Efficient Adaptive Multi-level Privilege Partitioning with RTrustSoC

Raphaële Milan, Lilian Bossuet, *Senior Member, IEEE,* Loïc Lagadec, Carlos Andres Lara-Nino, Brice Colombier and Théotime Bollengier

***Abstract*—In recent years, heterogeneous SoCs—comprised of multiple processor cores and programmable logic—have greatly progressed both complexity and performance. From a security point of view, this leads to an expansion of the attack surface exposed to adversaries. To address this issue, in this article, we propose a novel heterogeneous SoC architecture called RTrustSoC. Our proposal includes an innovative fully-reconfigurable post-deployment strategy for partitioning the SoC architecture into multiple exclusion levels—worlds—with customizable degrees of privilege. We aim to provide SoC designers with fine control over the security of the system by segregating trusted hardware components from third-party IPs with "on-demand" hardware isolation. Therefore, we expect that an RTrustSoC instance could evolve from a multi-world SoC to a fully trusted platform as IPs progressively develop. RTrustSoC also proposes a dynamic reconfigurable penalty system to monitor the third-party IPs and take measures in case of a detected abnormal behavior. Our experimental testing on an AMD-Xilinx Zynq-7000 SoC-FPGA showed the penalty of the proposed isolation strategy to be small, up to 1% in LUT and 0.7% Flip Flop utilization, thus enabling to an efficient security solution. RTrustSoC introduces a novel design paradigm, evolving from the binary notion of security—trusted vs untrusted—into a flexible set of worlds that can be adapted to any scenario. We demonstrate a real case scenario of RTrustSoC use on time-based cache memory attacks with implementation results.**

***Index Terms*—Secure system-on-chip, Hardware IP core protection, Hardware security, Secure architecture**

I. Introduction

HETEROGENEOUS System-on-a-Chip (SoC) platforms are found in multiple application domains thanks to their high performance and flexibility. Their use cases range from general-purpose to critical military applications: high-frequency trading, cloud services, telecommunications, among others. To adapt to an even wider range of environments, the number and variety of components inside the SoC are constantly increasing. Today, typical heterogeneous SoCs embed one or several general-purpose processor cores, one or several application-specific processor cores, several hardware accelerators (which may be reconfigurable or not), large memories, power management units, communication interfaces, analog components, and so on. SoC-FPGAs are among the more complex heterogeneous SoCs. Contemporary examples include the Intel Agilex FPGA·SoC, and the AMD-Xilinx Zynq UltraScale+ MPSoC. Although this paper focuses on this generation of devices, our work can be extrapolated to other SoC platforms.

The increasingly complex design of SoCs poses a greater challenge for security auditors. The attack surface available for a malicious entity expands with the continuous inclusion of additional components in the device creating multiple and varied risks. As the market pushes for seemingly yearly technology releases [1], manufacturers are hard pressed to meet such deadlines. This leads designers to reuse hardware descriptions and software libraries with the aim of reducing development time. Quite often, these modules are created by third parties and then licensed to be used in the SoC. If even a single component of the SoC is not designed in-house, the chain-of-trust may be broken. This is usually not an issue as long as the third parties are reliable. However, by chance, their intentions are not legitimate, the consequences can be severe [2]–[4].

This paper addresses these security issues by presenting a novel lightweight solution for securing heterogeneous SoCs we named RTrustSoC. We introduce an innovative way of segregating the heterogeneous SoC resources into multiple levels, or worlds, with varying degrees of privilege. This novel approach was inspired by the ARM TrustZone technology [5]. In TrustZone, the components of the system can be declared as "trusted" or "untrusted." The trusted modules have full access to the platform while untrusted components have restricted access. A secure monitor is responsible for performing the context switches. TrustZone effectively performs spatial partitioning of the device into two regions. However, a binary system of classes is too coarse to classify all the components of modern SoCs. To overcome the problem, RTrustSoC provides a customizable number of worlds that can have an equally large set of customizable privileges.

This idea relies on configurable small reconfigurable security monitors that "wrap" each component and audit its interaction with the SoC bus. These security wrappers enforce the on-demand hardware isolation required by each world via security policies to prevent any unauthorized behaviors and restrict unauthorized accesses to other resources. They can even disable the ability of a component (a hardware IP or a software application) to make communication requests via a reconfigurable penalty system. Another limitation of ARM's TrustZone is that it offers support for generic SoC architectures, which reduces its usefulness in systems with custom

R. Milan, L. Bossuet, and B. Colombier are with the Université Jean Monnet Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023 in SAINT-ETIENNE, France. (email: {raphaele.milan, lilian.bossuet, b.colombier}@univ-st-etienne.fr). C.A. Lara-Nino is with the Universitat Rovira i Virgili, Departament d'Enginyeria, Informàtica i Mathemàtiques, TARRAGONE, Spain. (email: carlos.lara@fundacio.urv.cat). L. Lagadec and T. Bollengier are with the Lab-STICC, ENSTA Bretagne, in BREST, France. (email: {loic.lagadec,theotime.bollengier}@ensta-bretagne.fr).

hardware accelerators. AMD-Xilinx addressed this issue by proposing a TrustZone extension that can also isolate the IPs in the FPGA through the AXI bus. However, the approach has been shown to be vulnerable to elementary physical attacks [6]. To solve this issue, RTrustSoC also extends the notion of secure worlds to hardware accelerators: co-processors or FPGA. As the multi-world partition is enforced by individual security monitors, this explicitly improves the "poor man's security" approach of AMD-Xilinx.

RTrustSoC can be fully integrated into contemporary heterogeneous SoCs that feature ARM cores as well as in novel platforms that intend to use those architectures. In this work, we use the former approach: we implement an RTrustSoC prototype in heterogeneous SoC. We use an AMD-Xilinx Zynq-7000 SoC-FPGA (XC7Z010-1CLG400C) as testing platform to conduct experiments and test for performance. Our findings show that RTrustSoC has a minimal resource overhead, with up to 1% LUT resource utilization and up to 0.7% FF utilization for the fully trusted communication system.

The rest of the paper is organized as follows. Section II presents the security context, related work and a comparison with the state of the art. Section III describes the threat model. Section IV presents RTrustSoC: the lightweight heterogeneous SoC architecture secure-by-design. Section V provides the estimated costs of the approach with the relevant implementation results. Section VI presents the application of RTrustSoC to a time-based cache memory attack scenario. Section VII concludes the paper.

II. State of the Art

A. Security Overview

The literature contains multiple reports of vulnerabilities on SoC platforms. Most are discovered and reported by security researchers and have no significant impact for the end users. But in other cases, manufacturers have been requested to redesign their products [7]. When looking at these records, we identified common failure points. For example, the SoC communication bus is a particularly important source of weakness in the security of the system. In [6], [8], the authors showed how to perform privilege escalation attacks by targeting a single bit of the bus. Another popular point of attack is the device's power distribution network (PDN). Works like [9] and [10] have demonstrated the feasibility of leveraging the shared power or clock trees of commercial heterogeneous SoCs as intrinsic channels for bypassing isolation policies. The memory architecture of SoCs also tends to be vulnerable. It has been shown that shared DDR can be exploited to cause faults in the system [11]. Cache memories can leak side-channel information concerning the operation of the platform [12] and also enable the covert communication of data [13]. In heterogeneous SoCs, the FPGA itself represents a major liability. Many authors have demonstrated how to leverage the reconfigurable fabric to cause faults in the system [14], transfer data covertly [10], and implement remote monitoring schemes that provide high quality side-channel information [15], [16].

Another relevant part of the literature focuses on countermeasures against the aforementioned attacks. Some works like [17]–[19] propose different approaches to secure a SoC through its communication bus. However, these solutions focus on processor architectures that are not native to commercial heterogeneous SoCs. Indeed, most of these platforms feature an array of ARM processors. This is consistent with current trends, as ARM is the leader in the smartphone processor market [20]. The solutions available in the literature also fail to consider the heterogeneous SoC as a whole, i.e. they do not provide security protections for all the heterogeneous SoC components. Lastly, they require major modifications of the architecture to prevent most vulnerabilities. This means their ideas are worth exploring for applications in future SoC designs, but they fail to address the challenges of platforms that are already in use.

Given that SoCs may be used to handle sensitive data, they have become prime targets for malicious attackers. The main aims of attacks on SoCs range from stealing sensitive data to creating a denial-of-service. These attacks are mainly software-based and target the processing system of the SoCs. They are partially possible because some SoC resources are shared between applications. To give an example, in some recent multi-processor SoC architectures the last level of cache is shared by different cores. Depending on the cache memory access time, a malware can leverage this characteristic to determine whether or not the target application has accessed the data [21]. This provides the adversary with helpful information concerning the target application. This attack is also feasible on heterogeneous SoCs [13].

