys, devices with Secure Boot offer the possibility to block reading or overwriting the keys once they have been programmed. This secure boot process represents the initial immutable root of trust on which all subsequent security measures are based.

7.2.2 Trust on FV's Devices and TTP

FPGA vendors (FVs) have access to their devices during manufacturing, which provides all fundamental roots of trust, including identification numbers and the secure key vault. Since the devices are commercially available and used, malicious behavior or weakness in them will likely be found and, consequently, damage their reputation. Therefore, it can be assumed that the devices are trustworthy. Secondly, several keys specific to secure boot and the security framework (SFW) are programmed by a trusted third party (TTP) during an FPGA device's enrollment before its use (see Section 7.5). All these security tasks imply that the TTP must be trustworthy. However, measures

are taken to reduce the degree of trust placed in the TTP, i.e., its access to the IPs is restricted (See Section 7.1 and 9.2). Ideally, the FV can also play the TTP role because fewer participants will reduce security leaks. However, this is not mandatory.

7.3 Threat Model

The most valuable assets in an IP licensing scenario are the bitstreams of IPs and cryptographic keys present in the device. Appropriate measures must be taken to ensure that both are protected against unauthorized access, as far as this is possible with devices available on the market. Based on the categorization of attackers and their capabilities, according to Abraham et al. [1], this work is limited to measures that protect against any attacks except those carried out by Funded Organizations. The latter have access to advanced technology and well-equipped laboratories that, for instance, can be used for sophisticated physical attacks. Protection against such attacks is not feasible with a mere licensing scheme and matching software alone but must be implemented directly in the hardware (i.e., the FPGA itself). Explicitly, however, the possibility of a non-trustworthy system designer is considered, who may use readback or similar attacks to extract protected IPs. This also includes anyone who has physical access to the device, e.g., a customer of the system developer. Furthermore, the possibility of a hostile core vendor (CV) is also considered who can insert a malicious functionality in their IP to attack SD's device.

7.4 Trusted Platform

7.4.1 Establish Trust on the Processing System (PS)

In general, the goal of a licensing scheme should be to secure the target platform against IP theft attacks and misuse, discussed in Chapter 3, without restricting the SD's access to the device and its features. The first step towards this goal is to distribute IPs as encrypted bitstreams and have decryption keys pre-configured on the devices. However, this approach has several shortcomings:

- SDs can readout configuration data using configuration interfaces, which is discussed in Section 3.1.4.
- IPs may include malicious content such as Trojans or tamper with the SD's design. A malicious bitstream can only be detected by analyzing the IP's decrypted form.
- IP-specific software applications cannot be protected.

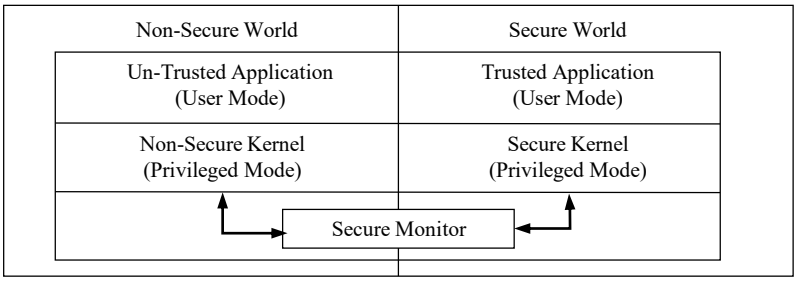

Figure 7.1: Secure and non-secure world concept

- All CVs either have to use the same key to encrypt their bitstreams, or a TTP needs to do this encryption step on CVs' behalf. However, this would lead to multiple security challenges [57].

For most SoCs, the PS is the main controller with access to all the device components. If SDs have full control of the PS, some of the above drawbacks cannot be countered, such as readback attacks. An alternative would be restricting SD's access to the PS; however, this would restrict many of the device features from being used by the SD. A solution would be partitioning the PS into secure and non-secure worlds (see Figure 7.1), where only the latter one will be in the control of the SD. This can be achieved using TrustZone technology (see Section 2.5.3), which provides a TEE-based trusted OS and a rich OS. Their access rights are presented below:

- Trusted OS has access to all components of the device. It acts as a secure master and cannot be accessed once running, apart from predefined APIs. SDs can neither control nor modify this OS.
- Rich OS is under SD's full control. They have direct access to most peripherals as long as they are configured as non-secure in the TEE. Other secure peripherals can only be accessed by making requests to the trusted OS.

Figure 7.1 shows the general software setup with ARM Trusted Firmware as a base layer that configures TrustZone on startup and acts as a security monitor once the system is running. The purpose of this security monitor is to separate trusted and rich OSes but still allow communication between them using security monitor calls, which provide rich OS applications' access to the APIs of the trusted OS.

Processing units, interfaces, and sections of the main memory can be defined as secure or non-secure. For example, for the secure configuration of an encrypted IP, two regions of the main memory are declared secure, as shown in Figure 8.1. One of them contains

the trusted OS, while the trusted OS uses the other to store the decrypted bitstreams. Since the rich OS does not have access to these memory regions, even a plain-text bitstream is secure.

As mentioned in Section 3.1.4, configuration access ports (PCAP or ICAP) can be used to launch the readback attack. This attack can be avoided if rich OS is restricted from accessing the configuration interfaces. Even though the target device and its development flow provide partitioning of resources into sub-systems (secure and non-secure), the same is not valid for PCAP, i.e., PCAP cannot be isolated from rich OS using isolation flow [151]. Secondly, the ICAP is on the PL, and restricting its access from the rich OS is not supported by the TEE implementation on the target device. More details on this are presented in the following section.

7.4.2 Extending TEE to the Programmable Logic (PL)

The basic concept of TEE is the isolation of PS resources. In this work, the discussion is focused on ARM TrustZone, which is an implementation of the TEE. As mentioned in Section 2.5.3, TrustZone is a system-wide approach to security with hardware-enforced isolation built into the PS. Among other security features such as isolated execution, this approach also offers secure, and non-secure read/write transactions that can be used to isolate access to the IPs on the PL. However, this approach does not offer any isolation or security for the configuration/programming of the PL.

The TEE concept can be extended to the configuration process by managing the access to the configuration interfaces and bitstream analysis. This is achieved by partitioning the PL into a secure and non-secure region, as shown in Figure 7.2. The configuration and readback of the secure PL region can only be issued by the secure masters (trusted OS). However, (re-)configuration and readback of the non-secure PL region are still possible from the non-secure master (rich OS). Here, SD's design and other open-source IPs can be configured/readback by the trusted OS using a predefined API, therefore, not restricting SD from using features of the device. This way, security is extended to the PL without affecting the available features like reconfiguration and readback. Here, the term non-secure only means that these PL resources are non-secure for the unprotected licensed IPs because SD has full access to them. In other words, SD's proprietary software in rich OS or hardware IPs in the non-secure PL region are secure from other parties. For example, in cases where the system is used by SD's customer.

The distinction between target regions of bitstreams is made by analyzing the addresses of frames they are writing. Frames are the smallest sections of the PL that can be

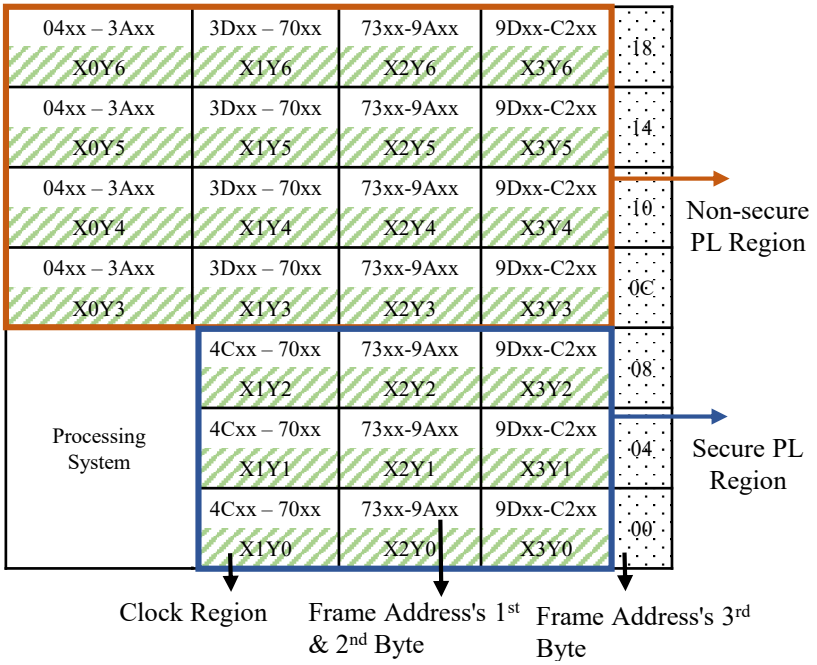

Figure 7.2: FPGA layout with clock regions, respective frame addresses and partition into Secure and Non-secure PL regions.

programmed individually. They are identified by their unique address, the frame address. Bitstreams mainly consist of a sequence of commands alternating between setting the target frame address and writing to it. The frame address ranges for the target device are shown in Figure 7.2.

The partitioning process begins with the analysis of the bitstream where frame address writes are identified. Here, a list of properties is presented that must be satisfied by the bitstream analysis process for a successful configuration.

1. Clock region X3Y1 can never be reconfigured or readback, i.e., partial bitstreams must not have any frame addresses specific to clock region X3Y1 because it is reserved for the SFW components. SFW components are PRC, TrustZone logic, and connection to ICAP. This clock region is selected for the SFW components because it has the physical connection between PL and ICAP.

2. SD's bitstreams must only have frame address specific to the non-secure PL region.

3. Bitstreams of the licensed IPs must only have frame addresses specific to the RRs reserved for them. RRs of the licensed IPs are created in the secure PL region except X3Y1 clock region.

7.5 Proposed IP Licensing Scheme

The protocol of the proposed IP licensing scheme consists of three steps, which are IP core enrollment, preparing SFW, and IP licensing:

7.5.1 IP Core Enrollment

Core vendors (CVs) enroll their IPs with the TTP by sharing their public key (PK_{CV}) and information about the IPs, such as cost, functionality, resource consumption, and throughput. This is shown as step 1 in Figure 7.3. Afterward, TTP enrolls the IP and publishes the details in its store. From where SDs can choose and initiate the licensing procedure.

7.5.2 Preparing Security Framework (SFW)

The second part of the scheme is preparing the SFW, which is from steps 3 to 12 in Figure 7.4. A system developer interested in licensing IPs can browse through a TTP's IP store and select the ones that meet its requirements shown in step 3. Based on its system and the IP's information available on the store, the SD specifies IP implementation requirements such as the area required on the PL, interface, etc. In step 4, the SD specifies the requirements, which are reserving PL resources for its designs and IPs, and interfaces among them. Then, it requests FPGA devices (F_i) along with the SFW that supports IP licensing, which is shown as step 5. Alternatively, an SDs can also request SFW for their own devices; however, they need to send the devices to the trusted third party (TTP) because physical access is necessary for programming the device keys (Step 10).

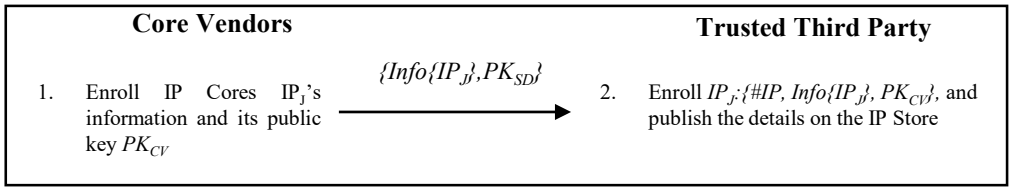

Figure 7.3: Steps of IP Enrollment

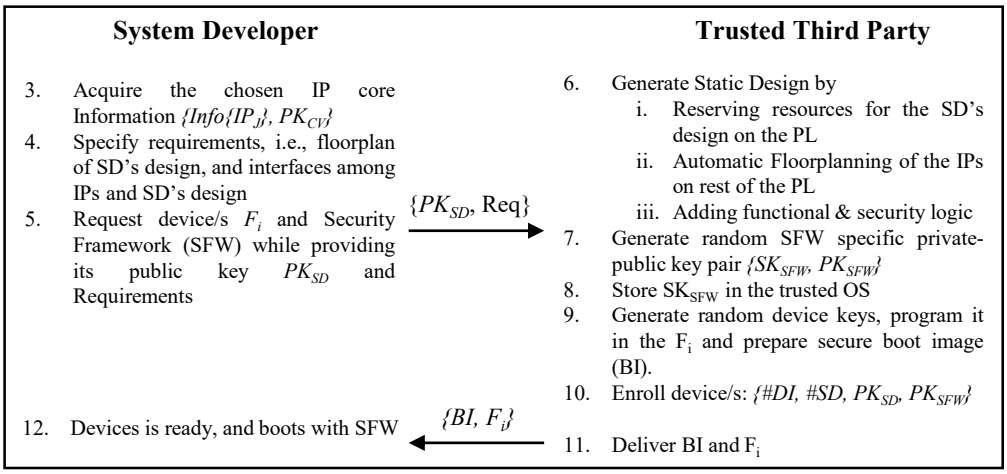

Figure 7.4: Steps of Preparing the SFW

Once the TTP receives the request, it performs step 6, which is generating the static design. This step requires reserving PL resources for the SD's design (See Chapter. 6 Standalone Generation of IPs), Automatic Floorplanning of the IPs (See Chapter. 5) and specify interfaces among the IPs and SD's design. This static design includes a secure and a non-secure PL region, as shown in Figure 8.4 and 8.1. The secure region contains components supporting the SFW, such as TrustZone logic, a configuration controller, and interconnects. Also, it has reconfigurable regions for all the licensed IPs. The non-secure one is reserved for the SD's design, which the SD specified in step 4.

