IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 December 2023, accepted 29 December 2023, date of publication 10 January 2024, date of current version 22 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3352114

RESEARCH ARTICLE

"Interrupting" the Status Quo: A First Glance at the RISC-V Advanced Interrupt Architecture (AIA)

FRANCISCO MARQUES, MANUEL RODRÍGUEZ, BRUNO SÁ, AND SANDRO PINTO
Centro ALGORITMI/LASI, Universidade do Minho, 4800-058 Guimarães, Portugal

Corresponding author: Francisco Marques (fmarques_00@protonmail.com)

This work was supported in part by the Secure Systems Research Center (SSRC) - Technology Innovation Institute (TII); in part by Zero-Day Labs; in part by FCT—Fundação para a Ciência e Tecnologia within the Research and Development Units Projec Scope under Grant UIDB/00319/2020; and in part by the Scholarships Project Scope under Grant SFRH/BD/07707/2021.

ABSTRACT Interrupt controllers are a crucial component in computing platforms. From cloud computing to embedded systems, interrupts and respective controllers enable more efficient management and operation of a platform's resources. Modern computer architectures incorporate hardware interrupt controllers (e.g., Arm Generic Interrupt Controller (GIC)) that are well-established in modern processors and system-on-chips (SoCs). This article describes our work and research on developing the first open-source RISC-V Advanced Interrupt Architecture (AIA) IP compliant with the recently ratified specification (v1.0). Our contribution is multifold and encompasses architecture, microarchitecture, and evaluation. In particular, we explored alternative designs and microarchitectural enhancements for the implemented IP to cope with mixed-criticality systems (MCS) requirements (e.g., real-time and predictability). From this exploration, we highlight the proposed Integrated Embedded AIA (IE-AIA) design. For each configuration, we assess the impact on hardware utilization and interrupt latency. Due to the increased proliferation of virtualization in MCS, we measured the interrupt latency for a system configuration built atop the Bao hypervisor. At the macro level (i.e., considering both hardware and software), and in particular for a full-blown virtualization stack, we observed a reduction of ∼99.5% in the average interrupt latency when comparing the Platform-Level Interrupt Controller (PLIC) IP (full trap and emulation) with the standard AIA IP (no hypervisor mediation due to IMSIC direct interrupt injection). For the IE-AIA, our evaluation focused on the micro view (i.e., hardware only), where we observed that under interference, the IE-AIA IP shows a reduction of $\sim 7\times$ in the average interrupt latency and deterministic behavior compared to the standard AIA implementation. We also provide the first empirical-based comparison between the RISC-V PLIC and the RISC-V AIA. Finally, all artifacts described in this article are open source to foster collaboration and further explore additional design configurations.

INDEX TERMS Advanced interrupt architecture, AIA, RISC-V, virtualization, interrupt controller, CVA6, microarchitecture, architecture, FPGA.

I. INTRODUCTION

In the realm of computer architectures, interrupts and respective hardware controllers play a pivotal role in guaranteeing efficient operations and optimal resource management. Interrupts refer to signals generated by various system-on-chip (SoC) components (e.g., DMA, timer, UART, USB) that require the central processing unit (CPU) attention. The responsibility of interrupt controllers is to manage these interrupts, which can be generated by peripherals and processors, and assign them to a specific CPU [1], [2], [3]. As a result, dedicated modules can handle particular tasks, freeing the CPU to perform other operations [4].

Modern computing systems implement advanced hardware interrupt controllers, including the Arm Generic Interrupt Controller (GIC) [5] and the Intel Advanced Programmable

The associate editor coordinating the review of this manuscript and approving it for publication was Tomas F. Pena.

Interrupt Controller (APIC) [6]. Over the years, these interrupt controllers have been evolving to cope with the ever-growing number of features and requirements [7], [8]. For example, both the APIC and GIC (since version 2) support Message Signalled Interrupts (MSIs) which are key for standards such as Peripheral Component Interconnect Express (PCIe). Also, with the increasing adoption of virtualization technology [9], mainstream interrupt controllers such as the APIC and GICv3/GICv4 started to add virtualization support.

RISC-V sets itself apart from the established classical mainstream Instruction Set Architectures (ISAs) by providing an open standard ISA. This unique ISA incorporates a modular and adaptable extension mechanism, enabling it to scale seamlessly from compact microcontrollers to supercomputers [10], [11], [12]. The RISC-V Platform-Level Interrupt Controller (PLIC) [13] was the first (standard) interrupt controller in RISC-V systems. However, its specification posed scalability and feature richness limitations, including a lack of support for message-signaled interrupts (MSIs) and virtualization. The RISC-V community has then been focusing on devising a new interrupt controller specification. The RISC-V Advanced Interrupt Architecture (AIA) [14] is the state-of-the-art reference specification for interrupt-handling functionality. This recently ratified specification is paving the way for integration into modern and next-generation RISC-V SoC blueprints. To the best of our knowledge, while some proprietary implementations of the RISC-V AIA (e.g., SiFive, Ventana, MIPS) have been designed, no functional open-source implementation has been found at the time of writing. Furthermore, in the context of the RISC-V AIA, no existing work has (i) performed a design exploration, in particular, driven by mixed-criticality systems (MCS) requirements and (ii) an empirical evaluation in terms of hardware costs and interrupt latency, focusing on predictability under interference in virtualization setups.

In this article, we describe our research on the RISC-V AIA. To carry out this study, we first developed a parametrizable Advanced Platform-Level Interrupt Controller (APLIC) IP and an Incoming Message-Signaled Interrupt Controller (IMSIC) IP - which together form the AIA IP - fully compliant with the RISC-V AIA specification v1.0. These IPs were integrated into a CVA6-based SoC with virtualization support [15]. Then, we explore alternative designs and microarchitectural enhancements for the implemented IP to cope with the MCS requirements (e.g., real-time and predictability). This research culminates into three new design options (IPs): (i) APLIC Minimal, (ii) IMSIC Island, and (iii) Integrated Embedded AIA (IE-AIA). The first two IPs are intended to reduce the hardware resources of the standard AIA IP implementation. The IE-AIA aims to provide deterministic interrupt latency in multi-core system configurations (and in the presence of other bus masters connected through the main system bus). Then, we perform an empirical evaluation (hardware resources and interrupt latency) of different AIA hardware configurations and microarchitectures, with a focus on modern embedded and MCS requirements (e.g., real-time, predictability) [9], [16]. The experiments carried out on a Genesys 2 FPGA board, show an improvement in the average interrupt latency of ∼99.5% between the AIA/IE-AIA IP and the PLIC. When comparing the AIA IP with the IE-AIA at the micro level (i.e., hardware only), results show a reduction of the interrupt latency worst-case execution time (WCET) by 96.43% while providing deterministic execution. We made our hardware design and RTL implementation open source[1] to (1) encourage the RISC-V AIA adoption among the RISC-V community (both academia and industry) and (2) foster research to validate and (potentially) improve the AIA specification in different use cases.

In summary, with this work, we make four major contributions. (1) We provide the first open-source RISC-V AIA IP, fully compliant with the RISC-V AIA specification v1.0 (Section III). (2) We explore alternative designs and microarchitectural enhancements targeting MCS requirements (Section IV). (3) We carry out an empirical evaluation of different AIA hardware configurations and microarchitectures, providing insights concerning hardware costs and interrupt latencies introduced in a virtualized system (Section V). (4) Lastly, based on the knowledge and experience gained while conducting this research, we discuss a set of alternative designs and architectural/microarchitectural changes to further optimize and enhance our RISC-V AIA IP in particular and the specification, in general (Section VI).

II. BACKGROUND

A. RISC-V PRIVILEGED ISA AND HYPERVISOR EXTENSION

The RISC-V privileged ISA [17] organizes its execution model into three distinct privilege levels: (i) *machine mode* (M-mode), the highest privileged level, executes software implementing the supervisor binary interface (SBI); (ii) *supervisor mode* (S-Mode) runs Unix type operating systems (OSes); and (iii) *user mode* (U-Mode) executes user applications.

The RISC-V Privileged Architecture specifies that a machine with multiple harts must provide (for each hart) an implementation-defined memory address that can be written to signal a machine-level software interrupt (major code 3) at that hart. Machine-level IPIs can consequently be sent to any hart in the form of machine-level software interrupts.

