一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2025-54,DC2025-126(2026-03)

System-Level Integration and Evaluation of RISC-V Advanced Interrupt Architecture towards Secure I/O on Multicore Platforms

Xinyuan FU†, Akihiro SAIKI†, and Keiji KIMURA†

† Faculty of Science and Engineering, Waseda University
Ohkubo 3–4–1, Shinjuku-ku, Tokyo, 169–8555 Japan

E-mail: †xinyuanfu@akane.waseda.jp, ††saiki@kasahara.cs.waseda.ac.jp, †††keiji@waseda.jp

Abstract As multicore processors and heterogeneous systems become mainstream, interrupt handling has emerged as a critical bottleneck affecting both scalability and system security. Interrupts cross privilege boundaries and directly influence control flow, making them a sensitive shared resource in systems hosting mutually untrusted software components. Traditional interrupt architectures, including legacy RISC-V PLIC-based designs, were not designed with strong isolation or large-scale multicore systems in mind. In parallel, Trusted Execution Environments (TEEs) and secure runtime systems are increasingly required to provide strong isolation while sharing hardware resources. For such systems, interrupt delivery must be precisely controlled to prevent unauthorized interference. These trends motivate the need for hardware-supported interrupt architectures that provide scalability, isolation, and configurability. While Message Signal Interrupt (MSI) satisfies these requirements and has been introduced in TEE, further investigation is needed in terms of hardware and software to achieve truly secure systems. However, there have been no hardware research platforms for it. To the best of our knowledge, this work presents the first open, Linux-ready multicore hardware evaluation platform that integrates the RISC-V Advanced Interrupt Architecture, enabling MSI. By providing a complete and scalable interrupt subsystem compliant with the AIA specification, the proposed platform enables execution of AIA-aware system software and serves as a practical foundation for Linux-based and security-oriented research on multicore RISC-V systems.

Key words Trusted Execution Environment, RISCV-AIA, multi-core, RISC-V, SoC architecture

1. Introduction and Motivation

1.1 Background and Motivation

In modern computing systems, ensuring correct, scalable, and isolated interrupt delivery is a fundamental requirement for both performance and security. Traditional interrupt architectures, including RISC-V ' s legacy Platform-Level Interrupt Controller (PLIC), were designed primarily for simple privilege separation and performance optimization, and they face limitations when deployed in multicore systems with stringent security requirements. For example, when multiple execution domains share hardware resources, unmediated interrupts can interfere across privilege boundaries, undermining isolation and trusted computing goals. Message Signal Interrupt (MSI) has been a popular interrupt mechanism to realize it.

At the same time, Trusted Execution Environments (TEEs) — hardware-assisted isolated execution contexts that protect sensitive code and data even from a compromised operating system — are gaining importance across cloud, edge, and IoT platforms. TEEs aim to provide a secure execution foundation by isolating execution contexts from untrusted software and enforcing confidentiality and integrity. However, as TEE use cases expand, the underlying system software and hardware must support fine-grained control of shared resources, including interrupts, to realize secure end-to-end execution environments. [1]

Research efforts such as the national " Universal TEE architecture " program in Japan emphasize the necessity of closely coordinated hardware, software, and theoretical foundations, in which hardware must expose flexible primitives that system software can compose into secure execution domains tailored to diverse threat models and performance/security trade-offs [1]. In this context, improving interrupt architecture is not only a hardware design problem but also a system-level enabler for secure execution environments.

1.2 Problem Statement

Given the increasing importance of TEEs, further investigation is needed to achieve truly secure systems. Interrupts in hardware and software are important research topics because they constitute a significant attack surface. RISC-V-based open-source hardware platforms are promising, as researchers can fully customize both hardware and software. However, they are equipped with traditional PLIC-based interrupt controllers rather than modern MSI-based controllers, such as the RISC-V AIA. They have several limitations:

- Lack of scalable, per-domain interrupt isolation: Most legacy controllers do not provide hardware-enforced ownership and delivery contexts across multiple privilege domains or secure partitions, making it difficult to guarantee that interrupts intended for one domain cannot affect others.
- Inefficient support for virtualized or secure execution stacks: Software mechanisms for isolating interrupts often rely on large hypervisor or OS intervention, which increases complexity, performance overhead, and trusted computing base (TCB) size.
- Absence of architectural primitives that meet emerging security-centric requirements: As hardware and software integrate in support of TEEs or similar isolation techniques, there is a gap between what modern secure system software expects and what interrupt hardware provides.