B. Protected SoC architectures

The best known strategy for protecting heterogeneous SoCs is ARM TrustZone [5]. The ARM TrustZone technology is available for heterogeneous SoCs with ARM processors such as the AMD-Xilinx SoC-FPGAs. This technology splits the resources of the processing system into two different worlds: secure and non-secure. Partitioning is then extended to the rest of the SoC: peripherals and memories, and in the case of AMD-Xilinx SoC-FPGAs also to the reconfigurable fabric [22]. Figure 1 is a didactic example of this technology applied to a heterogeneous SoC: the red blocks represent the non-secure world and the green blocks the secure one. This protection strategy is applied to the processing system (called PS in Fig. 1), to the memory resources, and also to the programmable logic (called PL in Fig. 1). The ARM TrustZone technology allows each CPU core to execute software applications in one of the two worlds, whereas with the extension proposed in the AMD-Xilinx SoC-FPGAs, each hardware IP embedded in the programmable logic is linked to one of the two worlds.

TrustZone enforces the policies in the SoC with an identifier called "NS bit" and some controllers. To do so, TrustZone uses the bus communication bus as shown in Fig. 1. Indeed, communications within a SoC pass through system buses like the Advanced Microcontroller Bus Architecture (AMBA). This technology is available for ARM cores, and both AMD-Xilinx and Intel SoC-FPGA use ARM cores. The Advanced eXtensible Interface bus (AXI) [23] is the main communication channel between the processing system and the programmable logic

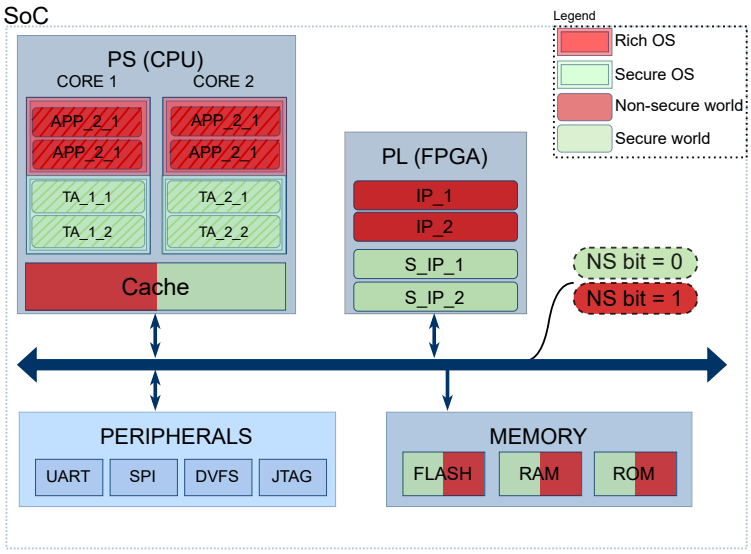

Fig. 1. An example of heterogeneous SoC architecture using ARM-TrustZone technology. The red blocks represent the non-secure world and the green blocks represent the secure world.

in the case of AMD-Xilinx SoC-FPGA. In these platforms, a proprietary IP (the AXI interconnect) acts as a translator between the ARM AXI and AMD-Xilinx' own specification: the AXI4 bus. In this type of communication bus system, the NS bit is sent on the AMBA buses and an AXI4 bus to allow the programmable logic to be aware of the world (secure/non-secure) in which the software application is running at any time. This prevents non-secure resources (in the processing system, programmable logic and memories) from accessing secure ones. The code and data within the secure world are assumed to be protected from intruders.

TrustZone was originally conceived as an efficient, holistic security approach, but despite its popularity, it has been shown to have many vulnerabilities that can be exploited to perform attacks and corrupt the security partitioning [6]–[8]. In [6], the authors target the communication bus of the SoC-FPGA, and they show that a hardware Trojan [24], [25] can modify the AXI communication signals and force an arbitrary value on the NS bit. This modification can jeopardize the rest of the system, leading to privilege escalation or denial-of-service attacks. In addition to this attack, the work reported in [8] uses power management of the heterogeneous SoC to perform covert transmission of data between secure and non-secure worlds despite TrustZone's isolation policies. In [7], the authors used a hardware Trojan to corrupt the secure boot and break the memory isolation. Modifying secure boot allows an attacker to change permissions to critical information, data or instructions, and can lead to privilege escalation. What is more, because TrustZone is proprietary, i.e. is not open source, it is difficult to improve its implementation.

In [18], Bahmani *et al.* propose an architecture called "CURE" containing three different types of software enclaves at different levels in the SoC: user-space, kernel-space and sub-space. The authors modify a RISC-V processor to support an enclave identifier. The rest of the system is also modified to support the enclave identifier. This identifier is then used in the rest of the system, and filtering blocks are installed to check the legitimacy of accesses to the enclave. A hypervisor (secure monitor) is used to configure the permissions. CURE also embeds protections against cache side-channel attacks. Policies are installed to supervise cache partitioning (L1), allocation and eviction. For each cache line in CURE, the enclave identifier is attached to it, allowing the hardware-coded arbiters to determine illegitimate accesses and prevent illegal eviction. The cache system is also flushed whenever there is a context change. CURE is mostly a software security proposal, as no hardware (logical) resources are embedded in the SoC. Thus, we consider their solution as unsuitable for heterogeneous SoCs since a secure architecture must take the whole SoC into consideration and provide both software and hardware protection.

In [17], Nasahl *et al.* present a secure architecture called "Hector-V". In their proposal, security is based on differentiating between the processors. The non-trusted applications operate in a rich execution environment in the application processor and the trusted applications operate in a trusted execution environment in the secure co-processor. To differentiate between illegitimate and legitimate communications, Hector-V uses identifiers (core ID, process ID and peripheral ID) and filtering blocks called "wrappers". The processors are modified to directly embed the identifiers. Hector-V uses AXI4 as a communication protocol, so the identifiers are propagated using the AXI4 user signals. The SoC communication buses are also distinguished into two communication channels: one for the data and one for the configuration. In Hector-V, the peripherals are bound to an entity and can only accept requests coming from it. The configuration channel is used to define the entity for each peripheral. A secure monitor is responsible for the configuration and for overseeing the operation of the communication between all peripherals and the processors. The authors argue that duplicating the resources in Hector-

TABLE I
COMPARISON OF PROTECTED SOC ARCHITECTURES. THE SYMBOLS INDICATE ○ NO SUPPORT, ◐ LIMITED SUPPORT, AND ● FULL SUPPORT.

Architecture	Type of processors	Trusted hardware IPs	Dynamic penalty system	Protections for the cache memory	Protections for the bus	Secure domains	Protections against DoS attacks
CURE [18]	RISC-V	○	○	●	●	3 types of enclaves	○
Hector-V [17]	RISC-V	○	○	◐	●	1	◐
ARM TrustZone [5]	ARM	◐	○	●	◐	1	○
Embedded policing [26]	ARM	◐	○	○	●	1 from ARM TZ	◐
E-IIPS [27]	DLX	○	○	○	●	0	●
ProMiSE [28]	RISC-V	○	●	●	◐	0	●
TrustSoC [29]	ARM	●	○	●	●	N worlds	○
RTrustSoC (this work)	ARM	●	●	●	●	N worlds	●

V can mitigate micro-architectural attacks and in part cache side-channel attacks. However, in a real use case where the constraints on resource use are high, this solution would not be entirely viable. What is more, their proposal does not allow programmable hardware resources to be embedded which is an essential part of SoC-FPGAs. Furthermore, the proposal in Hector-V is only a dual world segregation, similar to ARM TrustZone.

Hagan *et al.* [26] propose a hardware-based pro-active policing and policy architecture. They use hardware modules called "security policy engines" at the system communication level. These modules act as hardware-coded firewalls with a list of permissions that actively monitor the AXI4 SoC communication bus. For every incoming request, the security policy engine uses the read and write address channels to determine the legitimacy of the transaction. They can either grant or deny access to the peripheral depending on the policies stored in a table. The policies are configured by SELinux and can be updated over time. The system also allows the integration of programmable logic (FPGA). In case an attack is detected, blocks called "security response engines" can initiate responses. The responses can go from erasure of secret keys to system reset. Furthermore, it uses ARM TrustZone to provide the designer with the possibility of having a secure domain in their architecture. Their architecture fails to provide several secure software environments to isolate the user trusted applications from each other. The active responses are neither very dynamic nor particularly flexible. Their proposal does not provide solutions against cache-based side-channel attacks.

