Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 46 No. 8
August 2025

Design of Fast Interrupt Controller for RISC-V Processor

Abhishek M[1], Dr. Roopa J[2]

[1] M. Tech. in VLSI Design and Embedded Systems, R V College of Engineering, Bengaluru, Karnataka, India
[2] Assistant Professor, Department of Electronics and Communication Engineering, R V College of Engineering, Bengaluru, Karnataka, India

Abstract: This work presents a Core Local Interrupt Controller (CLIC) for the RISC-V processor that achieves low-latency and efficient interrupt handling. By combining Selective Hardware Vectoring (SHV), a priority-aware arbitration tree, and a kill-ack handshake, the design services interrupts in 1–3 clock cycles, reducing latency by 50% over CLINT and 80% over PLIC. Simulation and APB configuration results validate its accuracy, flexibility, and suitability for real-time and safety-critical applications.

Introduction: For current processors to handle numerous asynchronous events with prompt CPU reaction, an interrupt controller is necessary. This study introduces a Core Local Interrupt Controller (CLIC) that uses a priority-aware arbitration tree and hardware vectoring to handle interrupts quickly and reliably. Selective Hardware Vectoring (SHV) with a kill-ack handshake allow for real-time preemption by removing software-managed handlers and servicing interrupts (IDs 32, 42, and 52) in three clock cycles. According to simulation studies, latency is reduced by 50% over CLINT and 80% over PLIC, confirming a constant three-cycle response. Functional accuracy was confirmed by APB activity during configuration, which makes the suggested CLIC perfect for latency-sensitive, real-time applications.

Objectives: The primary objective of this work is to design an efficient Core Local Interrupt Controller (CLIC) for the RISC-V processor. The proposed design is simulated and validated in Xilinx Vivado, with waveform analysis used to ensure functional accuracy. The focus is on minimizing interrupt response time by reducing the number of clock cycles required for servicing, thereby achieving lower interrupt latency and enhancing real-time performance for latency-critical applications.

Methods: The Core Local Interrupt Controller (CLIC) was designed and implemented using SystemVerilog and validated through Xilinx Vivado 2023.2 simulation. The design incorporated a priority-aware arbitration tree, hardware vectoring, and a kill-ack handshake mechanism to minimize interrupt response time and ensure real-time preemption. A dedicated testbench environment was developed to evaluate functional accuracy. Multiple interrupt sources (IDs 32, 42, and 52) with varying priorities were applied to validate arbitration accuracy and latency. Key signals, including irq_valid, irq_id, irq_level, irq_priv, and handshake signals, were monitored in waveforms to confirm proper sequencing and reliable register interactions over the APB interface.
Performance evaluation focused on measuring clock cycles from interrupt trigger to handler initiation, with comparisons against baseline controllers like CLINT and PLIC. Results consistently demonstrated reduced latency, validating the efficiency and reliability of the proposed architecture.

Results: According to simulation studies, the suggested CLIC design consistently reduces latency by about 50% over CLINT and 80% over PLIC by servicing interrupts within 1–3 clock cycles. In order to provide precise and predictable interrupt handling, waveform analysis confirmed appropriate arbitration, hardware vectoring, and dependable APB register access.

Conclusions: The proposed CLIC design significantly reduces interrupt response time and latency, demonstrating superior performance over CLINT and PLIC, making it well-suited for real-time and latency-sensitive applications.
Keywords: RISC-V, Core Local Interrupt Controller (CLIC), hardware vectoring, interrupt latency, Selective Hardware Vectoring (SHV), arbitration tree, real-time systems, APB interface, context switching, nested interrupts

1. Introduction

The rapid advancement of VLSI technology has allowed for the integration of increasingly sophisticated functionalities into System-on-Chip (SoC) architectures, considerably improving computing efficiency, real-time responsiveness, and power optimization. Among these integrated

components, interrupt controllers are crucial in managing asynchronous events that require the processor's immediate attention. Efficient processing of such events is critical in embedded systems, where numerous peripherals may generate interrupt requests simultaneously. Conventional interrupt controllers like the Platform-Level Interrupt Controller (PLIC) and Core Local Interruptor (CLINT) are mostly based on software-controlled interrupt service routines (ISRs), leading to increased latency and limited preemption support. To overcome these limitations, this paper introduces a hardware-tuned Core Local Interrupt Controller (CLIC) based on Selective Hardware Vectoring (SHV), priority-aware arbitration tree, and optimized kill-ack handshake mechanism to offer predictable interrupt service in 3 clock cycles.