These limitations are particularly acute on multicore SoCs, where tight coupling between cores, devices, and interrupts is necessary for both performance and security, yet legacy mechanisms are insufficient for emerging trust models.

1.3 Research Objectives

The central objective of this paper is to design, implement, and evaluate a multicore interrupt architecture that meets modern scalability and isolation requirements while serving as a hardware foundation for secure system software, including TEEs or similar execution environments.

The main contributions of this paper are:

(1) Integrate the RISC-V Advanced Interrupt Architecture (AIA) — including APLIC and IMSIC components — into a multicore SoC environment. [2]
(2) Provide hardware-level mechanisms for secure, isolated interrupt delivery and ownership across privilege domains and cores.
(3) Validate the architectural behavior through cycle-accurate RTL simulation (using Verilator) and identify real-world design challenges.
(4) Establish a platform that can be extended in future work toward full secure execution environments with FPGA deployment and system-software integration.

2. Background of RISC-V Interrupt Architecture

2.1 Interrupt Handling in Multicore Systems

In a typical multicore system-on-chip (SoC), multiple devices may generate interrupts that must be routed to specific processor cores based on software-defined policies. These policies may depend on workload distribution, power management, or execution context. As the number of cores increases, centralized interrupt handling mechanisms often become bottlenecks, leading to contention and increased latency. Consequently, modern interrupt architectures, such as MSI, increasingly favor distributed or per-core interrupt handling models.

From a system perspective, interrupts directly affect control flow and privilege transitions. An interrupt can preempt currently executing code and transfer control to a handler operating at a higher privilege level. This property makes interrupts both powerful and potentially dangerous: improper isolation or routing can lead to unintended interference between software components, especially when multiple execution domains coexist on the same hardware platform. Therefore, interrupt handling is not merely a performance concern but also a critical architectural element that influences system correctness and security.

2.2 Limitations of Legacy RISC-V Interrupt Architecture

The original RISC-V interrupt architecture primarily relies on the Platform-Level Interrupt Controller (PLIC) [3] to handle external interrupts. While the PLIC design is sufficient for simple systems with a small number of cores and limited privilege separation, it exhibits several limitations when applied to modern multicore and security-sensitive environments.

First, the PLIC employs a largely centralized design, where interrupt prioritization and routing decisions are managed through shared state. As core counts increase, this shared structure becomes a scalability bottleneck and complicates fine-grained control over interrupt delivery. Furthermore, the PLIC provides limited native support for per-core or per-domain interrupt contexts, thereby making it difficult to enforce strong hardware isolation.

Second, legacy PLIC-centric designs assume a relatively static mapping between interrupts and privilege modes. Although basic masking and prioritization mechanisms are available, the architecture lacks explicit support for interrupt ownership across multiple execution domains. As a result, software must rely on additional layers of indirection—such as hypervisors or privileged monitors—to virtualize or isolate interrupts. This approach increases system complexity and expands the trusted computing base.

Finally, the legacy interrupt model does not align well with emerging system software requirements that demand fine-grained, hardware-enforced separation between different software components. In environments where multiple mutually untrusted workloads share a multicore platform, the lack of architectural support for isolated interrupt handling becomes a fundamental limitation.

These shortcomings motivate the need for a new interrupt architecture that can scale with core count, support flexible routing policies, and provide hardware primitives suitable for secure system design.

2.3 Overview of RISC-V Advanced Interrupt Architecture

To address the limitations of legacy interrupt mechanisms, the RISC-V Advanced Interrupt Architecture (AIA) [4] introduces a modular and scalable approach to interrupt handling. AIA decouples interrupt generation, routing, and delivery into distinct architectural components, enabling fine-grained control and improved isolation.

At a high level, AIA separates global interrupt management from per-core interrupt delivery. This separation allows interrupt routing decisions to be made independently of the execution context on each core, while maintaining per-hart (hardware thread) interrupt state.

2.3.1 Advanced Platform-Level Interrupt Controller (APLIC)

The Advanced Platform-Level Interrupt Controller (APLIC) manages interrupt sources and routes them to their appropriate targets. Unlike the legacy PLIC, APLIC supports multiple interrupt domains, each of which can represent a distinct privilege or security context.

