

A Design Flow to Securely Isolate FPGA Bus Transactions in Heterogeneous SoCs

Niko Salamini*, Sara Alonso Salazar[†], Gabriele Serra[‡], Giorgiomaria Cicero[‡], Pietro Fara[‡],
Federico Aromolo*, and Alessandro Biondi*

*Scuola Superiore Sant'Anna, Pisa, Italy

[†]University of the Basque Country, Leioa, Bizkaia, Spain

[‡]Accelerat s.r.l, Pisa, Italy

Abstract—Embedded computing systems are becoming increasingly complex. Modern system-on-chips come with heterogeneous designs that integrate diverse processing systems and a large variety of peripherals. When considering software with mixed and independent security and criticality levels, the heterogeneity of modern computing platforms poses considerable challenges in achieving strong isolation between execution domains. Tackling these challenges is even more difficult in platforms that integrate Field-Programmable Gate Array (FPGA) fabrics, which, due to their wide flexibility, introduce new security- and safety-related threats that can jeopardize isolation. As a matter of fact, if no proper countermeasures are in place, hardware accelerators (HAs) deployed on FPGA can be exploited to break the isolation capabilities implemented in a system by issuing dangerous bus transactions. This research proposes a design flow for heterogeneous platforms to strongly isolate bus transactions issued by HAs. The design flow is then specialized for the AMD Zynq UltraScale+ platform, leveraging the virtualization-related features of the Arm System Memory Management Unit (SMMU). The proposed solution jointly combines two new IPs for enforcing information transported by the AXI bus, a tool to verify the FPGA design, a principled configuration of the SMMU driver, and a secure boot flow. The proposal is evaluated with an industry-relevant use case related to embedded machine learning applied for the railway domain, in which isolation is established between two AMD Deep Learning Processor Units (DPU) and a set of FPGA HAs dedicated to a real-time critical application.

Index Terms—Xilinx UltraScale+, Field-programmable gate array (FPGA), Deep Learning Processor Unit (DPU)

I. INTRODUCTION

Embedded systems play a pivotal role in applications such as nuclear power plants, railways, automotive systems, and robotics, where they are often constrained by real-time, power, and reliability requirements. To meet the growing industry demands for processing power, semiconductor manufacturers designed heterogeneous multi-core systems-on-a-chip (SoCs), which include processors and programmable accelerators, such as GPUs and FPGAs. An FPGA-based SoC platform combines a *Processing System* (PS) with an FPGA unit, also referred to as *Programmable Logic* (PL). Software tasks run on processors, while the PL can host hardware accelerators (HAs). Hardware acceleration is essential to meet performance and real-time constraints beyond the capabilities of CPUs. HAs and processors share resources, including memories, peripherals, and system registers, and communicate via PL-PS and PS-PL interfaces using a bus such as the Advanced

Microcontroller Bus Architecture (AMBA) AXI protocol [1]. Heterogeneous platforms enable the consolidation of systems that previously required dedicated processing units. Modern SoCs hence support systems with *Multiple and Independent Levels of Security/Safety* (MILS), allowing both untrusted (low-critical) and trustworthy (high-critical) applications to coexist on a single chip, thereby enhancing resource efficiency. Modern embedded systems often require a general-purpose environment for user applications alongside a secure, real-time environment for safety-critical software. The challenge lies in maintaining predictable behavior while hosting applications with different levels of criticality and security on the same hardware. In this context, an SoC can be seen as a collection of distinct *Hardware Domains* (HDs), each defined by its own set of computing resources, bus managers, security levels, peripherals, and memory address space regions. Specifically, the PL may be subject to severe security threats: if not properly managed, HAs in PL may gain unauthorized access to the system resources [2] [3] [4] [5], hence compromising isolation. Existing isolation techniques [6]–[12] fail to address key threats: (i) compromised isolation when multiple HAs share the same PL-PS interface, (ii) lack of control over critical bus signals from HAs, and (iii) the introduction of vulnerabilities at the PL design stage. Furthermore, most solutions often lack compliance with technologies like Arm TrustZone and IOMMU virtualization.

Contribution. In this paper, we propose a *secure-by-design* workflow to properly isolate HDs at the PL level by making the following contributions:

- 1) A design flow for enforcing isolation between different HDs within a heterogeneous SoC based on:
 - An IP called *AXI Enforcer* to securely control critical AXI bus signals at the design stage.
 - An IP called *AXI ID Mapper* (AIM) enabling PL-PS port sharing between HAs belonging to different HDs.
 - Compliance with Arm TrustZone technology and IOMMU virtualization.
 - A firmware component for securely configuring the Arm SMMU.
 - A tool to verify the PL design to avoid hardware vulnerabilities and/or unintended misconfigurations.
- 2) An experimental evaluation based on an industry-relevant

use case based on the AMD/Xilinx Zynq UltraScale+ MPSoC for a railway system including a real-time safety-critical HD, managing different protocol interfaces, alongside two virtual machines (VMs) with dedicated HDs, each accessing an *AMD Deep Learning Processor Unit* (DPU) core. It jointly addresses *response times* and *latency*, *bandwidth*, and *area* requirements.

Paper Organization. This paper is structured as follows: Section II introduces key concepts related to the architecture of heterogeneous SoC designs and resource isolation when hosting software with mixed levels of security and criticality. Section III presents the system and threat model. Section IV discusses the related work and a corresponding taxonomy. Section V presents the proposed design flow. Section VI discusses implementation issues for the UltraScale+ architecture. Section VII illustrates our case study, while Section VIII reports the corresponding experimental results. Finally, Section IX concludes the paper and discusses future work.

II. ESSENTIAL BACKGROUND

Modern embedded platforms often consist in heterogeneous SoCs that integrate a PS unit with an FPGA PL component within a single device. Fig. 1 illustrates the typical architecture of a heterogeneous SoC. The PS includes various processing units, memory elements, and peripherals, while the PL can host multiple HAs. These components are interconnected via the AMBA AXI bus protocol, the de facto standard for bus communication on modern platforms. The PS-PL and PL-PS ports enable communication between the PS and PL.

The flexibility of heterogeneous systems lies in their capability to host and execute software with varying levels of security and criticality on the same platform. In this context, HDs can be defined based on the separation of resources according to security- and safety-related levels. Security level separation is achieved through Trusted Execution Environment (TEE) technology, which isolates secure applications from untrusted ones. Safety-critical separation is enforced using hardware protection units (HPUs) and the IOMMU, which implement strict access controls and prevent unauthorized memory access.

AXI protocol. The AXI protocol consists of five channels: write/read address, write/read data, and write response. Address channels include the address of the target transaction (AxADDR) plus some control signals (AxBURST, AxLEN, AxSIZE, etc.). Additional control signals include the TrustZone security level (AxPROT), quality of service (AxQOS), and cache control (AxCACHE). Data channels include the actual data (WDATA for writes, RDATA for reads) along with byte strobes (WSTRB) for write operations. Each transaction follows a handshake mechanism based on valid (VALID) and ready (READY) signals, ensuring proper synchronization between master and slave. The write response channel (BRESP) provides feedback on the outcome of write operations, indicating success or errors. Furthermore, the protocol supports user-defined signals (AxUSER, WUSER, RUSER, etc.), which

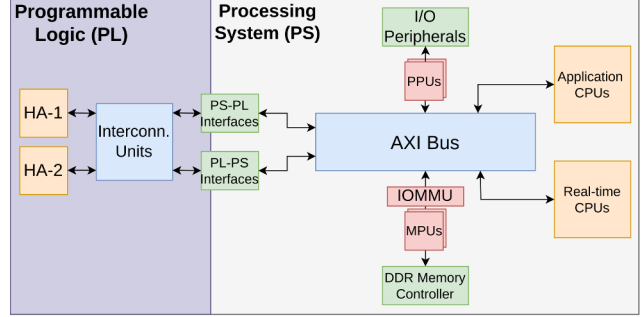


Fig. 1. Modern SoC architecture with hardware accelerators in the programmable logic.

allow custom metadata to be passed alongside transactions for specialized applications.

TEEs. To enhance security, manufacturers released hardware support for implementing TEEs [13] [14]. Many embedded SoCs rely on the Arm TrustZone technology [15], which allows trusted applications to run in a TEE. To protect the execution of trusted applications, TrustZone separates the system resources into the Secure World and the Non-Secure World. Memory regions, memory management unit (MMU) page tables, peripherals, and interrupts are designated as either Secure or Non-Secure, ensuring that sensitive data and operations remain accessible to the Secure World only. Moreover, AXI bus transactions are marked with a security attribute to recognize the world the bus managers belong to. This setup allows TrustZone to protect secure applications and data, even if the Non-Secure World (e.g., a rich OS) is compromised.