This architectural enhancement significantly reduces interrupt response time and supports layered interrupt handling by enabling high-priority interrupts to preempt lower-priority interrupts. The controller is connected using the Advanced Peripheral Bus (APB) protocol, which offers a lightweight and low power setup interface for SoC peripherals. The simulation findings validate register access and corroborate the low latency interrupt service mechanism. Compared to previous implementations, the suggested CLIC-based design reduces latency by 50% over CLINT and 80% over PLIC, making it ideal for real-time and latency-sensitive embedded applications.

2. Objectives

The primary objective of this research is to design and implement a Core Local Interrupt Controller (CLIC) for the RISC-V processor to achieve efficient and low-latency interrupt handling. The proposed design is simulated and validated in Xilinx Vivado, with waveform analysis performed to ensure functional accuracy. A key focus of the work is to minimize the number of clock cycles required for interrupt servicing, thereby reducing overall interrupt latency and improving response time. Furthermore, the performance of the proposed architecture is compared against baseline controllers such as CLINT and PLIC to demonstrate its suitability for real-time and latency-critical embedded applications.

3. Methods

The proposed Core Local Interrupt Controller (CLIC) is intended to provide quick and deterministic interrupt handling by incorporating Selective Hardware Vectoring (SHV), a priority-aware arbitration tree, and a kill-ack handshake mechanism for nested interrupt preemption. This section describes the architectural components, the configuration protocol, and the interrupt handling flow. Figure 1 shows the block diagram of the CLIC-based interrupt controller. The controller is set up using the Advanced Peripheral Bus (APB) interface, which allows for low-power access to control/status registers. All APB signals (paddr, pwrite, pwdata, penable, psel, and prdata) are simulated to ensure proper register mapping and protocol compliance.

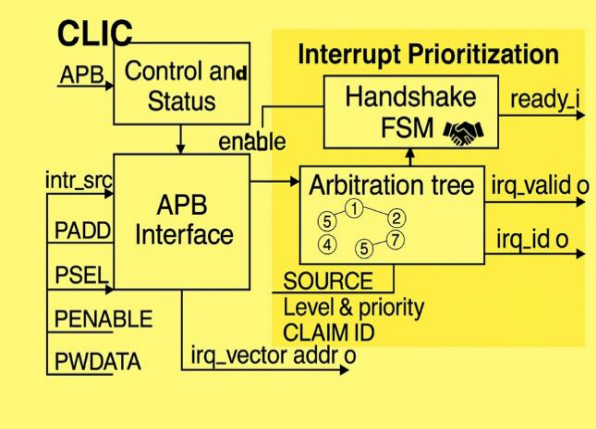

Fig. 1. Block Diagram of CLIC Controller

A RISC-V-based processor system can effectively handle up to 256 interrupt sources thanks to the Core Local Interrupt Controller (CLIC) architecture shown in figure 1. The internal organization of the CLIC is depicted in the block diagram, which highlights the handling, prioritization, and processor dispatch of interrupt sources. The CLIC module receives a 256-bit input vector containing the interrupt sources (for example, IDs 32, 42, and 52). The underlying logic continuously monitors these inputs for active interrupt requests. Each source is allocated a configurable priority and interrupt level using APB-configured registers. The priority-aware arbitration tree assesses all pending interrupts and chooses the one with the highest priority to service. Upon selection, the irq_valid signal is asserted, together with irq_id_o, irq_level_o, and

irq_priv _o, showing the metadata of the active interrupt. The SHV (irq_shv) signal allows the processor to handle vectored interrupts, skipping standard software ISR lookup.