In [27], Basak *et al.* propose a flexible framework to implement security policies in a SoC based on two types of communication controllers. The first one is centralized and is a plug-and-play module implementing the SoC policies called "extended infrastructure IP for security". It is microcontroller-based and can be updated via firmware code. The second controllers comprise small "security wrappers" connected to each IP block. They identify security events relevant to the security policies they enforce and communicate any violations of security policies to the centralized controller. Upon detection of a violation, the system can take action for example, by disabling the IP interface. However, like in [17], the authors duplicate the resources with the two types of communication controllers, meaning this solution would not be entirely feasible on a system where the constraints on resource utilization are high. Additionally, the architecture does not provide multiple secure domains nor dynamic responses to attacks. Their proposal fails to provide solutions against cache-based side-channel attacks.

In [28], Singh *et al.* presented a hardware framework to quantify the security health value of a device inside a zero-trust network with a trust score. This is done by hardware-coded blocks that monitor the CPU cores at any time during execution and log all the operation information. A dedicated secure co-processor then uses the gathered data and executes a threat estimation model that computes a trust score. This score evolves over time and the closer it gets to zero, the more untrustworthy the program it is linked to is considered. The system can take immediate action according to the value of the trust score and provides dynamic access policies. However, the proposal is mainly based on software and the design lacks of programmable resources. This architecture is designed for network applications which is not our target, plus it uses neural networks for the computation. This type of calculation is not viable for embedded systems where the constraints on resource use are high. Therefore, in our opinion, their solution is not suitable for heterogeneous SoCs.

In addition to these works, we can find other solutions, hardware or software, to secure the cache memory against time-based side-channel attacks. However, we are not going to cite them in our comparison since these works only focus on the cache memory, whereas we target the full architecture of the SoC. In [30], the authors present a multi-compartment solution applied to an embedded system and more precisely to the cache memory. Similarly to CURE, each cache line is assigned an identifier and with the help of hardware-coded firewalls a logic check of access rights is performed preventing any illegitimate access or modification. Only cache lines belonging to a certain identifier are flushed when needed and not the entire cache memory. This allows a performance gain. In [31], the authors concentrate on the software with the proposition of automated compiling techniques that eliminate key-dependent conditional instructions. Another possibility is to use constant-time techniques such as [32], so the adversary is not able to observe cache access behavior during the

execution. The attacker is not able to recover any sensitive information. The downside of the software-based solutions to mitigate side-channel attacks on the cache memory is the loss of performance of the system. Nonetheless, they are easier to implement than hardware-based propositions.

Table I provides a qualitative comparison of the state of the art solutions that are the most relevant to this work. Note that the TrustSoC [29] listed in Table 1 is a first initial version of RTrustSoC with less protections and flexibility features than the most recent version. We consider solutions that use ARM cores to be more viable because they are native to most heterogeneous SoCs available on the market today and therefore easier to integrate unlike RISC-V processors. Most of the solutions embed bus protections and cache memory protections, but not many consider the programmable logic, and even fewer propose a concept of trusted hardware IPs. Some works include protection against DoS attacks, but only one instance [28] proposes a penalty system where the penalization parameters are reconfigurable at runtime. None of the solutions in the literature propose a multi-world-based system. CURE [18] proposes different types of enclaves but limited to only three, and the remainder propose either no secure domain at all, or just one, whereas we propose up to N with different levels of privileges (the number is only limited by the resources of the system).

This paper aims to extend the distributed hardware monitoring to cover the heterogeneous SoCs as a whole, while accounting for both factors: software (processing system, operating system, boot, etc.) and hardware (programmable logic, bus, hardware IP, etc.). It also offers the designer more flexibility via a multi-world on-demand segregating method with more than one secure world. RTrustSoC extends the notion of trusted applications to the programmable logic with trusted hardware IPs. We also propose a dynamic penalty reconfigurable system that can respond in case of a detected attack performed by a third party. We target threats introduced during the SoC design which are detailed in the following threat model.

III. Threat model

In this paper, we consider several threats from remote software along with hardware attacks. We called our proposal RTrustSoC. We consider threats that are relevant to the SoC design process [33], in particular with the reuse of hardware IP blocks or software applications. Indeed, as time-to-market tends to progressively shorten, designers do not have the time to develop every software or hardware component, and consequently use third-party blocks. These components may contain malicious routines or circuits that can be used to perform an attack on the system. These malicious entities can affect the system in various ways. They can be passive, i.e. collect secret information that is meant for another hardware IP or software application. They can also interfere more deeply with the system by changing the contents of communications or by taking control of a communication that is not intended for them. This can cause severe damage to the system. For these reasons, RTrustSoC considers malicious hardware IPs or software applications that are introduced during the design stage. These malicious entities can perform the attacks cited above such as: illegitimate accesses, modifications of the memory contents, modifications of the communication contents [6] or take control of the power supply to perform covert channels [8]. RTrustSoC considers time-based cache memory side-channel attacks. The attack can be performed by malicious third-party hardware IPs targeting a trusted software application running in a secure world [13]. The malicious IP can modify the contents of the cache memory contents and access secure regions.

Unlike TrustSoC [29], RTrustSoC also covers denial-of-service attacks. The goal of these attacks is to prevent legitimate users from using the system resources. In the case of RTrustSoC, we consider a malicious entity flooding the SoC communication bus with illegitimate requests thereby preventing legitimate users from using a resource.

The threats we consider are relevant and correspond to the process of heterogeneous SoC design. RTrustSoC mitigates these threats by introducing minimal additional components for each hardware IPs enforcing policy to guarantee that no abnormal behavior can take place in the architecture such as illegal accesses, modifications of the memory contents, changes in privileges, etc.

We assume that the CAD toolchain is trusted and cannot be used to perform illegal modifications of the design. The synthesis tool is responsible for the components added to each hardware IPs. The additional components are trusted and cannot be modified by the synthesis tool. We also assume that the SoC and the founder are trusted. No physical modification can be made to the circuit. Attacks that require physical access to the architecture are beyond the scope of this paper.

IV. RTrustSoC

This section describes the secure-by-design heterogeneous SoC architecture we named RTrustSoC. It is based on a multi-world segregating method: from one non-secure world up to N secure worlds with the possibility of having different privilege levels (N chosen by the designer and only limited by the resources in the system). The idea behind RTrustSoC is to extend the notion of trusted applications and their trusted environments available in ARM TrustZone in the CPU to the whole SoC and to be able to multiply these environments to suit the designer's needs. We want to provide designers with isolated, safe and trusted environments in the programmable logic so they have hardware IPs they can trust. Trusted IPs have the same guarantees as trusted applications, i.e. confidentiality of their design and data, privacy, integrity and control over access. Their design and data are isolated from the non-secure world and the segregation, and access controls are added. The security of the SoC modules is enforced by on-demand hardware isolation and control.

RTrustSoC is a flexible and scalable architecture that can be adjusted to the designer's requirements. Figure 2 presents the RTrustSoC concept. The processing system includes k cores but could be extended to support different architectures. RTrustSoC also embeds a programmable logic region with m