HPUs. While TEEs enable resource partitioning by security level, system resources can also be partitioned to support varying levels of criticality. This allows real-time, safety-critical tasks to run alongside less critical tasks managed by a rich OS. Heterogeneous SoCs include HPUs for enforcing the isolation of system resources. The Memory Protection Unit (MPU) enforces access control for memory regions, while a Peripheral Protection Unit (PPU) manages access to the peripheral address space. Designers can configure these units to allow or deny access based on the bus manager ID value, and the units will act accordingly (e.g., by generating an interrupt) in case of a violation.

IOMMU. The IOMMU is a hardware component designed to virtualize memory address spaces for peripherals and, more generally, for bus managers. Specifically, the IOMMU maps device addresses to physical addresses. The latest implementations of this unit support TEE, allowing distinct translation regimes to be defined according to security levels.

III. SYSTEM AND THREAT MODEL

This section presents the system and threat model adopted in this work.

System Model. We consider a heterogeneous system including a PS and a PL FPGA unit that can host multiple HAs. All components of the PS, including CPUs and peripherals, are aware

of TEE technology (e.g., Arm TrustZone) and communicate over the bus using the AXI protocol. The PS includes two multi-core CPUs. On the first CPU, either native applications or virtual machines (VMs) managed by a hypervisor can be executed. The second CPU runs a real-time operating system (RTOS) where multiple tasks are scheduled. The HPUs are configured to ensure exclusive access of the CPUs to specific memory regions and peripherals. Both virtual machines (or native applications) and RTOS tasks have access to a subset of HAs in PL. The HAs need to share PL-PS ports for accessing the PS. A secure boot process employs encryption and authentication schemes for all the software partitions (including the PL bitstream). In the following, we assume that, when access to PL is shared between VMs running on the same CPU, the hypervisor is responsible for supervising the access to HAs from each VM, so that VMs can only interact with the HAs they have been granted access to.

Threat Model. We assume that secure boot cannot be compromised, e.g., the system is resistant to differential power analysis attacks targeting the hardware root of trust (HRoT) keys [16], [17]. We also assume that the adversary can take control at run-time of the user applications and the OSes. The trusted computing base is composed of boot loaders, the hypervisor (if any), and privileged firmware (e.g., Arm Trusted Firmware). In this context, threats are related to the vulnerabilities and potential issues associated with the state-of-the-art design workflows of FPGA-based heterogeneous platforms:

- **T1. HA-Driven Manipulation of AXI Bus Signals:** The HAs within the system can manipulate AXI signals, which determine access to secure resources (AxPROT), access priority on the DDR memory controller (AxQOS), and control over access to processor caches (AxCACHE). Malicious or compromised HAs could exploit these signals to bypass isolation.
- **T2. Lack of Isolation on Shared PL-PS Ports:** Commonly (as it is the case for the Ultrascale+), when HAs share PL-PS ports, HPUs are unable to enforce distinct access controls over memory and peripheral address spaces. Instead, a single access control is applied to the entire PL-PS port based on the bus manager ID. The IOMMU cannot in turn be configured to apply separate translation regimes (i.e., sets of rules for translating virtual to physical addresses) for the HAs. This limitation arises because the IOMMU associates translation regimes either with a single ID linked to each PL-PS port or with an ID obtained by combining the former with an HA-controlled ID. This limitation allows an adversary to exploit a compromised HA to access protected memory regions or peripherals designated for other HAs or software components.
- **T3. Insufficient Protection of the IOMMU Configuration Address Space:** The IOMMU configuration address space is generally memory-mapped and accessible to all bus managers. When multiple bus managers require

access to configure the unit, the integrity of translation regime configurations cannot be guaranteed. MPUs lack the granularity to restrict access to specific page tables, as they can only control access over the entire IOMMU configuration address space.

- **T4. Lack of PL Design verification:** The source code of HAs is vulnerable at the development stage, and it can be manipulated to introduce hardware vulnerabilities or trojans [3].

The design flow proposed in this research addresses **T1** and **T2** by integrating two new IPs in the PL design: AXI Enforcer and AXI ID Mapper (see Section V-A). **T3** is handled by a firmware component included in the boot process, which first configures the translation regimes and then locks the IOMMU configuration address space (Section V-B). **T4** is addressed through a verification tool that analyzes the PL design and checks for possible misconfigurations (Section V-C).

IV. RELATED WORK

Various solutions were proposed in previous work for extending the concept of isolation to the PL, and different approaches were used in practice. Before analyzing the related work in this field, a taxonomy is introduced to classify the existing solutions.

A. Taxonomy

1) *Isolation Method.* It defines the technology used for ensuring isolation. Solutions proposed in the literature mainly leverage one or more technologies: MPU, IOMMU, TEE realized in PL, or a new architectural approach.

2) *Isolation Level.* It refers to the layer at which resource isolation is enforced within a system. The isolation level can be categorized as follows:

- Application Level: The OS assigns HAs to applications and ensures isolation.
- VM Level: VMs share access to the PL and the hypervisor ensures the isolation of the HAs and their corresponding memory regions.
- HD Level: HDs are isolated by the hardware design and configuration.

3) *AXI Signal Control.* It refers to the ability to control the following AXI signals for HAs: AxPROT (TrustZone), AxQOS, and AxCACHE (specific to Arm platforms).

5) *PL-PS port sharing.* It refers to whether PL-PS ports can be shared among multiple HAs without jeopardizing isolation.

6) *Virtualization.* It refers to the support for virtualization of the memory address space exposed to HAs.

7) *IOMMU runtime protection.* It refers to whether the IOMMU unit is protected from runtime reconfiguration.

8) *PL Design Verification.* It refers to whether the PL design is verified at the development stage.

B. Related work analysis

The works reviewed in this section adopted different isolation techniques for HAs in PL.

TABLE I
CLASSIFICATION OF RELATED WORK

Related work	Isolation Method	Isolation Level	AXI Signal Control	PL-PS Sharing	Virtualization	IOMMU runtime prot.	PL Design verification
Olson <i>et al.</i> [6]	IOMMU	App.	✗	✓	✓	✗	✗
Alam <i>et al.</i> [7]	IOMMU	App.	✗	✓	✓	✗	✗
Pham <i>et al.</i> [8]	IOMMU	VM	✗	✗	✓	✗	✗
Karabulut <i>et al.</i> [9]	MPU	HD	TZ	✗	✗	-	✗
Donchez <i>et al.</i> [10]	MPU	HD	TZ	✗	✗	-	✗
Milan <i>et al.</i> [11]	MPU	HD	TZ	✓	✗	-	✗
Armanuzz. <i>et al.</i> [12]	TEE	HD	✗	✗	✓	-	✓
Ren <i>et al.</i> [18]	TEE	App.	✗	✗	✓	-	✗
Benhani <i>et al.</i> [19]	Architect.	HD	TZ	✗	✗	-	✓
Gouveia <i>et al.</i> [20]	Architect.	HD	✗	✓	✓	-	✗
Asmussen <i>et al.</i> [21]	Architect.	HD	✗	✓	✓	-	✗
<i>This Paper</i>	IOMMU	HD	✓	✓	✓	✓	✓

-: Not applicable, TZ: Only AxPROT TrustZone control.

One common approach is utilizing the IOMMU to enforce isolation in multi-tenant environments within the PL. Olson *et al.* [6] proposed a solution called Border Control, which leverages the IOMMU's Address Translation Service (ATS) to ensure that HAs adhere to memory access permissions defined in the page table. This method confines accelerators to the virtual memory space of their corresponding OS processes. Alam *et al.* [7] presented an enriched IOMMU design, called CryptoMMU, for improving scalability when dealing with multiple untrusted accelerators. In this solution, cached IOMMU translations related to HAs are authenticated using secure hash functions, with authentication keys defined for each pair of the form {accelerator, OS process}. Pham *et al.* [8] proposed an FPGA OS that ensures memory isolation for accelerators on the Xilinx UltraScale+ architecture. Their system employs a kernel driver to configure the IOMMU and map each accelerator to a dedicated translation regime.

Another possibility is to use an MPU-like approach to protect access to shared memory. Emre Karabulut *et al.* [9] extended the AXI crossbar switch by incorporating access control functionality to secure AXI subordinate access, supporting dynamic reconfiguration privileges and compatibility with Arm's TrustZone. Donchez *et al.* [10] leveraged the XMPU_PL (Xilinx Memory Protection Unit for PL isolation) [22] to safeguard access to HAs and restrict them to their allocated memory regions. Milan *et al.* [11] presented an SoC architecture that extends TrustZone to support multiple environments that can be associated with different HAs.