If a higher-priority interrupt is detected during service, the kill-ack handshake mechanism (irq_kill_req_o, irq_kill_ack_i) preempts the current interrupt, allowing for fast context switching and nested interrupt support.

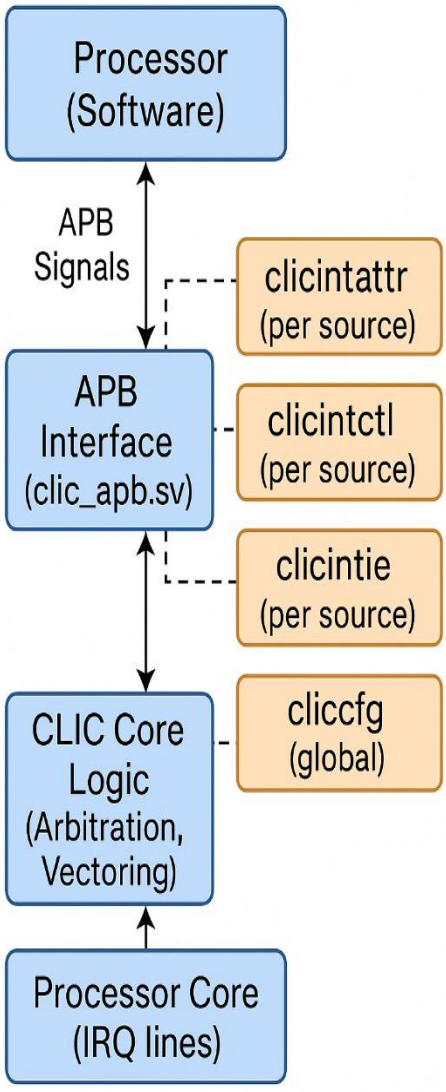

Fig. 2. APB interface connecting the processor with the CLIC register file and logic.

The figure 2 depicts the architectural flow of data and control between the processor and the Advanced Peripheral Bus (APB) and the Core Local Interrupt Controller (CLIC) is shown in fig 2. It shows how the internal registers of the CLIC module are accessed and configured by the processor software to dynamically control interrupt behavior. The architecture places a strong emphasis on keeping the processor's main interrupt response mechanism, interrupt arbitration, and register control logic separate.

Processor (Software): The processor's software side is represented by this block, which creates APB transactions in order to setup the CLIC. It reads and writes to internal CLIC registers using the APB protocol, including clicintattr, clicintctl, and cliccfg, enabling software-level control over SHV mode selection, levels, and interrupt prioritization. Interrupt vectoring is implemented in hardware instead of software when the SHV bit is enabled.

APB Interface (clic_apb.sv): The internal CLIC register set and APB signals are inter-faced by this module. The APB addresses are decoded (paddr), handles read/write operations using pwrite, penable, psel, and pwdata, It routes the data accordingly to registers such as clicintattr, clicintctl, clicintie, and cliccfg.

CLIC Registers: To customize each interrupt source, these programmable registers—such as the clicintattr, clicintctl, clicintie, and cliccfg registers—are required. The CLIC core circuitry uses these registers internally for interrupt selection and can access them via the APB interface.

CLIC Core Logic (Arbitration, Vectoring): The logic for assessing pending interrupts, resolving priority problems, and selecting the interrupt with the highest priority to handle is implemented in this block. Based on the configuration saved in the system, it decides whether to use software or hardware vectoring(clicintattr).

Processor Core (IRQ lines): When the CPU receives the interrupt signals produced by the CLIC core logic, the flow reaches its conclusion.

Among them are irq_valid, irq_id, irq_level, irq_priv, and irq_shv. The CPU can identify, prioritize, and appropriately handle incoming interrupts thanks to these signals. Low latency, software-controlled interrupt handling in real-time is supported by this design. The system provides the flexibility and performance needed for embedded and time-sensitive applications by giving the CLIC hardware control over interrupt priority and vectoring decisions and allowing customization through a straightforward APB interface.