The Hypervisor extension is a RISC-V privileged architecture extension to support virtualization in RISC-V platforms efficiently. This extension restructures the execution model. The S-Mode is transformed into an extended *hypervisor mode* (HS-mode), accommodating both type-1 and type-2 hypervisors. Additionally, it introduces two new privileged modes, enabling the execution of guest OS in *virtual supervisor mode* (VS-mode) and *virtual user mode* (VU-mode).

[1] https://github.com/zero-day-labs/riscv-aia

B. RISC-V PLIC

The Platform-Level Interrupt Controller (PLIC) was the first interrupt controller available for RISC-V architectures, offering a naive solution for interrupt management. However, the PLIC specification presents several limitations in terms of scalability and feature-richness.

First, memory-mapped registers require a large part of the physical address space. Second, the global configuration registers are shared across two privilege rings: M-mode and S-mode. Third, the PLIC lacks support for MSIs, which are interrupts implemented as write operations to special interrupt controller registers and propagated through the system interconnect. This deficiency limits the flexibility of the interrupt controller signaling mechanisms. Fourth, the PLIC does not provide the capability to change the Interrupt Request (IRQ) line sensing configuration at run-time. Finally, the PLIC also lacks virtualization support, resulting in hypervisors needing to rely on techniques like trap-and-emulate, increasing the interrupt delivery latency to Virtual Machines (VMs) [16].

C. RISC-V AIA: AN OVERVIEW

In response to the well-known PLIC limitations, the RISC-V community developed a new interrupt controller specification. The RISC-V AIA is the novel reference specification for interrupt-handling functionality.

The AIA consists of (i) an extension to the RISC-V privilege ISA [17], i.e., Smaia/Ssaia, (ii) two standard interrupt controllers, i.e., the APLIC and the IMSIC, and (iii) a set of requirements for other system components (e.g., I/O memory management unit (IOMMU)). Furthermore, the protection against undesirable accesses is guaranteed at the core level, via the physical memory protection PMP [17], and at the system level via either the I/O physical memory protection (IOPMP) [18], and RISC-V IOMMU [19] (or any other customized system-level memory protection controller), specifying rules with devices (or groups of devices), and access types that should be blocked. Some rules may be hard-coded in the microarchitecture, but usually, they are configured in software. The PMP and the IOPMP also have a lock feature, i.e., they can ignore modifications to a rule or set of rules. Once locked, the (IO)PMP entries remain locked until the system is reset. Figure 1 shows an SoC configuration that implements the AIA.

1) AIA PRIVILEGED EXTENSIONS

The AIA introduces i) a few Control and Status Registers (CSRs) to interface with the IMSIC IP, ii) a mechanism that permits the major interrupt priorities configuration and allows mixing major and minor interrupts, and iii) a set of registers to support HS/VS modes.

2) AIA IMSIC

The IMSIC architecture includes a collection of interrupt files that collectively provide the essential framework to facilitate the utilization of MSIs. Within a single interrupt file, interrupt priorities are determined directly from interrupt identity numbers, i.e., lower identity numbers have higher priority. At its core, each interrupt file constitutes a hardware module with two arrays: one to monitor pending interrupts and another responsible for their activation. The arrangement is such that for every privilege level and individual virtual hardware hart capable of receiving MSIs, there is a dedicated interrupt file within the hart's corresponding IMSIC. This mechanism is also used for the virtualization support. The number of guest interrupt files is exactly the number of supported guest external interrupts (GEILEN), as defined by the hypervisor extension. The GEILEN corresponds to the maximum number of active virtual harts (within a hart) that can receive interrupts directly. The IMSIC controller, per the specification, must be located in close proximity to the hart.

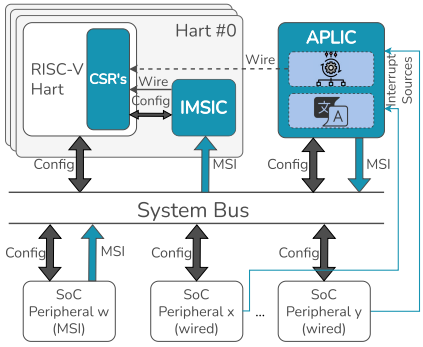

FIGURE 1. Simplified view of a generic multi-core SoC microarchitecture with AIA-related components (highlighted in blue).

3) AIA APLIC

The APLIC controller introduces two distinct delivery modes: (i) direct mode and (ii) MSI mode. In the direct mode, the APLIC has the same role as the pre-existing PLIC, i.e., effectively operating as an external interrupt controller. However, when in pair with the IMSIC(s), the APLIC acts as a translator of (physical) wired interrupts into MSIs. Subsequently, these MSIs are dispatched to the IMSIC for further processing. Regarding the APLIC internal structure, it encompasses a structured collection of interrupt domains. Each domain can be assigned for either M-mode or S-mode, i.e., guaranteeing that interrupt propagation is exclusive to a specific privilege level. The APLIC presents a more versatile interrupt priority configuration mechanism, allowing the software to define, for each interrupt source, a specific priority, with lower priority numbers having higher priority.

4) AIA IPIs

The AIA provides interprocessor interrupt (IPI) support via the IMSIC subsystem. Similarly to regular MSIs, an IPI can be sent to a hart by writing to the destination hart's IMSIC interrupt file (*seteipnum_[l/b]e*). Subsequently, when a RISC-V hart does not implement an IMSIC, it receives IPIs through the software major interrupt IRQ line. Within

our scope, heterogeneous platforms can be divided into i) RISC-V-only heterogeneous platforms, i.e., harts are implemented with different IPI mechanisms and ii) multi-ISA (e.g., RISC-V and Arm) heterogeneous platforms. RISC-V-only heterogeneous platforms may implement various harts with distinct use cases and requirements (e.g., real-time, general purpose). It is possible that different RISC-V harts implement IPIs via i) the IMSIC mechanism or ii) the RISC-V privileged architecture. In these heterogeneous systems, the privileged architecture's mechanism guarantees the compatibility between AIA and non-AIA subsystems. In multi-ISA platforms (e.g., AMD Versal Adaptive SoC), several subsystems support IPIs differently. The RISC-V AIA would be just another interrupt controller, as is the case with the Arm GIC. Consequently, sending IPIs between architecturally distinct subsystems depends exclusively on the platform's interrupt system that manages the various interrupt controllers.

D. CVA6

The CVA6 [20] is an application class RISC-V core that implements RV64 and RV32 RISC-V instruction sets. To support Unix-like OSes, this core fully supports the M/S/U privilege modes and provides hardware support for virtual memory by implementing a Memory Management Unit (MMU). Internally, the CVA6 microarchitecture encompasses a 6-stage pipeline, single issue, in-order CPU. The MMU has separate TLBs for data and instructions, and the page table walker implements the Sv39 and Sv32 translation modes. Recent enhancements to the CVA6 include an energy-efficient Vector Unit co-processor [21] and hardware virtualization support (a.k.a. RISC-V Hypervisor extension) [22].

III. RISC-V AIA: BUILDING AN IP

In this section, we discuss the design and implementation of the AIA IP, focusing on the implementation challenges and the main insights gained through the journey. Our goal is to complete an AIA implementation that is both modular and scalable, suitable for a wide range of market segments ranging from embedded and MCS to general-purpose platforms. The implementation of AIA IP was carried out in SystemVerilog.

A. AIA PRIVILEGED EXTENSIONS

The AIA specification adds a set of CSRs at the various execution levels of the RISC-V core (M, S, HS, and VS). To include the new CSRs, we modified the CVA6 CSR module. The CVA6 decoder module was also modified to support the new interrupt delivery scheme to the core. As depicted in Table 1, the registers related to configuring the major interrupt's priority (e.g., *mvip*, *hviprio1*) have been just partially implemented. Thus, setting major interrupts priority in software is not possible. The priority order is enforced by the default priorities table present in the specification. This decision aligns with the current upstream software support, i.e., openSBI or Linux does not currently support this feature.

TABLE 1. Current state of AIA CSRs/IMSIC/APLIC registers implemented in the CVA6: ● fully-implemented; ◐ partially implemented; ○ not implemented.