Another relevant approach is building customized TEEs inside the PL. BYOTee (by Armanuzzaman *et al.* [12]) is a hardware-software co-design infrastructure for creating enclaves with customized hardware in the PL, establishing a dynamic root of trust for secure execution of sensitive appli-

cations. Ren *et al.* [18] proposed extending TEEs for multi-tenant cloud environments by introducing an architecture that uses the AccGuard [23] framework for secure and encrypted communication between host and PL-based TEEs. AccGuard is a hardware enclave framework supporting isolation and remote attestation. The authors also extended the Coyote framework [24] to enable virtualization. This framework provides OS abstractions and a variety of features for enhancing the FPGA capabilities.

Different architectural approaches to guarantee isolation of accelerators were proposed in previous work. Benhani *et al.* [19] presented six distinct attacks to TrustZone's security mechanisms in FPGA-based SoCs and two new security controllers to enhance access control and mitigate attacks. Gouveia *et al.* [20] introduced Midir, a manycore capability-based architecture that uses hardware-based trusted components, called T2H2, to manage access to shared resources. Asmussen *et al.* [21], [25] presented M3, a hardware/software co-design that implements isolation between cores and accelerators in a heterogeneous system, leveraging a Network-on-Chip (NoC) as a communication subsystem. Kurth *et al.* [26] proposed an IP called ID Remapper to set the AXI IDs to dynamic values. However, if used in our context, crucial isolation requirements would not be satisfied.

Table I classifies the related work according to the taxonomy presented above. Solutions that rely on software components only do not implement HD-level isolation and do not adhere to our threat model. Many approaches also lack isolation when PL-PS ports are shared between HAs, which is critical, especially when the number of HAs exceeds the number of PL-PS ports. IOMMU-based approaches overlook the possibility of runtime reconfiguration of translation regimes, highlighting the need to secure the IOMMU configuration address space. MPU-based solutions, on the other hand, cannot provide memory virtualization. While using a dedicated TEE in PL is an excellent strategy for establishing isolation, challenges emerge when dealing with resource-constrained SoCs that have limited PL area/resources. When a large number of HAs must reside in the PL, the resources required to build the TEE can hardly be available. Custom architectural solutions face similar limitations, as they require substantial re-implementation of the bus and memory subsystems. Most of those proposed in previous work are also incompatible with Arm's IOMMU and TrustZone.

Most importantly, none of the previous works provided control of critical AXI signals, such as AxQOS, which regulates access priority to the DDR memory controller, and AxCACHE, which controls access to the processor caches, thereby overlooking the potential risk for malicious HAs freely manipulating these signals. In this regard, Bossuet *et al.* [27] demonstrated the feasibility of Flush+Reload and Evict+Time attacks by exploiting cache coherency mechanisms between the PS and PL in modern SoCs. Some of the works provided support for TrustZone by controlling the AxPROT signal only. Finally, all but one of the works we reviewed did not address PL design verification, leaving them susceptible to hardware

trojans that may be introduced by designers.

In summary, although various approaches were proposed to isolate HAs, *none of them implemented strong and resource-efficient isolation at the HD level while enabling memory virtualization and providing support for controlling critical AXI signals and PL design verification.* This work fills this relevant gap.

V. DESIGN FLOW

The design flow presented in this section addresses the challenge of achieving HD-level isolation while offering all the features specified in the taxonomy of Sec. IV-A. To do so, the proposed flow ensures the following requirements.

R1. Protected access from/to HAs. HAs need to be protected from unauthorized access by PS managers while HAs themselves must be prevented from performing unauthorized accesses to PS resources.

R2. Dedicated IOMMU translation regimes for HAs. Modern IOMMUs, such as the Arm's SMMU, can implement isolation by configuring separate translation regimes based on an identifier and TrustZone security level. In the AXI protocol, the identifier is called *Stream ID*. Typically, in heterogeneous platforms including a PL unit, Stream IDs are determined by (i) a PS hard-wired part and (ii) a flexible part set by PL devices. The latter is specified through an attribute called AXI ID. The HAs hence need to be associated with a unique set of Stream IDs and a TrustZone security level, so that the IOMMU can be configured to enforce a per-HA translation regime, successfully implementing virtualization and TrustZone compliance.

R3. Explicit management of critical AXI bus signals. HAs can freely control critical AXI signals such as cache control and QoS. As stated in the previous section, a malicious HA can manipulate these signals to compromise the isolation of the HDs. It is therefore required to control the TrustZone, QoS, and caching AXI bus signals for each HA at the design stage.

R4. Configuration and locking of the IOMMU. In principle, the IOMMU configuration address space can be accessed by all bus managers. As such, the IOMMU translation regimes must be configured during at boot time and locked at runtime. This is crucial to prevent any component within an HD from reconfiguring the IOMMU, which could compromise isolation.

R5. Anti-tampering. Software partitions and the PL bitstream (i.e., its configuration) must be authenticated and encrypted using an HRoT to prevent system tampering.

The proposed design flow is composed of three steps. **(I)** To match requirements R1, R2, and R3 it requires integration of two IPs in PL, which are presented in Section V-A. **(II)** To match requirement R4, it requires to install a firmware component, detailed in Section V-B. **(III)** To match requirement R5, it mandates secure boot and PL design verification, which are addressed in Section V-C. Specialized implementations of these steps are discussed later in Section VI. Note that, since the IPs are statically configured at design time, the design flow

cannot be applied in cases of dynamic partial reconfiguration that require changes to their configuration.

A. Two new IPs to isolate HAs

We designed and implemented two new lightweight IPs: **AXI Enforcer** and **AXI ID Mapper (AIM)**. They are meant to work in conjunction to fix the AXI attributes and dedicate a set of Stream IDs to each HA. The PS hard-wired part of a Stream ID is associated with the PL-PS port, while the PL flexible part is specified in the AXI ID. AXI IDs are used to implement multithreading in the AXI protocol, and their values are either set by interconnection units, or by the IPs themselves when directly connected to the PL-PS interface. Consequently, the resulting Stream ID in the PS is out of the control of designers.

The idea of the proposed approach consists in statically assigning a set of AXI IDs to each HA in the PL design. In this way, each HA will generate a set of unique Stream IDs to be associated with a dedicated translation regime in the IOMMU. Translation regimes are also matched based on the AXI TrustZone security level. Fig. 2 provides a logical overview of the proposed solution. First, an MPU unit is required to protect the access to the HAs from the PS managers. The AXI enforcer is then used to fix the TrustZone security level signal (AxPROT), the AxQOS signal, and the AxCACHE signal. It is also used to set the AXI user signal (AxUSER) to the manager ID of the connected HA. Bus transactions then pass through the interconnection unit to the AIM, where the AxUSER signal identifies the manager interface and maps incoming AXI IDs to a unique set for each HA. In this way, the resulting Stream IDs will be uniquely associated with the HA. The IOMMU can be configured to match the HAs transactions to dedicated translation regimes while preserving the TrustZone's Secure/Non-Secure resource separation. The next sections specify the details regarding the IP functionality and configuration parameters.

1) *AXI Enforcer:* HAs are free to manipulate critical AXI signals, being directly generated by them. Furthermore, when a new transaction arrives from the manager port of an interconnect unit, it is common that AXI IDs do not consistently match those of the source managers. For instance, this happens with all the AMD PL interconnects we tested. To address these issues, AXI Enforcer can be employed to *enforce the information transmitted on critical bus signals* such as security level (AxPROT), QoS (AxQOS), caching (AxCACHE), and user-defined (AxUSER), on both read and write channels. All the other AXI signals are propagated as they are.