The tail-chaining technique of a Core Local Interrupt Controller (CLIC), which allows for efficient processing of numerous back-to-back interruptions with low

latency and reduced context-switching costs, is shown in Flowchart 3. When a higher-priority interrupt comes in right after the current one is handled, tail-chaining comes in quite handful.

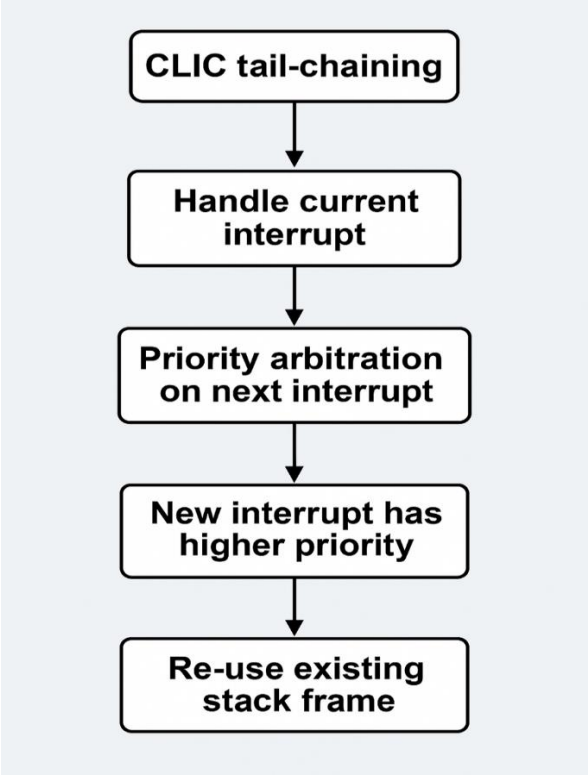

Fig. 3. Tail Chaining process in CLIC

By enabling smooth transitions between interrupt handlers, the tail chaining technique of the Core Local Interrupt Controller (CLIC) enhances the handling of numerous pending interrupts.

The CLIC tail-chaining process begins by initiating the tail-chaining operation, a hardware optimization in which the controller chain-loads the next higher-priority interrupt without returning to the main execution flow. Once an interrupt is detected, the CPU processes the currently pending interrupt by entering the corresponding Interrupt Service Routine (ISR), saving the necessary context, and executing the required service operations. While the current ISR is running, the CLIC logic continues to scan for new interrupt requests. If a new interrupt is detected, the controller performs priority arbitration to determine whether the new request has a higher priority than the one currently being serviced. When a higher-priority interrupt is identified, the controller seamlessly transitions to handle it using the existing stack frame, avoiding redundant register reload and save operations. This approach significantly reduces context-switching overhead, enabling faster transitions between ISRs and lowering latency between consecutive interrupts.

4. Results

To benchmark the performance of the given Core Local Interrupt Controller (CLIC), two simulation waveforms are compared. The first is normal interrupt handling with hardware vectoring disabled (SHV disabled) and the second is vectored interrupt handling with SHV enabled for some of the interrupt sources. Both the simulations were carried out in Vivado simulation.

5. Discussion

To benchmark the performance of the given Core Local Interrupt Controller (CLIC), two simulation waveforms are compared. The first is normal interrupt handling with hardware vectoring disabled (SHV disabled) and the second is vectored interrupt handling with SHV enabled for some of the interrupt sources. Both the simulations were carried out in Vivado simulation.

Fig4: Standard Interrupt Handling of non-vectored CLIC Controller.