Within each domain, interrupt sources can be configured with domain-specific properties such as enablement, priority, and target selection. This design enables hardware-enforced interrupt ownership, ensuring that each interrupt source is associated with exactly one domain at a time. By providing explicit domain separation, APLIC enables isolation between different software components without requiring complex software mediation.

Furthermore, APLIC supports both direct interrupt delivery and MSI generation. This flexibility allows the interrupt architecture to adapt to different system designs and integration strategies, particularly in multicore environments.

2.3.2 Incoming Message-Signaled Interrupt Controller (IMSIC)

The Incoming Message-Signaled Interrupt Controller (IMSIC) handles interrupt delivery at the per-hart level. Each hardware thread is equipped with its own IMSIC, which maintains local interrupt state and delivers interrupts directly to the core.

IMSICs use memory-mapped and CSR-based interfaces to receive message-signaled interrupts, enabling efficient and scalable interrupt delivery without shared global state. By maintaining per-hart interrupt contexts, IMSICs eliminate contention between cores and allow interrupt handling to scale naturally with the number of cores.

Importantly, IMSICs support multiple privilege modes, enabling separate interrupt contexts for different execution levels. This capability provides a hardware foundation for isolating interrupt handling across privilege domains, reducing reliance on software-based virtualization mechanisms.

3. System Architecture and Design

3.1 Target Multicore SoC Architecture

The target platform for this work is a multicore RISC-V SoC

designed to support scalable interrupt handling and architectural experimentation. The SoC comprises multiple RISC-V processor cores (harts), a shared interconnect fabric, memory subsystems, and peripheral devices capable of generating external interrupts. Each core supports multiple privilege modes, forming the basis for system-level isolation and secure execution.

The interrupt subsystem is integrated as a first-class hardware component within the SoC and interacts directly with processor cores through well-defined architectural interfaces. External devices connect to the interrupt controller via dedicated interrupt-source signals, whereas configuration and control are performed via memory-mapped registers accessible via the system interconnect. This organization enables the precise definition of interrupt routing and delivery behavior at the hardware level.

3.2 AIA Integration Overview

An open-source RISC-V Advanced Interrupt Architecture (AIA) [2]is integrated into the SoC by decomposing interrupt handling into two primary components: the Advanced Platform-Level Interrupt Controller (APLIC) and the Incoming Message-Signaled Interrupt Controller (IMSIC). This separation reflects the architectural intent of AIA and enables scalable and isolated interrupt management.

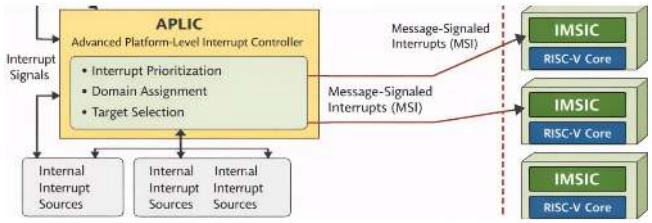

Fig. 1 Conceptual system architecture of RISC-V Advanced Interrupt Architecture

Figure 1 provides an overall architectural view of the proposed multicore interrupt system based on the RISC-V AIA. This highlights the conceptual separation between global interrupt source management and per-hart interrupt delivery, which is central to the AIA design philosophy. In this architecture, interrupt sources, such as RISC-V harts for interprocessor interrupt (IPI) and IO devices having traditional wire-based interrupt mechanisms, are first handled by the APLIC, which is responsible for interrupt configuration, prioritization, and domain-based routing. Interrupts are then delivered to target processor cores via message-signaled interrupts and received by per-hart IMSIC instances. By assigning a dedicated IMSIC to each hart, interrupt state and delivery are fully decentralized, enabling scalable and isolated interrupt handling across the multicore system. This architectural overview clarifies the roles and interactions of AIA components, independent of their concrete implementation details, thereby providing a foundation for the system design discussed in the remainder of this chapter.

The integration adheres strictly to the AIA specification while allowing flexibility in internal implementation details, such as register organization and interconnect attachment, to suit the target SoC architecture.

3.3 APLIC Design and Domain Configuration

3.3.1 APLIC Architecture

The APLIC is designed as a centralized but domain-aware interrupt controller. It manages a configurable number of interrupt sources, each corresponding to an external device or internal event. Internally, the APLIC maintains per-source state, including pending status, enable bits, and priority information.

To support isolation and flexible interrupt ownership, the APLIC organizes interrupt sources into domains. Each domain represents an independent interrupt management context that can be associated with a particular privilege level or execution environment. At any given time, an interrupt source belongs to exactly one domain, ensuring exclusive ownership.