The AXI Enforcer must be installed between the PL manager interface and the subordinate port of interconnects. Differently from the AXI ID, the interconnection unit does not apply any modification to AxUSER signals. Therefore, AXI Enforcer can set AxUSER signals to identify managers at later stages on the bus. Enforcing the AxPROT signal to a value chosen by the designer prevents HAs from changing their TrustZone security level on the bus, thereby preserving TrustZone-assisted isolation. As DDR memory controllers

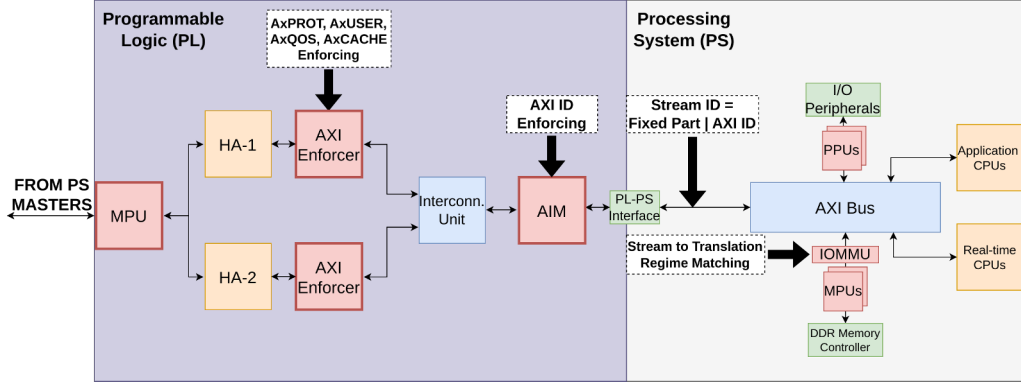


Fig. 2. Design flow to isolate the transaction of HAs sharing a port using an interconnection unit.

TABLE II
AXI ENFORCER CONFIGURATION PARAMETERS

Parameter Name	Description	Value Range
AXI_ADDR_WIDTH	The AXI address channel width	12-64
AXI_DATA_WIDTH	The AXI data channel width	32-512
AxPROT_value	The value of the AxPROT channel	0-7
AxUSER_value	The value of the AxUSER channel	0-1023
AxQOS_value	The value of the AxQOS channel	0-15
AxCACHE_value	The value of the AxCACHE channel	0-15

prioritize AXI transactions based on the AxQOS values, it is also crucial to enforce AxQOS to values chosen by designers to prevent HAs from altering their access priority to memory. Finally, also enforcing the AxCACHE signal allows restricting unauthorized access to the processor's caches. The values to be enforced can be set by designers in the configuration parameters of AXI Enforcer, reported in Table II.

2) *AXI ID Mapper (AIM)*: The interconnection unit is directly connected to an IP called AXI ID Mapper (AIM). The AIM can be configured to dedicate a set of unique AXI IDs to each HA for both read and write channels. When a transaction arrives, the AXI ID provided by the interconnection unit is mapped based on the AxUSER signal value configured by AXI Enforcer. The resulting Stream IDs on the PS will uniquely identify the HA that has originated the transaction. On the response, the AIM maps the incoming AXI ID to the original value. In this way, the interconnection unit can correctly route the response to the HA according to its AXI thread assignment. The configurable parameters of AIM are summarized in Table III.

AIM works on *pools*, which are unique sets of AXI IDs dedicated to an HA. The number of pools in AIM is configurable at design time, based on the number of HAs to be supported (each HA is assigned a pool). For each pool with index i , a corresponding configuration parameter AxUSER_MAP_ i , which defines the maximum number of AXI IDs associated with the HA, is provided (see Table III) to match with the value carried by the AxUSER signal. Each pool hence contains

TABLE III
AIM CONFIGURATION PARAMETERS

Parameter Name	Description	Value Range
AXI_ADDR_WIDTH	The AXI address channel width	12-64
AXI_DATA_WIDTH	The AXI data channel width	32-512
POOL_SIZE	The pool size, i.e., number of unique AXI IDs per manager	1-64
NUMBER_OF_MANAGERS	Number of isolated managers	1-64
AxUSER_MAP_ i	AxUSER channel value associated with the i^{th} pool	0-1023
WRITE_REQ_BUF_SIZE	Write request channel buffer size	2-64
WRITE_BURST_BUF_SIZE	Write burst channel buffer size	2-64
WRITE_RSP_BUF_SIZE	Write response channel buffer size	2-64
READ_REQ_BUF_SIZE	Read request channel buffer size	2-64
READ_BURST_BUF_SIZE	Read burst channel buffer size	2-64

POOL_SIZE AXI ID values, which are incremental. The first pool contains values from 0 to (POOL_SIZE - 1), the second from POOL_SIZE to $2 \times (\text{POOL_SIZE} - 1)$, and so on. The larger the pool size, the larger the number of out-of-order transactions that the AIM can handle. In the design flow, it is sufficient to fix the POOL_SIZE parameter to the number of threads the interconnection unit can generate. Given the pool size, the maximum number of HAs that can be isolated is given by

$$N_{\text{managers_max}} = \frac{2^{\text{AxID_Width}}}{\text{POOL_SIZE}}, \quad (1)$$

where AxID_Width is the length in number of bits of the AXI ID (e.g., for the AMD UltraScale+, AxID_Width=6).

When setting POOL_SIZE=1, the maximum number of HAs is 64. However, as interconnection units usually have a limited number of subordinate ports, a cascade of interconnections is required to sustain a large number of HAs.

The AIM unit includes FIFO buffers to store pending transactions for write and read operations on both the AXI

response, burst, and request channels. This buffering mechanism helps maintain a continuous flow of transactions between AXI endpoints by temporarily storing information when one side is not ready to receive or process them immediately. This reduces the idle time and maximizes the utilization of available bandwidth. When a new manager transaction reaches the AIM, the target subordinate interface may not be ready to process it. This is established by checking the xREADY channels. Instead of waiting for the subordinate to be ready, the AIM keeps accepting transactions and stores them in internal FIFO buffers. When the subordinate is ready, the AIM will send the stored transactions in FIFO order. The buffer parameters in Table III (i.e., those that include the `BUF` label) define the size of the buffers for each channel. The larger the buffer, the more transactions the AIM can process without initiating an xREADY handshake. Once the buffer is full, the AIM asserts its xREADY signal to halt the manager. The AIM complies with the following behavioral rules:

- 1) The AIM can receive both AXI read and write transactions. It handles them using separate FIFO buffers.
- 2) Upon receiving a valid request, if the target request buffer is not full, the transaction is buffered and the original AXI ID is replaced with a new value from the assigned pool selected from the `AxUSER` value identifying the manager.
- 3) Upon receiving a valid response, if the target response buffer is not full, the transaction is buffered and the AXI ID is reset to its original value.
- 4) On the rising edge of the clock, if the target subordinate interface is ready, the AIM sends the first buffered transaction, if any, and removes it from the buffer.
- 5) The AIM sets the corresponding xREADY signal when the buffer for that channel is full.
- 6) In the event of a design configuration error (e.g., the AIM subordinate interface receives an `AxUSER` that is not mapped), an interrupt notification signal is sent.

In bus communication, both the PL-PS interface and the interconnection unit may assert ready signals to indicate they cannot receive a response or request. AIM Buffers are used to *minimize the latency by AIM* during these events. Larger buffers allow the AIM to store more transactions, reducing ready handshakes and increasing area utilization. This trade-off is analyzed with experiments in Section VIII.

B. Configuring and locking the IOMMU

A firmware component is in charge of managing the IOMMU. As any AXI manager can potentially access and modify the translation regime of the HAs, compromising HD isolation, the IOMMU configuration must be protected. The firmware is hence required to first configure the IOMMU with the desired translation regimes and then lock the IOMMU register address space, so that no unauthorized modifications can be performed to jeopardize isolation. The First-Stage Boot Loader (FSBL), commonly present in SoCs, is responsible for initializing the hardware and loading the second stage boot-loader (e.g., U-Boot), setting up the basic system configuration and secure environment before transitioning to the next stage

of the boot process. Our firmware component can hence be executed at the end of FSBL operations. After it executes, no modifications to the memory virtualization scheme are allowed unless required by privileged software components that are part of the trusted computing base, e.g., a hypervisor.

C. Secure Boot and PL design integrity

When considering the proposed design flow, it is important to address the potential threats that could compromise the security of the system. The design flow enables HD-level isolation within the PL design. When working with complex heterogeneous systems running mixed-criticality applications, designers may either need to deal with untrusted (low-critical) HAs or inadvertently introduce vulnerabilities by means of malicious HAs (e.g., hardware trojans). To address these issues, our design flow integrates a verification tool to automatically validate the PL design, ensuring that both AIM and AXI Enforcer are installed and properly configured. Malicious users with access to the physical system may also attempt to modify one of the boot partitions or the PL bitstream to compromise the isolation between HDs. For this reason, the last step of the design flow consists in configuring a secure boot for enabling the authentication and encryption of the boot partitions, including the validated PL design. At the core of a secure boot process is the concept of HRoT. A HRoT is a secure, immutable component embedded within a SoC that serves as the foundation for establishing the system's integrity. It performs the authentication and decryption of the boot partitions to ensure that the system only boots with trusted software. A realistic secure boot process integrated with our design flow, based on the AMD UltraScale+'s HRoT implementation, is described in detail in the following section.