In step 7 and 8, TTP generates an asymmetric key pair for the SFW, a private- (SK_{SFW}) and a public key (PK_{SFW}), and stores SK_{SFW} in the trusted OS, respectively. Then, random device keys are generated and programmed into the device. These keys are used to generate a protected bootable image (BI) of the SFW along with the bootloader, firmware, and the static design (full bitstream) using the secure boot feature [142]. This is shown as step 9 in Figure 7.4.

In step 10, TTP enrolls devices into their database. For each device, the TTP creates an entry in their database that has information about the device identity ($\#DI$), SD's identity ($\#SD$), SD's public key (PK_{SD}), and the public key of the SFW PK_{SFW}. Then, the devices and matching bootable images are delivered to the SD, shown as step 11 in Figure 8.1. For higher security, unique keys can be generated for each device, which means a unique bootable image needs to be created for each device. Also, each trusted OS can have a unique private key (SK_{SFW}). Alternatively, keys can be

generated for a batch of devices, which would simplify the boot image generation because a single image can be used for all the devices in a specific batch, provided they all have the same SK_{SFW}. Finally, shown in step 11, SD has the devices running the SFW.

Since $\#DI$ is unique for every device, it can be used to verify whether a device is legally produced or not during enrollment. However, the TTP needs a list of legally produced devices from the FPGA vendors (FVs) for this to work.

7.5.3 IP Licensing

The third part of the scheme is acquiring the IP in bitstream form from the CV. In step 13, the SD makes a licensing request to the CV by providing its identity ($\#SD$), the identities of the device ($\#DI$) and IP ($\#IP$). After receiving this request, in step 14, the CV verifies the identities and uses $\#SD$ and $\#DI$ to acquire the static design and public keys of SD and SFW from the TTP. Since SD and CV have each other's public keys, they can start an authenticated secure communication.

Afterward, in step 15, CV generates the requested IP's bitstream (IP_j) using the static design. CV can also add a fingerprint [17, 20, 69] in the IP that includes SD's and CV's information. This can help in identifying the source after IP theft is discovered.

This work considers two delivery paths for bitstream-based IPs. The preferred way is to establish a direct communication channel between a device and a CV, e.g., via the internet. As mentioned earlier, every device has a public-private key pair, where the private key (SK_{SFW}) is pre-programmed in the device's SFW, and the corresponding public key (PK_{SFW}) is made public. Also, CV as an organization is assumed to have a private-public key pair. Since both CV and device have public keys, they can start mutually authenticated and encrypted communication using network security protocols, e.g., transport layer security (TLS) [31]. With such secure communication, bitstreams can be transferred as plain-text without compromising them during transfer. On the target device, the communication channel is established by the trusted OS, which means IPs are stored in a secure region of the main memory when received. Therefore, IPs cannot be accessed by the rich OS.

Alternatively, IP bitstreams can be delivered using a non-secure channel. This delivery method is shown in Figure 7.5 from step 16 to 20. In this approach, a random symmetric bitstream key (K_B) is generated, and IP_j is encrypted using this key shown as step 16. Afterward, in step 17, K_B is encrypted with the public key of the SFW (PK_{SFW}) running on the device for whom this IP was requested. Then, in step 18, the encrypted IP_j

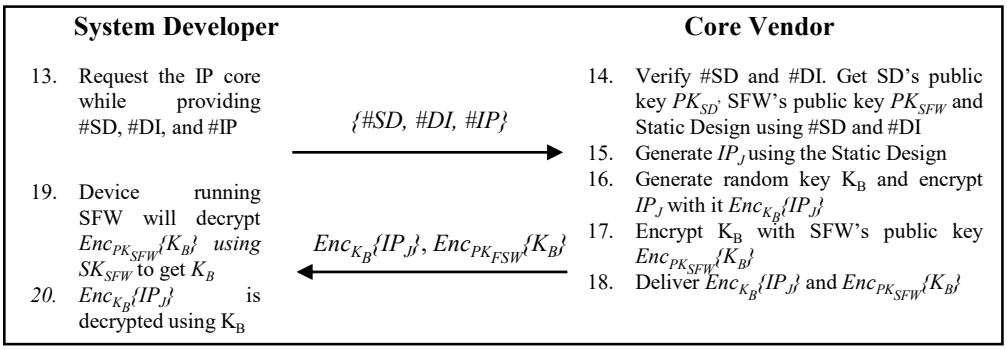

Figure 7.5: Steps of Licensing IPs

($Enc_{K_B}\{IP_j\}$) is delivered along with the encrypted bitstream key K_B ($Enc_{PK_{SFW}}\{K_B\}$) to the SD.

An SD can provide both $Enc_{K_B}\{IP_j\}$ and $Enc_{PK_{SFW}}\{K_B\}$ to the device after booting it up. The trusted operating system running on the device can then decrypt $Enc_{PK_{SFW}}\{K_B\}$ using SK_{SFW}. Afterward, it can use K_B to decrypt $Enc_{K_B}\{IP_j\}$. These are the two final steps of the schemes, shown as step 19 and 20 in Figure 7.5.

8 Implementation

The work described in this chapter was published in [177, 179] and is joint work with co-authors Sven Nitzsche, Asier Garciandia López, Jorge Castro-Godínez, Shixiang Xue, Jörg Henkel, and Jürgen Becker. More details on contributions is found in Section 1.5.

This chapter presents the implementation of the proposed licensing scheme discussed in the previous chapter. Furthermore, IPs used in the implementation are floorplanned using the Automatic Floorplanner presented in Chapter 5, and are generated independent of the static design using the flow presented in Chapter 6. The proposed licensing scheme can be realized on any platform that supports the features presented in Section 7.2. In this work, a Xilinx Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC is chosen as the target device, which has a quad-core Arm Cortex-A53, a dual-core Arm Cortex-R5F real-time processor, a Mali 400 MP2 graphics processing unit (GPU), and a PL fabric in a single device. Xilinx Vivado Design Suite 2018.3 and corresponding PetaLinux Tools are used to create the final implementation.

The implementation starts with creating a project for the target device, where a block design is defined. The block design instantiates a Zynq Ultrascale+ MPSoC, partial reconfiguration controller (PRC), AXI interconnects, TrustZone logic, ICAP primitive, SD's protected design (PP_{SD}), SD's non-protected design (NP_{SD})), and licensed IPs are instantiated. Among them, PRC, TrustZone logic, and the connection to ICAP support the SFW and are referred to as SFW components. This main design will be used to create the static design (see Section 6.3), which will be shared with all the CVs and the SD. They will use it to generate partial bitstreams of their respective designs. Therefore, all designs that need protection should only be declared as reconfigurable designs (RDs), which are SFW components, IPs, and PP_{SD}. Static design can be generated without including implementation details of its sub-modules that are defined as reconfigurable. However, interface definition of the sub-modules is required. The block diagram of the project is shown in Figure 8.1.

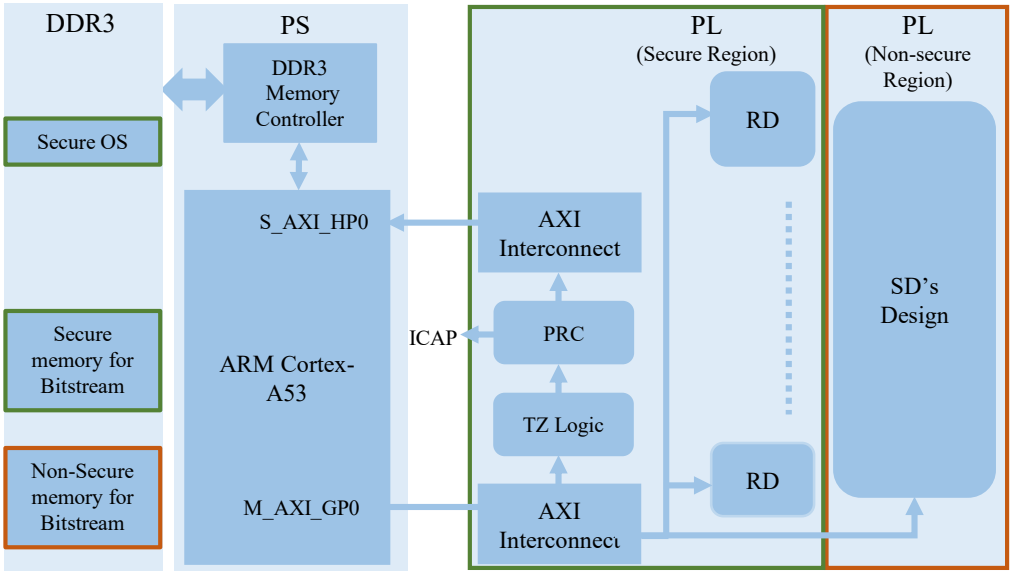

Figure 8.1: Block diagram of PS and PL showing memory and PL partitioning into secure and non-secure region

For an overview of this chapter, an outline of the implementation aspects of the proposed work is presented below

- In the current implementation, IP's encrypted and decrypted (plain-text) bitstreams are stored in the DDR memory. Since the plain-text one can be accessed by the rich OS, it needs to be stored in a non-secure partition of the memory. This partitioning of memory into secure and non-secure regions is discussed in Section 8.1.
- As explained in Section 3.1.4, configuration interfaces can be used to readback the configuration data from the PL. This is avoided by isolating the configuration interfaces from the rich OS and is discussed in Section 8.2.
- Afterward, the use of independent IP generation (IIPG) flow (see Section 6) for the generation of the static design, full and partial bitstreams is discussed (see Section 8.3). This section also includes the use of automatic floorplanning (see Section 5) for the current implementation (see Section 8.3.2).
- Section 8.4 provides details about the Trusted and Rich OSes.
- After that, the implementation details of several trusted applications (TAs) are presented in Section 8.5.

- All the components outlined so far can be collectively called the security framework (SFW). In Section 8.6, integration of all the sub-components of the SFW is discussed. Furthermore, this section ends with the generation of a Secure Boot Image of the SFW.
- Finally, Section 8.7 provides the execution flow of the two use cases of the proposed work, which is the configuration of an encrypted and plain-text bitstream.

8.1 Memory Partitioning

The PS setting of Xilinx Vivado offers an isolation feature, which is used to define a secure sub-system. In this sub-system, parts of the main memory are declared as secure. This feature utilizes AXI transaction inhibitors in addition to the ARM Trust zone infrastructure. For the Zynq Ultrascale+ MPSoC, these inhibitors use the Xilinx memory and peripheral protection units to block transactions between AXI masters and slaves [149, pp. 50]. As shown in Figure 8.1, 256 MB of memory, starting from address 0x60000000, is reserved for the trusted OS, and 16 MB of memory, starting from 0x45000000, is reserved for storing the decrypted bitstream of an IP. These settings can only be configured by the first stage boot loader (FSBL) [151], [149, pp. 50, 58], which means a custom FSBL needs to be generated rather than using the default one generated by Petalinux. Once the block design, including the PS, is configured, a hardware platform specification is generated as an HDF file and is exported to the Xilinx software development kit (XSDK). XSDK is used to generate the FSBL that manages the memory partitioning specific settings on the target device.

8.2 Restricting Configuration Interfaces

The configuration data can be written to or read out from the PL using the configuration interfaces, e.g., ICAP or PCAP. If the rich OS has access to these interfaces, it can perform the readback attack or write to a secure PL region. These attacks can be avoided by restricting access to these interfaces from the rich OS. However, the target device and the corresponding vendor tools do not provide a straightforward way of restricting these interfaces.

8.2.1 Blocking PCAP

In this implementation, PCAP is used to configure the full bitstream (static design). Afterward, it is disabled by enabling ICAP, which is then used to configure SD's protected design and IPs. Enabling ICAP requires clearing the *CSU_PCAP_CTRL* register in function *XFsbl_HookAfterBSDownload* of the FSBL source code. However, the

rich OS still can enable PCAP using the FPGA manager driver, which in turn has access to the *Xilfpga* library of the platform management unit's (PMU) firmware. This library runs at the highest privilege level (EL3) and will give the rich OS access to enable PCAP and configure the PL [152, pp. 598-600]. Therefore, removing this library from the PMU firmware is also necessary. This is done by generating custom PMU firmware using XSDK. By utilizing both custom FSBL and PMU firmware, PCAP is disabled for both OSes.

8.2.2 Isolating PRC from rich OS

Since PCAP is disabled for both OSes, the next step is to restrict rich OS's access to ICAP while still allowing the trusted OS to perform PL configuration. Configuring the PL via ICAP is done using Xilinx's PRC [144]. It has an AXI4-Lite register interface to the PS, as shown in Figure 8.1. Once a bitstream is copied to a specific RAM address, the PRC's bitstream address and size register can be updated with that information. Afterward, a software or hardware trigger to the PRC can start the PL configuration process [144].

The AXI Interconnect IP connecting the PS and the PRC can be set to declare that the latter is a secure slave. However, even after applying this setting, the non-secure master (rich OS) has access to the PRC. Furthermore, PRC's AXI input interface does not have TrustZone specific signals such as AWPRORT. As shown in Figure 8.1, a custom TrustZone logic is placed between the AXI interconnect and the PRC to block rich OS's access to the PRC. TrustZone logic only allows secure AXI read/write transactions to the PRC registers, i.e. when the *AWPROT(1)* signal is low. This way, only the secure master (trusted OS) can read/write PRC's registers.

Furthermore, an API is provided to the rich OS, using which it can set PRC's registers to only non-secure memory, where SD's plain-text bitstream can be copied. Using the API, rich OS can trigger the configuration process. However, the configuration will only start if bitstream analysis (discussed in Section 8.5.5) of the SD's bitstream confirms that it targets a non-secure PL region. The same is true for the readback operation. Individual steps for configuring an encrypted bitstream on the secure and a plain-text bitstream on the non-secure PL regions are explained in Section 8.7.

8.3 Standalone Generation of IPs and SD's Design

The section presents the independent IP generation (IIPG) flow implemented for the target device. The main design is real-time object labeling (ROL) which is a stream-based

Table 8.1: Synthesis Report of the Designs for the PL.