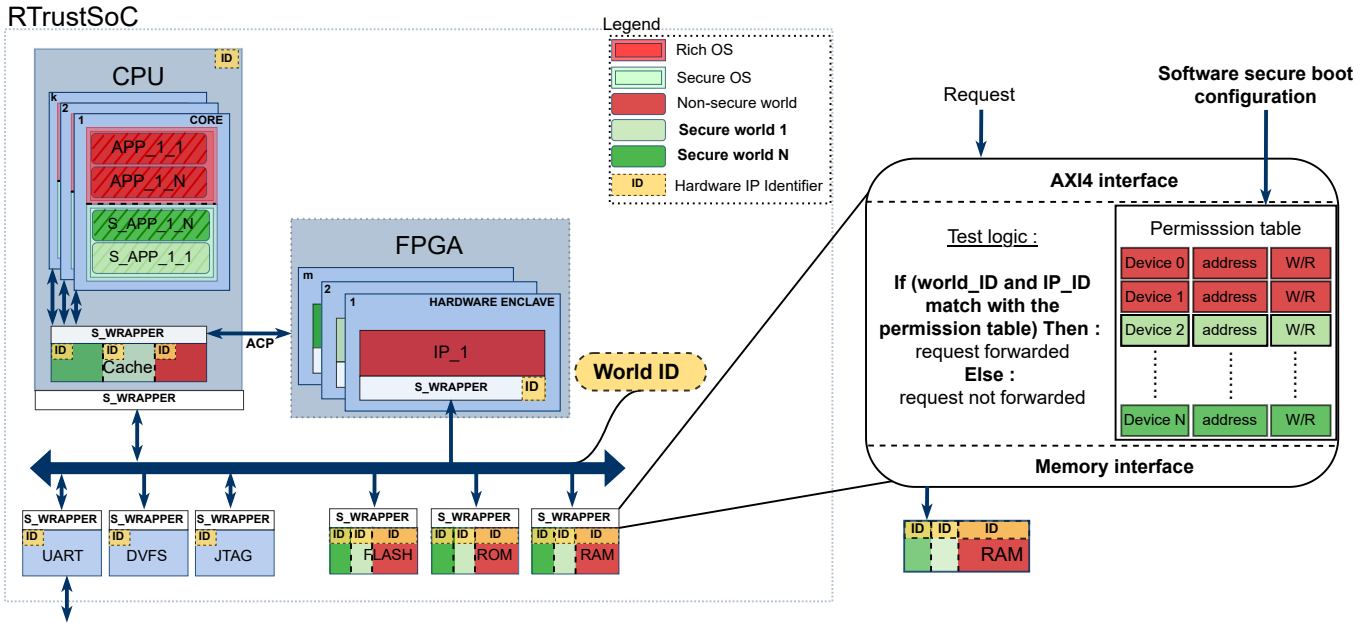

Fig. 2. Example of the proposed RTrustSoC architecture

hardware enclaves, a communication bus, several peripherals and shared memories. Each core has a non-secure world, shown in red, and *N* secure worlds, shown in green. Finally, RTrustSoC embeds tiny, distributed communication controllers called *"s_wrapper"*, for security wrappers, to enforce RTrustSoC's security policies. These policies allow the security wrappers to distinguish between legitimate and illegal access with the help of hardware-coded identifiers. With the extension of the security wrappers, RTrustSoC can also take immediate action when an attack is detected. It does so through a system of dynamic penalties that prevent DoS attacks on the system. The security wrappers enforce the isolation, the integrity and the safe environments required by the trusted hardware IPs and trusted applications. However, to propose a secure-by-design architecture, we must provide a set of essential security features. We present our security features in the following subsection.

A. *RTrustSoC security features*

SF.1: Operating rules: RTrustSoC comes with a set of operating rules that must be enforced as policies to prevent any unwanted behavior and create trust between the components during operation.

SF.2: Extended secure multi-worlds: RTrustSoC provides segregated on-demand multiple secure domains with the possibility of having different privilege levels to allow designers more flexibility for their design. In contrast to ARM TrustZone technology, RTrustSoC allows the designers to choose the number of secure worlds they want in their design. This extends the notion of trusted execution environments to the rest of the SoC, not only to the software applications but also to the hardware IPs.

SF.3: Programmable logic in the security resources: RTrustSoC fully integrates the programmable logic in the security resources by using a unique identifier for each hardware IP, again offering a secure environment during execution.

SF.4: Trusted communications inside the SoC: RTrustSoC establishes secure communications between hardware IPs and software applications inside the heterogeneous SoC. Thanks to this security functionality, RTrustSoC does not have to rely on a third-party's security features and guarantees that the IPs introduced are operating as intended. RTrustSoC also guarantees that no malicious entity can take control of the SoC communications: change communications or monopolize the SoC communication bus.

SF.5: Side-channel cache time-based attack resistance: RTrustSoC embeds protections against remote time-based side-channel attacks targeting the cache memory by restricting access to the cache either from the processing system or the programmable logic. This is done with the use of identifiers and by creating different isolated cache partitions for each world. The isolation is also enforced with the communication controls that prevent any illegal accesses. The operating rules of RTrustSoC also stipulate flushing the sensitive cache partitions at each context switch or at the end of the use of the cache memory by a given sensitive application. Even if flushing cache memory partitions leads to performance loss, it will enhance the security.

B. *RTrustSoC security wrappers*

The RTrustSoC prototype proposed in this paper embeds ARM Cortex processors. This choice is motivated by the fact that ARM has a strong presence in the SoC and heterogeneous SoC markets. ARM processors can be found in the main heterogeneous AMD-Xilinx and Intel SoCs.

RTrustSoC uses small, distributed hardware-coded security wrappers to create trusted communications between the hardware accelerators, peripherals and applications. The security

wrappers aim to distinguish between illegal and legitimate transactions. To establish this secure communication, RTrustSoC assigns an IP identifier and a world identifier to each hardware resource in the SoC. These identifiers are different, unique and hardware-coded. They are assigned prior to synthesis and cannot subsequently be changed. Each security wrapper comes with a set of permissions that specifies the access rights of each hardware resource to the underlying component. The security wrapper is then able to perform the control access to the underlying resource and guarantee the on-demand hardware isolation.

The module identifiers are transported through the communication bus, which is an AXI4 in the prototype presented in this work. AXI4 is a slave/master protocol [23]. It has five separate channels: write address (AWADDR), write data (WDATA), write response (BRESP), read address (ARADDR), read data (RDATA) and the optional read response (RRESP). The AXI protocol operates on handshake mechanisms with *Xready* and *Xvalid* signals for each channel (*X* represents the channel). The response channels (BRESP and RRESP) indicate the state of the transaction to the master: OKAY if the transaction is successful, SLVERR or DECERR when an error occurs. A transaction can only be initiated by a master interface and only occurs if the channel handshake signals are high at the same time.

In addition, the AXI4 protocol makes it possible to use user signals to transport extra information up to 1024 bits with no overhead. We leverage this feature in RTrustSoC: each request submitted to the SoC communication bus has its IP identifier and its world identifier added through the AXI4 user signals. The width of the identifiers depends on the number of components and worlds in the SoC. For the hardware identifiers, encoding is given by $\lceil \log_2(\max(\text{components})) \rceil$ bits, excluding the zero. Similarly, for the identifier of the worlds, we use $\lceil \log_2(\max(\text{worlds})) \rceil$ bits. The world identifier essentially extends the NS bit of the ARM TrustZone. Since it is hardware-coded and we assume that the CAD toolchain is trusted the world ID cannot be changed, hence preventing attacks described in [6].

Figure 3 shows how the security wrappers operate. When a security wrapper receives a request, it compares the IP and world identifiers with its list of access policies (read/write). RTrustSoC allows the access rights to be changed at boot time via a software secure configuration. This secure configuration gives the designer more flexibility. After the secure boot configuration, the policies are set and can subsequently not be changed. The reconfiguration of security policies during runtime is excluded. After comparison, if a request conveys the correct hardware and world identifiers, plus if it follows the security policies, it is forwarded to the underlying component. If an anomaly is detected, the wrapper discards the data, sends a null response, and signals an error via the AXI bus using the response XRESP signals. The distributed security wrappers also embed simple security policies to oversee the operation of the IP. For example, there is a reset after every use of the component to prevent reuse of data. As these hardware-coded identifiers and access rights cannot be modified by an attacker to perform an attack (illegal accesses, change in communications, etc.), they therefore confirm the **SF.4** security feature. Likewise, the secure boot configuration makes it possible to change the access rights thereby giving the designer more flexibility.

C. RTrustSoC penalty system for trusted communications

Figure 3 also shows how a security wrapper behaves when it is attached to a malicious entity performing unauthorized communications. This is a new security feature of RTrustSoC that is not available in TrustSoC [29]. The security wrapper has a counter that is incremented each time the IP makes an illegal request over a specified period of time. To determine if a request is illegal, the security wrapper analyzes the *XRESP* signals of the AXI bus, if an error is detected the illegal request counter is incremented. When the value of the illegal request counter exceeds a given threshold called *Max*, then the ability of the IP to communicate is disabled for a specified period of time, called *Tblock*. *Max* and *Tblock* are initialized during the secure boot configuration and then become dynamic during execution. They follow a penalty system. They start at the values fixed during configuration and then the higher the frequency of attempts by the IP underneath the security wrapper to perform illegal requests, the more the *Tblock* time value is incremented and the threshold *Max* is decremented. So, the more malicious behavior is pursued by the IP, the more it will be penalized, whereas, if the IP behaves the way it is supposed to, no penalties will be applied. This system provides more flexibility while nevertheless enforcing **SF.1** and **SF.4** security features.

D. RTrustSoC multi-world partitioning

Figure 4 illustrates the operation of RTrustSoC applied to any design. The non-secure world is shown in red. There can be up to N secure worlds, which are identified in green. Figure 4 illustrates the state of the system when it is operating in the non-secure world (Fig. 4 (a)) and in one of the secure worlds (Fig. 4 (b)). Figure 4 (a) shows which resources of the system are accessible to the non-secure world when it is operating. None of the secure world resources, encoded world ID = “secure world 1” (#sec_world_1) to world ID = “secure world N” (#sec_world_N), are accessible to the non-secure world. No hardware IP is directly connected to the system communication bus because a security wrapper is placed between the component and the bus. The security wrapper prevents any unauthorized communication between hardware and software components in the system. It also prevents the occurrence of covert channels since each action must be authorized by the communication controllers. Additionally, the components cannot access or modify a memory partition without authorization. This applies to all operations in the different worlds. The authorizations are enforced by the distributed security wrappers and their policies.