The APLIC exposes a memory-mapped register interface for configuration and control. Through this interface, software can enable or disable interrupt sources, assign them to domains, and configure routing targets. Importantly, once configured, domain ownership and routing decisions are enforced entirely in hardware.

3.3.2 Domain-Based Interrupt Routing

Domain configuration plays a central role in enabling isolation. For each domain, the APLIC maintains separate enable and priority control, preventing cross-domain interference. Interrupts generated by a source are evaluated only within the context of their assigned domain.

When an interrupt becomes pending, the APLIC determines whether it is eligible for delivery based on domain-level configuration and priority resolution. If delivery is permitted, the interrupt is translated into a message-signaled interrupt targeting a specific hart or set of harts. This mechanism allows interrupts to be dynamically routed without exposing shared mutable state between domains.

By enforcing interrupt ownership and routing decisions in hardware, the APLIC reduces reliance on software mediation and provides a strong foundation for secure and predictable interrupt handling.

3.4 IMSIC Integration and Per-Hart Interrupt Contexts

Each processor hart in the SoC is equipped with a dedicated Incoming Message-Signaled Interrupt Controller (IMSIC). The IMSIC maintains all interrupt-delivery states locally, thereby eliminating contention between cores and enabling scalable interrupt handling.

The IMSIC architecture is based on a per-hart interrupt file that records pending interrupts, enable bits, and priority information. Interrupts are delivered to the IMSIC via message-signaled writes generated by the APLIC or MSI-capable peripheral devices via the system bus. Upon receiving a message, the IMSIC updates its local state and signals the processor core according to the configured interrupt mode.

The IMSIC supports multiple privilege modes by maintaining separate interrupt contexts. This allows interrupts to be delivered to the appropriate privilege level without requiring software to manually multiplex the interrupt state. As a result, privilege separation is enforced directly by hardware mechanisms.

The integration of IMSICs into the SoC is straightforward and uniform across cores, making the design naturally extensible to systems with higher core counts.

4. Implementation

4.1 Implementation Platform and Framework Selection

The implementation of the multicore SoC equipped with AIA is based on a set of mature open-source hardware projects, chosen to ensure architectural correctness, extensibility, and reproducibility. Rather than developing a complete system from scratch, this work focuses on integrating and extending existing, widely used hardware

frameworks to realize a practical AIA-enabled multicore SoC.

Figure 2 illustrates the overall implementation structure of the proposed multicore SoC, highlighting the integration of these open-source components. The figure shows how the OpenPiton SoC framework [5] serves as the system-level backbone, into which multiple CVA6 RISC-V processor cores are integrated. The open-source RISC-V Advanced Interrupt Architecture (AIA) components [2], including the global APLIC and per-hart IMSIC instances, are incorporated as first-class hardware modules and connected through the system interconnect. This visualization provides a concrete overview of the hardware architecture and clarifies the interactions among processor cores, interrupt controllers, and the surrounding SoC infrastructure.

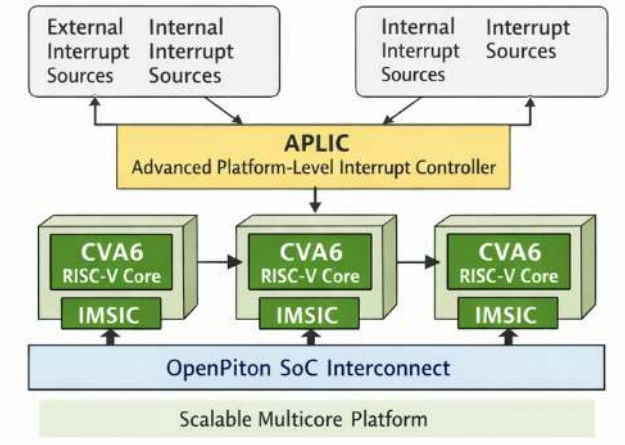

Fig. 2 Multicore SoC Architecture with RISCV AIA

The overall SoC integration framework is based on OpenPiton, an open-source manycore research platform developed by Princeton University. OpenPiton provides a scalable tile-based architecture, a configurable interconnect, and well-defined integration points for processor cores and system peripherals. Its modular design makes it well suited to architectural experimentation and multicore system research.