Module	Register	Status
RISC-V AIA Smaia/Ssaia	miselect/siselect/mireg/sireg	●
	mtopei/stopei/mtopi/stopi	●
	mvien/mvip	◐
	vsiselect/vsireg/vstopei/vstopi	●
	hvien/hvictl/hviprio1/hviprio2	◐
RISC-V AIA IMSIC	iprio	◐
	seteipnum_le	●
	seteipnum_be	○
	eidelivery/eithreshold/eip/eie	●
RISC-V AIA APLIC	domaincfg/sourcecfg	●
	mmsiaddrcfg/mmsiaddrcfgh	◐
	smsiaddrcfg/smsiaddrcfgh	◐
	setip/setipnum/in_clrip/clripnum	●
	setie/setienum/clrie/clrienum	●
	setipnum_le/genmsi/target	●
	setipnum_be	○

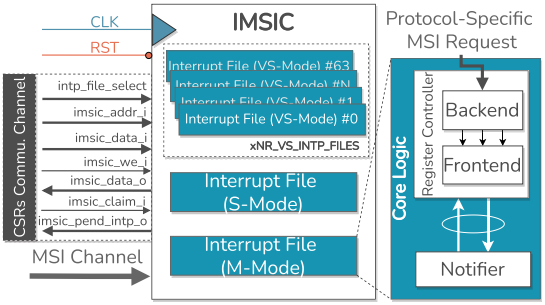

FIGURE 2. Simplified view of a generic IMSIC top-level and interrupt file design.

B. AIA IMSIC

Figure 2 depicts the developed IMSIC interface and composition design. Our design allows the IMSIC IP to have an arbitrary number of VS-files through RTL parameterization. The number of interrupt files implemented include: (i) at least 2 for M-mode and S-mode and (ii) a user-defined (at design time, by modifying a module parameter) number of VS-files, with a maximum of 63 for RV64 or 31 for RV32. The IMSIC IP was also designed to facilitate the protocol replacement for receiving MSIs on the system interconnect. These design choices minimize the requirement for extensive modifications to the RTL code, guaranteeing a smoother and more adaptable integration process of IMSIC into new platforms.[2]

1) INTERRUPT FILES

The IMSIC is composed of a set of interrupt files. Each interrupt file includes a register controller and a notifier, which together form the interrupt file core logic. For modularity, the IMSIC's interrupt file is generic and adaptable to each privilege mode, facilitating the IMSIC configurability.

[2]In the near future, we plan to develop a Python-based tool that utilizes templating to generate designs, offering increased flexibility and better integration with other protocols.

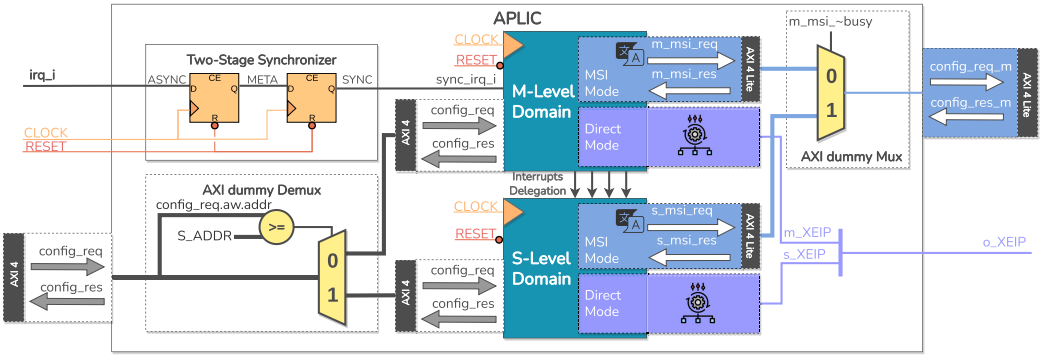

FIGURE 3. APLIC top-level module design with one M-level domain and one S-level domain.

2) CORE LOGIC: REGISTER CONTROLLER

The register controller is conceptually divided into a frontend and a backend module to isolate the IMSIC logic from a particular communication protocol. The register controller's backend is the protocol-dependent module. This module must translate a specific protocol request from the MSI channel into discriminated signals for the frontend. The frontend, in turn, is responsible for processing incoming MSIs, modifying the pending array, and sending it to the notifier module.

3) CORE LOGIC: NOTIFIER

The notifier module receives the IMSIC pending and enabled arrays and searches for the highest interrupt, notifying the hart if an interrupt is both pending and enabled. Since in the IMSIC interrupt sources, a higher priority corresponds to a lower interrupt ID, we implement the notifier module with a linear algorithm[3] that iterates over each interrupt source. The algorithm checks if an interrupt is both pending and enabled, stopping the search when it encounters the first interrupt available to be notified to the hart.

4) IMSIC INTERFACES

Figure 2 also shows the two IMSIC communication interfaces: (i) the CSRs configuration channel and (ii) the MSI reception channel. The CSRs configuration channel is a non-standard channel that enables the RISC-V hart to configure the IMSIC through CSR reads and writes. This channel provides a direct mechanism for the RISC-V hart to manage the IMSIC's operation. The second communication channel is responsible for receiving MSIs (and, therefore, IPIs). This channel i) is unidirectional, i.e., the MSI flows from the system interconnect to the IMSIC, and ii) does not include a handshake mechanism between agents, e.g., if hart 0 sends an IPI for hart 1, writing to the latter's IMSIC, hart 0 will never receive any confirmation if hart 1 received the IPI. However, as the interface is implemented with the AXI4 lite protocol, the MSI transaction uses the AXI handshake mechanism to guarantee that the MSI arrives successfully at the IMSIC. We implement this channel using the AXI4 lite protocol as proof of concept due to its widespread use and maturity in RISC-V platforms. It is, however, worth noting that the implementation is not dependent on this protocol and can be replaced by another without the rest of the IMSIC being modified.

[3]Given the IMSIC IP modular design, the internal logic (algorithm) of the notifier can be easily modified. We defer such exploration for future activities.

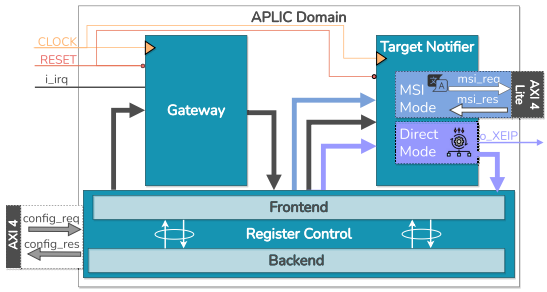

FIGURE 4. APLIC domain top-level design overview, comprised by a gateway, a register controller, and a target notifier. The target notifier only implements one of the coloured modules (light blue or purple), depending on the domain operation mode.

C. AIA APLIC

Figure 3 depicts the APLIC top-level module. As can be seen, the APLIC IP is the instantiation of a set of domains. In this implementation, we create an APLIC with two domains, i.e., M and S-mode. It is also visible in the figure the existence of a two-stage synchronizer. The synchronizer prevents metastability issues, preventing against data corruption or loss in the interrupt signal, i.e., providing stability and reliability of the interrupt source line connected to the root domain. To keep the APLIC interface simple, the APLIC is implemented with only one configuration port and, when supporting the MSI mode, with one MSI channel. Thus, in the top-level module, it is necessary to use a demultiplexer to correctly map a configuration request to its domain and a multiplexer to forward MSIs from the correct domain.

Figure 4 depicts an APLIC domain containing three modules: (i) gateway, (ii) notifier, and (iii) register control.

1) REGISTER CONTROLLER

Similar to the IMSIC implementation, the APLIC register controller is also divided into two modules, i.e., frontend and backend, to facilitate the adaptability of the IP with various

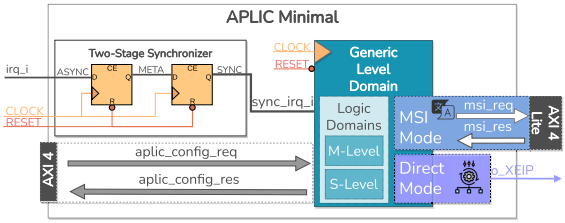

FIGURE 5. APLIC minimal top-level module design with one generic physical domain.