Designs	CLBs			BRAMs	DSPs
	Luts	Regs	CARRY8		
SD's Main Design and IPs					
ROL	2220	2721	0	27	0
lzrw1-comp	1088	1186	24	0	1
kvcordic	377	254	0	0	0
xtea	320	162	0	0	0
b163-arith	245	489	0	0	8
Licensed IPs					
RGB2HSV	555	616	74	0	0
CED	2667	3933	350	17	0
MDCT	1220	1932	80	0	0
DCT_AAN	644	550	32	0	4
Sha256	1169	1031	52	0	0
AES-T	471	137	0	9	0
AES-S	264	253	0	0	0
AES-R1	1336	533	0	0	0
AES-R2	613	312	0	5	0

video processing design that performs connected component analysis and labeling [9]. Its sub-modules are run length encoder, label selection, merger resolution, and feature calculation [153]. The design is considered to be developed by an SD who would like to add more functionality to it in the future. Also, there are other functionalities that the SD requires, for which external IPs are acquired. These IPs are either internal or taken from open sources. The IPs are lzrw1-comp [115], kvcordic [55], xtea [103], b163-arith [8], RGB2HSV core, canny edge detector (CED) [7], MDCT Core [65], DCT Core [129], Sha256 [102] and several AES implementations namely AES-T, AES-S, AES-R1, and AES-R2. Synthesis results of the main designs and IPs are presented in Table 8.1.

8.3.1 Design Partitioning

As shown in Table 8.1, ROL, lzrw1-comp, kvcordic, xtea, and b163-arith are considered to be SD's designs. The rest of the IPs belong to CVs. First SD's designs are divided into PP_{SD} and NP_{SD} (see Section 6.3). ROL is considered to be PP_{SD} and the rest of the SD's IPs are considered to be NP_{SD}. PP_{SD} and CV's IPs need to be protected from

Table 8.2: Resources assigned to each Reconfigurable Region.

Designs	CLBs	CLBs+25%	BRAMs	DSPs
RGB2HSV	74	93	0	0
CED	350	438	18	0
MDCT	153	192	0	0
DCT_AAN	81	102	0	4
Sha256	143	179	0	0
AES-T	59	74	10	0
AES-S	33	42	0	0
AES-R1	167	209	0	0
AES-R2	77	97	5	0

sharing, so they are declared as RDs. NP_{SD} and interfaces between PS and the RDs are made part of the static design. The next step is the floorplanning of all the RDs, which is explained in the following section.

8.3.2 Floorplanning

Chapter 5 discusses in detail the proposed floorplanner. As mentioned earlier, the resource consumption of BRAMs and DSPs for each RD can be taken directly from their synthesis report. The report shows CLBs consumption in terms of LUTs, FFs, and CARRY blocks, which are presented in Table 8.1.

The resource consumption information taken from synthesizing the designs has some inaccuracy, and for some designs, the FV tool cannot run a successful placement. Secondly, the designs of IPs and the static design could be complex and require more routing resources. Therefore, 25% of CLB resources are added for each reconfigurable region (RR), which in turn will also increase the routing resources. The 25% margin will guarantee that the designs can be routed in all cases. The fixed margin value is applied to all RRs; however, some can be routed with lower values. But the added advantage in resource waste is less than the effort required to run the entire flow several times. Resources calculated from Equation (5.1) and 25% added margin for CLBs are shown in Table 8.2.

As mentioned in Section 5.2, floorplanner results are the left-most, top-most coordinates: x_i, y_i and the width, height : w_i, h_i for each RR R_i. These values are used to create constraint files where RRs are defined on FPGA's layout, and they are assigned to the

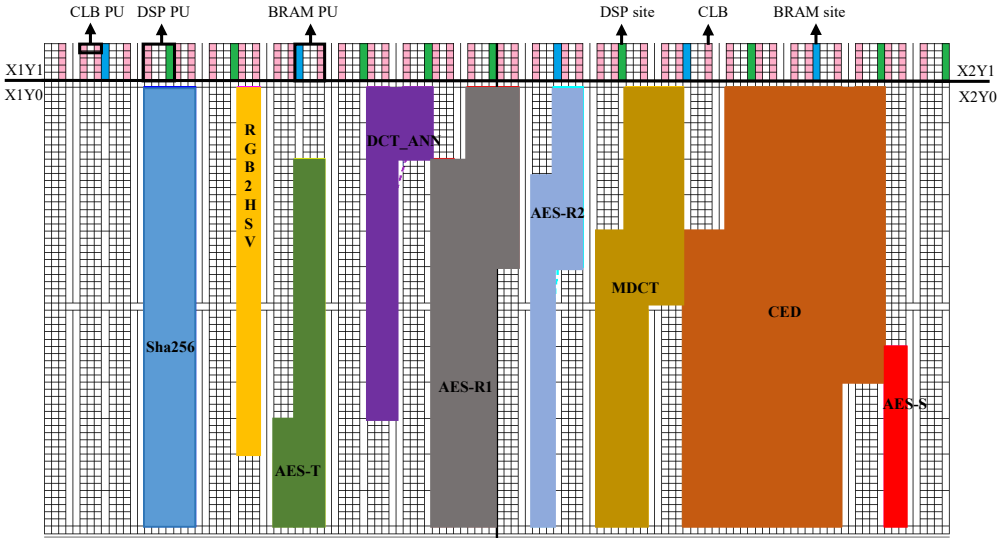

Figure 8.2: Floorplanned IPs on the X1Y0 and X2Y0 clock regions of the target device.

corresponding RDs. The generated floorplan of the RRs is shown in Figure 8.2. Since the licensed IPs are smaller and can fit into two clock regions, they are floorplanned only on X1Y0 and X2Y0 clock regions. Part of the X1Y0 and X2Y0 is also shown where resource types and smallest PUs can be seen. Similarly, an RR is placed on X1Y1 for SD's protected design.

Resource consumption of the licensed IPs is 61% of the target region (4.4% of the device). Since the target area and resource consumption are small, an optimal solution is found in 20 seconds. For the IP licensing use case, usually, a large part of the PL is reserved for SD's design. If floorplanning of the licensed IPs is restricted to few clock regions, an optimal solution for the target area can found in a very short time. However, in cases where most of the PL is covered with licensed IPs, IPs can be divided into several groups, and each can be floorplanned while targeting different regions. This way, an optimal solution for each group specific to their target region can be found. Alternatively, the optimization model can be solved with a time limit to find a sub-optimal solution.

8.3.3 Static Design Generation

Once the floorplanning is done, a static design can be generated that will be shared with all the CVs. Major FPGA vendors support the partial reconfiguration feature for their devices, which allows to dynamically change a sub-module implementation within an active design [147]. The flow requires defining an RR on the PL for each reconfigurable

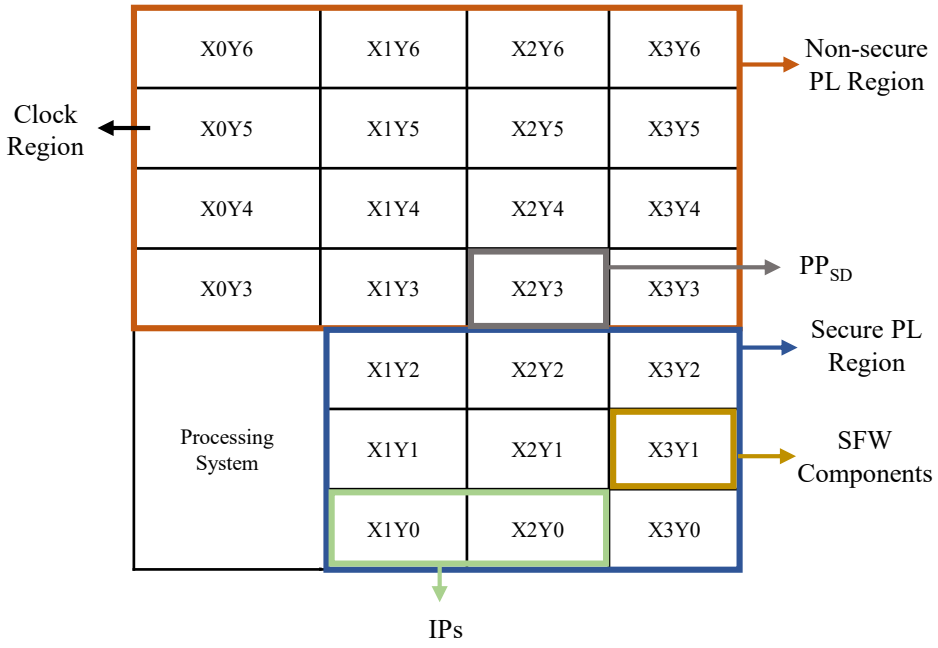

Figure 8.3: Clock regions used for the floorplanning of PP_{SD}, Licensed IPs and the SFW components

module (RM). Furthermore, the flow requires full design instantiating the RM and several RM implementations. The flow ultimately generates a full bitstream for the active (full) design and partial bitstreams for each RM implementation.

Standard DPR flow supported by Xilinx Vivado [147] requires an HDL file with only name and I/O port definition for the reconfigurable designs (RDs). SFW components, IPs, and SD's protected part are RDs that are defined in Verilog HDL. The previously created design has an instances of all RDs, a Zynq Ultrascale+ MPSoC, an ICAP primitive, and SD's non-protected design. The interface used for the licensed IPs is AXI4-Lite, and SD's protected design is wrapped with an AXI4-Stream interface [138]. All the RDs are connected with the PS, and other system settings are adjusted.

As discussed in Section 7.4.2, PL is partitioned into secure and non-secure PL region for extending the TrustZone concept to the PL. The licensed IPs and SFW components will be placed in the secure PL regions. That is why the IPs are floorplanned using the Automatic Floorplanner on X1Y0 and X2Y0 clock regions, as shown in Figure 8.3. The floorplanner is used to create a constraint file that is added in the flow where all the RRs for the IPs are defined and assigned to their respective RDs. The next step is

defining an RR for the SFW components. This RR is created in the X3Y1 clock region because it is the secure PL region (see Figure 8.3). Furthermore, the ICAP interface is physically located here. Further details about this are presented in Section 9.

It is important to mention that the term non-secure only means that the PL resources are non-secure for the unprotected licensed IPs or SFW components because SD has full access to them. In other words, SD's hardware IPs in the non-secure PL region are secure from other parties. For example, in cases where SD's customer uses the system. Here, the goal is to offer flexibility to the SD in using their device (for details, see Section 7.4.2). The final floorplanning step is defining an RR for the SD's protected design (PP_{SD}) in the non-secure PL region (see Figure 8.3) and assigning it to the PP_{SD} in the constraint file.

The next step is converting the block design to HDL files. After this, synthesis, optimization, placement and routing of the block design is done using Vivado commands *synth_design*, *opt_design*, *place_design*, and *route_design*, respectively [146]. After these steps, the design is stored as xilinx design checkpoint (DCP) [146] (*full_design.dcp*). *full_design.dcp* can be used to generate full bitstream at this stage, and it can be shared with several parties who can generate their respective bitstreams with it. Furthermore, *full_design.dcp* does not have any implementation information specific to SD's protected design, SFW components or IPs.

The earlier discussed HDF file (hardware platform specification) is also generated at this stage and used in a PetaLinux project (See the following section).

8.3.4 SD's Protected Design

SD synthesizes the protected design for the target device and writes it as a DCP file (*pp_synth.dcp*). After this, SD opens *full_design.dcp* and updates their protected black-box design with *pp_synth.dcp* while inserting buffers for all the other CV's design. Then, optimization, placement, and routing are done using Vivado's commands *opt_design*, *place_design*, and *route_design*. In the end, bitstreams are generated for the PP_{SD} and the static design.

8.3.5 CV's Design

CV synthesizes their IP for the target device and writes it as a DCP file (ip_i*_synth.dcp*). They have the static design *full_design.dcp*, which is opened. Afterward, CV updates their IP's black-box instance in the static design with ip_i*_synth.dcp* and does optimization,

placement, routing, and generate a bitstream for their IP.

The static design's bitstream is called full bitstream, while PP_{SD} and IP's bitstreams are called partial bitstream. They all are compatible because they are generated from the same implemented static design.

8.4 Trusted and Rich Operating Systems

PetaLinux is a software development kit targeting Xilinx SoCs. For the trusted OS, the open portable trusted execution environment (OP-TEE) is used, which is an open-source project maintained by TrustedFirmware.org (previously by Linaro [78]). OP-TEE's architectural details can be found in [75]. It was designed as a companion to a non-secure Linux Kernel running on an ARM processor using TrustZone technology. OP-TEE implements TEE Internal Core API v1.1.1 and the TEE Client API v1.0 [36]. The former one is the API exposed to the trusted applications, and the latter is the API that describes how to communicate with a TEE [77].

Among the OP-TEE components, we have used *build*, *optee_os* and *optee_client*. They are part of an open-source project whose repository is available at [76]. *build* is a full OP-TEE developer setup for the OP-TEE project. We have created a local repository of *build*, *optee_os*, *optee_client*, and trusted applications. The Makefile that is part of the *build* setup creates a PetaLinux project whose configuration is imported from the HDF file. The script adds *optee_os* to the board support package (BSP) recipe and *optee_client* and all applications to the application recipes of the project. The secure memory address and size are set according to Section 8.1 during the configuration of the project. The script downloads *optee_os*, *optee_client* and applications, and builds the project. The build process generates an EL3 runtime firmware (bl31.elf), the rich OS (u-boot.elf), and the trusted OS (bl32.elf). It also generates an FSBL and a PMU firmware. However, they will later be replaced with custom versions, as discussed previously.

8.5 Trusted Applications (TAs)

There are two ways to implement trusted applications, user-mode TAs (UMTAs) and pseudo TAs (PTAs). UMTAs are fully featured TAs as specified by the GlobalPlatform API TEE specifications [36]. PTAs are implemented directly in the OP-TEE core tree and are statically built into the OP-TEE core blob. Details on the communication, invoking, locations, and privileges of UMTAs and PTAs can be found in [79].

Registering and read-write operations on physical address spaces are implemented as PTA and are built into the OP-TEE core blob. Among others, PRC registers and two 16 MB memory blocks, a secure and a non-secure one, are registered with OP-TEE OS. The isolation of both memory blocks is discussed in Section 8.1 and shown in Figure 8.1. The rest of the functionalities are implemented as UMTAs, and they are presented below.