VI. IMPLEMENTATION

The implementation of the design flow is carried out on the AMD/Xilinx UltraScale+ MPSoC platform, which matches the characteristics of the typical heterogeneous SoC presented in the background section. The PL available on this platform can host multiple HAs, such as processors for artificial intelligence (AI), crypto accelerators, DMAs, and various protocol interfaces (e.g., SPI, UARTs, CAN). The proposed design flow can be utilized by designers seeking to implement HD-level isolation while ensuring full compliance with the features outlined in the taxonomy. Fig. 3 illustrates the steps when specializing the proposed design flow for the AMD UltraScale+, starting from a block design that includes AXI Enforcer and AIM modules. The isolation capabilities of the design flow have been validated through software programs that instruct HAs to attempt illegal access to memory regions across different HDs and to use forbidden AXI signal control values.

Portability. The design flow is conceived for AXI in general and Arm-based SoCs. While the implementation is specialized for UltraScale+ in the following, it can be adapted to other platforms (e.g., those produced Intel/Altera).

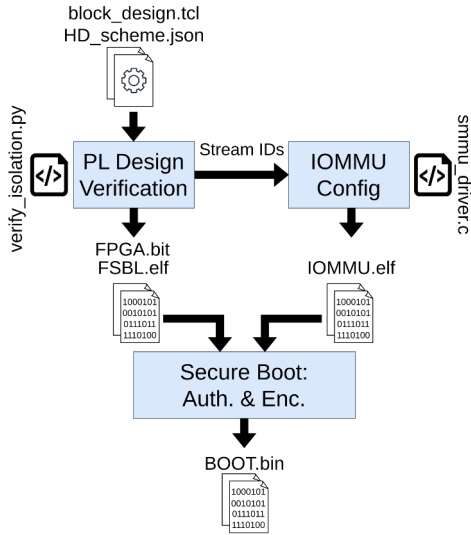


Fig. 3. Block diagram of the proposed design flow steps for AMD UltraScale+. The final output is a BOOT.bin file containing authenticated and encrypted executable binaries along with the authenticated and encrypted PL bitstream.

A. Verified PL Design

Configuring the PL design to match the architecture in Fig. 2 requires considerable effort from designers. To simplify and automate this process, we developed a tool (written in Python) compatible with the AMD Vivado design suite. This tool takes two input configuration files that the designers must provide to specify the isolation requirements for the HDs:

- 1) An exported block design from the Vivado project in Tool Command Language (TCL).
- 2) A JSON file specifying the HD scheme, including:
 - The set of HAs assigned to each HD, identified by a unique number.
 - The number of AXI threads used by each HA.
 - The values of AXI signals AxUSER, AxPROT, AxCACHE, and AxQOS for each HA (see Section V-A1).
 - The desired access bus priority for each HA.

The tool parses the TCL block design and matches each IP configuration based on its name in the design. It then verifies the connections and validates the AIM and AXI Enforcer parameters. Finally, it outputs the set of Stream IDs associated with each HA and generates warnings if the priority access scheme defined in the configuration file does not align with the AxQOS-assigned values. Fig. 4 shows an example template of the JSON configuration file. Fig. 5 illustrates the Vivado design used to isolate two central DMA (CDMA) units. The AXI Enforcer fixes the AXI signals reported above, while the SmartConnect interconnection unit from AMD/Xilinx [28] manages data flow. Finally, the AIM maps the transaction to a predefined set of AXI IDs, resulting in a dedicated set of Stream IDs in the PS. Notably, the AXI SmartConnect does not enforce AXI ID signals [28, p. 17], making the AIM essential

```

1  {
2    "IP_Name": "<value>",
3    "Hardware_Domain_ID": "<value>",
4    "AXI_Threads": "<value>",
5    "AxUSER": "<value>",
6    "AxPROT": "<value>",
7    "AxCACHE": "<value>",
8    "AxQOS": "<value>",
9    "Bus_Priority": "<value>"
10 }

```

Fig. 4. Example template of the JSON configuration for the PL design verifier.

for enforcement.

B. PS Design

A subset of the PL-PS interfaces of the UltraScale+ are subject to SMMU virtualization. The HP and HPC ports are the only PL-PS interfaces passing through the SMMU, while the remaining ports cannot be utilized for enabling memory virtualization. The stream-to-context mapping feature of the SMMU is used to map Stream IDs to different translation contexts. The SMMU can be configured to match the Stream IDs of transactions generated by HAs traversing the PL-PS interfaces. This enables the association of different contexts with the PL HAs to apply a dedicated translation regime. Secure context banks can be dedicated to the HAs flagged as TrustZone-secure. Translations can be organized in one or two levels. This is useful for a two-stage translation when running VMs upon a hypervisor.

When considering the PL-PS ports in the UltraScale+, the Stream ID has the following structure:

- Translation buffer unit (TBU) number (bits 14 to 10).
- Manager ID (bits 9 to 6).
- AXI ID from the PL manager (bits 5 to 0).

Each PL-PS port is connected to a single TBU unit and it is identified by a unique manager ID. The AXI ID serves two purposes: identifying the transaction's originating manager in the PS via the Stream ID and implementing transaction threads within the AXI protocol. Transactions with the same AXI ID must be returned in order, while those with different AXI IDs can be completed out of order. When connecting multiple managers to a single PL-PS port, a SmartConnect is needed. The AXI IDs originated by the connected HAs are modified by SmartConnect according to one of two possible configurations, (i) *Single-ordered mode* or (ii) *Multi-threaded mode*. In single-ordered mode, the transactions issued by HAs are propagated and returned in order. The propagated AXI ID has always a value of 0. On the other hand, in *Multi-threaded mode* the transactions issued by HAs share a set of AXI IDs that are dynamically assigned by the SmartConnect. In multi-threaded mode, the SmartConnect remaps the original AXI ID of the HA into one of the free AXI IDs available in a set. The size of the set specifies the number of threads that the SmartConnect can generate. When a transaction ends, the AXI ID is freed up and becomes available for a new transaction, which could be initiated by a different HA. In both configurations, HAs

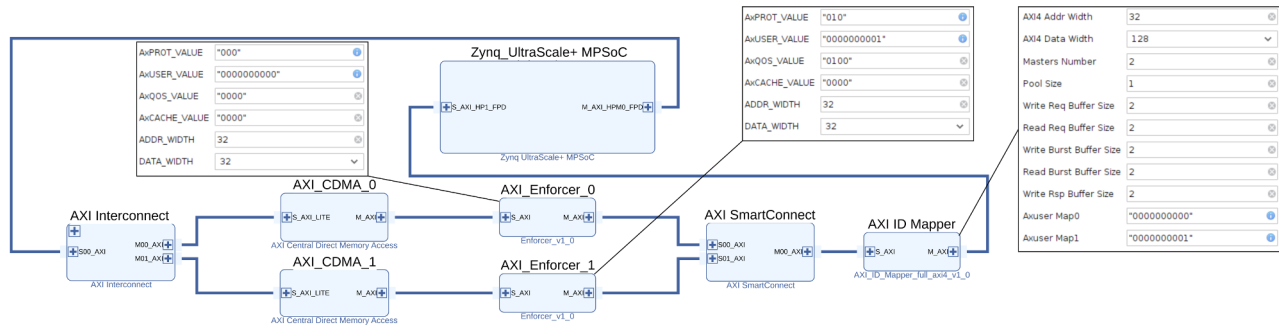


Fig. 5. Vivado PL block design showing the configuration for the isolation of two CDMA units belonging to different HDs.

connected to the SmartConnect share the same set of AXI IDs, resulting in identical Stream IDs in the PS. For this reason, the two IPs are necessary to bind each HA to a unique set of AXI ID values, resulting in a set of unique Stream IDs in the PS.

C. Secure Boot

At boot time, a firmware component first configures the translation regimes on the SMMU context banks and then locks the register address space. This is accomplished by configuring the XMPU PS to block access from any manager to the memory-mapped region of the SMMU register address space. The secure boot feature in the UltraScale+ is offered by two hardware units: the platform management unit (PMU) and the configuration secure unit (CSU). The secure boot starts with dedicated hardware validating the PMU ROM and performing PMU registers zeroization. The PMU performs additional checks on the CSU ROM code and releases its reset signal. The CSU decrypts and authenticates the FSBL partition. Finally, the FSBL decrypts and verifies the proposed firmware component, provided in a separate partition, and the PL FPGA bitstream using the CSU cryptographic accelerators. The authentication scheme is the HRoT proposed by AMD/Xilinx [29], which is based on two pairs of public-private keys. The encryption method is based on a key-rolling scheme still proposed by AMD/Xilinx, in which the entire encrypted image is composed of smaller AES-encrypted blocks. The PL bitstream file is too large to be contained inside the tightly-coupled memories of the platform. For this reason, encryption and authentication are performed by splitting the bitstream into 8 MB blocks, which are loaded from the external DDR memory [30].