The non-secure world components cannot access resources in the secure worlds, but the restriction does not apply to the secure worlds. For example, an application running in a secure world could delegate some computations to a non-secure hardware accelerator. In this case, when processing is

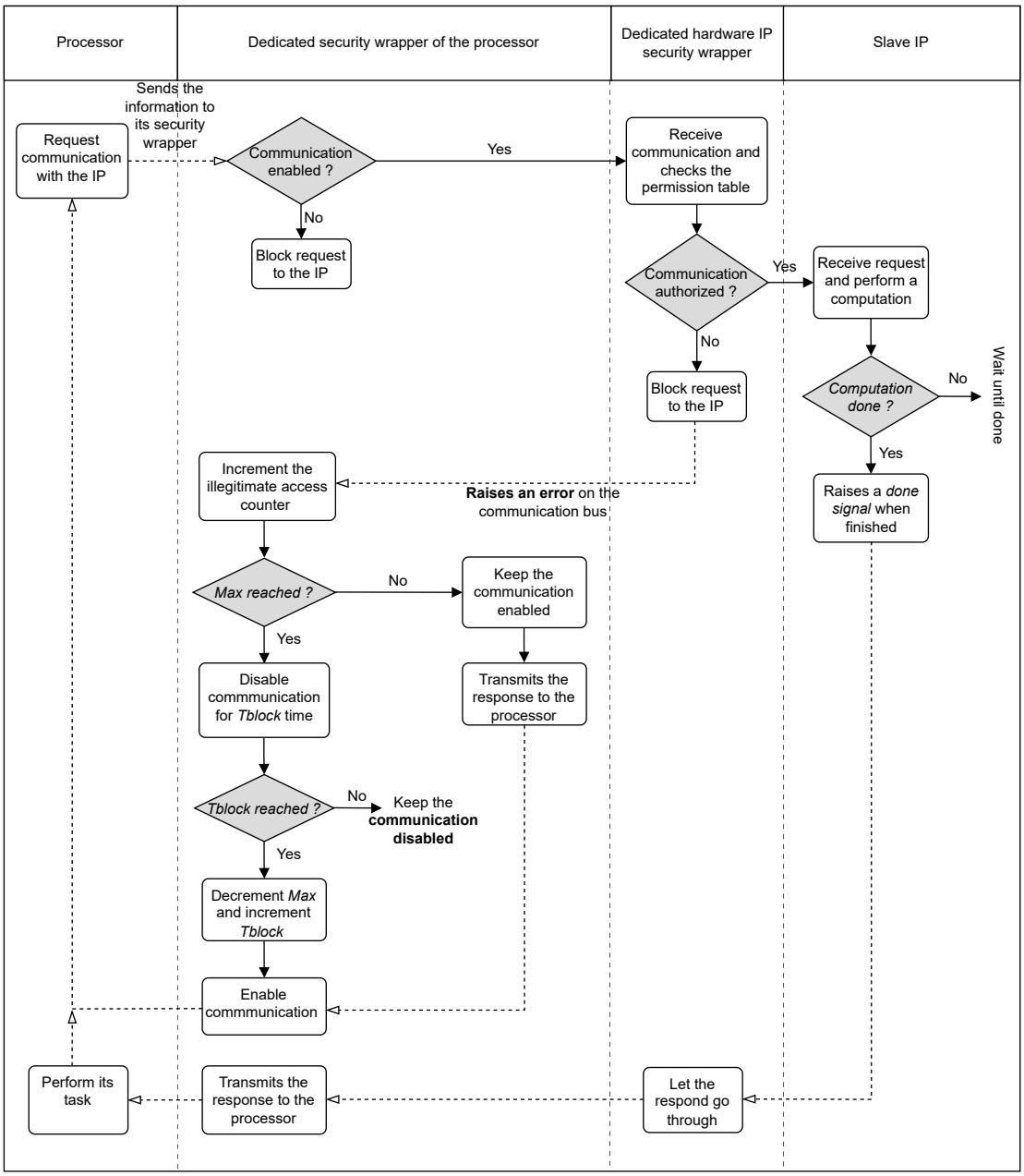

Fig. 3. Flow chart of how the RTrustSoC security wrappers operate.

complete, the IP is automatically reset by its security wrapper to prevent the misuse of sensitive data. This rule also applies to cache partitions which are flushed when switching from one world to another. This reduces overall performance but provides a better level of security and contributes to the **SF.5** security feature. Our trusted communication system and operating rules address the vulnerabilities of the basic ARM TrustZone technology. The on-demand isolation we provide between the worlds makes it impossible for a malicious entity to obtain information on a victim that resides in one of the secure worlds. Neither could a potential attacker modify the identifiers needed to illegally access a world where it does not belong.

E. RTrustSoC memory protection and cache memory protection

One interesting feature of RTrustSoC is its ability to protect memory resources from illegitimate access and isolate the different world memory partitions from each other. This is enabled by security wrappers that are placed between the AXI4 communication bus and the memories. Each request originated from the bus is verified by the security wrapper to be sure it complies with the security policies. The security wrappers compare the identifiers of the transaction sender and the access rights table that belongs to the memory. If the rights are confirmed, the security wrappers accept and forward the transaction, otherwise they discard the data and raise an error on the bus through the SLVERR signal. With the security

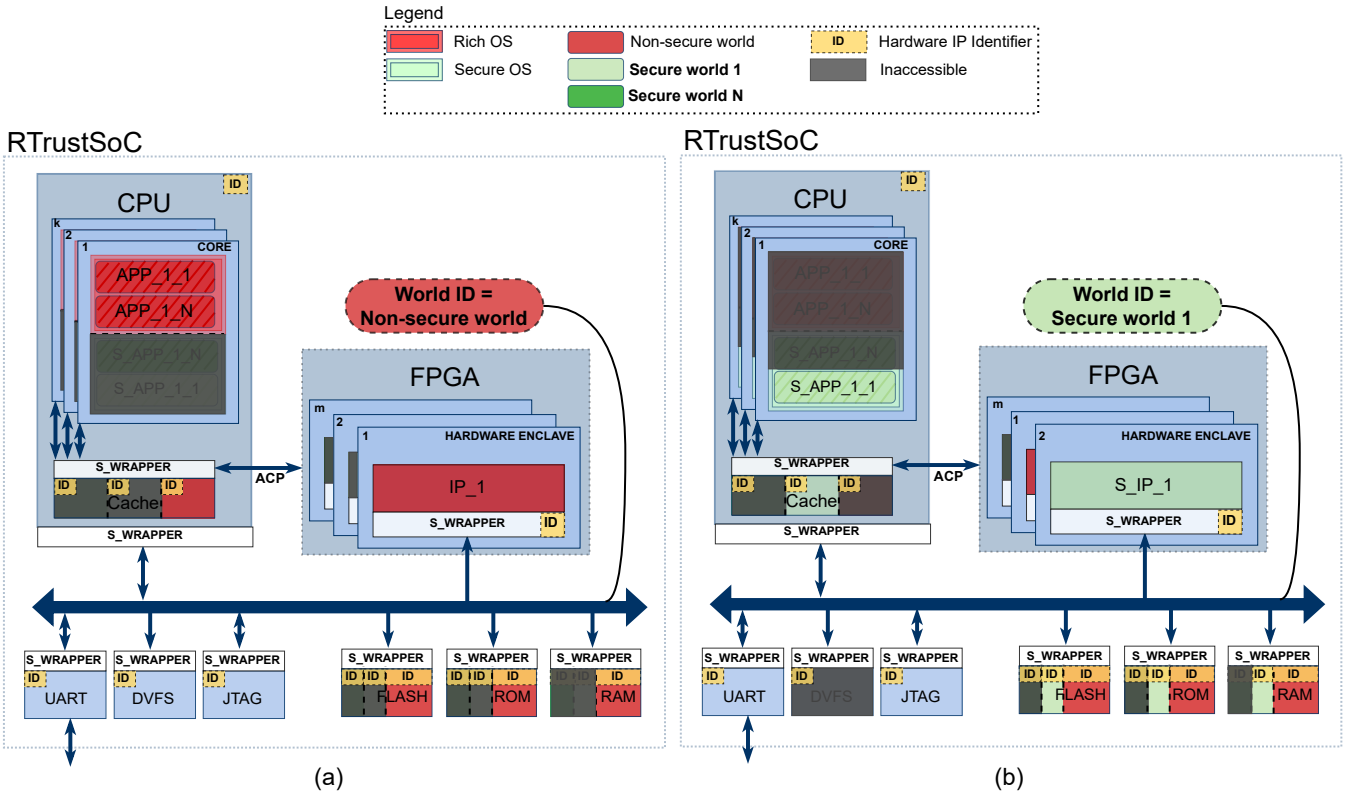

Fig. 4. Modes of operation of RTrustSoC architectures in (a) the non-secure world, (b) in one of the secure worlds (#sec_world_1)

wrapper controls, it is impossible for a malicious entity to illegally access the memory (cache and external memories). RTrustSoC's rules stipulate that the different world cache memory partitions are isolated from each other. However, RTrustSoC allows a secure world to have access to non-secure external memory (FLASH, ROM, RAM) partitions for performance purposes. For example, if the secure world needs to use of an accelerator to speed up a calculation. We discuss the cache memory issue with a real use case scenario in another Section VI.