The RISC-V processor core used in this work is CVA6, an open-source, application-class 64-bit RISC-V core [6]. CVA6 supports multiple privilege modes and is designed for Linux-capable systems, making it appropriate for evaluating advanced interrupt architectures in realistic system settings.

The implementation of the Advanced Interrupt Architecture (AIA) is based on the open-source riscv-aia project developed by Zero-Day Labs. This project provides RTL implementations of APLIC and IMSIC modules that closely follow the RISC-V AIA specification and serve as a solid foundation for system-level integration.

By combining these three projects, this work constructs a multicore SoC platform that is both standards-compliant and suitable for detailed architectural evaluation.

4.2 RTL Integration of AIA Components

The AIA components are integrated into the OpenPiton-based SoC at the RTL level, forming a coherent interrupt subsystem that interfaces with both external devices and processor cores.

The APLIC module is instantiated as a global interrupt controller within the SoC. It receives interrupt signals from peripheral devices that use traditional wire-based interrupt mechanisms and internal interrupt generators. The APLIC is connected to the system interconnect as a memory-mapped peripheral, allowing configuration registers to be accessed through standard load/store transactions.

For each processor hart, a dedicated IMSIC instance is instantiated and tightly coupled with the corresponding CVA6 core. The IMSIC interfaces directly with the core ' s interrupt input signals and control/status registers, enabling efficient delivery of message-signaled interrupts without shared global state.

Integration at the RTL level required careful coordination of signal interfaces, clocking, and reset behavior. Particular attention was paid to ensuring that interrupt routing, delivery timing, and privilege enforcement followed the AIA specification while remaining compatible with the existing OpenPiton and CVA6 interfaces

4.3 Multicore Support and Interconnect Considerations

Multicore support is a central requirement of the implementation. In the OpenPiton framework, each tile contains a processor core and associated local components, while shared resources are connected through a scalable interconnect.

The APLIC is connected to the global interconnect, allowing it to communicate with all tiles in the system. Interrupt delivery from the APLIC to IMSICs is implemented via a message-signaled mechanism that avoids shared interrupt lines and scales naturally with the number of cores. In addition to its connection with APLIC, IMSIC receives MSI from MSI-capable peripherals via the system bus.

Each IMSIC instance maintains per-hart interrupt state, ensuring that interrupt handling remains independent across cores. This design eliminates contention among cores during interrupt delivery and simplifies reasoning about correctness and isolation in a multicore environment.

The interconnect integration ensures that configuration traffic, interrupt messages, and normal memory accesses coexist without interference, preserving system functionality while enabling flexible interrupt routing.

4.4 Configuration and Control Interfaces

Configuration and control of the interrupt subsystem are performed through memory-mapped registers exposed by the APLIC and IMSIC components. These registers allow software to define interrupt domains, enable or disable interrupt sources, configure priorities, and select routing targets.

In this implementation, configuration access is limited to privileged software, ensuring that interrupt ownership and routing policies are established in a controlled manner. Once configured, enforcement of these policies is handled entirely in hardware.

The IMSIC exposes per-hart control registers that manage local interrupt state and privilege-level interrupt contexts. These interfaces enable precise control over interrupt delivery without requiring software to manually multiplex or virtualize interrupt state.

By relying on standardized memory-mapped interfaces, the implementation remains compatible with existing firmware and minimal system software while preserving architectural clarity.

5. Verification and Evaluation

5.1 Objectives of Verification and Evaluation

Here, we validate the correctness, isolation properties, and scalability of the proposed multicore interrupt architecture. Since the focus of this work is on hardware design and system integration, verification and evaluation are conducted using RTL-level simulation rather than full-system deployment.

5.2 Experimental Setup

The simulated platform used in this paper includes:

- An OpenPiton-based multicore SoC framework
- CVA6 RISC-V processor cores
- Integrated APLIC and per-hart IMSIC components
- A configurable number of interrupt sources and domains

Verilator is chosen for its ability to efficiently simulate large

RTL designs while providing detailed visibility into internal signals. This enables precise observation of interrupt source state, routing decisions, and delivery behavior at the hardware level.

Testbenches are designed to inject interrupt events, configure interrupt domains, and observe resulting interrupt delivery across multiple cores and privilege contexts. Simulation traces and waveform analysis are used to confirm expected behavior.

5.3 Functional Verification of Interrupt Delivery

5.3.1 Basic Interrupt Routing Correctness

The first set of verification tests focuses on basic interrupt routing functionality. Interrupt sources are individually asserted, and the APLIC is configured to route each source to a designated target hart.