VII. USE CASE: MIXED-CRITICALITY HARDWARE ACCELERATORS FOR A RAILWAY SYSTEM

Diverse applications can coexist on heterogeneous SoCs, such as the UltraScale+, each with its own safety and security level, provided that applications with higher criticality or security levels are not adversely impacted by those with lower levels. This section presents an industrial-grade use case of this kind from the railway domain, which includes three HDs:

- A real-time, safety-critical, and secure (RTSC) HD, characterized by a set of tasks running on FreeRTOS on the Real-time Processing Unit (RPU) of the UltraScale+ (2xCortex-R5).
- A virtual machine VM_0 HD that utilizes an AI-based image processing algorithm for a driver alert system [31] (mid criticality), running on 2 cores of the Application Processing Unit (APU) of the UltraScale+ (Cortex-A53).
- A virtual machine VM_1 HD that leverages AI-based image processing to gather and analyze data about rail conditions to support maintenance decisions [32] (low criticality), running on other 2 cores of the APU.

The system is designed as a *2-out-of-2* redundant architecture with two replicas, a configuration commonly used in the railway domain for fault detection. Fault detection is implemented using a watchdog unit periodically refreshed by the replicas. When the voting differs, the watchdog stops actuation and transitions into the fail-safe state [33]. In case of matching votes, one of the two replicas is configured to apply the actuation. In our work, one UltraScale+ MPSoC platform implements a replica (see Fig. 6). The two replicas are connected via two wired *inter-replica communication interfaces*: one for sending and one for receiving the data which should be *voted* using a consensus voting protocol. Only *vital outputs*, i.e., critical data handled by the RTSC HD, undergo the voting protocol. The PL of each replica includes several communication interfaces associated with components belonging to the RTSC HD, which are discussed next.

SPI Interfaces. Two instances of the Serial Peripheral Interface (SPI) protocol are used for implementing distributed replica voting. Both interfaces operate at 100 MHz:

- The first interface, `SPI_Send_Vote`, is responsible for transmitting the data to be voted on.
- The second interface, `SPI_Recv_Vote`, receives the data to be voted on from the other replica.

These separate interfaces are essential to prevent contention between sending and receiving tasks.

CAN Interfaces. Two Controller Area Network (CAN) interfaces operate at 10 MHz and are used for communication with the braking and traction control systems: `CAN_Brake` for the braking system; `CAN_Traction` for traction control.

UART Units. Two Universal Asynchronous Receiver-Transmitter (UART) units are included, each connected to a display to show system status: namely UART_Status0 and UART_Status1.

AXI CDMA Units. Each SPI, CAN, and UART unit is paired with an AXI Central Direct Memory Access (CDMA) IP [34] running at 200MHz, which facilitates efficient data transfer from the PL devices to the DDR memory used by the tasks.

Each HA in the RTSC domain is managed by a dedicated task running on FreeRTOS. The TrustZone security level for the protocol interfaces is set to secure, ensuring access to secure memory regions through a secure translation regime on the SMMU. The QoS values are adjusted based on the criticality associated with each task. QoS values between 12 and 15 are ideal for low-latency applications [35]. The caching for the HAs is set to 0, as the AxCACHE signal dictates coherency on the APU caches, while the tasks run on the RPU processors. This is crucial to avoid potential data leaks.

Additionally, the PL of each replica hosts two AMD DPUs, which are soft cores specifically designed to accelerate *convolutional neural networks* [36]. DPUs are well-suited for embedded AI applications due to their scalability in both resource and power consumption [37]. The two DPUs, referred to as DPU0 and DPU1, are each equipped with two data interfaces (DATA0 and DATA1) and an instruction fetch interface (IF). The APU processors host CLARE-Hypervisor [38], which manages two VMs, each running PetaLinux, a specialized Linux distribution for AMD SoCs. Each VM can access one of the two DPUs implemented in the PL, which are used to perform AI-enabled image classification. In both VM_0 and VM_1, all transactions are marked as non-secure. The AxQoS priority values are set to 0 to minimize interference with the RTSC HD and the APU processors. The AxCACHE values are not enforced, allowing the DPUs to optimize their transactions through caching as needed. The configuration of the HD elements is summarized in Table IV. It is important to recall that the Ultrascale+'s SMMU can only apply virtualization to high-performance (HP0, HP1, HP2, HP3) and high-performance coherent (HPC0, HPC1) ports. Due to the limited availability of PL-PS port interfaces, HAs from different hardware domains HDs need to share a port.

The design flow proposed in this work can hence be used to guarantee isolation between the HDs. The PL design was verified using our tool, proving its effectiveness in a realistic use case. AXI Enforcer and AIM were used to dedicate unique sets of AXI IDs to the HAs, resulting in unique Stream IDs in the PS. The SMMU is configured to apply separate translation regimes for each HA. The software partitions are encrypted and authenticated using the secure boot feature of the UltraScale+ platform, which is configured according to the scheme outlined in Section VI.

VIII. EXPERIMENTAL EVALUATION

This section presents an experimental evaluation of the design flow implemented on an AMD/Xilinx Kria KR260 board based on the use case in Sec. VII. The proposed design

TABLE IV
RAILWAY SYSTEM HD CONFIGURATION

HD	PL-PS Port	HA	AxPROT	AxQOS	AxCACHE
RTSC	HP0	SPI_Recv_Vote	S	15	0
	HP0	SPI_Send_Vote	S	15	0
	HP1	CAN_Brake	S	15	0
	HP1	CAN_Traction	S	14	0
	HP2	UART_Status0	S	13	0
	HP2	UART_Status1	S	13	0
VM_0	HP3	DPU0.DATA_0	NS	0	-
	HPC0	DPU0.DATA_1	NS	0	-
	HPC1	DPU0.IF	NS	0	-
VM_1	HP3	DPU1.DATA_0	NS	0	-
	HPC0	DPU1.DATA_1	NS	0	-
	HPC1	DPU1.IF	NS	0	-

-.: not configured, S: TrustZone Secure, NS: TrustZone Non-Secure.

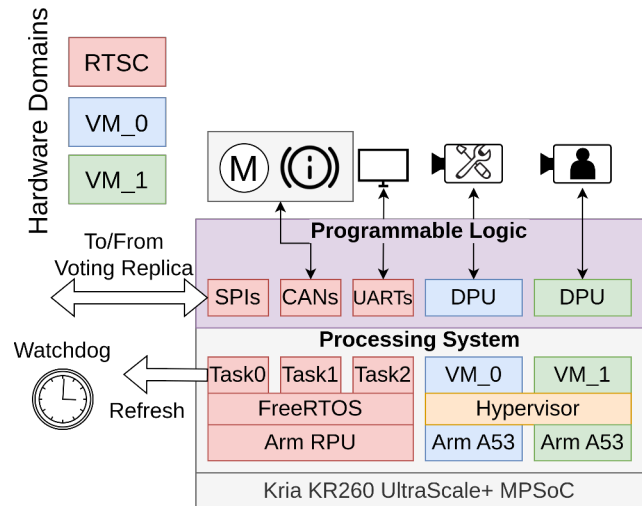


Fig. 6. Block diagram of one replica of the 2-out-of-2 architecture of our railway case study based on AMD/Xilinx Kria KR260 boards.

flow introduces latency, power consumption, and area utilization overhead. The AXI Enforcer is a simple combinatorial logic IP that leaves all AXI signals unchanged except for AxUSER, AxPROT, AxQOS, and AxCACHE, which are set to fixed values. The Vivado synthesizer identifies that the high-level description (HDL) of the IP consists of only assigning constant values to specific signals. Consequently, in the gate-level netlist, this IP is synthesized as a set of connections to constants. As such, in the bitstream, this IP does not utilize any area or impact the resource usage of the PL. When either subordinate interface sets one of the xREADY signals to 0, the AIM buffers the transactions issued by managers. The AIM can be configured with different request, burst, and response buffer sizes. As anticipated in Section V-A2, increasing the sizes of these buffers helps minimize the idle time induced by

additional xREADY handshakes. The next section analyzes the performance impact of the AIM for increasing buffer sizes. It is important to note that, since the SMMU translation regimes are statically configured, translation processing has minimal overhead. This is because the TBU units efficiently cache the static translations. In the following evaluations, the overhead associated with SMMU translation is included.