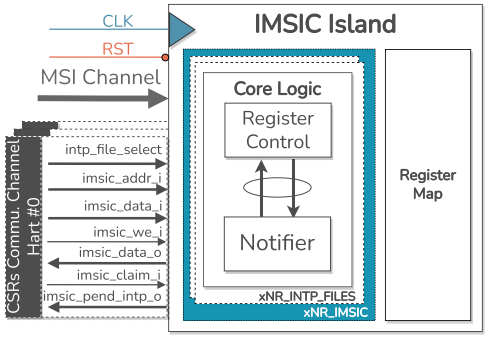

FIGURE 6. IMSIC Island top-level module design.

communication protocols (Figure 4). The backend depends on the protocol that connects the APLIC configuration interface with the system interconnect. The frontend is responsible for managing the APLIC registers independently of the protocol used in the backend. The backend module is implemented using an AXI4 lite protocol for the very same reasons advocated in the IMSIC (widespread use and maturity).

2) GATEWAY

The gateway module is responsible for receiving the interrupt source and its configuration and evaluating if an interrupt source can become pending. The gateway also implements the source mode for each interrupt source (i.e., level-sensitive high and low, edge-sensitive high and low, inactive, and detached). In the current implementation, to simplify APLIC's parameterization interface, the source mode is the same for all interrupts. The APLIC's line sensings to implement can be discovered by grouping the required source modes in each interrupt source. This design choice only impacts the hardware resource utilization. The capability to change the IRQ line sensing configuration at run-time remains unaffected. It is worth noting that in future iterations, the IP will be adapted to allow each interrupt to individually select the modes it wants to implement, making the IP suitable for application-centric platforms.

3) TARGET NOTIFIER

The notifier implementation depends on the APLIC delivery mode. If the APLIC is in direct mode, i.e., highlighted in purple in Figure 4, in that case, the notifier implements an algorithm to discover the highest pending and enabled interrupt, notifying the hart. If the APLIC is in MSI mode, represented in light blue in Figure 4, the notifier forwards newly pending and enabled interrupts to a hart's IMSIC. In the last, the notifier design is split into frontend and backend. Notwithstanding, the backend must provide the discriminated signals to the frontend. The frontend will then translate these signals into an actual protocol request, sending them through the system interconnect. The frontend module is implemented using an AXI4 lite protocol.

IV. RISC-V AIA: MICROARCHITECTURAL ENHANCEMENTS

In the following section, we propose a set of microarchitectural design enhancements that aim to address embedded and MCS requirements. The enhancements focus on optimizing hardware resources, minimizing interruption latency, and providing deterministic execution. This exploration led to the development of three IPs: (i) APLIC Minimal, (ii) IMSIC Island, and (iii) IE-AIA, which we describe next.

A. APLIC MINIMAL

In the APLIC IP described so far, we have implemented an internal physical instance per domain, i.e., as depicted in Figure 3, there is a physical M-level interrupt domain and a physical S-level interrupt domain (two independent internal hardware subsystem). However, we noted that in the overall APLIC behavior, only one domain is active at a given point in time. Thus, we optimized the APLIC IP to provide a single physical domain multiplexed among multiple logical domains, i.e., the same physical domain may multiplex, for example, M-level and S-level. This results in further hardware optimizations since registers like *sourcecfg*, *setip*, *setie*, and target are not duplicated for all physical domains. There is, however, a small additional logic that was added, which is related to the correct interrupt masking to the respective domain.

The APLIC minimal leverages the vanilla APLIC top-level module. As illustrated in Figure 5, we implemented only one physical domain, referred to as the generic level domain, that handles two logical domains (M-mode and S-mode). As a result of this optimization, we eliminate the AXI *mux* and *demux* modules present in the vanilla APLIC (Figure 3). We also modified the register controller frontend module, highlighted in Figure 4. It is important to note that the domain top-level, gateway, and target modules were kept almost identical (very few changes), as they already received the processed data from the register controller in the vanilla APLIC IP.

B. IMSIC ISLAND

The IMSIC IP presented in Section III is designed to be instantiated near the hart. So, it means that the number of IMSICs is proportional to the number of harts, and each IMSIC instantiation reflects into an extra connection to the system bus interconnect, directly increasing the hardware resources consumption. To mitigate this, we created a new IP that groups all the IMSICs inside a single logical module,

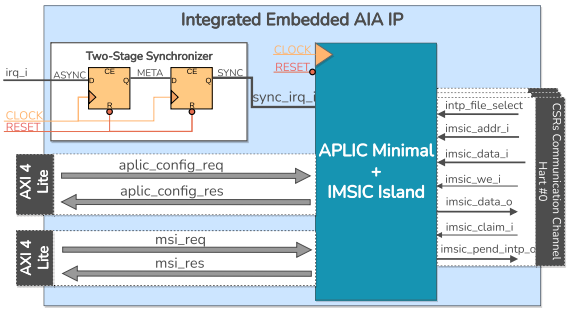

FIGURE 7. Integrated Embedded AIA top-level design.

exposing a single interface to connect to the system bus. This design is possible in the context of embedded and MCS, as mainstream SoCs typically embed up to four cores. Therefore, IMSICs are anyway placed in proximity to the harts.[4]

As illustrated in Figure 6, we modified the IMSIC IP to offer configurability on the number of interrupt files and IMSICs. This design reduces the number of protocol-dependent interfaces connecting to the system interconnect and decreases the overall size of the IMSIC register controller. Due to its ability to support multiple IMSICs within a single module, we refer to this IP as IMSIC Island.

C. INTEGRATED EMBEDDED AIA

As discussed in Section II-C, the AIA virtualization support is provided via the IMSIC. In a typical virtualization-based software for MCS, the hypervisor manages and provides isolation for the consolidated VMs, guaranteeing the overall system integrity. Thus, in a system lacking virtualization support at the interrupt controller (i.e., IMSIC IP with VS interrupt file), the hypervisor must rely on techniques like trap-and-emulate to securely deliver interrupts to the VMs. This approach significantly impacts the overall system performance, mainly interrupt latency [9], [16]. Therefore, implementing the IMSIC IP is highly recommended for embedded and MCS. However, as seen in Section III-B, the IMSIC module interfaces with interrupts in the form of MSIs. Transmitting these messages across the standard system bus (per the suggestion of the AIA spec) may open a venue for issues related to interference, which has a very negative effect on predictability and determinism (in some cases, may potentially result in some denial-of-service (DoS) attack) [23], [24].

The IE-AIA IP, Figure 7, reflects the proposed AIA microarchitecture for embedded and MCS. This design merges the APLIC Minimal and the IMSIC Island described above. Regarding IP scalability, we reiterate that this microarchitecture strongly focuses on embedded and MCS. Therefore, the centralization of AIA components does not constitute a severe microarchitectural scalability problem, as these platforms typically have 2-4 cores. The design logic focuses on instantiating the IMSICs within the APLIC target notifier. In this design, the APLIC can send MSIs directly to the target IMSIC, avoiding the system interconnect and eliminating the necessity for the target notifier frontend. Since the MSIs are delivered directly to their IMSIC without passing through any interconnect, the interrupt latency is expected to be lower and deterministic. To materialize such a design, we made the following modifications: one to the APLIC Minimal and other to the IMSIC Island.

Firstly, we had to (i) adapt the APLIC Minimal's target notifier by removing the frontend module that converts MSI requests into protocol-specific requests and (ii) instantiate the IMSIC Island. Since SoCs may support interrupts in the form of MSIs, we kept the MSI channel interface (although its implementation is now optional). Please note that we also modified the APLIC top-level interface to add this extra communication interface. Secondly, for the APLIC to send an MSI directly to an IMSIC, we added a new communication channel to the IMSIC interface named APLIC Channel. This channel consists of 3 signals: (i) one to indicate the pending and enabled interrupt source, (ii) another to indicate the target interrupt file, and (iii), finally, a control signal that ensures the data validity.

V. EVALUATION

System and Tools: The evaluation was conducted using a single-core CVA6-based SoC (@50 MHz), with the RISC-V hypervisor extension, on a Genesys 2 FPGA board. The software stack includes OpenSBI (version 1.1), Bao hypervisor (version 1.0) [25], [26], and Linux (version 6.1-rc4). Bao and OpenSBI were compiled using SiFive's GCC version 10.1.0 for riscv64 unknown targets, and Linux was compiled using GNU RISC-V GCC version 10.3.0 for riscv64 Linux targets. We compiled the entire software stack with the optimization level -O2. The (A)PLIC interrupt controller and the IMSIC interrupt files IPs were generated to support 32 and 64 interrupt sources, respectively. The design was synthesized using Vivado version 2021.2.