In the figure 4, With the Selective Hardware Vectoring (SHV) feature turned off, the first waveform is the typical Core Local Interrupt Controller (CLIC) operation, or clicintattr[i]. shv = 0. In this setup, the mtvec.BASE address is used to route all interrupts to a common software-defined trap handler. Because the intr_src_i signal represents a 256-bit wide interrupt source vector, where each bit corresponds to a distinct interrupt line, it appears as a large decimal value in the waveform. Unless specifically formatted, the entire 256-bit binary vector is automatically interpreted and shown in the simulator as a single large decimal number rather than as separate bits. When wide vectors are grouped for compact display or the signal radix is not set to binary, simulation tools such as Vivado often exhibit this behavior. When an interrupt occurs, the processor first enters a generic trap handler, then, using the decoded interrupt ID, explicitly branches to

the relevant Interrupt Service Routine (ISR) after reading the mcause register to determine the interrupt's source. Because it requires several instructions fetches and conditional branching before the ISR is executed, this software-mediated redirection mechanism inevitably adds latency to the interrupt response cycle.

The simulation waveform shows that there is a latency of roughly 5–6 clock cycles between the assertion of intr_src_i and the emergence of a valid interrupt signal (irq_valid_o) and its acknowledgment (irq_ready_i). When several interrupts, like those with ID's 42, 52, and 32, are asserted in quick succession, this delay is even more noticeable. The overall interrupt response time is prolonged due to increased software overhead and arbitration time caused by the lack of hardware-level prioritization and vectoring logic. As a result, high interrupt traffic has a substantial negative influence on the controller's efficiency, particularly when SHV is not used.

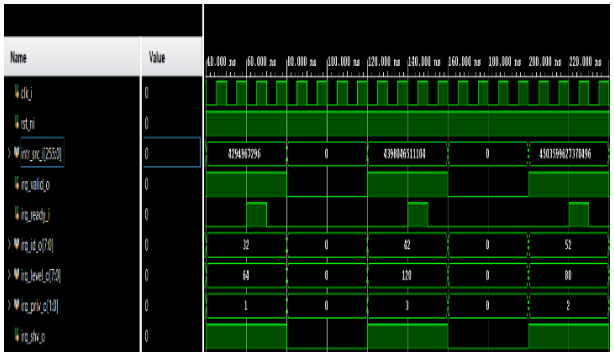

Fig. 5. Optimized Interrupt Handling of non- vectored CLIC Controller.

- The second waveform in Fig 5, illustrates the given CLIC design setup with:
- SHV globally enabled through {cliccfg.shv = 1}
- Per-interrupt vectoring {clicintattr[i]. shv = 1}
- Priority levels set through {clicintctl[i]}
- A fixed interrupt threshold {clicintthresh}.

Here, interrupt 42, 52, and 32 are asserted sequentially. Due to hardware-level priority arbitration, interrupt 42 (highest priority) is handled first. The {irq_valid_o} signal is asserted within 1–2 clock cycles of the interrupt assertion, and the processor branches to the associated ISR vector directly without additional software decoding. The CLIC architecture supports global and local Selective Hardware Vectoring (SHV), enabling vectored interrupt handling when both the global cliccfg.shv bit and the per-interrupt clicintattr[i].shv bit are set. If either bit is disabled, the controller defaults to non-vectored mode, where interrupt handling follows a software-polling routine. When an interrupt request arrives via intr_src_i[i], the controller examines clicintctl[i] to determine its priority and compares it against clicintthresh. Interrupts with priority levels below the threshold are masked, while valid interrupts proceed to arbitration. During arbitration, the CLIC selects the highest-priority pending interrupt and outputs its irq_id_o, along with associated properties such as priority level, privilege mode, and SHV configuration.

For vectored mode (SHV=1), the hardware calculates the ISR address as mtvec.BASE + 4 × irq_id, allowing the CPU to directly jump to the interrupt service routine within one to two clock cycles. This APB-configured CLIC architecture delivers a highly responsive and programmable interrupt handling mechanism that reduces latency, enhances flexibility, and supports dynamic reconfiguration of priorities, modes, and thresholds.

Its deterministic and efficient operation makes it ideal for embedded, safety-critical, and low-power real-time applications, while significantly reducing software overhead compared to conventional fixed interrupt controllers.