V. Prototyping and Testing

In this section, we provide the results of the implementation on an AMD-Xilinx Zynq-7000 SoC-FPGA (XC7Z010-1CLG400C). We used the Xilinx Vivado 2020.2 toolchain to implement the RTrustSoC prototype. The distributed security wrappers presented in subsection IV-B were described in VHDL. We used these distributed wrappers to protect five different IPs from cryptographic and signal processing applications: Sobel filter, ASCON, Karatsuba-128, AES-128 and Montgomery-128. We also prototyped the security wrappers that enforce the penalty system presented in subsection IV-C. The results of the implementation are shown in Fig. 5. Both slave and master implementations were done for a small system (small number of worlds and components) to give the logic costs. Then we implemented a security wrapper protecting a BRAM memory to explore the costs of our solution for bigger systems, the results are shown in Table II.

A. Evaluation of the RTrustSoC security wrappers

The size of the hardware IPs ranges from 2,747 to 4,875 LUTs. All the hardware IPs we used are open source and found in online repositories. We evaluated our hardware implementations with and without the security wrapper, using the hardware utilization in LUTs, FFs, and the maximum frequency achievable by the design as metrics. As shown in Fig. 5, the resources overhead caused by the security wrapper in number of LUTs is very small, at most 2.54% and in number of registers, at most 0.54% compared to the baseline implementation costs of the IPs. This resource overhead can be explained by the logic we add to each IP in order to implement our distributed security wrappers. In our experiment, we were limited by the size of the fabric in the AMD-Xilinx Zynq-7000. For example, the largest multiplier instances we were able to fit in this board used 128-bit operands. However, for cryptography applications one would expect to use up to 512-bits operands. Such larger instances would obviously dwarf the hardware costs of the security wrapper in comparison. The overhead in terms of maximum frequencies of operation as shown in Fig. 5 is not significant. Indeed, the security wrapper does not affect the critical path of the hardware accelerators and hence does not affect the maximum achievable frequencies. We suspect that

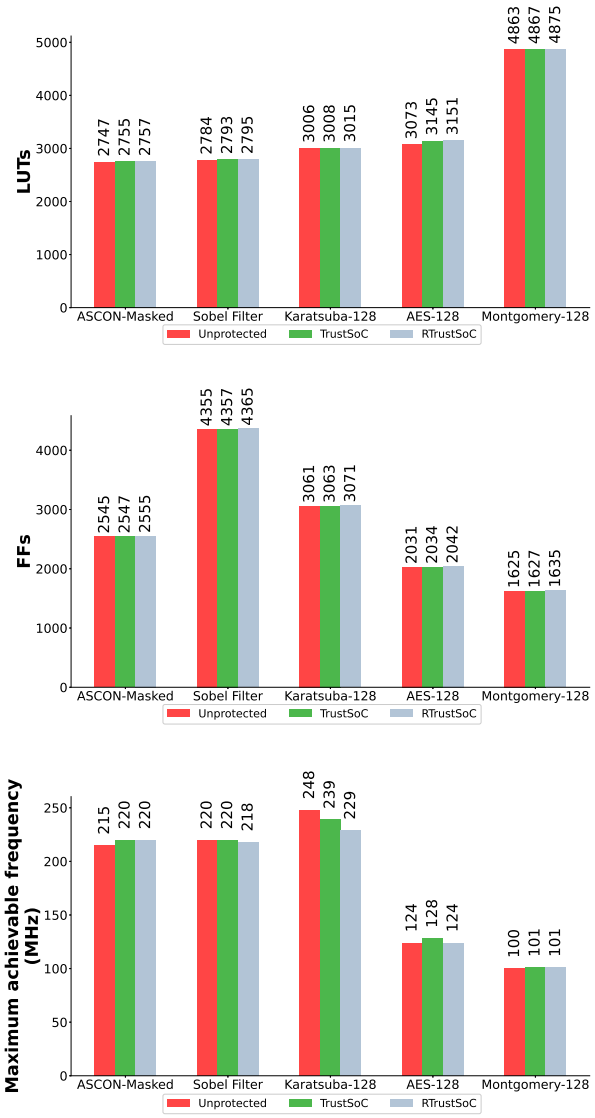

Fig. 5. Results of implementation of a RTrustSoC security wrapper for five different hardware IPs from an AMD-Xilinx Zynq-7000 SoC-FPGA.

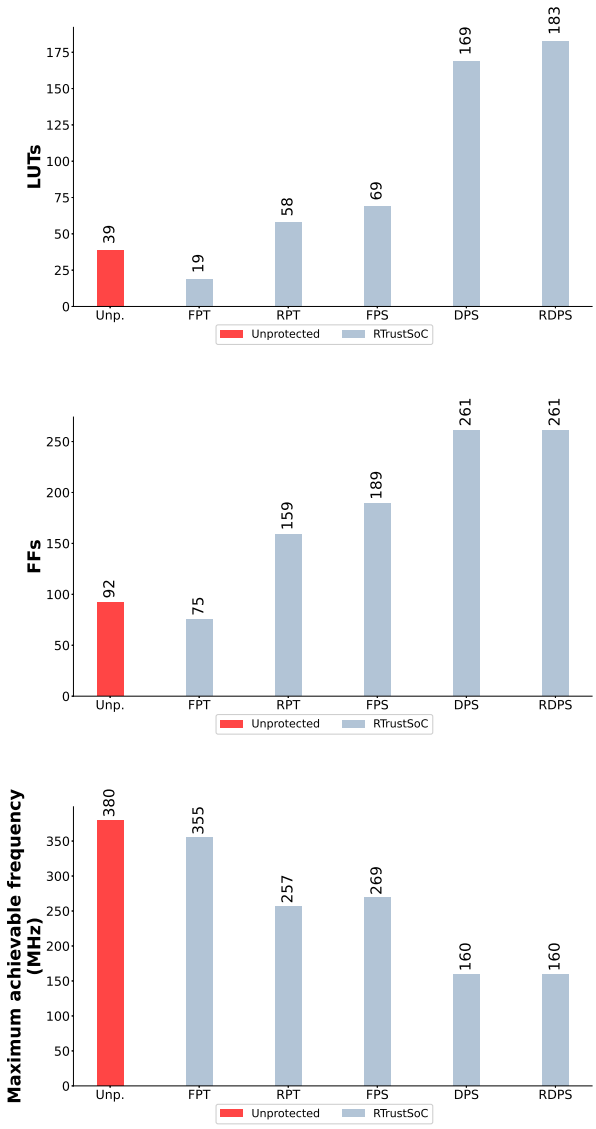

Fig. 6. Results of the implementation of a RTrustSoC security wrapper connected to a master interface (Unp. for Unprotected, FPT for fixed permission table, RPT for reconfigured permission table, FPS for fixed penalty system with fixed Max and Tblock, DPS for dynamic penalty system and fixed Max and Tblock, RDPS for dynamic penalty system and reconfigured Max and Tblock values) from an AMD-Xilinx Zynq-7000 SoC-FPGA.

the fluctuations that appear in Fig. 5 are due to the non-deterministic nature of the synthesis process. In conclusion, the resources and performance overhead incurred by security wrapper presented in RTrustSoC is negligible compared with a significant improvement in security. Indeed, RTrustSoC and its security wrappers mitigate the threats induced by the use of third-party hardware IPs or software applications while simultaneously offering the designer more flexibility.