Simulation results confirm that:

- Interrupts are correctly detected by the APLIC
- Routing decisions follow the configured domain and target settings
- Message-signaled interrupts are delivered to the correct IMSIC instance
- The corresponding processor core observes the interrupt at the expected cycle

These tests establish baseline correctness of the interrupt delivery path.

5.3.2 Per-Hart IMSIC Behavior

To verify per-hart isolation, multiple IMSIC instances are exercised simultaneously. Interrupts targeting different harts are injected concurrently to evaluate whether the interrupt state remains local to each hart.

The results show that:

- Each IMSIC maintains an independent interrupt state
- Interrupt delivery to one hart does not affect the pending or enabled state of other harts
- Concurrent interrupt delivery scales linearly with the number of cores

```
Info: spc(0) thread(0) Hit Good trap
41175250: Simulation -> PASS (HIT GOOD TRAP)
- /piton/verif/env/manycore/pc_cmp.tmp.v:2210: Verilog $finish
41175250: Simulation -> PASS (HIT GOOD TRAP)
- /piton/verif/env/manycore/pc_cmp.tmp.v:3256: Verilog $finish
- /piton/verif/env/manycore/pc_cmp.tmp.v:3256: Second verilog $finish, exiting
sims: sim_stop
sims: "perf > perf.log"
sims: "regreport -1 > status.log"
sims: stop_time
```

Fig. 3 Final simulation log

As shown in Fig. 3, the system reaches a Good trap condition, which is the predefined success state in the verification framework. The subsequent PASS messages and Verilog finish calls confirm that the simulation terminates normally without error,which confirms that the IMSIC-based delivery mechanism eliminates shared global interrupt state at the delivery stage.

5.4 Domain Ownership Enforcement

To evaluate domain isolation, interrupt sources are assigned to different APLIC domains. Test cases attempt to trigger interrupts from one domain while observing behavior in other domains.

Simulation results demonstrate that As shown in Figure 4:

- Interrupts are evaluated only within their assigned domain
- Cross-domain delivery does not occur
- Domain configuration errors result in interrupts being masked rather than misrouted

5.5 Privilege-Aware Interrupt Delivery

The next set of tests evaluates interrupt delivery across privilege modes. Interrupts are configured to target different privilege-level contexts supported by the IMSIC.

Also as Fifure 4 illustrating, Simulation confirms that:

- Interrupts are delivered only to the configured privilege context
- Privilege transitions occur as expected upon interrupt entry
- No unintended privilege escalation occurs due to interrupt delivery

This behavior is essential for systems that rely on strict separation between execution contexts, such as secure monitors or trusted runtime environments.

```
RISC-V AIA (IMSIC + APLIC + Smaia/Ssaia) BASIC
test
Configuring APLIC...
Configuring IMSIC M File...
Configuring IMSIC S File...
Configuring interrupts...
cond_ctl:  1
cond_ctl:  1
end
```

Fig. 4 AIA Privilege Domain test log

5.6 Multicore Scalability Evaluation

To evaluate the scalability of the proposed interrupt architecture, the system is configured with up to four processor cores, each equipped with a private IMSIC instance. Although the core count is intentionally limited, the objective of this evaluation is to validate the structural scalability of the AIA-based design rather than to maximize core count.

Interrupt sources are generated concurrently and routed to different target harts through the APLIC. Various configurations are evaluated, including simultaneous interrupts targeting multiple cores and domains, to observe potential contention or interference in the interrupt delivery path.

The evaluation results demonstrate that interrupt routing and delivery remain correct and deterministic as the number of active cores increases to four. Each hart handles interrupts independently via its local IMSIC, and no shared interrupt-delivery state is observed. In addition, no functional bottlenecks are introduced at the delivery stage, confirming that the per-hart IMSIC design effectively decouples interrupt handling across cores.

Although the evaluated core count is modest, the results confirm that the architecture scales by construction. The absence of centralized delivery arbitration and the use of message-signaled interrupts indicate that the design can be extended to larger multicore systems without fundamental architectural changes.

5.7 Linux-Ready Multicore Platform Validation

Beyond functional correctness and scalability, an important outcome of this work is the establishment of a Linux-ready multicore hardware evaluation platform with integrated RISC-V Advanced Interrupt Architecture.