Methodology: We started by performing a functional validation of the AIA IP. We then focused on evaluating the interrupt latency. We borrowed the same methodology as in works [9] and [16], i.e., we use a custom lightweight baremetal benchmark to measure the interrupt latency and variance. This is a well-established approach used among the real-time systems community. This benchmark measures the latency of periodic interrupts triggered by the timer within the CVA6 SoC. Within the context of this benchmark, the timer operates in auto-reload mode, continuously triggering interrupts at 10 ms intervals and capturing 100,000 samples.

A. FUNCTIONAL VALIDATION

To guarantee a minimal level of soundness of our AIA IP implementation, we conducted a comprehensive functional validation, which encompassed the following key components. (i) Hardware unitary tests: we performed unitary tests,

[4]We acknowledge that placing the IMSICs at the chip level is not precisely the same as that at the hart level, but this comparison is in perspective with designs for HPC and the cloud, where we may have hundreds of harts.

TABLE 2. Hardware resources comparison (in a Genesys 2) between SoC configurations: (1) Original SoC with PLIC, (2) AIA APLIC Scalable, and (3) AIA APLIC Minimal. The percentual values marked with * translate the comparison between (3) and (2).

SoC Configuration	Resource	Utilization
PLIC (1)	**LUT**	**83630**
	FF	**55796**
APLIC Scalable (2)	LUT	90529 (+8.24%)
	FF	58819 (+5.42%)
APLIC Minimal (3)	LUT	86858 (+3.86%)(-4.06%*)
	FF	57352 (+2.79%)(-2.49%*)

created from scratch, using the Cocotb framework [27]. (ii) Basic software framework: we expanded upon the framework developed in [16] to validate the hypervisor extension by incorporating AIA baremetal tests. These low-level test cases cover scenarios such as external interrupt line triggering and MSI interrupt generation. This framework enables the first validation of the AIA IP in a software stack environment. (iii) Custom baremetal application: we built a baremetal application with AIA support and made a blob with openSBI. (iv) Linux boot and execution: we successfully booted and ran the vanilla Linux for RISC-V. (v) Custom baremetal application atop Bao hypervisor: with the same baremetal used in (iii), we built and ran a virtualization-based software stack based on Bao. (vi) Linux atop of Bao: we successfully booted the Linux OS on top of Bao.

B. HARDWARE RESOURCES

Tables 2 and 3 summarise FPGA resource utilization across various SoC configurations. These numbers aim at offering an initial reference point for system designers.[5] Configuration (1) represents the original SoC with the RISC-V PLIC, while in configurations (2) and (3), we replace the PLIC with the AIA APLIC IP and AIA APLIC Minimal IP, both in direct mode, respectively. Configuration (4) includes the complete AIA IP (APLIC Minimal IP + 1 IMSIC IP). Configuration (5) implements the IE-AIA IP with 1 IMSIC. Finally, to demonstrate the advantages of the IE-AIA in an SoC that is closer to what is used in high-end platforms for MCS [25], configurations (6) and (7) implement 4 IMSICs (representing a 4-core setup). For configurations (4), (5), (6), and (7), the IMSICs implement three interrupt files: the mandatory M-mode and S-mode, as well as the VS-mode file to support virtualization for one guest. Tables 2 and 3 yields three key conclusions.

1) HARDWARE RESOURCES: APLIC

Firstly, when comparing SoC configuration 1 (vanilla PLIC) with configuration 2 (APLIC IP), the utilization of FPGA Look-Up Table (LUT) resources increased by 8.24% and Flip-Flops (FF) by 5.42%. These results decrease when replacing the APLIC IP with the APLIC Minimal (configuration 3). Thus, when comparing configuration 1 with 3 (i.e., PLIC vs APLIC Minimal), we observe an increase

TABLE 3. Hardware resources (in a Genesys 2) comparison between SoC configurations: (3) AIA APLIC Minimal and (4) AIA IP 1 IMSIC (w/ 1 VS-File), (5) IE-AIA 1 IMSIC (w/ 1 VS-File), (6) AIA IP 4 IMSIC (w/ 1 VS-File), and (7) IE-AIA 4 IMSIC (w/ 1 VS-File).

SoC Configuration	Resource	Utilization
APLIC Minimal (3)	**LUT**	**86858**
	FF	**57352**
AIA (w/ 1 IMSIC) (4)	LUT	89226 (+2.73%)
	FF	58119 (+1.34%)
IE-AIA (w/ 1 IMSIC) (5)	LUT	88309 (+1.67%)
	FF	58107 (+1.32%)
AIA (w/ 4 IMSIC) (6)	LUT	92632 (+6.65%)
	FF	59175 (+3.18%)
IE-AIA (w/ 4 IMSIC) (7)	LUT	90312 (+3.97%)
	FF	58842 (+2.60%)

of 3.86% in LUTs and 2.79% in FFs. Notably, this aligns with expectations, as the APLIC implements two domains, leading to an anticipated reduction of approximately 50% in both LUTs and FFs between APLIC Scalable and APLIC Minimal IPs. However, it is important to highlight that there is still a slight increase in hardware resource consumption between the PLIC (configuration 1) and the APLIC Minimal (configuration 3). This is justified by (i) the additional functionalities offered by APLIC compared to PLIC and (ii) our focus on modularity over optimization.

2) HARDWARE RESOURCES: SINGLE-CORE AIA

Secondly, from Table 3, when comparing configurations 4 and 3 (AIA w/ 1 IMSIC vs APLIC Minimal), we perceived an increase of 2.73% in LUTs and 1.34% in FFs. The observed hardware increase results from adding AIA ISA extensions to the CVA6 hart and implementing an IMSIC IP with three interrupt files. This extra hardware cost can be minimized with the IE-AIA design (configuration 5). Therefore, from the comparison between configurations 5 and 3, we observe a total increase of 1.67% in LUTs and 1.32% in FFs. Based on these results, we can conclude that using the IE-AIA IP (configuration 5), rather than the AIA IP (configuration 4), results in less utilization of FPGA resources. This reduction is attributed to the integration of the IMSIC within the APLIC, eliminating the necessity to implement hardware for translating an MSI into a protocol-specific request.

3) HARDWARE RESOURCES: MULTI-CORE AIA

Thirdly, adopting the IE-AIA design instead of AIA IP further benefits from the increasing number of harts in the system. While public and open-source SoCs based on the CVA6 target single-core configuration,[6] we have built two SoCs that implement the AIA and IE-AIA IPs with support for up to four harts, i.e., AIA w/ 4 IMSIC and IE-AIA w/ 4 IMSIC, respectively (no cache coherence - just for hardware resource evaluation purposes). As observed, configuration 6 (AIA w/ 4 IMSIC) presents a rise of 6.65% in LUTs and 3.18% in FFs when compared to configuration 3 (APLIC Minimal).

[5] There may be deviations in hardware cost assessment between FPGA and ASIC synthesis. We plan to conduct this study in future work by integrating our AIA IP into an evolving version of Shaheen [15].

[6] OpenPiton-Ariane [28] and ESP [29] provide multi-core support but do not target MCS. There is ongoing work to enable embedded multi-core CVA6 configurations based on snoop-based coherence, but it is not finished yet.

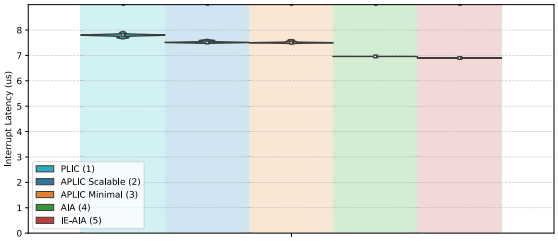

FIGURE 8. Base interrupt latency for the lightweight baremetal application benchmark for each SoC configuration: (1) Original SoC with PLIC, (2) AIA APLIC Scalable, (3) AIA APLIC Minimal, (4) AIA IP 1 IMSIC (w/ 1 VS-File), and (5) IE-AIA 1 IMSIC (w/ 1 VS-File).

Notably, when comparing configurations 7 and 3 (IE-AIA w/ 4 IMSIC vs APLIC Minimal), we observe an increase of only 3.97% in LUTs and 2.6% in FFs. The observed difference is explained by the fact that with AIA IP (configuration 6), each of the IMSICs will be individually connected to the system interconnect, whereas, with the IE-AIA (configuration 7), the implementation of an additional IMSIC does not increase the number of interfaces connected to the system interconnect.