8.5.1 Asymmetric Encryption/Decryption

The asymmetric cipher RSA is implemented in the trusted OS using the following functions [37]:

- *TEE_AllocateTransientObject* is used to allocate an uninitialized transient object that is used to hold a cryptographic key of size 4096 bits.
- *TEE_PopulateTransientObject* is used to populate the transient object with the SFW private key SK_{SFW}.
- The algorithm argument in *TEE_AllocateOperation* function is set to $TEE\_ALG\_RS-AES\_PKCS1\_V1\_5$.
- Finally, *TEE_AsymmetricDecrypt* is used to perform the decryption.

8.5.2 Authentication of Encrypted IPs

The authentication of encrypted IP is implemented in the trusted OS using the following functions [37]:

- A digest or hash handle of type *TEE_OperationHandle* is defined and initialized.
- The digest handle is updated using *TEE_AllocateOperation*, where the algorithm argument is set to *TEE_ALG_SHA256*, which represents secure hash algorithm 2 (SHA-2) with a hash value of 256 bits (SHA256).
- Then, *TEE_DigestUpdate* and *TEE_DigestDoFinal* functions are used to calculate the hash of the encrypted IPs.
- Finally, the *TEE_AsymmetricVerifyDigest* function is used to compare the signed hash provided to the trusted OS with the hash computed in the previous step. The signed hash is calculated on a workstation and signed with the SFW public key (PK_{SFW}). The transient object holding SK_{SFW}, created in steps 1 and 2 of Section 8.5.1, is used for the comparison (verification) of the provided signed hash and the hash computed in step 3 of this section.

8.5.3 Symmetric Encryption/Decryption

Similarly, the symmetric cipher AES is implemented in the trusted OS with cipher block chaining mode and a key size of 256 bits, using the following functions [37]:

- *TEE_AllocateTransientObject* is used to allocate an uninitialized transient object.
- *TEE_PopulateTransientObject* is used to populate the transient object that will hold the AES key.
- In *TEE_AllocateOperation*, the algorithm argument is set to $TEE\_ALG\_AES\_CBC\_NOPAD$.
- Finally, *TEE_CipherInit*, *TEE_CipherUpdate* and *TEE_CipherDoFinal* functions are used to perform encryption or decryption.

The AES key used here is provided by the rich OS side, which is encrypted with the SFW public key (PK_{SFW}).

8.5.4 PRC Configuration and Trigger

As already discussed, a PRC is instantiated in Vivado and will be used to configure partial bitstreams. Physical addresses of the PRC's internal registers are taken from the address editor window of Vivado. As the functionality to read and write physical addresses is already implemented as PTA, that functionality can be used to implement PRC configuration and trigger registers [144].

8.5.5 Partitioning of the Programmable Logic (PL)

Figure 8.4 shows that the layout of the PL consists of 25 clock regions. Each clock region has multiple columns of reconfigurable resources and interconnects. Configuration data specific to a column is known as the configuration frame. Each column is uniquely addressed, called frame address [147, pp. 133-134]. Frame addresses are 32-bit wide, where the 16 least significant bits (LSB) (1st and 2nd byte) uniquely identify a column on the x-axis. The next 8 bits (from 16 to 23) represent the clock region row, which is shown as the last column of Figure 8.4. For example, columns inside the clock region X2Y5 are addressed from 1473xx to 149Axx.

Partial bitstreams of IPs are generated by assigning a reconfigurable region (RR) to them, where RRs represent IP's placement location on the PL. If the partial bitstream of an IP is generated whose RR was placed in the clock region X2Y5, it will have only frame addresses from 1473xx to 149Axx. In this work, PL is divided into two regions. The top four rows (total 16 clock regions) are defined as the non-secure region and the

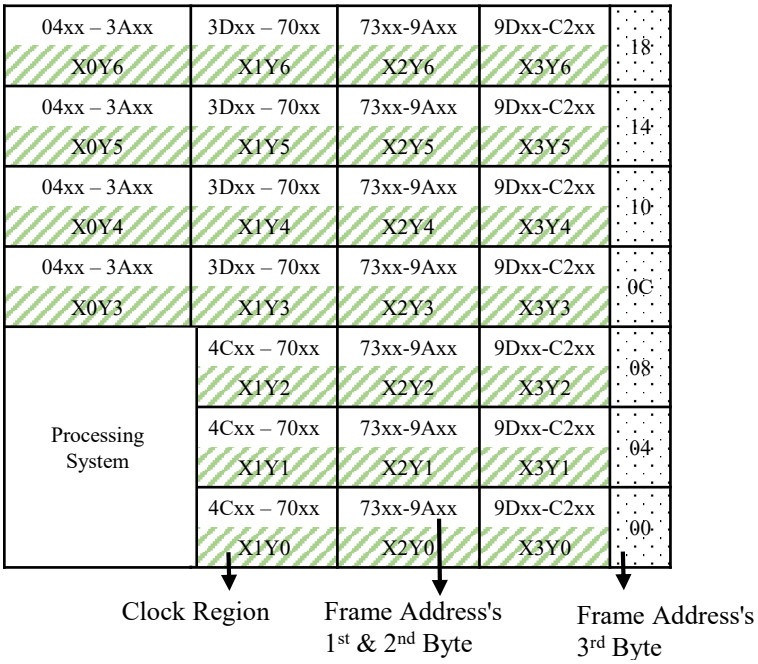

Figure 8.4: FPGA layout with clock regions and respective frame addresses.

lower three rows as the secure. PL partitioning is enforced with a trusted application that performs bitstream analysis. This application goes through a bitstream's content and extracts frame addresses that are preceded by the opcode of the "Write frame address register" command (opcode 30002001 in hex) [145, pp. 156].

As discussed in Section 7.4.2, PL's partitioning into secure and non-secure regions is only for readback and reconfiguration. An SD can use these features only for the non-secure PL region using the rich OS. Bitstream analysis of SD's bitstream will stop the configuration if it has frame addresses that belong to the secure region. Also, licensed IPs will only be configured on the PL if its bitstream has frame addresses specific to the region assigned to it. This will avoid licensed IPs' placement into the non-secure region, which will secure them against IP theft attacks. Also, configuration of licensed IPs will not tamper other PL regions (e.g., SD's or other CVs' designs). An additional security measure of the bitstream analysis module restricts the reconfiguration of the X3Y1 clock region after the initial configuration. This is because the design element that gives access to the ICAP is located in this clock region. In addition, SFW components (PRC, TrustZone logic, and connection to ICAP) are also placed in this clock region. This security measure will guarantee that these components cannot be tampered with,

and connectivity to the ICAP is only allowed from them. A detailed discussion on this is provided in Sections 3.1.4 and 7.4.2.

8.6 Integration and Secure Boot

In the integration process, the secure boot feature [142] supported by the target device is used to create an authentic and encrypted boot image with the following components:

- Custom FSBL and PMU Firmware generated by XSDK (see Section 8.2.1).
- The full bitstream of the PL design generated by Vivado (see Section 8.3.3).
- EL3 runtime firmware (bl31.elf), rich OS (u-boot.elf) and trusted OS (bl32.elf) generated by Petalinux tools (see Section 8.4).

Afterward, a 256-bit bitstream encryption key (K_B) and a 128-bit initialization vector (IV) are generated, and partial bitstreams are encrypted using AES in cipher block chaining mode. As discussed in Section 8.5.1, a 4096-bit public-private key pair was generated where the private key is hardcoded in the trusted applications used for decryption and authentication. The public key of that pair is used to encrypt the bitstream key K_B and IV using RSA. The OpenSSL toolkit is used in both cases.

8.7 Application Execution

8.7.1 Encrypted Bitstream

In the IP licensing use case, core vendors provide their IP as an encrypted bitstream $Enc_{K_B}\{IP_j\}$. The bitstream encryption key (K_B) is encrypted with the PK_{SFW} of the public-private key pair, where SK_{SFW} is inside the trusted OS. Furthermore, a hash of $Enc_{K_B}\{IP_j\}$ is generated and signed with PK_{SFW}. The encrypted IP, the signed hash, and encrypted bitstream key are provided to the platform. The execution flow, including authentication, decryption, and configuration on the target platform, are presented below.

- $Enc_{PK_{SFW}}\{K_B\}$, $Enc_{K_B}\{IP_j\}$ and signed hash are read from the storage device into a buffer.
- A hash of the $Enc_{K_B}\{IP_j\}$ is computed and verified against the provided signed hash. If the verification process passes, the process continues with the following steps; otherwise, it aborts.
- $Enc_{PK_{SFW}}\{K_B\}$ is decrypted with SK_{SFW}, and $Enc_{K_B}\{IP_j\}$ is decrypted using K_B.

- Then bitstream analysis is performed on IP_J, and if it has no malicious content, e.g., a target-location mismatch, the execution process will proceed to the next step. Otherwise, it will be aborted.
- IP_J is moved from the trusted OS buffer to the secure memory shown in Figure 8.1.
- PRC registers are updated with the address and size of IP_J's location in RAM.
- Finally, PRC's software trigger register is written, which initiates the PL configuration process.

8.7.2 Plain-text Bitstream

The implemented application can configure a plain-text bitstream IP_J on the PL. This feature is for configuring a bitstream in a non-secure region of the PL. In the IP licensing use case, this bitstream could be either SD's IP or an open-source one, whose protection from the SD is not required. The execution flow of the configuration is presented below.

- IP_J is read from the storage device into a buffer.
- IP_J is moved to the secure memory, shown in Figure 8.1, to avoid tampering during the configuration or bitstream analysis process.
- Then bitstream analysis is done on IP_J. If it is addressed to the non-secure PL region, the next step will be performed. Otherwise, this execution process will be aborted.
- PRC registers are updated with the address and size of IP_J's location in RAM.
- Finally, PRC's software trigger register is written, which initiates the PL configuration process.

9 Security Analysis and Possible Security Enhancements

The work described in this chapter was published in [179] and is joint work with co-authors Sven Nitzsche, Asier Garciandia López, and Jürgen Becker. More details on contributions is found in Section 1.5.

The purpose of the proposed trusted platform is to protect assets against any attacker according to the assumptions and threat model presented in Section 7.3. The main assets in the presented scheme are:

- IP bitstreams provided by core vendors (CVs).
- The security framework (SFW) cryptographic keys.
- The device cryptographic keys for decrypting and authenticating the boot-loader, firmware, full bitstream, and the trusted OS.

Among those, IP bitstreams are the most important asset, and everything else solely exists to protect them.

In general, three entities are participating in the considered IP licensing scenario. Their communication is shown in Figures 7.3, 7.4 and 7.5. Sensitive data such as the assets described above are always protected using encryption and authentication when they are transmitted among participants or stored on external non-volatile memory. Therefore, an attacker can only obtain them in encrypted form. Since the cryptographic algorithms used in this scheme are considered computationally secure, breaking them, if possible at all, would require effort beyond the financial benefit.

Therefore, an attacker's only way to access IPs is to extract the device keys using physical attacks such as side-channel or fault-injection attacks and use them to decrypt the SFW. Afterward, he needs to reverse engineer the SFW to extract its private key. Then, he needs to emulate a device with a valid device identity, for which a specific IP is

licensed. However, only the participants of the licensing process know this information. If the attacker gets this information, too, he can get access to the plain-text bitstream. Furthermore, the stolen IP is tailored for a specific device and SD design; the attacker cannot easily perform cloning or tampering attacks. He may learn about the IP's novelty by reverse engineering. In short, access to the device to do physical and multiple reverse engineering attacks makes this scenario highly unlikely.

Anyone with physical access to the device can perform physical attacks, including system developers (SDs) and their customers. The following section will discuss possible physical attacks for each participating entity, assess them, and propose possible countermeasures.

9.1 Malicious System Developer

Among the participants, SDs can benefit the most from successful IP theft attacks to avoid licensing fees. For example, they could pay for an IP for a single device and then use it for multiple. Since the device is in their possession, they can perform several attacks on different assets. These attacks are discussed in the following sections.

9.1.1 SCAs on Decryptions

There are three types of keys that can be compromised if a successful side-channel attack is performed. The first are device keys used for secure boot. If compromised, an SD can get access to the plain-text SFW. Then they could extract the SFW's private key (SK_{SFW}) and consequently access plain-text IP bitstreams. However, the secure boot feature of modern devices typically includes protection against physical attacks. For example, Xilinx's Ultrascale and later devices include asymmetric authentication, side-channel attack protection, and other anti-tamper features [143]. Therefore, the secure boot process can be considered sufficiently trustworthy.

The other assets that can be compromised using SCAs are the keys used in decryptions, e.g., decryption of the bitstream key ($Enc_{PK_{SFW}}\{K_B\}$) using SFW's private key (SK_{SFW}), and decryption of the encrypted IP ($Enc_{K_B}\{IP_j\}$) using bitstream key (K_B). Both decryptions are implemented in software and are part of a trusted application. An attacker could observe side-channel information of the decryption process and use it to extract the corresponding keys. A possible countermeasure would be an SCA resistant software [111] implementation of the encryption algorithms.

Alternatively, an SCA resistant hardware [44] implementation can also be used for the decryption processes. Chapter 4 presents one of the contributions of the thesis, where a reference AES design is modified with two countermeasures against SCAs. These countermeasures offer substantial resistance against SCAs in cases where power is either recorded using an EM probe over the chip or by measuring the voltage drop across an internal resistor. As presented in the evaluation, a security gain of factor 95 is achieved with an 80% additional CLB resources. The security strength can be further improved by scaling the countermeasures. Another possibility is applying these countermeasures to masked SBOX designs to achieve a higher level of protection. Suppose a first-order masking scheme gives resistance against one million power traces. Then, these countermeasures would make the adversary acquire probably more than 100 million traces. A system composed of masking and hiding would also be resistant against second-order attacks, as it has been shown in related work [26], especially with a higher number of reconfigurable modules.