B. Evaluation of the RTrustSoC's penalty system

We have implemented a security wrapper that enforces the penalty system presented in subsection IV-C. We evaluated the security wrapper using the hardware utilization in LUTs, FFs, and the maximum frequency achievable by the design as metrics. We developed several implementations with and without the penalty system, with and without the secure boot reconfiguration. The results are presented in Fig. 6 and shown in the following order: (Unp.) unprotected, (FPT) fixed permission tables, (RPT) reconfigurable permission tables, (FPS) fixed penalty system, (DPS) dynamic penalty system and (RDPS) reconfigurable dynamic penalty system. The resources overheads incurred by the security wrapper in number of LUTs and FFs seem very high. The security wrapper can take a lot more resources with the dynamic penalty system, but the baseline for comparison is very small so the implementation costs appear to be really high although in reality they are not. If we were to compare with the total resources available in the AMD-Xilinx Zynq-7000 SoC-FPGA, our mean LUT implementation result is 100 LUTs which represents only 0.57 % of the total LUTs, our mean FF result is 189 which represents only 0.54 % of the total FFs. So, our penalty system has a very small overhead compared to the confidence it

TABLE II
RESULTS OF IMPLEMENTATION OF A SECURITY WRAPPER ATTACHED TO A BRAM IN LUTS FROM AN AMD-XILINX ZYNQ-7000 SOC-FPGA.

Components \ Worlds	2	4	8	16
2	7	11	20	29
4	9	15	28	50
8	15	17	48	83
16	23	29	77	152
32	53	89	157	291
64	92	160	296	1112

adds to the architecture. For the first evaluation, the security wrapper with fixed permission tables, we have results that are inferiors to the unprotected version, this is due to the use of constants in the implementations which then induced optimizations from the CAD toolchain. The overhead in terms of maximum achievable frequency of operation is shown in Fig. 6, the dynamic penalties impact the system because of the use of a counter to block the communications for a period defined by the parameter *Tblock*. To improve our results, we could use another type of counter like the Johnson counter.

C. Evaluation of the RTrustSoC's memory protection

We implemented a security wrapper attached to a BRAM in order to demonstrate how our system works and the costs of world partitioning on a memory block. We implemented the BRAM security wrapper and tested it with a varying number of worlds (2, 4, 8, 16) as well as a varying number of IPs that require access to the memory (2, 4, 8, 16, 32, 64). Table II shows the results of our implementations. This prototype allowed us to explore the costs and scalability of our proposal. The overhead in resources is due to the size of the access rights table that evolves quadratically in large systems (16 worlds and 64 components). In addition, in larger systems, the AXI logic increases with the extension of the USER signal width. This explains the higher overhead for the larger number of worlds with the most hardware identifiers. Currently our implementation uses LUTRAMS, but it is also possible to use BRAMS, which would enable a dramatic reduction in the overhead in terms of the number of LUTs. The results of the timing criteria are not shown since the variation is negligible. From the results in Table II, we can conclude that the costs of the security wrapper and world partitioning lead to a small resource overhead on the protected system, which is more than acceptable given the high degree of protection provided by the proposed solution. The security wrappers prevent any malicious entity from illegally accessing memory.

VI. USE CASE SCENARIO

In this section, we present one use case scenario for RTrustSoC. More concretely, we demonstrate how RTrustSoC can be used to secure the cache memory from malicious hardware IPs embedded inside the programmable logic. In this example, we use the threat model presented in [13]. It targets a heterogeneous SoC architecture where the TrustZone technology is enabled. It consists of a software application inside the CPU that runs a vulnerable to time-based cache attack AES-128 T-table implementation. The software application is linked to the secure world. Then, a malicious hardware IP belonging to the non-secure world is embedded inside the FPGA. The malicious logic tries via the cache coherency port to access the cache memory and perform time-based cache attacks to recover the encryption key. The authors in [13] use *Flush+Reload* and *Evict+Time* attacks. This attack is made possible in most systems due to the fact that slave processing system interfaces are often not configured to deny access to secure regions in TrustZone-enabled SoC architectures. The authors leverage this particular feature to run the attack from the PL to the cache memory.

TABLE III
IMPLEMENTATION COSTS FOR THE SCENARIO PROPOSED IN FIG. 7 FROM AN AMD-XILINX ZYNQ-7000 SOC-FPGA.

	LUT	FF	Fmax (MHz)
Cache security wrapper	63	116	212
Utilization (%)	0.36	0.33	–

In Fig. 7, we show this scenario applied to a RTrustSoC-enabled architecture. The software implementation of the vulnerable AES-128 T-table is inside the CPU and belongs to one of the secure worlds identified in dark green. The malicious hardware IP is embedded inside the FPGA inside the non-secure world identified in red. The malicious hardware IP is then connected to the snoop control unit (SCU) via the ACP port. The SCU connects the processor cores, and the ACP PL interfaces to the cache memory and offers support to the cache coherency. Since it is not possible to modify inside the ARM CPU to add our hardware-coded monitor for the cache memory, we placed it inside the PL to the ACP connection for our demonstration. Both *Flush+Reload* and *Evict+Time* attacks are based on the ability of the master interface to be able to evict a cache line. However, with RTrustSoC and its enhanced control, this is not possible. The permission tables are set to make it impossible for a non-secure resource to access secure cache memory partitions. The permission tables also cannot be changed since they are only managed by the security wrapper which is trustworthy. We implemented the cache memory security wrapper with its permission table according to the scenario (Fig. 7: 2 worlds and 2 components) on an AMD-Xilinx Zynq-7000 SoC-FPGA. There is only one secure world in this scenario, but it could be easily extended to match a multi-world SoC architecture. The overheads of our implementation are shown in Table III. The security wrapper utilizes 0.36 % of LUT and 0.33 % of Flip Flop resources of the SoC-FPGA.

VII. CONCLUSION

In this work, we propose a reconfigurable on-demand multi-world-based partitioning system on heterogeneous SoCs. As RTrustSoC partitioning is flexible, the designer can choose

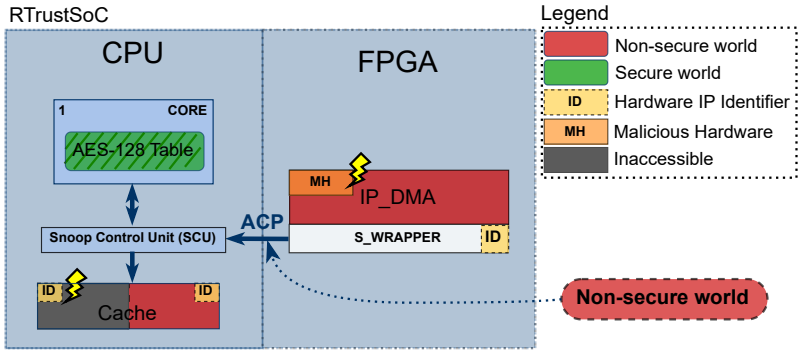

Fig. 7. Attack scenario based on [13] where a malicious hardware IP inside PL tries to perform cache timing attacks on an AES algorithm inside the PS implemented on an AMD-Xilinx Zynq-7000 SoC-FPGA.

the number of secure worlds. Our trusted communication system, which is based on hardware-coded operating rules, identifiers and security wrappers, enforces strict separation inside the heterogeneous SoC between the different secure domains proposed by RTrustSoC. The security wrappers make it impossible for a malicious entity to get information concerning a victim that may reside in one of the secure worlds. Neither could a potential attacker modify the identifiers to gain illegal access to a world where it does not belong. The dedicated security wrappers control all accesses to the memories, thus also making it impossible for a malicious entity to illegally access memory. When a malicious activity is detected, the dedicated security wrappers can also take proactive measures using a penalty system. The parameters of this penalty system are configurable through a secure boot configuration and evolve over time depending on the behavior of the entity. With these security features, RTrustSoC transposes the concept of trusted applications inside the processing system to trusted hardware IPs inside the programmable logic. The security wrappers guarantee privacy, confidentiality, integrity and access control, services required by the trusted components.

The proposed architecture has been prototyped on an AMD-Xilinx Zynq-7000 SoC-FPGA. Our experiment demonstrated that the hardware overheads of the communication monitoring and the dynamic penalty system are small in relation to the size of the targeted domains of application. The operating frequency of the system would not be affected, and only a small latency overhead is incurred for a slave interface which is not the case for a master interface, but this can be mitigated by using a Johnson counter. Concerning memory protections, we have shown that there is a lineal relationship between the hardware overhead, and the number of secure worlds and the components considered in the system. This can be mitigated by using the dedicated memories available in most modern platforms. Finally, we have demonstrated with a real case scenario of time-based attack on the cache memory, the interest of using RTrustSoC in heterogeneous SoC architectures.

Acknowledgments

This work was conducted in the framework of the TrustSoC project funded by the French *Agence de l'Innovation de Défense* (AID). This work was also supported by the French government through the *Agence Nationale de la Recherche* in the framework of project ARSENE (ANR-22-PECY-0004).

References

[1] A. Lanxon, "Why We Don't Need New Phone Launches Every Year," 2023. [Online] https://www.cnet.com/tech/mobile/why-we-dont-need-new-phone-releases-every-year/.