C. INTERRUPT LATENCY

To recap, as aforementioned, we resort to the same methodology described in [9] and [16] to assess the interrupt latency. We developed a customized lightweight baremetal application benchmark. We split our evaluation into three parts for the various SoC configurations presented in Section V-B. First, we measured the base interrupt latency by running the benchmark as a baremetal application directly on top of each SoC. Second, we repeated the same experiments for a virtualization-based software stack using the Bao hypervisor. In these experiments, the benchmark runs as a baremetal VM atop Bao for each SoC configuration. Finally, we extended the virtualization-related experiments and focused our research on comparing AIA IP and IE-AIA IP when the system interconnect is under interference.

1) BASE LATENCY

Figure 8 depicts the violin plots for the base interrupt latency of the custom application benchmark running atop each SoC configuration. For configuration 1 (PLIC), we measure an average interrupt latency of 7.80 us. This was the highest base interrupt latency measured among all the SoC configurations. At the same time, we can notice some variation in the interrupt latency, which, in the worst case, reaches 7.90 us. Next, for SoC configurations 2 (APLIC Scalable IP) and 3 (APLIC Minimal), we observe a decrease in average interrupt latency of about 3.72% (from 7.80 us to 7.51 us). The reduction in latency between configuration 1 (PLIC) and 2/3 (APLIC Scalable and Minimal, respectively) is justified by the interrupt handling process, i.e., in the PLIC it requires two stages - claim and complete - whereas, in the APLIC, this process completes in a single stage. Regarding configurations 2 and 3, we see that, as expected, they do not show different interrupt latency results since the

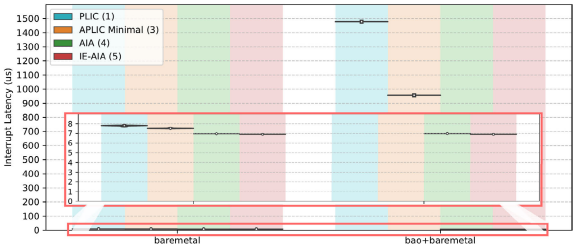

FIGURE 9. Interrupt latency in a virtualization stack environment - benchmark application running atop Bao hypervisor for SoCs: (1) Original SoC with PLIC, (3) AIA APLIC Minimal, (4) AIA IP 1 IMSIC (w/ 1 VS-File), and (5) IE-AIA 1 IMSIC (w/ 1 VS-File).

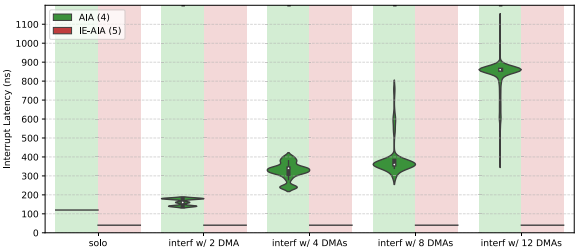

FIGURE 10. Hardware interrupt latency under interference for SoCs: (4) AIA IP 1 IMSIC (w/ 1 VS-File) and (5) IE-AIA 1 IMSIC (w/ 1 VS-File).

redesign of the latter focuses solely on better management of hardware resources. Therefore, in the remaining parts of this subsection, only configuration 3 (APLIC Minimal) will be taken into consideration. In configuration 4 (AIA IP), compared to the base configuration 1 (PLIC), we observed a decrease in the average interruption latency of about 10.77% (from 7.80 us to 6.96 us). Such as with the APLIC, the process of handling an interrupt with the AIA IP just one step. The difference found between configuration 4 and configuration 2/3 (APLIC Scalable and Minimal, respectively) is due to the fact that with AIA, the software handles the interrupt by writing and reading to CSRs instead of reading and writing to memory-mapped regions (MMIO). Finally, configuration 5 (IE-AIA) had the lowest average interruption latency of all the configurations. Compared to PLIC, there was a reduction of 11,41% (from 7.80 us to 6.91 us). Although it is already clear that there is a difference between the average interruption latency in configurations 4 (AIA) and 5 (IE-AIA), these will be further explored in the last part of this subsection.

2) VIRTUALIZATION STACK

Figure 9 shows the results for interrupt latency when using a virtualization stack (the baseline results from Figure 8 are also included to facilitate the comparison). When using a software stack with virtualization, we observe that the PLIC (base configuration 1) introduces a latency of ∼1489 us. This latency primarily arises from the traps between the application and the hypervisor during the claim and complete processes [16]. In configuration 3 (APLIC Minimal), there is a reduction of ∼35.80% (from ∼1489 us to ∼956 us) due to the APLIC's design that allows the interrupt to be handled with a single operation. This happens because reading the APLIC's *claimi* register simultaneously clears the pending bit for the reported interrupt. Therefore, when the

TABLE 4. RISC-V AIA spec implementation status and features: ✓- supported; X - not supported; ○- under development; ? - no information available.

Organization	Processor/Family	License	APLIC	Interrupt Sources	IMSIC	Target Application	Status
John Hauser (RISC-V)	—	Apache 2.0	✓	1-1023	X	General use	WiP
SiFive	P400/P600/P800	Commercial	✓	?	✓	High-performance edge and cloud	Done
Ventana	Veyron V1/V2	Commercial	✓	?	✓	Data-centers	Done
Cobham Gaisler	NOEL-V	Commercial	✓	32	○	Space	WiP
MIPS	eVocore P8700/I8500	Commercial	✓	256	X	High-performance heterogeneous computing	Done
Our Work	—	**Solderpad Hardware License v2.1**	✓	**1-1023**	✓	**General use and embedded/MCS**	**Done**

baremetal benchmark runs atop Bao hypervisor and receives an interrupt, it will only trap once. Finally, for configurations 4 and 5 (AIA and IE-AIA), the interrupt delivery has a latency similar to the respective baseline value (6.98 us and 6.90 us, respectively). Therefore, when comparing configuration 4/5 (AIA/IE-AIA) with configuration 1 (PLIC), we observe a reduction of ∼99.5% (from ∼1489 us to 6.98/6.90 us). This massive reduction results from the guest's direct access to the IMSIC's VS interrupt file, not requiring hypervisor intervention and mediation. Next, we focus on the interrupt latency and predictability of AIA and IE-AIA design approaches.

3) AIA VS IE-AIA

Due to the lack of an embedded/MCS multi-core SoC (i.e., snooping-based coherent multi-core design) based on the CVA6, we added 12 PULP iDMAs [30] to the SoC configurations 4 and 5 (AIA and IE-AIA) to create configurations where the system interconnect is under interference. For the AIA vs IE-AIA comparison, we focused the evaluation at the micro level, i.e., only on the hardware, because the software component will always be dependent on the software stack and identical for both setups. The obtained results are depicted in Figure 10. As it can be seen, with 2 DMAs causing interference to the AIA IP (configuration 4), we observe an increase in average interrupt latency of ∼1.42 × and notice a non-negligible increase of the interrupt variance. For the experiments with 4 and 8 DMAs (still for configuration 4 - AIA), the average interrupt latency increases by ∼2.92 × and ∼2.95 ×, respectively. However, the variance is higher when the 8 DMAs cause interference. Finally, for the experiment with 12 DMAs, we observed an increase in average interrupt latency of ∼7.17 ×, along with an accentuated rise in the variation of interrupt latency, ranging from a minimum of 380 ns to a maximum of 1120 ns. For the same experiments, the IE-AIA (configuration 5) microarchitecture guarantees no additional interrupt latency, since the APLIC does not have to send the MSI through the system interconnect and is, therefore, not affected by interference. Thus, considering the WCET, we perceive a reduction of interrupt latency by 96.43%,[7] comparing to the IE-AIA (from 1120 ns to 40 ns). Therefore, IE-AIA presents a completely deterministic behavior combined with lower interrupt latencies compared to the standard AIA.

[7]Please note this reduction only takes into account the hardware component that contributes to the overall interrupt latency (which typically is the sum of the hardware and software components). Considering both components, we calculate a theoretical reduction of the WCET of 14,02%.