Also, a security monitor IP core can be used to counter SCA attempts [139] by monitoring device temperature, power supply voltages, and user clocks.

9.1.2 SCAs on DDR Memory

Even though DDR memory is volatile, it retains its contents for several seconds after losing power. Therefore, an attacker can do a hard reset of the target machine and acquire memory contents. This attack is commonly known as a cold-boot attack [47]. The attack can be launched on the example implementation to extract bitstream keys inside trusted OS, which is in the DDR memory's secure part. However, this can be avoided by running OP-TEE (trusted OS) entirely in on-chip memory, which is also the preferred way suggested by its developers [78]. Alternatively, sensitive code and data can be protected against cold boot attacks by encrypting the trusted application before storing it in the DDR memory and only decrypting it within the PS before execution as proposed in [159].

Secondly, a cold boot attack can also be performed on the secure part of the DDR memory, where bitstreams are stored in plaintext form. However, this could be easily avoided using one of the following two workarounds. The first would be updating the trusted application that handles decryption of bitstreams to configure the PL on the fly, i.e., not using DDR memory and reconfiguration controller (PRC) for the PL configuration. However, this would require Xilfpga library-like functionality, discussed in Section 8.2.1, in the trusted OS to configure bitstream via PCAP. The second workaround is specific to the PRC configuring PL via ICAP. Here, decryption can be done using AES core

(preferably the protected one presented in Chapter 4) followed by the configuration via PRC. In these cases, a cold boot attack would only yield encrypted data and, therefore, would be effectively mitigated.

9.1.3 Readback Attack

Readback is a debug feature that can be used to read out the configuration data even if the device's security features are enabled. It is because configuration interfaces (PCAP/ICAP) are considered trusted channels. This feature and its usage as an attack are discussed in Section 3.1.4. Since none of the existing IP licensing schemes restricts SD's control on their device, they all are prone to this attack (see Table 7.1).

In the proposed work, SD's access to these interfaces is restricted. Section 8.2.1 provides implementation details on how access to the PCAP is stopped from both trusted and rich OSes, which will avoid reading out configuration data from this interface. If access to the second interface (ICAP) is also blocked from both OSes, the device will not be able to perform partial reconfiguration or readback. Without the former feature, IP licensing will not work. Therefore, access to ICAP is only allowed from the trusted OS. An API implemented in the trusted OS is made available to the rich OS for indirect access to the ICAP. Here, the purpose of REE's indirect access is to allow SDs to have both readback and reconfiguration features, i.e., the scheme is less restrictive in terms of available features.

The indirect access or isolation of ICAP from the rich OS is made by extending the TEE to the PL. The concept is presented in Section 7.4.2, and implementation details are in Sections 8.5.5 and 8.2.2. The basic idea is to partition the PL into secure and non-secure regions. rich OS has full access to the reconfiguration and readback features of the non-secure PL region and no access to the secure PL region. On the other hand, trusted OS has full access to the entire PL.

Since SD has full access to the non-secure PL region, he can try to perform the readback attack by placing a reconfiguration controller on the non-secure PL region while connecting it to ICAP. However, this scenario would not work because PL's connection to the ICAP lies physically in the X3Y1 clock region. Reconfiguration of this clock region is not allowed in any case (see Property 1 in Section 7.4.2), and therefore he cannot configure the routing resources that provide connection to the ICAP.

Similarly, regions defined for licensed IPs can launch this attack in a potential CV-SD collaboration case. However, this scenario is also not possible because reconfiguration of clock region X3Y1 is never allowed. Furthermore, Property 3 defined in Section 7.4.2 states that bitstreams of the licensed IPs must only have frame addresses specific to the RRs reserved for them.

9.2 Breach of Trust by the TTP

As discussed in Section 7.2.2, the proposed scheme places trust in the trusted third party (TTP). However, measures are taken to reduce the degree of this trust. Section 7.1 defines three levels of trust in the TTP for IP licensing schemes. The first level is high trust, which applies to the schemes where TTP has access to both keys and plaintext IPs. Schemes where the TTP only has access to keys are considered to trust moderately. Finally, schemes where TTP is not involved or does not have access to keys and IPs, place minimal trust in the TTP. The proposed scheme makes sure that IPs are not accessible to the TTP; therefore, it places moderate trust in the TTP. Other schemes use core installation modules (CIMs) to reduce the amount of trust in the TTP [82, 157], which introduces an extra reverse engineering step. The proposed scheme can easily adopt the utilization of CIMs. In this case, CVs configure their CIMs on the device, which includes the bitstream decryption key (K_B).

In limited use cases, where devices are connected to a public network such as the internet, a malicious functionality in the trusted OS could communicate plain-text IPs to the TTP; however, these IPs are device-specific and PL's region-specific. TTP could do minor damage by a cloning attack or learn some novelty using reverse engineering, which of course, would require effort.

9.3 Malicious Core Vendor

Since core vendors do not have physical access to the device, their attack possibilities are limited. The only attack option would be to hide a malicious sub-circuit in their IP core that damages or manipulates the SD design. Such hardware Trojans may be detected at run-time during bitstream analysis as part of the decryption and configuration process. A possible Trojan detection method using machine learning is presented in [110]. Such a machine learning algorithm can be trained remotely and then deployed as part of the SFW. Based on the Trojan taxonomy of Shakya et al. [117], this would detect any Trojan inserted during the IP design phase, no matter whether they are RTL-based or gate-level. Furthermore, modification of the SD's design and the secure PL region (Other IPs regions and SFW components) is avoided by analyzing the IP's bitstream.

IPs are only configured on the PL if they have frame address specific to the RRs reserved for them (see Property 3 in Section 7.4.2).

9.4 TrustZone

Regarding the security of TrustZone-based TEEs, Machiry et al. [81] presented a class of vulnerabilities called BOOMERANG. These vulnerabilities allow the REE application to read and write any memory location. Furthermore, they developed an automated framework and used it to detect bugs within popular TEEs such as QSEE, Trustonic, OP-TEE, Huawei-TEE, and SierraTEE. In another work, Gross et al. in [40] exploited a faulty TZ implementation to break the memory isolation on the target device. A detailed discussion on the security of TZ and TZ-based TEEs are presented by Pinto and Santos in [104]. The proposed work assumes that ARM TZ's implementation on the respective target device, including the supplied firmware, is secure. Attacks due to a faulty implementation are not considered.

9.5 Variants of the Scheme

The proposed scheme can have multiple variants. For example, a batch of devices can be programmed with the same keys, and the same security framework (SFW) can be booted on all those devices. This would make the implementation more manageable, however, at the cost of reduced security. Alternatively, a unique key per device and a unique public-private key pair for the SFW can be programmed. This approach will require more effort but will also provide higher security.

The scheme can also be used to license IPs per device on a duration base, which would require IP's delivery via a secure channel and a continuous internet connection. Furthermore, license validity monitoring functionality needs to be implemented in the trusted OS or secure PL region.

9.6 Performance Evaluation

The overhead on the device for the proposed scheme is minimal. The PL of the device has no metering or core installation module (CIM) bitstream [25, 42, 82]. If a malicious TTP/FV is considered and a CIM is used, like in [157], the programmable resources can be occupied by other designs once the CIM decrypts the respective IP. The only overhead will be on the PS, where a lightweight trusted OS is running. The execution times of the trusted applications depend on the IP size, encryption algorithms, and the

Table 9.1: Decryption of bitstream encryption key (K_B) using RSA.

Size of the RSA key	1024 bits	4096 bits
Size of the K_B (AES key)	256 bits	
Size of encrypted K_B ($Enc\{K_B\}_{PK_{SFW}}$)	1024 bits	4096 bits
Execution time for decrypting $Enc\{K_B\}_{PK_{SFW}}$	207 ms	812 ms

key-widths.

As shown in Table 9.1, an encrypted bitstream key ($Enc\{K_B\}_{PK_{SFW}}$) is decrypted in 207 ms when RSA with a key width of 1024 bits is used. The decryption time increases to 812 ms when RSA with a key width of 4096 bits is used. As mentioned in Section 8.5.2, authentication is the generation of a hash of the encrypted IP and verifying it against the provided signed one. During authentication, only the signature verification can be affected when RSA keys of different widths are used. However, with both key widths (1024 and 4096 bits), the verification took 6 ms. The execution time of hash computation only varies with the varying size of the IP. Therefore, the execution time of the authentication step is shown in Table 9.2, where results are provided for two IPs with sizes of 379 and 720 KB. The Table also shows the execution time of all the functionalities affected by changing the size of the encrypted IPs.

As presented in Table 9.2, the encrypted IP ($Enc\{IP_J\}_{K_B}$) of 379 KB is authenticated and decrypted in 6 and 3 ms, respectively. Its bitstream analysis part takes 38 ms, and moving to the physical memory takes roughly 53 ms. On the other hand, $Enc\{IP_J\}_{K_B}$ of 720 KB is authenticated in 8 ms, decrypted in 5 ms, analyzed in 51 ms, and moved to the physical memory in 61 ms. Partial reconfiguration controller (PRC) registers can be updated in 112 ms, and TA's invoking overhead is 45 ms, which is the same for both the IPs. Configuration of IPs using the PRC (ICAP), with a bandwidth of 3.2 Gb/s at 100 MHz [148], takes 0.97 and 1.84 ms for IPs of 379 and 720 KB size, respectively. The overall execution times for all four combinations with two RSA key widths and two bitstream sizes are also presented in Table 9.2. The PRC registers update step can be skipped if the registers are set at the design time, which will reduce the overall execution time for each case by 112 ms.

Table 9.2: The execution time of the functionalities along with different RSA keys and encrypted IPs ($Enc\{IP_J\}_{K_B}$) sizes. Bitstream encryption key (K_B) width is 256 bits.

Size of IP_J and $Enc\{IP_J\}_{K_B}$	**379 KB**	**720 KB**
Authentication of $Enc\{IP_J\}_{K_B}$	6 ms	8 ms
Execution time for decrypting $Enc\{IP_J\}_{K_B}$	3 ms	5 ms
Execution time for PL partitioning	38 ms	51 ms
Moving IP_J to physical memory	53 ms	61 ms
PRC's registers update	112 ms	
Overhead of invoking TA	45 ms	
Configuration time of IP_J using PRC	0.97 ms	1.84 ms
Total execution time with RSA key of 1024 bits	464 ms	1069 ms
Total execution time with RSA key of 4096 bits	489 ms	1094 ms

10 Conclusion

This dissertation, in a nutshell, tries to answer the question: can one trust a device with his intellectual property (IP)? A continued examination and improvement of security mechanisms will ensure that they are resistant to future security threats. Organizations like Globalplatform or Trusted Computing Group are created for this reason. They publish and revise the specification of TEEs and TPMs, respectively. For this work, the same philosophy is followed where publicly available security mechanisms are used to realize trust on a device. The target use case is licensing FPGA-based IPs, where multiple IP core vendors (CVs) can deliver their protected IPs directly to the device. Since the IPs are only decrypted inside the trusted device, CVs can be confident that their IPs will not end up with an adversary.

Trust establishment starts with an IP licensing model where practicality, security, and least restrictiveness are the main focuses. The device is equipped with a TEE-based security framework, which isolates some device assets to provide a trusted environment for security-related tasks. The rest of the device assets are available to the system developer (SD) for their IPs. A significant contribution of this work is the extension of this concept to the programmable logic (PL) of the device. Some of the assets on the PL are isolated and used only for security-related tasks, while the rest are used by the SD's design. This isolation on the PL is enforced by limiting access of the SD to the configuration interfaces and bitstream analysis. The bitstream analysis functionality ensures that licensed IPs are only configured on the PL regions reserved for them. This is done to protect SD's logic from modification/damage from a malicious licensed IP. Bitstream analysis also ensures that SD design does not tamper or readback licensed IPs or other security-related implementation on the PL.

This dissertation considers everyone as a potential attacker except the trusted third party (TTP). The broader attacker model is used so that all the security threats can be analyzed and necessary countermeasures can be either developed or proposed. Also, a malicious TTP scenario is considered, and countermeasures are proposed from the

literature. This work also considers threats from malicious CVs whose intention can be to damage or leak information from the system.

Unlike other related IP licensing models, the focus of this dissertation is not limited to only security threats. Here, challenges specific to practicality are also considered, which are floorplanning of IPs and generation of IPs independent of the static design. The floorplanning challenge is solved by a fine-grained MILP-based floorplanner that supports the latest heterogeneous FPGA devices. The floorplanner shows a clear improvement in resource wastage when compared with other existing floorplanners. The second practice issue, IPs' independent generation, is also addressed. The main solution to this challenge is a use case-specific flow, which only uses FV tools to generate the bitstream of IP and the static design.

Last but not least, the dissertation also provides two countermeasures against side-channel attacks (SCAs). The countermeasures are based on moving target and implementation diversity concepts. The basic idea of the moving target is physically moving a target function of the encryption algorithm on the PL using DPR. Moving target is effective against EM-based SCAs as they usually use a small probe, which is moved over the chip to find the suitable position (i.e., the target function). Randomly moving the target function to different positions makes traces from all positions mixed up; therefore, it counters position-based attacks. The implementation diversity works by having multiple implementations of the target function that are functionally equivalent but have different physical layouts. These implementations are continuously reconfigured on the same location, which results in varying dynamic power consumption. This acts as a countermeasure against power measurement-based SCAs.

10.1 Future Work

The future work can be considered from two viewpoints. The first one is improving the trust in the device, and the second is developing applications running on the device that offers specific functionality.

10.1.1 Trust in Devices

In this dissertation, trust in devices is established using a security framework (SFW). This trust can be further extended by adding remote attestation functionality for the system developer (SD) and IP core vendors (CV). Remote attention is a technology, which offers evidence to remote entities about the authentication of its hardware and

software configuration. A multi-tenet concept of remote attestation can be investigated to find a trade-off between resource overhead and extension of trust. This multi-tenet concept will allow CVs to monitor the state of their IP (both software and FPGA one), and SDs can monitor the overall state of the device.