[2] H. Li, A. Abdelhadi, R. Shi, J. Zhang, and Q. Liu, "Adversarial Hardware With Functional and Topological Camouflage," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 68, no. 5, pp. 1685–1689, 2021.

[3] J. Robertson and M. Riley, "The Long Hack: How China Exploited a U.S. Tech Supplier," 2021. [Online] https://www.bloomberg.com/features/2021-supermicro/.

[4] A. Greenberg, "How a Shady Chinese Firm's Encryption Chips Got Inside the US Navy, NATO, and NASA," 2023. [Online] https://www.wired.com/story/hualan-encryption-chips-entity-list-china/.

[5] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," *ARM Inf. Q.*, vol. 3, no. 4, pp. 18–24, 2004.

[6] E. M. Benhani, L. Bossuet, and A. Aubert, "The Security of ARM TrustZone in a FPGA-Based SoC," *IEEE Trans. Comp.*, vol. 68, no. 8, pp. 1238–1248, 2019.

[7] M. Gross, N. Jacob, A. Zankl, and G. Sigl, "Breaking TrustZone memory isolation and secure boot through malicious hardware on a modern FPGA-SoC," *J. Cryptogr. Eng.*, vol. 12, no. 2, pp. 181–196, 2022.

[8] E. M. Benhani and L. Bossuet, "DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, IEEE, 2018.

[9] D. R. E. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-Based Covert Channels Using FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 6, 2021.

[10] L. Bossuet and C. A. Lara-Nino, "Advanced Covert-Channels in Modern SoCs," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 80–88, IEEE, 2023.

[11] R. Elnaggar, S. Chen, P. Song, and K. Chakrabarty, "Securing SoCs With FPGAs Against Rowhammer Attacks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 7, pp. 2052–2065, 2022.

[12] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *2014 USENIX Security Symposium*, pp. 719–732, USENIX Association, 2014.

[13] L. Bossuet and E. M. Benhani, "Security Assessment of Heterogeneous SoC-FPGA: On the Practicality of Cache Timing Attacks," in *2021 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 1–6, IEEE, 2021.

[14] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2020, no. 3, p. 169–195, 2020.

[15] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *2018 IEEE Symposium on Security and Privacy (S&P)*, pp. 229–244, IEEE, 2018.

[16] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An Inside Job: Remote Power Analysis Attacks on FPGAs," *IEEE Des. Test*, vol. 38, no. 3, pp. 58–66, 2021.

[17] P. Nasahl, R. Schilling, M. Werner, and S. Mangard, "HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment," in *2021 ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pp. 187–199, ACM, 2021.
[18] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "CURE: A Security Architecture with CUstomizable and Resilient Enclaves," in *2021 USENIX Security Symposium*, pp. 1073–1090, USENIX Association, 2021.
[19] C. Andriamisaina, F. Thabet, J.-R. Coulon, G. Chauvon, A. C. Aldaya, N. Tuveri, M. C. Martínez-Rodríguez, and P. Brox, "Secure platform for ICT systems rooted at the silicon manufacturing process," in *2023 RISC-V Summit Europe*, RISC-V International, 2023.
[20] A. Fitch, "Arm-Based Chips Make Inroads With Apple, Amazon," 2023. [Online] https://www.wsj.com/articles/arm-based-chips-make-inroads-with-apple-amazon-11674436002.
[21] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *2006 Cryptographers' Track at the 2006 RSA Conference (CT-RSA)*, pp. 1–20, Springer, 2005.
[22] AMD-Xilinx, "Programming ARM TrustZone Architecture on the Xilinx Zynq-7000 All Programmable SoC," User Guide UG1019 (v1.0), AMD-Xilinx, Santa Clara CA, USA, 2014.
[23] ARM, "AMBA AXI and ACE Protocol Specification," White paper ARM IHI 0022H (ID040120), Arm Limited, Cambridge, England, 2020.
[24] A. Shabani and B. Alizadeh, "Enhancing Hardware Trojan Detection Sensitivity Using Partition-Based Shuffling Scheme," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 68, no. 1, pp. 266–270, 2021.
[25] M. Sabri, A. Shabani, and B. Alizadeh, "SAT-Based Integrated Hardware Trojan Detection and Localization Approach Through Path-Delay Analysis," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 68, no. 8, pp. 2850–2854, 2021.
[26] M. Hagan, F. Siddiqui, and S. Sezer, "Policy-Based Security Modelling and Enforcement Approach for Emerging Embedded Architectures," in *2018 IEEE International System-on-Chip Conference (SoCC)*, pp. 84–89, IEEE, 2018.
[27] A. Basak, S. Bhunia, and S. Ray, "A flexible architecture for systematic implementation of SoC security policies," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 536–543, ACM, 2015.
[28] N. Singh, S. Pal, R. Leupers, F. Merchant, and C. Rebeiro, "PROMISE: A Programmable Hardware Monitor for Secure Execution in Zero Trust Networks," *IEEE Embedded Syst. Lett.*, pp. 1–4, 2024.
[29] R. Milan, L. Bossuet, L. Lagadec, C. A. Lara-Nino, and B. Colombier, "Trustsoc: Light and efficient heterogeneous soc architecture, secure-by-design," in *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1–6, IEEE, 2023.
[30] J. Porquet, C. Schwarz, and A. Greiner, "Multi-compartment: A new architecture for secure co-hosting on SoC," in *2009 International Symposium on System-on-Chip (SoC)*, pp. 124–127, IEEE, 2009.
[31] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *2009 IEEE Symposium on Security and Privacy (S&P)*, pp. 45–60, IEEE, 2009.
[32] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert, "Software mitigations to hedge AES against cache-based software side channel vulnerabilities," Preprint 2006/052, IACR Cryptology ePrint Archive, 2006.
[33] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, "An Overview of Hardware Security and Trust: Threats, Countermeasures, and Design Tools," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 40, no. 6, pp. 1010–1038, 2021.

Raphaële Milan received her Master's degree in electronics and embedded systems from the Télécom Saint-Étienne and Ecole Centrale de Lyon in 2021. Since 2021, she has been pursuing her PhD at the Hubert Curien Laboratory of the University Jean Monnet, France. Her research interests include reconfigurable architecture and hardware security.

Lilian Bossuet (Senior Member, IEEE) received his MSc degree in electrical engineering at INSA, Rennes, France, in 2001, and his PhD in electrical engineering and computer sciences from the University of South Britanny, Lorient, France, in 2004. Since 2017, he has been a professor at the Jean Monnet University of Saint-Etienne, where he is the head of the Computer Science Department of the Hubert Curien Laboratory, he is also the head of the Secured Embedded Systems and Hardware Architecture Group of this laboratory. His main research interests include hardware security, security of embedded systems, IP protection, PUF design and characterization, secure-by-design crypto-processor, and reconfigurable architecture. He has published more than 200 refereed publications in these areas.

Loïc Lagadec received his Ph.D. in computer science from the University of Rennes 1, Rennes, France, in 2000, and a Habilitation to Supervise Research (HDR) from the University of Brest, Brest, France, in 2009. He is a Full Professor at the Laboratory-STICC (CNRS), ENSTA Bretagne, Brest, where he is also Head of Research of the IT Department. His current research interests include software tools for reconfigurable computing, cyber security, and interpreted languages. Prof. Lagadec has been a Guest Editor for several special issues of scientific journals.

Carlos Andres Lara-Nino received his Master and Ph.D. degrees in Computer Sciences from CINVESTAV (Mexico) in 2016 and 2020, respectively. He is a researcher with the Security & Privacy Research Group (CRISES) of the Rovira i Virgili University (Spain). From 2021 to 2024 he was a postdoctoral fellow with the Hubert Curien Laboratory of the Jean Monnet University and the CNRS (France). His academic interests include digital systems design, data processing with reconfigurable hardware, information and hardware security, and cryptography.

Brice Colombier received his Master's degree in electronics and embedded systems from Télécom Saint-Étienne and INSA Lyon in 2014 and his Ph.D. degree in microelectronics from the University of Lyon in 2017. Since 2022, he has been Associate Professor at the Jean Monnet University in Saint-Etienne, France. He has been an Associate Editor of Journal of Cryptographic Engineering (Springer) since 2020. His research interests include hardware security, physical attacks, and post-quantum cryptography.

Théotime Bollengier received the engineering degree from ENSEIRB-MATMECA, Bordeaux, France in 2013 and his PhD degree his PhD degree in computer science from the University of Bretagne Loire in 2018. His interests include hardware design, reconfigurable computing architectures. Since 2018, he has been an engineer at Lab-STICC laboratory, ENSTA Bretagne, France.