VI. DISCUSSION AND FUTURE DIRECTIONS

A. AIA VALIDATION AND OPTIMIZATIONS

As of this writing, the software setup and the functional validation focused exclusively on a single hypervisor, i.e., Bao. Thus, we will perform additional functional validation with alternative hypervisor solutions, namely KVM [31] and XVisor [32], to guarantee seamless integration within a wide range of software. Furthermore, the current implementation of APLIC focuses on high modularity, which allows excellent compatibility with plenty of protocols and better RTL code maintainability. However, the need for generic interfaces to ensure IP modularity also results in higher utilization of hardware resources. Therefore, we have started a streamline of work towards implementing an optimized APLIC IP focused on low resource usage.

B. PLIC VS AIA

As demonstrated in Section V, APLIC requires more hardware resources than the PLIC. This increase is justified by the new features, including the robust privilege mode isolation. Notwithstanding, the APLIC presents advantages concerning interrupt latency (virtualization system setup) compared to PLIC (fewer traps needed). However, given that both controllers do not provide intrinsic support for virtualization, this has a non-negligible impact on interrupt latency and predictability. Thus, both solutions are unsuitable for use in time-sensitive applications, such as MCS. The RISC-V AIA specification adds virtualization support via the IMSIC. However, the specification suggests implementing separate APLIC and IMSIC components, resulting in the AIA microarchitecture from configuration 4. We identified a drawback in the suggested approach, as the APLIC communicates with the IMSIC via the main platform system bus interconnect. This has implications on the interrupt latency and respective predictability, particularly under the interference of other bus masters. To address such an issue, we developed the IE-AIA microarchitecture, which presented empirical evidence on the advantages of embedded and mixed-criticality systems.

VII. RELATED WORK

Interrupt controllers have a longstanding presence within computing platforms. In academia, several studies have

been conducted to enhance these components with specific features for embedded and MCS [1], [2], [3], [4], [33], [34], [35], [36], [37]. In industry, there are several well-established interrupt controllers targeting mainstream computing ISAs. On the x86/x64 spectrum, Intel has the PIC [38] and its successor APIC [6]. Arm, in turn, has specified the NVIC [39] for the Cortex-M MCU family and the several versions of the GIC (GICv1 [40], GICv2 [41], GICv3, and GICv4 [5], [42]) for the Cortex-A APU family. The AMD Versal has a system interrupt that manages multiple and architecturally distinct interrupt controllers (e.g., GICv2, GICv3, PSM MicroBlaze), allowing their co-existence onto the same platform [43].

The RISC-V AIA specification was just recently ratified (June 2023). Notwithstanding, there are already (or there is work in progress) a few commercial and open-source designs/IPs available: (i) the open-source APLIC IP from John Hauser, the major contributor for the AIA spec [14]; (ii) the commercial APLIC and IMSIC IPs for P400, P500, and P870 series, from SiFive [44], [45]; (iii) the commercial APLIC and IMSIC for Veryon V1/V2, from Ventana; (iv) the commercial APLIC and IMSIC for the space-grade NOEL-V [46], [47], from Cobham Gaisler; and (v) the commercial APLIC for eVocore P8700 and I8500 series, from MIPS [48].

Table 4 summarizes existing and ongoing efforts related to the RISC-V AIA. As observed, the IPs developed by SiFive, Ventana, and MIPS are primarily oriented toward high-performance systems within their respective processor families. For instance, MIPS has introduced the APLIC component, but there is no available information regarding future IMSIC implementations. Cobham Gaisler has implemented an APLIC focusing on space applications with its high-performance and fault-tolerant design. Currently, the IMSIC IP is under development, and version 9 of NOEL-V is expected to feature the entire AIA specification. All these implementations are either closed-source and/or maintained under a commercial license. Ongoing efforts to provide an open-source AIA design are minimal. John Hauser is developing a RISC-V APLIC in System Verilog to offer an open-source, generic reference implementation to the RISC-V community. As of this writing, this IP has not been made publicly available. With our work, we provide a complete open-source AIA reference design for both general use (standard AIA IP) and for embedded and MCS (IE-AIA IP).

VIII. CONCLUSION

This work described our work and research on the RISC-V AIA. We implemented the first open-source AIA IP fully compliant with the recently ratified specification. We then explored microarchitectural design enhancements for mixed-criticality systems (e.g., real-time and predictability). Based on our exploration, we proposed the IE-AIA IP. We also conducted a comprehensive evaluation to collect the first empirical evidence regarding hardware costs and interrupt latencies introduced in a virtualized system. We concluded by discussing further optimizations and other design alternatives to enhance the RISC-V AIA IP and the specification.

ACKNOWLEDGMENT

The authors would like to thank John Hauser, the major driving force behind the AIA specification, for the discussions, insights, and research directions that greatly influenced the content of this article.

REFERENCES

[1] W. Chipin, L. Z. lin, Z. Qingwei, Y. Jianfei, and L. Shenglong, "Design of a configurable multichannel interrupt controller," in *Proc. 2nd Pacific-Asia Conf. Circuits, Commun. Syst.*, vol. 1, Aug. 2010, pp. 327–330.
[2] A. Tumeo, M. Branca, L. Camerini, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "An interrupt controller for FPGA-based multiprocessors," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling Simulation*, Jul. 2007, pp. 82–87.
[3] T. Gomes, P. Garcia, F. Salgado, J. Monteiro, M. Ekpanyapong, and A. Tavares, "Task-aware interrupt controller: Priority space unification in real-time systems," *IEEE Embedded Syst. Lett.*, vol. 7, no. 1, pp. 27–30, Mar. 2015.
[4] T. Harnath and K. L. Kishore, "IDevelopment of customized interrupt controller logic," *Int. J. VLSI Des. Commun. Syst.*, vol. 3, no. 10, pp. 1446–1449, Dec. 2015.
[5] *Arm Generic Interrupt Controller Architecture Specification GIC Architecture Version 3 and Version 4*, Arm, Cambridge, U.K., 2022.
[6] *Intel64 and IA-32 Architectures Software Developer's Manual*, Intel, Santa Clara, CA, USA, 2023.
[7] M. Bechtel and H. Yun, "Denial-of-service attacks on shared cache in multicore: Analysis and prevention," in *Proc. IEEE RTAS*, Apr. 2019, pp. 357–367.
[8] S. Pinto, H. Araujo, D. Oliveira, J. Martins, and A. Tavares, "Virtualization on trustzone-enabled microcontrollers? Voilà!" in *Proc. IEEE RTAS*, Apr. 2019, pp. 293–304.
[9] J. Martins and S. Pinto, "Shedding light on static partitioning hypervisors for arm-based mixed-criticality systems," in *Proc. IEEE RTAS*, May 2023, pp. 40–53.
[10] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen, C. Li, Y. Pu, J. Meng, X. Yan, Y. Xie, and X. Qi, "Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension : Industrial product," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*. Chicago, IL, USA: Industrial, May 2020, pp. 52–64.
[11] L. Valente, Y. Tortorella, M. Sinigaglia, G. Tagliavini, A. Capotondi, L. Benini, and D. Rossi, "HULK-V: A heterogeneous ultra-low-power Linux capable RISC-V SoC," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Apr. 2023, pp. 1–6.
[12] F. Ficarelli, A. Bartolini, E. Parisi, F. Beneventi, F. Barchi, D. Gregori, F. Magugliani, M. Cicala, C. Gianfreda, D. Cesarini, A. Acquaviva, and L. Benini, "Meet Monte cimone: Exploring RISC-V high performance compute clusters," in *Proc. 19th ACM Int. Conf. Comput. Frontiers*, May 2022, pp. 207–208.
[13] *RISC-V Platform-Level Interrupt Controller Specification, Document Version 1.0.0*, RISC-V International, 2023. [Online]. Available: https://github.com/riscv/riscv-plic-spec/blob/master/riscv-plic-1.0.0.pdf
[14] J. Hauser. *RISC-V Advanced Interrupt Architecture (AIA)*, RISC-V, 2023. [Online]. Available: https://github.com/riscv/riscv-aia
[15] L. Valente, A. Veeran, M. Sinigaglia, Y. Tortorella, A. Nadalini, N. Wistoff, B. Sa, A. Garofalo, R. Psiakis, M. Tolba, A. Kulmala, N. Limaye, O. Sinanoglu, S. Pinto, D. Palossi, L. Benini, B. Mohammad, and D. Rossi, "Shaheen: An open, secure, and scalable RV64 SoC for autonomous nano-UAVs," in *Proc. IEEE Hot Chips 35 Symp. (HCS)*, Aug. 2023, pp. 1–12.
[16] B. Sá, J. Martins, and S. Pinto, "A first look at RISC-V virtualization from an embedded systems perspective," *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2177–2190, Sep. 2022.
[17] A. Waterman, K. Asanovic, and J. Hauser, Eds., "The RISC-V instruction set manual, volume II: Privileged architecture, document version 20211203," RISC-V Int., Dec. 2021. [Online]. Available:https://github.com/riscv/riscv-isa-manual/releases/download/Privv1.12/riscv-privileged-20211203.pdf