The device's internal decryption engines (DDEs) offer several protection mechanisms like secure storage, tamper, or SCA resistance. Further research can be done on these engines, where they use public-key cryptography, including unique key generation and decryption. Furthermore, it must be ensured that the private key never leaves the DDE and is stored in secure storage. The public key can be made available via an interface with a specified protocol. In the presence of these details, mechanisms can be used to issue digital certificates to such devices. IP core vendors can establish a secure communication channel with the devices, simplifying IP licensing models. Where IPs can be securely delivered to these devices.

Shepherd et al. in [118] defined three On-Device adversaries, namely Application, Kernel, and Hardware adversaries. The proposed work is focused mainly on hardware adversaries, and throughout the work, attacks due to a faulty TEE implementation are not considered. Protection against application and kernel adversaries could also be another future direction.

10.1.2 Application Development

Examples of applications running on the trusted device are detecting malicious functionality in an IP and decryption engines. These functionalities can be both in software or hardware. The first example, Malicious IPs, is presented in Section 3.3. The malicious functionality can be tampering configured-designs or hardware Trojans. The tampering of configured design is implemented using bitstream analysis. However, for the detection of Trojans, only ideas are presented in Section 9.3.

Trojan detection for FPGA-based IPs is an exciting research area and can be one of the future research directions. Shakya et al. [117] categorized Trojan detection techniques into pre-silicon-, post-silicon detection, and design for trust. Post-silicon detection includes techniques that detect Trojans in manufactured chips. These techniques usually apply to the physical layout of the chip and can be considered in cases where the hardware manufacturer is considered a potential attacker. The other two techniques can be further investigated.

The pre-silicon detection includes functional testing [117] and structural analysis [160] techniques, which analyze HDLs for redundant code or conditional statements that are rarely triggered. These techniques can only be used with the proposed trusted device if the device's TEE delivers IP's bitstream to a trusted cloud, where the IPs are reverse engineered to netlist [11, 95].

The design for trust techniques includes run-time monitoring, detecting an increase in side-channel activity, or machine learning-based Trojan detection in the bitstream (Section 9.3). These techniques can be further researched and implemented as an application on the device.

Even though detecting an increase in the side-channel activity is discussed specifically for hardware Trojan, this technique can be generalized to detect SCAs performed on the decryption engines of the device. This could be another topic for future research where an application provides security detection/protection functionality.

Abbreviations

AES	advanced encryption standard
AES-GCM	AES - Galois/counter mode
AXI4	Advanced eXtensible Interface 4
CIM	core installation module
CLB	configurable logic block
CSU	configuration security unit
CV	IP core vendor
DDE	dedicated decryption engine
DES	data encryption standard
DPR	dynamic partial reconfiguration
DSP	digital signal processing
EM	electromagnetic
EPROM	erasable programmable read-only memory
FPGA	Field programmable gate array
FSBL	first stage boot loader
FV	FPGA vendor
HDL	hardware description language
HSM	hardware security module
IC	integrated circuit
ICAP	internal configuration access port
IIPG	independent IP generation
IP	intellectual property
LUT	lookup table
MAC	message authentication code
MILP	mixed-integer linear programming
NG	Noise Generator
NIST	National Institute of Standards and Technology
NVM	non-volatile memory
PCAP	processor configuration access port

PL	programmable logic
PLD	programmable logic device
PMU	platform management unit
PRC	partial reconfiguration controller
PS	processing system
PU	programmable unit
RD	reconfigurable design
REE	rich execution environment
RM	reconfigurable module
RNG	random number generator
ROT	root of trust
RR	reconfigurable region
RTL	register transfer level
SBOX	substitution byte
SCA	side-channel attack
SD	system developer
SFW	security framework
SoC	system-on-chip
SRAM	static random access memory
TA	trusted application
TEE	trusted execution environment
TFR	Target Function Relocation
TPM	trusted platform module
TTP	trusted third party

List of Figures

List of Tables

Bibliography

[1] Abraham, D. G., G. M. Dolan, G. P. Double, and J. V. Stevens (1991): Transaction Security System. *IBM Syst. J.* **30**(2), pp. 206–229. doi: 10.1147/sj.302.0206. url: https://doi.org/10.1147/sj.302.0206.

[2] Akram, R. N., K. Markantonakis, and K. Mayes (2014): An Introduction to the Trusted Platform Module and Mobile Trusted Module. In: *Secure Smart Embedded Devices, Platforms and Applications*. K. Markantonakis and K. Mayes eds. New York, NY: Springer New York, pp. 71–93. doi: 10.1007/978-1-4614-7915-4_4. url: https://doi.org/10.1007/978-1-4614-7915-4_4.

[3] Alves, T. (2004): TrustZone : Integrated Hardware and Software Security. In:

[4] Amano, H., ed. (2018): *Principles and Structures of FPGAs*. Springer. doi: 10.1007/978-981-13-0824-6. url: https://doi.org/10.1007/978-981-13-0824-6.

[5] Android (n.d.): *Android Key Store*. Accessed: Mar. 25, 2020. url: https://developer.android.com/training/articles/keystore.

[6] *Arm security technology-building a secure system using trustzone technology* (n.d.). Accessed: April. 26, 2021. url: https://developer.arm.com/documentation/genc009492/c/TrustZone-Hardware-Architecture/Overview.

[7] Bacchini, A. (n.d.): *OpenCore Canny Edge Detectore*. Online; accessed 16-03-2020. url: https://opencores.org/projects/canny_edge_detector.

[8] Bacchini, A. (n.d.): *OpenCores B-163 EC Arithmetic*. Online; accessed 16-03-2020. url: https://opencores.org/projects/b163arith.

[9] Bailey, D. G., C. T. Johnston, and Ni Ma (2008): Connected components analysis of streamed images. In: *2008 International Conference on Field Programmable Logic and Applications*, pp. 679–682. doi: 10.1109/FPL.2008.4630038.

[10] Beckhoff, C., D. Koch, and J. Torresen (2012): Go Ahead: A Partial Reconfiguration Framework. In: *IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 37–44. doi: 10.1109/FCCM.2012.17.

[11] Benz, F., A. Seffrin, and S. A. Huss (2012): Bil: A tool-chain for bitstream reverse-engineering. In: *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 735–738. doi: 10.1109/FPL.2012.6339165.

[12] Boivie, R. (2011): SecureBlue + + : CPU Support for Secure Execution. In:

[13] Bolchini, C., A. Miele, and C. Sandionigi (2011): Automated Resource-Aware Floorplanning of Reconfigurable Areas in Partially-Reconfigurable FPGA Systems. In: *2011 21st International Conference on Field Programmable Logic and Applications*, pp. 532–538. doi: 10.1109/FPL.2011.104.

[14] Bossuet, L., G. Gogniat, and W. Burleson (n.d.): Dynamically configurable security for SRAM FPGA bitstreams. In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* Pp. 146–. doi: 10.1109/IPDPS.2004.1303128.

[15] Bow, I., N. Bete, F. Saqib, W. Che, C. Patel, R. Robucci, C. Chan, and J. Plusquellic (2020): Side-Channel Power Resistance for Encryption Algorithms Using Implementation Diversity. *Cryptography* **4**(2). doi: 10.3390/cryptography4020013. url: https://www.mdpi.com/2410-387X/4/2/13.

[16] Brier, E., C. Clavier, and F. Olivier (2004): Correlation Power Analysis with a Leakage Model. In: *Cryptographic Hardware and Embedded Systems - CHES 2004*. M. Joye and J.-J. Quisquater eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 16–29.

[17] Caldwell, A. E., Hyun-Jin Choi, A. B. Kahng, S. Mantik, M. Potkonjak, Gang Qu, and J. L. Wong (2004): Effective iterative techniques for fingerprinting design IP. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **23**(2), pp. 208–215. doi: 10.1109/TCAD.2003.822126.

[18] Canright, D. (2005): A Very Compact S-Box for AES. In: *Cryptographic Hardware and Embedded Systems – CHES 2005*. J. R. Rao and B. Sunar eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 441–455.

[19] Chami, C. A. (n.d.): *Pseudo random generator Tutorial*. Accessed: Mar. 16, 2019. url: https://fpgasite.wordpress.com/2016/08/09/pseudo-random-generator-tutorial/.

[20] Chang, C. and L. Zhang (2014): A Blind Dynamic Fingerprinting Technique for Sequential Circuit Intellectual Property Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **33**(1), pp. 76–89. doi: 10.1109/TCAD.2013.2282282.

[21] Chhabra, S., B. Rogers, Y. Solihin, and M. Prvulovic (2011): SecureME: a hardware-software approach to full system security. In: *Proceedings of the international conference on Supercomputing*, pp. 108–119.

[22] Choudhary, V. (2007): Comparison of Software Quality Under Perpetual Licensing and Software as a Service. *Journal of Management Information Systems* **24**(2), pp. 141–165. doi: 10.2753/MIS0742-1222240206. eprint: https://doi.org/10.2753/MIS0742-1222240206. url: https://doi.org/10.2753/MIS0742-1222240206.

[23] Costan, V. and S. Devadas (2016): Intel SGX Explained. *IACR Cryptology ePrint Archive* **2016**(086), pp. 1–118.

[24] Diffie, W. and M. Hellman (1976): New directions in cryptography. *IEEE Transactions on Information Theory* **22**(6), pp. 644–654. doi: 10.1109/TIT.1976.1055638.

[25] Drimer, S., T. Güneysu, M. Kuhn, and C. Paar (2008): Protecting multiple cores in a single FPGA design. Draft.

[26] Druyer, R., L. Torres, P. Benoit, P. V. Bonzom, and P. Le-Quere (2015): A survey on security features in modern FPGAs. In: *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pp. 1–8. doi: 10.1109/ReCoSoC.2015.7238102.

[27] Economist, T. (n.d.): *Universal service?* Accessed: Mar. 24, 2020. url: https://www.economist.com/business/2006/04/20/universal-service.

[28] Evtyushkin, D., J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley (2018): Flexible Hardware-Managed Isolated Execution: Architecture, Software Support and Applications. *IEEE Transactions on Dependable and Secure Computing* **15**(3), pp. 437–451.

[29] Fitzek, A., F. Achleitner, J. Winter, and D. Hein (2015): The ANDIX research OS — ARM TrustZone meets industrial control systems security. *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pp. 88–93.

[30] Five, H. (n.d.): *Hex Five Announces General Availability of MultiZone™ Security for Linux – The First Commercial Enclave for RISC-V processors*. Accessed: Jan. 13, 2020. url: https://riscv.org/2019/12/hex-five-announces-general-availability-of-multizone-security-for-linux-the-first-commercial-enclave-for-risc-v-processors/.

[31] Force, I. E. T. (n.d.): *The Transport Layer Security (TLS) Protocol Version 1.3*. retrieved from https://tools.ietf.org/html/rfc8446.

[32] FREUNDSCHAFTER (n.d.): *About AMD TrustZone, AMD Platform Security Processor (PSP), AMD Secure Technology*. Accessed: Jan. 13, 2020. url: https://freundschafter.com/research/about-amd-trustzone-amd-platform-security-processor-psp-amd-secure-technology/.

[33] Gandolfi, K., C. Mourtel, and F. Olivier (2001): Electromagnetic Analysis: Concrete Results. In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ç. K. Koç, D. Naccache, and C. Paar eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 251–261.

[34] Gandolfi, K., C. Mourtel, and F. Olivier (2001): Electromagnetic Analysis: Concrete Results. In: *Cryptographic Hardware and Embedded Systems — CHES 2001*. Ç. K. Koç, D. Naccache, and C. Paar eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 251–261.

[35] Gaspar, L., V. Fischer, T. Güneysu, and Z. C. Jouini (2012): Two IP protection schemes for multi-FPGA systems. In: *2012 International Conference on Reconfigurable Computing and FPGAs*, pp. 1–6. doi: 10.1109/ReConFig.2012.6416790.

[36] Globalplatform (n.d.). retrieved from https://globalplatform.org/.

[37] Globalplatform (2016): *GlobalPlatform Device Technology TEE Internal Core API Specification Version 1.1.1*. retrieved from https://globalplatform.org/.

[38] Globalplatform (2018): *Introduction to Trusted Execution Environments*. retrieved from https://globalplatform.org/.

[39] Gora, M. A., A. Maiti, and P. Schaumont (2010): A Flexible Design Flow for Software IP Binding in FPGA. *IEEE Transactions on Industrial Informatics* **6**(4), pp. 719–728. doi: 10.1109/TII.2010.2068303.

[40] Gross, M., N. Jacob, A. Zankl, and G. Sigl (2019): Breaking TrustZone Memory Isolation through Malicious Hardware on a Modern FPGA-SoC. In: *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*. ASHES'19. London, United Kingdom: Association for Computing Machinery, pp. 3–12. doi: 10.1145/3338508.3359568. url: https://doi.org/10.1145/3338508.3359568.

[41] Guajardo, J., S. S. Kumar, G.-J. Schrijen, and P. Tuyls (2007): FPGA Intrinsic PUFs and Their Use for IP Protection. In: *Cryptographic Hardware and Embedded Systems - CHES 2007*. P. Paillier and I. Verbauwhede eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 63–80.

[42] Güneysu, T., B. Moller, and C. Paar (2007): Dynamic Intellectual Property Protection for Reconfigurable Devices. In: *2007 International Conference on Field-Programmable Technology*, pp. 169–176. doi: 10.1109/FPT.2007.4439246.

[43] Güneysu, T. and A. Moradi (2011): Generic Side-Channel Countermeasures for Reconfigurable Devices. In: *Cryptographic Hardware and Embedded Systems – CHES 2011*. B. Preneel and T. Takagi eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 33–48.

[44] Güneysu, T. and A. Moradi (2011): Generic Side-Channel Countermeasures for Reconfigurable Devices. In: *CHES*. Vol. 6917. Lecture Notes in Computer Science. Springer, pp. 33–48. doi: 10.1007/978-3-642-23951-9_3. url: https://www.iacr.org/archive/ches2011/69170033/69170033.pdf.

[45] *GUROBI OPTIMIZATION Documentation* (n.d.). Online; accessed 16-03-2020. url: https://www.gurobi.com/documentation/9.0/refman/mipfocus.html.