The implemented platform provides a complete AIA-compliant interrupt subsystem, including APLIC-based interrupt source management and per-hart IMSIC-based delivery. The hardware interfaces, memory-mapped control registers, and privilege-aware interrupt routing mechanisms are designed to align with the requirements of AIA-aware system software.

As a result, the platform supports the execution of modern operating systems, such as Linux, that rely on the Advanced Interrupt

Architecture for scalable interrupt handling in multicore environments. While full Linux deployment and benchmarking are beyond the scope of this paper, the hardware foundation required for such system software has been fully implemented and validated at the RTL level.

To the best of our knowledge, this work presents the first open, Linux-ready multicore hardware evaluation platform that integrates RISC-V AIA. This platform enables realistic experimentation with AIA-based interrupt handling and provides a practical basis for future research on secure, virtualized, and trusted execution environments.

6. Conclusion and Future work

This paper presented the design, integration, and evaluation of the RISC-V Advanced Interrupt Architecture (AIA) on a multicore System-on-Chip platform. The primary contribution of this work is the successful integration of AIA into an open-source multicore SoC, demonstrating how modern interrupt architectures can be practically deployed to support scalability and isolation at the hardware level.

Specifically, this work:

- Integrated RISC-V APLIC and IMSIC components into a RISC-V multicore SoC based on OpenPiton
- Adapted and connected the AIA subsystem to CVA6 RISC-V processor cores
- Designed a configurable and extensible interrupt routing framework
- Built an RTL-level verification environment using Verilator
- Validated functional correctness, domain isolation, and multicore scalability

Through this work, the paper provides a concrete reference implementation and evaluation of AIA in a realistic multicore setting.

A natural next step is deploying the integrated platform on an FPGA. This would enable measurement of real-world interrupt latency, throughput, and resource utilization, providing quantitative performance evaluation beyond simulation. FPGA deployment would also allow validation of clock-domain interactions and bus timing behavior that cannot be fully captured in RTL simulation.

Integrating full system software stacks, such as Linux with AIA support, lightweight hypervisors, or secure runtime environments, and executing them on an FPGA is an important task. This would allow evaluation of how the hardware interrupt architecture interacts with scheduling, virtualization, and secure context management in realistic workloads.

Such integration would also enable end-to-end validation of interrupt-driven execution paths, from hardware event generation to software handling.

Building on the current platform, future research could focus on supporting a universal or extensible TEE system. In this context, the AIA-based interrupt architecture could serve as a hardware root for secure I/O channels, ensuring that interrupts destined for secure execution contexts cannot be intercepted or redirected by untrusted software.

This direction represents a convergence of hardware architecture and secure system design, with the potential to reduce software complexity while strengthening system security guarantees.

Acknowledgements This work was supported by JST K Program Grant Number JPMJKP24U4, Japan.

References

[1] 石川裕, 木村啓二, 河野健二, 光来健一, 五島正裕, 塩谷亮太, 須崎有康, 関山太朗, 高前田伸也, 竹房あつ子, 中條拓伯, 古川潤, 宮澤慎一. ハードウェア・ソフトウェア・理論の連携によるユニバーサル tee アーキテクチャの実現に向けて—システムソフトウェアの観点から—. Technical report, May 2025.

[2] Francisco Marques, Manuel Rodríguez, Bruno Sá, and Sandro Pinto. “interrupting” the status quo: A first glance at the risc-v advanced interrupt architecture (aia). *IEEE Access*, Vol. 12, pp. 9822–9833, 2024.

[3] RISC-V International. *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*. RISC-V International, 2023.

[4] RISC-V International. *RISC-V Advanced Interrupt Architecture (AIA) Specification*. RISC-V International, 2023.

[5] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. Openpiton: An open source manycore research framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, p. 217–232, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Bruno Sá, Luca Valente, José Martins, Davide Rossi, Luca Benini, and Sandro Pinto. Cva6 risc-v virtualization: Architecture, microarchitecture, and design space exploration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 31, No. 11, pp. 1713–1726, 2023.

[7] 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 Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, No. 11, pp. 2629–2640, Nov 2019.

[8] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. Openpiton: An open source manycore research framework. *SIGARCH Comput. Archit. News*, Vol. 44, No. 2, p. 217–232, March 2016.

[9] Linux Kernel Developers. Linux kernel risc-v documentation, 2023.

[10] Linux Kernel Developers. Kernel-based virtual machine (kvm) documentation, 2023.