[18] *RISC-V IOPMP Architecture Specification. Version 1.0.0*, IOPMP Task Group, 2023. [Online]. Available: https://github.com/riscv-non-isa/iopmp-spec/blob/main/riscv_iopmp_specification.pdf
[19] *RISC-V IOMMU Specification Document. Version 1.0*, IOMMU Task Group, 2023. [Online]. Available: https://github.com/riscv-non-isa/riscv-iommu/blob/v1.0/riscv-iommu.pdf
[20] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2629–2640, Nov. 2019.
[21] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, "Ara: A 1-GHz+ scalable and energy-efficient RISC-V vector processor with multiprecision floating-point support in 22-nm FD-SOI," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 530–543, Feb. 2020.
[22] B. Sá, L. Valente, J. Martins, D. Rossi, L. Benini, and S. Pinto, "CVA6 RISC-V virtualization: Architecture, microarchitecture, and design space exploration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 11, pp. 1713–1726, Nov. 2023.
[23] J. P. Cerrolaza, R. Obermaisser, J. Abella, F. J. Cazorla, K. Grüttner, I. Agirre, H. Ahmadian, and I. Allende, "Multi-core devices for safety-critical systems: A survey," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–38, Jul. 2021.
[24] P. Radojković, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla, "On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 1–25, Jan. 2012.
[25] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "BAO: A lightweight static partitioning hypervisor for modern multi-core embedded systems," in *Proc. Workshop Next Gener. Real-time Embedded Syst. (NG-RES)*, 2020, pp. 1–14.
[26] M. A. Alomari, H. Aris, M. Ghaleb, Y. Almurtadha, G. A. Alkawsi, I. A. A. Al-Hadi, Y. Baashar, and K. Samsudin, "Embedded devices security: Design and implementation of a light RDBMS encryption utilizing multi-core processors," *IEEE Access*, vol. 11, pp. 19836–19848, 2023.
[27] B. J. Rosser, "Cocotb: A Python-based digital logic verification framework," Micro-Electron. Section Seminar, CERN, Geneva, Switzerland, 2018.
[28] J. Balkind, K. Lim, F. Gao, J. Tu, D. Wentzlaff, M. Schaffner, F. Zaruba, and L. Benini, "OpenPiton+Ariane: The first open-source, SMP Linux-booting RISC-V system scaling from one to many cores," in *Proc. Workshop Comput. Archit. Res. RISC-V (CARRV)*, 2019, pp. 1–6.
[29] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile SoC development with open ESP: Invited paper," in *Proc. IEEE/ACM ICCAD*, Nov. 2020, pp. 1–9.
[30] T. Benz, M. Rogenmoser, P. Scheffler, S. Riedel, A. Ottaviano, A. Kurth, T. Hoefler, and L. Benini, "A high-performance, energy-efficient modular DMA engine architecture," 2023, *arXiv:2305.05240*.
[31] S. Zhao, "Trap-less virtual interrupt for KVM on RISC-V," KVM Forum, 2020. [Online]. Available: https://www.youtube.com/watch?v=KIJuqtHC4bQ
[32] A. Patel, M. Daftedar, M. Shalan, and M. W. El-Kharashi, "Embedded hypervisor xvisor: A comparative analysis," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Mar. 2015, pp. 682–691.
[33] A. de Gloria, P. Faraboschi, and M. Olivieri, "A self timed interrupt controller: A case study in asynchronous micro-architecture design," in *Proc. 7th Annu. IEEE Int. ASIC Conf. Exhib.*, Sep. 1994, pp. 296–299.
[34] J. E. Amiri and M. Kargahi, "A predictable interrupt management policy for real-time operating systems," in *Proc. CSI Symp. Real-Time Embedded Syst. Technol. (RTEST)*, Oct. 2015, pp. 1–8.
[35] I. Behnke, L. Pirl, L. Thamsen, R. Danicki, A. Polze, and O. Kao, "Interrupting real-time IoT tasks: How bad can it be to connect your critical embedded system to the Internet?" in *Proc. IEEE 39th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2020, pp. 1–6.
[36] M. De Alba, A. Andrade, J. González, J. Gómez-Tagle, and A. D. García, "FPGA design of an efficient and low-cost smart phone interrupt controller," *Latin Amer. Appl. Res.*, vol. 37, no. 1, pp. 59–63, 2007.
[37] B. Li, J. Lu, D. Wu, and G. Liu, "Design of many core interrupt controller based on ARMv8 architecture," *WSEAS Trans. Circuits Syst.*, vol. 14, pp. 468–473, 2015.
[38] C. Y. Sia, B. A. Rosdi, and M. C. Lee, "Synchronous design of 8259 programmable interrupt controller," in *Proc. IEEE ICCAIE*, Dec. 2011, pp. 195–200.
[39] *Cortex-M3 Technical Reference Manual*, Arm, Cambridge, U.K., 2010.
[40] *Application Note GIC Stream Protocol Interface*, Arm, Cambridge, U.K., 2020.
[41] *ARM Generic Interrupt Controller Architecture Version 2.0—Architecture Specification*, Arm, Cambridge, U.K., 2008.
[42] *Locality-Specific Peripheral Interrupts Arm Generic Interrupt Controller V3 and V4*, Arm, Cambridge, U.K., 2022.
[43] *Versal Adaptive SoC Technical Reference Manual (AM011)*, AMD, Santa Clara, CA, USA, 2023.
[44] SiFive. (2023). *SiFive Performance P400-Series*. [Online]. Available: https://sifive.cdn.prismic.io/sifive/60ff3edf-baa2-4656-b067-68f3b006ded6_sifive-p400-datasheet.pdf
[45] SiFive. (2022). *SiFive Performance P600-Series*. [Online]. Available: https://sifive.cdn.prismic.io/sifive/7be0420e-dac1-4558-85bc-50c7a10787e7_p600_datasheet.pdf
[46] Gaisler. (2022). *GRLIB IP Core Users Manual*. [Online]. Available: https://www.gaisler.com/products/grlib/grip.pdf
[47] J. Andersson, "Development of a NOEL-V RISC-V SoC targeting space applications," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2020, pp. 66–67.
[48] MIPS. (2023). *eVocore P8700 Multiprocessing System*. [Online]. Available: https://tinyurl.com/mips-product-brief

FRANCISCO MARQUES received the bachelor's degree in electronics and computer engineering. He is currently pursuing the M.Sc. (master's) degree in embedded systems and micro and nanotechnologies with the Embedded Systems Research Group, University of Minho, Portugal. His research interests include computer architecture, digital systems design, low-level programming, operating systems, and cybersecurity.

MANUEL RODRÍGUEZ received the bachelor's degree in electronic and computer engineering, in 2021. He is currently pursuing the M.Sc. degree in embedded systems and micro/nanotechnologies with the Embedded Systems Research Group, University of Minho, Portugal. His research interests include computer architecture, hardware design, embedded virtualization, operating systems, and automotive.

BRUNO SÁ received the master's degree in electronics and computer engineering with specialization in embedded systems and automation, control and robotics. He is currently pursuing the Ph.D. degree with the Embedded Systems Research Group, University of Minho, Portugal. His research interests include operating systems, virtualization for embedded systems, computer architectures, the IoT systems, and artificial intelligence.

SANDRO PINTO received the Ph.D. degree in electronics and computer engineering. He is currently an Associate Research Professor with Centro ALGORITMI, Universidade do Minho, Portugal. He has a deep academic background and several years of industry collaboration focusing on operating systems, virtualization, security for embedded, CPS, and the IoT systems. He has published more than 90 peer-reviewed articles and is a skilled presenter with speaking experience in top-tier academic and industrial conferences.