[46] *Gurobi Optimizer* (n.d.). Version 9.0.0. url: https://www.gurobi.com/products/gurobi-optimizer/.

[47] Halderman, J. A., S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten (2009): Lest We Remember:

Cold-Boot Attacks on Encryption Keys. *Commun. ACM* **52**(5), pp. 91–98. DOI: 10.1145/1506409.1506429. URL: https://doi.org/10.1145/1506409.1506429.

[48] Haroldsen, T., B. Nelson, and B. Hutchings (2015): RapidSmith 2: A Framework for BEL-Level CAD Exploration on Xilinx FPGAs. In: *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '15. Monterey, California, USA: Association for Computing Machinery, pp. 66–69. DOI: 10.1145/2684746.2689085. URL: https://doi.org/10.1145/2684746.2689085.

[49] Hettwer, B., J. Petersen, S. Gehrer, H. Neumann, and T. Güneysu (2019): Securing Cryptographic Circuits by Exploiting Implementation Diversity and Partial Reconfiguration on FPGAs. In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 260–263. DOI: 10.23919/DATE.2019.8714801.

[50] Hofmann, O. S., S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel (2013): InkTag: Secure Applications on an Untrusted Operating System. *SIGPLAN Not.* **48**(4), pp. 265–278. DOI: 10.1145/2499368.2451146. URL: https://doi.org/10.1145/2499368.2451146.

[51] Huang, A. (2002): Hacking the Xbox: an introduction to reverse engineering.

[52] Huss, S. and O. Stein (2017): A Novel Design Flow for a Security-Driven Synthesis of Side-Channel Hardened Cryptographic Modules. *Journal of Low Power Electronics and Applications* **7**(1), p. 4. DOI: 10.3390/jlpea7010004. URL: http://dx.doi.org/10.3390/jlpea7010004.

[53] Iida, M. (2018): What Is an FPGA? In: *Principles and Structures of FPGAs*. Springer, pp. 23–45.

[54] K., S. K., S. Sahoo, A. Mahapatra, A. K. Swain, and K. K. Mahapatra (2017): A Flexible Pay-per-Device Licensing Scheme for FPGA IP Cores. In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 677–682. DOI: 10.1109/ISVLSI.2017.123.

[55] Kavvadias, N. (n.d.): *OpenCores Universal multi-function CORDIC*. Online; accessed 16-03-2020. URL: https://opencores.org/projects/kvcordic.

[56] Kean, T. (2001): Secure Configuration of a Field Programmable Gate Array. In: *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, pp. 259–260.

[57] Kean, T. (2002): Cryptographic Rights Management of FPGA Intellectual Property Cores. In: *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*. FPGA '02. Monterey, California, USA: Association for Computing Machinery, pp. 113–118. DOI: 10.1145/503048.503065. URL: https://doi.org/10.1145/503048.503065.

[58] Khan, N. (n.d.): *Milp-Formulation*. URL: https://github.com/khankit/Milp-Formulation.

[59] Kim, Y., R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu (2014): Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 361–372. doi: 10.1109/ISCA.2014.6853210.

[60] Koch, D., C. Beckhoff, and J. Teich (2008): ReCoBus-Builder—A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS. In: *2008 International Conference on Field Programmable Logic and Applications*. IEEE, pp. 119–124.

[61] Kocher, P. C. (1996): Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: *Advances in Cryptology — CRYPTO '96*. N. Koblitz ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 104–113.

[62] Kocher, P. C. (1996): Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: *Advances in Cryptology — CRYPTO '96*. N. Koblitz ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 104–113.

[63] Kocher, P., J. Jaffe, and B. Jun (1999): Differential Power Analysis. In: *Advances in Cryptology — CRYPTO' 99*. M. Wiener ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 388–397.

[64] Kocher, P., J. Jaffe, and B. Jun (1999): Differential Power Analysis. In: *Advances in Cryptology — CRYPTO' 99*. M. Wiener ed. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 388–397.

[65] Krepa, M. (n.d.): *OpenCores Discrete Cosine Transform core*. Online; accessed 16-03-2020. url: https://opencores.org/projects/mdct.

[66] Kumar, S. S., J. Guajardo, R. Maes, G. Schrijen, and P. Tuyls (2008): Extended abstract: The butterfly PUF protecting IP on every FPGA. In: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 67–70.

[67] Kwong, A., D. Genkin, D. Gruss, and Y. Yarom (2020): RAMBleed: Reading Bits in Memory Without Accessing Them. In: *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 695–711. doi: 10.1109/SP40000.2020.00020. url: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00020.

[68] La, T. M., K. Matas, N. Grunchevski, K. D. Pham, and D. Koch (2020): FPGADefender: Malicious Self-oscillator Scanning for Xilinx UltraScale + FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* **13**(3). doi: 10.1145/3402937. url: https://doi.org/10.1145/3402937.

[69] Lach, J., W. H. Mangione-Smith, and M. Potkonjak (2001): Fingerprinting techniques for field-programmable gate array intellectual property protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **20**(10), pp. 1253–1261. doi: 10.1109/43.952741.

[70] Lal, V. (n.d.): *LFSR-Random number generator*. Accessed: Mar. 16, 2019. URL: https://opencores.org/projects/lfsr_randgen.

[71] Lavin, C., M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings (2010): Rapid prototyping tools for FPGA designs: RapidSmith. In: *2010 International Conference on Field-Programmable Technology*, pp. 353–356. DOI: 10.1109/FPT.2010.5681429.

[72] Lavin, C., M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings (2011): RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs. In: *2011 21st International Conference on Field Programmable Logic and Applications*, pp. 349–355. DOI: 10.1109/FPL.2011.69.

[73] Lavin, C. and A. Kaviani (2018): RapidWright: Enabling Custom Crafted Implementations for FPGAs. In: *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 133–140. DOI: 10.1109/FCCM.2018.00030.

[74] Lee, R. B., P. C. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang (2005): Architecture for protecting critical secrets in microprocessors. In: *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, pp. 2–13.

[75] Linaro (n.d.): *Architecture*. Accessed Oct. 26, 2020. URL: https://optee.readthedocs.io/en/latest/architecture/index.html.

[76] Linaro (n.d.): *OP-TEE*. URL: https://github.com/OP-TEE.

[77] Linaro (n.d.): *Open Portable Trusted Execution Environment*. Accessed Mar. 25, 2020. URL: https://optee.readthedocs.io/en/latest/general/about.html.

[78] Linaro (n.d.): *Projects*. Accessed July. 15, 2020. URL: https://www.linaro.org/projects/.

[79] Linaro (n.d.): *Trusted Applications*. Accessed Mar. 06, 2020. URL: https://optee.readthedocs.io/en/latest/architecture/trusted%7B%5C_%7Dapplications.html.

[80] Liu, G. (n.d.): *Gaussian Noise Generator (GNG) Verilog IP Core*. Accessed: Mar. 16, 2019. URL: https://github.com/liuguangxi/gng.

[81] Machiry, A., E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Krügel, and G. Vigna (2017): BOOMERANG: Exploiting the Semantic Gap in TEEs. In: *NDSS*.

[82] Maes, R., D. Schellekens, and I. Verbauwhede (2012): A Pay-per-Use Licensing Scheme for Hardware IP Cores in Recent SRAM-Based FPGAs. *IEEE Transactions on Information Forensics and Security* **7**(1), pp. 98–108. DOI: 10.1109/TIFS.2011.2169667.

[83] Mangard, S., E. Oswald, and T. Popp (2007): *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Berlin, Heidelberg: Springer-Verlag.

[84] Mavrovouniotis, S. and M. Ganley (2014): Hardware Security Modules. In: *Secure Smart Embedded Devices, Platforms and Applications*. K. Markantonakis and K. Mayes eds. New York, NY: Springer New York, pp. 383–405. doi: 10.1007/978-1-4614-7915-4_17. url: https://doi.org/10.1007/978-1-4614-7915-4_17.

[85] McGillion, B., T. Dettenborn, T. Nyman, and N. Asokan (2015): Open-TEE–An Open Virtual Trusted Execution Environment. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE, pp. 400–407.

[86] Mentens, N., B. Gierlichs, and I. Verbauwhede (2008): Power and Fault Analysis Resistance in Hardware through Dynamic Reconfiguration. In: *Cryptographic Hardware and Embedded Systems – CHES 2008*. E. Oswald and P. Rohatgi eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 346–362.

[87] Mentens, N. (2017): Hiding side-channel leakage through hardware randomization: A comprehensive overview. In: *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pp. 269–272. doi: 10.1109/SAMOS.2017.8344639.

[88] Moghaddam, E., N. Mukherjee, J. Rajski, J. Tyszer, and J. Zawada (2016): On Test Points Enhancing Hardware Security. In: *2016 IEEE 25th Asian Test Symposium (ATS)*, pp. 61–66. doi: 10.1109/ATS.2016.24.

[89] Montone, A., M. D. Santambrogio, and D. Sciuto (2010): Wirelength driven floorplacement for FPGA-based partial reconfigurable systems. In: *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, pp. 1–8. doi: 10.1109/IPDPSW.2010.5470756.

[90] Moradi, A. and T. Schneider (2016): Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series. *IACR Cryptol. ePrint Arch.* **2016**, p. 249. url: http://eprint.iacr.org/2016/249.

[91] Mulnix, D. (n.d.): *Intel® Trusted Execution Technology (Intel® TXT) Enabling Guide*. Accessed: Mar. 24, 2020. url: https://software.intel.com/en-us/articles/intel-trusted-execution-technology-intel-txt-enabling-guide.

[92] Nane, R., V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels (2016): A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **35**(10), pp. 1591–1604.

[93] Nguyen, T. T., M. Thevenin, A. Mouraud, G. Corre, O. Pasquier, and S. Pillement (2018): High-Level Reliability Evaluation of Reconfiguration-Based Fault Tolerance Techniques. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 202–205. doi: 10.1109/IPDPSW.2018.00038.

[94] Nomata, Y., M. Matsubayashi, K. Sawada, and A. Satoh (2016): Comparison of side-channel attack on cryptographic cirucits between old and new technology

FPGAs. In: *2016 IEEE 5th Global Conference on Consumer Electronics*, pp. 1–4. DOI: 10.1109/GCCE.2016.7800555.

[95] Note, J.-B. and É. Rannaud (2008): From the Bitstream to the Netlist. In: *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*. FPGA '08. Monterey, California, USA: Association for Computing Machinery, p. 264. DOI: 10.1145/1344671.1344729. URL: https://doi.org/10.1145/1344671.1344729.

[96] Note, J.-B. and É. Rannaud (2008): From the Bitstream to the Netlist. In: *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*. FPGA '08. Monterey, California, USA: Association for Computing Machinery, p. 264. DOI: 10.1145/1344671.1344729. URL: https://doi.org/10.1145/1344671.1344729.

[97] O'Flynn, C. and Z. Chen (2013): A Case Study of Side-Channel Analysis Using Decoupling Capacitor Power Measurement with the OpenADC. In: *Foundations and Practice of Security*. J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, A. Miri, and N. Tawbi eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 341–356.

[98] Owusu, E., J. Guajardo, J. McCune, J. Newsome, A. Perrig, and A. Vasudevan (2013): OASIS: On achieving a sanctuary for integrity and secrecy on untrusted platforms. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 13–24.

[99] Paar, C. and J. Pelzl (2010): *Understanding cryptography: a textbook for students and practitioners*. Berlin: Germany:Springer-Verlag.

[100] Pappala, S., M. Niamat, and W. Sun (2012): FPGA based key generation technique for anti-counterfeiting methods using Physically Unclonable Functions and artificial intelligence. In: *IEEE*, pp. 388–393. DOI: 10.1109/FPL.2012.6339209.

[101] *PartialReconfigurationUser Guide (UG702)* (2013). Xilinx Inx.

[102] de la Piedra, A. (n.d.): *OpenCores SHA-256 Core*. Online; accessed 16-03-2020. URL: https://opencores.org/projects/sha256core.

[103] de la Piedra, A. (n.d.): *OpenCores XTEA Core*. Online; accessed 16-03-2020. URL: https://opencores.org/projects/xteacore.

[104] Pinto, S. and N. Santos (2019): Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* **51**(6). DOI: 10.1145/3291047. URL: https://doi.org/10.1145/3291047.

[105] Pocklassery, G., W. Che, F. Saqib, M. Areno, and J. Plusquellic (2018): Self-authenticating secure boot for FPGAs. In: *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 221–226. DOI: 10.1109/HST.2018.8383919.

[106] *Pseudo random number generator Tutorial* (n.d.). Accessed: Mar. 16, 2019. URL: http://fpgasite.blogspot.com/2017/04/pseudo-random-generator-tutorial.html.

[107] Rabozzi, M., J. Lillis, and M. D. Santambrogio (2014): Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming. In: *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 186–193. DOI: 10.1109/FCCM.2014.61.

[108] Rabozzi, M., G. C. Durelli, A. Miele, J. Lillis, and M. D. Santambrogio (2017): Floorplanning Automation for Partial-Reconfigurable FPGAs via Feasible Placements Generation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **25**(1), pp. 151–164. DOI: 10.1109/TVLSI.2016.2562361.

[109] Reorda, M. S., L. Sterpone, and A. Ullah (2017): An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems. *IEEE Transactions on Computers* **66**(6), pp. 1022–1033. DOI: 10.1109/TC.2016.2607749.

[110] Rettkowski, J., S. Mahmood, A. Shallufa, M. Hübner, and D. Göhringer (2019): Inspection of Partial Bitstreams for FPGAs Using Artificial Neural Networks. In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 83–86.

[111] Rivain, M., E. Prouff, and J. Doget (2009): Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In: *Cryptographic Hardware and Embedded Systems - CHES 2009*. C. Clavier and K. Gaj eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 171–188.

[112] Santos, N., H. Raj, S. Saroiu, and A. Wolman (2014): Using ARM TrustZone to build a trusted language runtime for mobile applications. In: *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, pp. 67–80.