A. Performance Evaluation of the VM DPUs: Inference Time Comparison on Image Datasets

Each DPU was tested using a ResNet50 model to perform inference on a dataset [39] consisting of 500 images, each sized at 224x224 pixels. The software process that controls the DPU runs at maximum priority on a single CPU. The overhead of the design flow is assessed by comparison with the base case, i.e., the PL design without the AXI Enforcer and AIM units. The DPU includes three manager interfaces, each connected to the chain of IPs specified in the design flow: an AXI Enforcer, a SmartConnect, and an AIM. Each AIM was tested with progressively larger buffer sizes to evaluate their impact on inference time. Buffer sizes were increased until xREADY handshakes ceased on both read and write channels. This was verified using an Integrated Logic Analyzer (ILA) IP core [40]. For the read response channel and all the write channels of the DPU interfaces, the required buffer size to stop xREADY handshakes, and achieve full bandwidth, is 2. The following list provides the parameter values for each AIM unit connected to the DPU interfaces, using the format *AIM_DPU-Interface.Parameter*:

- `AIM_{DATA0-1, IF}.WRITE_REQ_BUF_SIZE: 2`
- `AIM_{DATA0-1, IF}.WRITE_BURST_BUF_SIZE: 2`
- `AIM_{DATA0-1, IF}.WRITE_RSP_BUF_SIZE: 2`
- `AIM_IF.READ_BURST_BUF_SIZE: 2`

The read burst channels of the DATA0 and DATA1 interfaces required larger buffer sizes on the connected AIM units to prevent xREADY handshakes. This was expected, as the read channels generate more transactions to read the parameter of the AI model and the input image from DDR memory, whereas write channels are used to write the inference results only. Starting from the above configuration of buffer sizes, Table V shows the results when increasing the value of `AIM_DATA0-1.READ_BURST_BUF_SIZE`. The last line represents the configuration parameters to get no xREADY handshakes on both channels. As it can be noted from the table, *the impact of the AIM is very marginal and not particularly affected by the buffer size*.

B. Performance Evaluation for the RTCS HD

All HAs within the RTCS HD rely on a CDMA unit configured with a 32-bit data width. The CDMA provides a total bandwidth, B_{CDMA} , calculated as:

$$B_{CDMA} = 32 \text{ bits} \times 200 \text{ MHz} = 6400 \text{ Mbps}. \quad (1)$$

Since SPI requires the highest data bandwidth among the protocols (compared to UART and CAN), it becomes the primary focus for our bandwidth analysis. SPI operates at a

TABLE V
AVERAGE INFERENCE TIME FOR DIFFERENT READ BURST BUFFER SIZES ON DATA0-1 DPU INTERFACES

DATA0 read buffer	DATA1 read buffer	Average inference time (ms)	Standard deviation
2	2	62.39	0.01422
3	3	62.37	0.01336
4	4	62.38	0.01253
8	8	62.37	0.01227
24	24	62.38	0.02146
24	48	62.37	0.02041
Base Case		62.08	0.01373

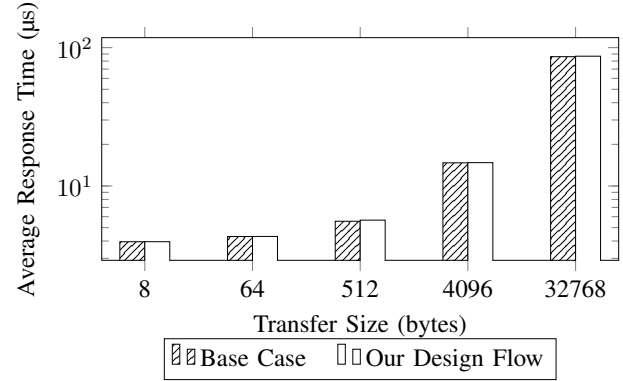


Fig. 7. Average response time for different CDMA transfer sizes

frequency of 100 MHz with a 16-bit data width, yielding a total bandwidth, B_{SPI} , given by:

$$B_{SPI} = 16 \text{ bits} \times 100 \text{ MHz} = 1600 \text{ Mbps}. \quad (2)$$

When implementing the proposed design flow, a delay on the CDMA transfers is introduced. Hence, we must verify that this delay does not heavily affect the communication with the protocol interfaces. To assess the impact on the CDMA units, we measured the overhead introduced across various transfer sizes. The average time was calculated on 10,000 trials. The results are shown in Fig. 7. The maximum response time for the same experiment is reported in Table VI. These experiments indicate an overhead on the transfer times in the order of microseconds. The bandwidth utilization was calculated using different transfer sizes. The results of this experiment are presented in Table VII. Analysis of the xREADY signals using the ILA core showed that they remained high (i.e., they did not transition to 0) even with the largest buffer sizes. The design flow achieved a bandwidth exceeding 3000 Mbps, demonstrating its ability to keep pace with the requirements of the protocol interfaces. To conclude, the observed overheads are minimal and do not have a significant impact on the timing behavior of the tasks.

C. Impact of AIM's Buffer Sizes

The above experiments show that increasing AIM buffer sizes does not significantly enhance performance, as most overhead comes from the AIM adding two clock cycles of

TABLE VI
MAXIMUM RESPONSE TIME FOR DIFFERENT CDMA TRANSFER SIZES

Transfer size (bytes)	Base case (μ s)	Design flow (μ s)
8	4.35	4.37
64	4.56	4.64
512	5.84	6.03
4096	14.95	15.13
32768	86.70	87.28

TABLE VII
BANDWIDTH COMPARISON IN MBPS (BUFFER SIZE 2)

Transfer size (Bytes)	Base case bandwidth (Mbps)	Design flow bandwidth (Mbps)
8	17.33	14.84
64	118.64	118.56
512	1131.12	1129.42
4096	1412.28	1410.78
32768	3019.69	3017.34

TABLE VIII
AREA UTILIZATION AND POWER WITH INCREASING WRITE REQUEST BUFFER SIZES

Buffer Size	CLB LUTs	CLB Reg	CARRY8	CLB	Power (W)
2	1819	1467	66	369	0.011
4	2064	1467	66	405	0.013
8	2362	1881	66	472	0.015
16	3806	2438	66	680	0.022
32	5200	3543	66	680	0.022
64	12239	5759	66	2093	0.052

latency for the first request or burst data, and one clock cycle for subsequent requests. This latency impacts communication between the HA and the PL-PS interface. The minimal transfer time improvements using larger buffer sizes are not worth the increased area and power usage. Table VIII shows that AIM area and power consumption increase with larger buffer sizes, using 40-bit addresses and 128-bit data widths. Parameters like NUMBER_OF MANAGERS and POOL_SIZE also affect area utilization, as shown in Table VI, which considers a fixed buffer size of 2.

The experiments demonstrate the feasibility of the proposed design flow in the use case. In the RTSC domain hosted on the RPU, unforeseen delays can jeopardize the mission of the system. However, the recorded overhead is in the order of microseconds and it can be accounted for at the stage of schedulability analysis to prevent deadline misses. The VMs running on the APU, consisting of two separate domains, do not execute safety-critical applications, and the increased latency only slightly affects the inference time.

IX. CONCLUSION

Heterogeneous embedded systems offer a wide range of isolation features that enable the deployment of applications with varying security and criticality levels. In this framework, multiple HDs, each with its own set of computing resources, can be defined to support these distinct application needs.

TABLE IX
AREA UTILIZATION AND POWER WITH VARIOUS MANAGER NUMBER AND POOL SIZE COMBINATIONS

# Managers	Pool Size	CLB LUTs	CLB Reg	CARRY8	CLB	DSPs	Power (W)
2	1	1799	1449	66	353	0	0.011
4	2	1818	1467	66	367	0	0.011
8	3	1990	1471	68	398	0	0.012
16	4	1897	1471	66	392	2	0.011

This paper addressed the problem of resource isolation when incorporating PL-based HAs into HDs. The proposed design flow, based on two novel AXI IP cores (AIM and AXI Enforcer) enforces security at the PL level using IOMMU virtualization features. This solution effectively addresses the challenge of preserving isolation when PL HAs share a PL-PS interface, overcoming the limitation of assigning a single HA per port. It also addressed the challenge of explicitly managing critical AXI bus signals, such as the TrustZone security level, QoS, and cache control settings. The proposed solution follows a *secure-by-design* approach, where the isolation of the HDs is guaranteed at design time through the use of a verification tool and secure boot. The feasibility of the design flow was validated through an industrial-grade use-case of a railway system. The experiments demonstrate that the overhead introduced by the proposed approach is negligible, ensuring no impact on the overall system performance and on the timing behavior of the real-time tasks. Future work should address the extension of the design flow to support partial reconfiguration capabilities. This would require a trusted software component to manage the reconfiguration of AIM/Enforcer settings for the reconfigured PL region. Additionally, the IPs would need to be extended with memory-mapped interfaces to access a configuration address space. The source code related to this project is publicly available on GitHub [41].