[113] Sasdrich, P., A. Moradi, O. Mischke, and T. Güneysu (2015): Achieving side-channel protection with dynamic logic reconfiguration on modern FPGAs. In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 130–136. DOI: 10.1109/HST.2015.7140251.

[114] Sasdrich, P., A. Moradi, and T. Güneysu (2017): Hiding Higher-Order Side-Channel Leakage. In: *Topics in Cryptology – CT-RSA 2017*. H. Handschuh ed. Cham: Springer International Publishing, pp. 131–146.

[115] Schrittwieser, L. (n.d.): *OpenCores LZRW1 Compressor Core*. Online; accessed 16-03-2020. URL: https://opencores.org/projects/lzrw1-compressor-core.

[116] Seyoum, B. B., A. Biondi, and G. C. Buttazzo (2019): FLORA: FLoorplan Optimizer for Reconfigurable Areas in FPGAs. *ACM Trans. Embed. Comput. Syst.* **18**(5s), 73:1–73:20. DOI: 10.1145/3358202. URL: http://doi.acm.org/10.1145/3358202.

[117] Shakya, B., M. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor (2017): Benchmarking of Hardware Trojans and Maliciously Affected Circuits. *Journal of Hardware and Systems Security* **1**, pp. 85–102.

[118] Shepherd, C., G. Arfaoui, I. Gurulian, R. P. Lee, K. Markantonakis, R. N. Akram, D. Sauveron, and E. Conchon (2016): Secure and Trusted Execution: Past, Present, and Future - A Critical Review in the Context of the Internet of Things and Cyber-Physical Systems. In: *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 168–177. doi: 10.1109/TrustCom.2016.0060.

[119] *Simple sine wave generator in VHDL* (n.d.). Accessed: Mar. 16, 2019. url: https://vhdlguru.blogspot.com/2010/03/simple-sine-wave-generator-in-vhdl.html.

[120] Simpson, E. and P. Schaumont (2006): Offline Hardware/Software Authentication for Reconfigurable Platforms. In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. L. Goubin and M. Matsui eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 311–323.

[121] Skorobogatov, S. and C. Woods (2012): In the blink of an eye: There goes your AES key. *IACR Cryptol. ePrint Arch.* **2012**, p. 296. url: http://eprint.iacr.org/2012/296.

[122] Sohanghpurwala, A. A., P. Athanas, T. Frangieh, and A. Wood (2011): OpenPR: An open-source partial-reconfiguration toolkit for Xilinx FPGAs. In: *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, pp. 228–235.

[123] Sun, H., K. Sun, Y. Wang, J. Jing, and H. Wang (2015): TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices. *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 367–378.

[124] Sun, P. and A. Cui (2019): A New Pay-Per-Use Scheme for the Protection of FPGA IP. In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5. doi: 10.1109/ISCAS.2019.8702721.

[125] Sundaramoorthy, N., N. Rao, and T. Hill (2010): AXI4 Interconnect Paves the Way to Plug-and-Play IP. *Xilinx White Paper WP379 v1. 0*.

[126] Tajik, S., H. Lohrke, J.-P. Seifert, and C. Boit (2017): On the Power of Optical Contactless Probing: Attacking Bitstream Encryption of FPGAs. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, pp. 1661–1674. doi: 10.1145/3133956.3134039. url: https://doi.org/10.1145/3133956.3134039.

[127] Tatas, K., K. Siozios, and D. Soudris (2007): A Survey of Existing Fine-Grain Reconfigurable Architectures and CAD tools. *Vassiliadis S., Soudris D. (eds) Fine-*

and Coarse-Grain Reconfigurable Computing. Springer, Dordrecht. DOI: https://doi.org/10.1007/978-1-4020-6505-7_1.

[128] Townsend, T. and B. Nelson (2017): Vivado design interface: An export/import capability for Vivado FPGA designs. In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–7.

[129] Uzenkov, O. (n.d.): *OpenCores Pipelined DCT/IDCT*. Online; accessed 16-03-2020. URL: https://opencores.org/projects/dct_idct.

[130] *verilog code for SINE PWM* (n.d.). Accessed: Mar. 16, 2019. URL: https://community.intel.com/t5/Intel-Quartus-Prime-Software/verilog-code-for-SINE-PWM/td-p/144210.

[131] Villar, J. C. (n.d.): *Opencores 128/192 AES*. Accessed: Mar. 16, 2019. URL: https://opencores.org/projects/systemcaes.

[132] Vipin, K. and S. A. Fahmy (2012): Architecture-Aware Reconfiguration-Centric Floorplanning for Partial Reconfiguration. In: *Proceedings of the 8th International Conference on Reconfigurable Computing: Architectures, Tools and Applications*. ARC'12. Hong Kong, China: Springer-Verlag, pp. 13–25. DOI: 10.1007/978-3-642-28365-9_2. URL: https://doi.org/10.1007/978-3-642-28365-9_2.

[133] Vliegen, J., N. Mentens, D. Koch, D. Schellekens, and I. Verbauwhede (2015): Practical feasibility evaluation and improvement of a pay-per-use licensing scheme for hardware IP cores in Xilinx FPGAs. *Journal of Cryptographic Engineering* **5**(2), pp. 113–122. DOI: 10.1007/s13389-014-0088-4.

[134] Wilkinson, K. (2018): *Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream*. (v1.3). Xilinx.

[135] Wilson, K. S. (2013): Conflicts Among the Pillars of Information Assurance. *IT Professional* **15**(4), pp. 44–49. DOI: 10.1109/MITP.2012.24.

[136] Xilinx (n.d.): *SignOnce IP Licensing*. Accessed: Jan. 13, 2020. URL: https://www.xilinx.com/content/xilinx/en/referenced-content/alliance/signonce.html?null.

[137] Xilinx (n.d.): *Xilinx Announces the World's Largest FPGA Featuring 9 Million System Logic Cells*. Accessed: Oct. 02, 2020. URL: https://www.xilinx.com/news/press/2019/xilinx-announces-the-world-s-largest-fpga-featuring-9-million-system-logic-cells.html.

[138] Xilinx (2011): *AXI Reference Guide*.

[139] Xilinx (2012): *SECURITY MONITOR IP*. Xilinx. URL: https://www.xilinx.com/support/documentation/product-briefs/security-monitor-ip-core-product-brief.pdf.

[140] Xilinx (2017): *Vivado Design Suite User Guide Programming and Debugging*. Xilinx.

[141] Xilinx (2017): *Zynq UltraScale+ DeviceTechnical Reference Manual*. Xilinx.

[142] Xilinx (2017): *Zynq UltraScale+ MPSoC: Embedded Design Tutorial: A Hands-On Guide to Effective Embedded System Design*. Xilinx.

[143] Xilinx (2018): *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs*. Xilinx.

[144] Xilinx (2018): *Partial Reconfiguration Controller v1.3, LogiCORE IP Product Guide*. Vivado Design Suite PG193. Xilinx.

[145] Xilinx (2018): *UltraScale Architecture ConfigurationUser Guide*. Xilinx.

[146] Xilinx (2018): *Vivado Design Suite User Guide Hierarchical Design UG905*.

[147] Xilinx (2018): *Vivado Design Suite User Guide: Partial Reconfiguration*. Xilinx.

[148] Xilinx (2018): *Vivado Design Suite User Guide: Partial Reconfiguration*. Xilinx.

[149] Xilinx (2018): *Zynq UltraScale+ MPSoC Processing System v3.2*. Xilinx.

[150] Xilinx (2018): *Zynq-7000 SoC Technical Reference Manual*. Xilinx.

[151] Xilinx (2019): *Isolation Methods in Zynq UltraScale+ MPSoCs*. (v2.0).

[152] Xilinx (2019): *Zynq UltraScale+ MPSoC Software Developer Guide UG1137 (v11.0)*. Xilinx.

[153] Yan Feng and D. P. Mehta (2006): Heterogeneous floorplanning for FPGAs. In: *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*. DOI: 10.1109/VLSID.2006.96.

[154] Zhang, J., Y. Lin, Q. Wu, and W. Che (2012): Watermarking FPGA Bitfile for Intellectual Property Protection. *Radioengineering* **21**(2).

[155] Zhang, J., Y. Lin, Y. Lyu, and G. Qu (2015): A PUF-FSM Binding Scheme for FPGA IP Protection and Pay-per-Device Licensing. *IEEE Transactions on Information Forensics and Security* **10**, pp. 1–1. DOI: 10.1109/TIFS.2015.2400413.

[156] Zhang, J. and G. Qu (2019): Recent Attacks and Defenses on FPGA-Based Systems. *ACM Trans. Reconfigurable Technol. Syst.* **12**(3). DOI: 10.1145/3340557. URL: https://doi.org/10.1145/3340557.

[157] Zhang, L. and C. Chang (2014): A Pragmatic Per-Device Licensing Scheme for Hardware IP Cores on SRAM-Based FPGAs. *IEEE Transactions on Information Forensics and Security* **9**(11), pp. 1893–1905. DOI: 10.1109/TIFS.2014.2355043.

[158] Zhang, L. and C. Chang (2015): Public key protocol for usage-based licensing of FPGA IP cores. In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 25–28. DOI: 10.1109/ISCAS.2015.7168561.

[159] Zhang, N., K. Sun, W. Lou, and Y. T. Hou (2016): CaSE: Cache-Assisted Secure Execution on ARM Processors. In: *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 72–90.

[160] Zhang, X. and M. Tehranipoor (2011): Case study: Detecting hardware Trojans in third-party digital IP cores. In: *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 67–70. DOI: 10.1109/HST.2011.5954998.

[161] Zhao, M. and G. E. Suh (2018): FPGA-Based Remote Power Side-Channel Attacks. In: *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 229–244. doi: 10.1109/SP.2018.00049.

[162] Zhao, M. and G. E. Suh (2018): FPGA-based remote power side-channel attacks. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 229–244.

[163] Ziener, D. and J. Teich (2008): Power Signature Watermarking of IP Cores for FPGAs. *Journal of Signal Processing Systems* **51**(1), pp. 123–136. doi: 10.1007/s11265-007-0136-8. url: https://doi.org/10.1007/s11265-007-0136-8.

[164] Ziener, D. (2018): Improving Reliability, Security, and Efficiency of Reconfigurable Hardware Systems. *arXiv preprint arXiv:1809.11156*.

Supervised Student Research

[165] Dlala, K.-B. (2018): *Functional Verification of Protected FPGA Designs*. (unveröffentlicht). Bachelorarbeit. Karlsruher Institut für Technologie.

[166] Frank, R. (2017): *Analysis and Assessment of Side-Channel-Information of State-of-the-Art FPGA Evaluation Boards*. (unveröffentlicht). Masterarbeit. Karlsruher Institut für Technologie.

[167] Khazi, R. (2021): *Extending the ARM TrustZone to the Dynamic Partial Reconfiguration of the Programmable Logic*. (unveröffentlicht). Masterarbeit. Technische Universität Chemnitz.

[168] López, A.-G. (2019): *Development of a Secure Framework with Remote Configuration of IP via TLS for FPGA-S*. (unveröffentlicht). Masterarbeit. Public University of Navarre.

[169] Nitzsche, S. (2018): *Security and Feasibility of Remotely Configuring Intellectual Property*. (unveröffentlicht). Masterarbeit. Karlsruher Institut für Technologie.

[170] Pachideh, B. (2019): *Securing IP against Theft in SoC FPGA*. (unveröffentlicht). Masterarbeit. Karlsruher Institut für Technologie.

[171] Ramachandra, K. (2019): *Investigation on Readback Attacks for Intellectual Property theft*. (unveröffentlicht). Masterarbeit. Hochschule Bremerhaven.

[172] Subbarao, S. (2019): *Reconfiguration of SBox in an AES as a countermeasure against Side Channel Attacks*. (unveröffentlicht). Masterarbeit. SRH Hochschule Heidelberg.

[173] Xue, S. (2020): *IP Generation Independent of the Static Design*. (unveröffentlicht). Masterarbeit. Karlsruher Institut für Technologie.

Own publications

[174] Khan, N., S. Nitzsche, and J. Becker (2019): A Secure Framework with Remote Configuration of Intellectual Property. In: *2019 International Conference on Information Systems Security and Privacy*.

[175] Khan, N., S. Nitzsche, R. Frank, L. Bauer, J. Henkel, and J. Becker (2019): Amplifying Side Channel Leakage by Hardware Modification of Xilinx Zynq-7 FPGA Evaluation Boards. In: *SECURWARE 2019 : The Thirteenth International Conference on Emerging Security Information, Systems and Technologies*.

[176] Khan, N., A. Silitonga, B. Pachideh, S. Nitzsche, and J. Becker (2019): Secure Local Configuration of Intellectual Property Without a Trusted Third Party. In: Springer International Publishing, pp. 137–146. doi: 10.1007/978-3-030-17227-5_11.

[177] Khan, N., J. Castro-Godínez, S. Xue, J. Henkel, and J. Becker (2021): Automatic Floorplanning and Standalone Generation of Bitstream-Level IP Cores. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **29**(1), pp. 38–50. doi: 10.1109/TVLSI.2020.3023548.

[178] Khan, N., B. Hettwer, and J. Becker (2021): Moving Target and Implementation Diversity Based Countermeasures Against Side-Channel Attacks. In: *Applied Reconfigurable Computing. Architectures, Tools, and Applications*. S. Derrien, F. Hannig, P. C. Diniz, and D. Chillet eds. Cham: Springer International Publishing, pp. 188–202.

[179] Khan, N., S. Nitzsche, A. G. López, and J. Becker (2021): Utilizing and Extending Trusted Execution Environment in Heterogeneous SoCs for a Pay-Per-Device IP Licensing Scheme. *IEEE Transactions on Information Forensics and Security* **16**, pp. 2548–2563. doi: 10.1109/TIFS.2021.3058777.

[180] Silitonga, A., Z. Jiang, N. Khan, and J. Becker (2019): Reconfigurable Module of Multi-mode AES Cryptographic Algorithms for AP SoCs. In: *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, pp. 1–7. doi: 10.1109/NORCHIP.2019.8906923.