ACKNOWLEDGMENT

This work was partially supported by the Project SERICS under the Ministry of University and Research (MUR) National Recovery and Resilience Plan funded by the European Union-NextGenerationEU under Grant PE00000014, the PRIN project RETICULATE (202249HCFS), and the PRIN-PNRR project OPERAND (P202284ZAT).

REFERENCES

- [1] ARM, "Learn the architecture - an introduction to amba axi," 2022. [Online]. Available: <https://developer.arm.com/documentation/102202/latest/>
- [2] M. Gross, N. Jacob, A. Zankl, and G. Sigl, "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*, ser. ASHES'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3–12. [Online]. Available: <https://doi.org/10.1145/3338508.3359568>
- [3] S. Chaudhuri, "A security vulnerability analysis of socfpga architectures," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3195970.3195979>

- [4] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [5] N. Jacob, C. Rolfes, A. Zankl, J. Heyszl, and G. Sigl, "Compromising fpga socs using malicious hardware blocks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, 2017, pp. 1122–1127.
- [6] L. E. Olson, J. Power, M. D. Hill, and D. A. Wood, "Border control: Sandboxing accelerators," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 470–481.
- [7] F. Alam, H. Lee, A. Bhattacharjee, and A. Awad, "Cryptommu: Enabling scalable and secure access control of third-party accelerators," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 32–48.
- [8] K. D. Pham, K. Paraskevas, A. Vaishnav, A. Attwood, M. Vesper, and D. Koch, "Zucl 2.0: Virtualised memory and communication for zynq ultrascale+ fpgas," in *Sixth International Workshop on FPGAs for Software Programmers*. Berlin, Germany: VDE VERLAG GMBH, 2019. [Online]. Available: <https://ieeexplore.ieee.org/document/8891818>
- [9] E. Karabulut, A. Awad, and A. Aysu, "Ss-axi: Secure and safe access control mechanism for multi-tenant cloud fpgas," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2023, pp. 1–5.
- [10] S. Donchez and X. Wang, "Memory isolation for multi-tenant data integrity in cloud mpsoe fpgas," in *2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2022, pp. 0515–0521.
- [11] 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)*, 2023, pp. 1–6.
- [12] M. Armanuzzaman, A.-R. Sadeghi, and Z. Zhao, "Building your own trusted execution environments using fpga," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1584–1599. [Online]. Available: <https://doi.org/10.1145/3634737.3637644>
- [13] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016. [Online]. Available: <https://api.semanticscholar.org/CorpusID:28642809>
- [14] A. Sev-Snp, "Strengthening vm isolation with integrity protection and more," *White Paper, January*, vol. 53, no. 2020, pp. 1450–1465, 2020.
- [15] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "Trustzone explained: Architectural features and use cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, 2016, pp. 445–451.
- [16] W. D. Mark Randolph, "Power side-channel attack analysis: A review of 20 years of study for the layman," *Cryptography*, 2020. [Online]. Available: <https://www.mdpi.com/2410-387X/4/2/15>
- [17] U. Ali, H. Omar, C. Ma, V. Garg, and O. Khan, "Hardware root-of-trust implementations in trusted execution environments," *Cryptology ePrint Archive*, Paper 2023/251, 2023. [Online]. Available: <https://eprint.iacr.org/2023/251>
- [18] W. Ren, W. Kozłowski, S. Koteshwara, M. Ye, H. Franke, and D. Chen, "Accshield: a new trusted execution environment with machine-learning accelerators," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
- [19] E. M. Benhani, L. Bossuet, and A. Aubert, "The security of arm trustzone in a fpga-based soc," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1238–1248, 2019.
- [20] I. P. Gouveia, M. Völpl, and P. Esteves-Verissimo, "Behind the last line of defense: Surviving soc faults and intrusions," *Computers & Security*, vol. 123, p. 102920, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404822003121>
- [21] N. Asmussen, M. Völpl, B. Nöthen, H. Härtig, and G. Fettweis, "M3: A hardware/operating-system co-design to tame heterogeneous many-cores," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 189–203.
- [22] Xilinx, "Memory and peripheral protection unit for pl isolation in zynq ultrascale+ devices," 2022. [Online]. Available: <https://docs.xilinx.com/t/en-US/xapp1353-pl-isolation>
- [23] W. Ren, J. Pan, and D. Chen, "Accguard: Secure and trusted computation on remote fpga accelerators," in *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2021, pp. 378–383.
- [24] D. Korolija, T. Roscoe, and G. Alonso, "Do OS abstractions make sense on FPGAs?" in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 991–1010. [Online]. Available: <https://www.usenix.org/conference/osdi20/presentation/roscoe>
- [25] N. Asmussen, S. Haas, A. Lackorzyński, and M. Roitzsch, "Core-local reasoning and predictable cross-core communication with m3," in *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024, pp. 199–211.
- [26] A. Kurth, W. Rönninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini, "An open-source platform for high-performance non-coherent on-chip communication," *IEEE Transactions on Computers*, vol. 71, no. 8, pp. 1794–1809, 2022.
- [27] L. Bossuet and E. M. Benhani, "Performing cache timing attacks from the reconfigurable part of a heterogeneous soc—an experimental study," *Applied Sciences*, vol. 11, no. 14, 2021. [Online]. Available: <https://www.mdpi.com/2076-3417/11/14/6662>
- [28] Xilinx, *SmartConnect v1.0 LogiCORE IP Product Guide*, Xilinx, 2018, last accessed: 3 Jan, 2024. [Online]. Available: <https://docs.xilinx.com/t/en-US/pg247-smartconnect>
- [29] Xilinx, "Hardware root of trust secure boot details," 2023. [Online]. Available: <https://docs.amd.com/t/en-US/ug1085-zynq-ultrascale-trm/Hardware-Root-Of-Trust-Secure-Boot-Details>
- [30] AMD, *Bootgen User Guide*, AMD, Inc., May 2024, version 1.0, pp. 93–94. [Online]. Available: https://docs.amd.com/viewer/book-attachment/3507hly7pahNVfn7_uSS_Q/kzDPGjPouYQx7CB_ak5uPA
- [31] J. Chen, H. Li, L. Han, J. Wu, A. Azam, and Z. Zhang, "Driver vigilance detection for high-speed rail using fusion of multiple physiological signals and deep learning," *Applied Soft Computing*, vol. 123, p. 108982, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1568494622003143>
- [32] G. D'Amico, M. Marinoni, F. Nesti, G. Rossolini, G. Buttazzo, S. Sabina, and G. Lauro, "Trainsim: A railway simulation framework for lidar and camera dataset generation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 12, pp. 15 006–15 017, 2023.
- [33] P. Fara, G. Serra, A. Biondi, and C. Donnarumma, "Scheduling Replica Voting in Fixed-Priority Real-Time Systems," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 13:1–13:21. [Online]. Available: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2021.13>
- [34] AMD, *AXI Central Direct Memory Access LogiCORE IP v4.1 Product Guide*, AMD, Inc., May 2022, version 4.1. [Online]. Available: <https://docs.amd.com/t/en-US/pg034-axi-cdma>
- [35] Xilinx, "Zynq ultrascale+ device technical reference manual-ug1085." [Online]. Available: <https://docs.amd.com/t/en-US/ug1085-zynq-ultrascale-trm/QoS-Priority>
- [36] Z. Du, W. Zhang, Z. Zhou, Z. Shao, and L. Ju, "Accelerating dnn inference with heterogeneous multi-dpu engines," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
- [37] J. Vandendriessche, B. Da Silva, and A. Touhafi, "Frequency evaluation of the xilinx dpu towards energy efficiency," in *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, 2022, pp. 1–6.
- [38] Accelerat Srl, "The CLARE Software Stack." [Online]. Available: <https://accelerat.eu/clare>
- [39] "Common objects day and night (codan) image classification dataset for zero-shot day-night domain." [Online]. Available: <https://github.com/Attila94/CODaN>
- [40] AMD, *Integrated Logic Analyzer v6.2 Product Guide (PG172)*, AMD, Inc., Oct. 2016, version 6.2. [Online]. Available: <https://docs.amd.com/v/u/en-US/pg172-ila>
- [41] N. Salami, "Zynqmp secure fpga isolation," <https://github.com/NikoSalami/zynqmp-secure-fpga-isolation>, 2025.