Fig 6. Depicts the clock cycle reduction of interrupt pending request from output of CLIC Controller.

Here, interrupt 42, 52, and 32 are asserted sequentially. Due to hardware-level priority arbitration, interrupt 42 (highest priority) is handled first. The {irq_valid_o} signal is asserted within 1–2 clock cycles of the interrupt assertion, and the processor branches to the associated

ISR vector directly without additional software decoding.

TABLE I
APB-ACCESSIBLE CLIC REGISTER MAP

Offset	Register	Function
0x0000	`cliccfg`	Enables SHV; sets global priority bits.
0x0004	`clicintctl[i]`	Sets priority/level for interrupt source `i`.
0x0008	`clicintattr[i]`	Enables SHV and configures edge/level triggering per source.
0x000C	`clicintthresh`	Blocks interrupts below the threshold.

Table 1 shows a close-up comparison of the proposed APB-configured CLIC versus the baseline CLIC with SHV disabled. It points out how the proposed design greatly improves interrupt handling. Through the addition of an APB register interface, each interrupt source can be separately configured for priority level, trigger mode, privilege, and vectoring behavior. Such per-interrupt flexibility is not present in the baseline CLIC, where they are either hardcoded or defined centrally.

Prominently, the design facilitates threshold-based interrupt masking, which masks only interrupts beyond a programmable priority threshold. This is crucial in systems that need to prevent being swamped by low-priority events. Furthermore, the hardware-level tail-chaining functionality reduces context-switching overhead by bypassing all the way back to the main execution flow and passing control to the subsequent pending interrupt directly, significantly reducing interrupt response latency.

TABLE II
COMPARISON OF NORMAL CLIC VS APB-CONFIGURED CLIC

Parameter	Normal CLIC	APB-Based CLIC
Interrupt Latency	5–6 cycles	1–3 cycles
Vectoring Mode	Software trap	SHV hardware vectoring
ISR Address	Decode `mcause`	`mtvec.BASE + 4×id`
Priority Control	Not supported	`clicintctl[i]` enabled
Threshold Filter	Unavailable	`clicintthresh` configurable
Runtime Config	No	Yes (via APB)
Arbitration	Static priority	HW preemption supported
Flip-Flop Use	IRQ state only	Pipelined, stable config
Simulation Behavior	Delayed, sequential	Fast, priority-aware
Scalability	Limited	High, fine-tuned
Applications	Basic embedded	Real-time, critical systems

Table 2 contrasts the new CLIC controller with conventional RISC-V interrupt schemes — CLINT (Core Local Interruptor) and PLIC (Platform-Level Interrupt Controller). Both CLINT and PLIC use software-based trap handling, wherein control is initially passed to an on-chip central trap handler, and the source of the interrupt is subsequently decoded in software.

This generates a significant amount of latency and constrains scalability, particularly when many interrupt sources are involved.

TABLE III
COMPARISON OF PLIC, CLINT, AND PROPOSED CLIC

Parameter	PLIC	CLINT	Proposed CLIC
Latency	10–12 clk	6–8 clk	1–3 clk
Vectoring	No	Not avail.	SHV supported
ISR Redirect	`mcause`	`mcause`	`mtvec + 4×id`
Priority Config	Static	None	`clicintctl`
Threshold	No	No	`clicintthresh`
Sources	1024+	Few	256+ scalable
Arbitration	SW logic	Trap logic	HW arbiter
Runtime Config	Low	None	High (APB)
FF Usage	Minimal	Minimal	Pipelined
RT Scaling	Limited	Poor	Excellent
Applications	OS-level	Timer/SW	RT embedded

Three interrupt controller architectures—the conventional PLIC, the legacy CLINT, and the proposed CLIC controller augmented with APB configuration and Selective Hardware Vectoring (SHV)—are compared in detail in Table 3. The table clearly indicates that the proposed CLIC outperforms the other two in real-time response, flexibility, and interrupt latency. With software-based redirection through the \\texttt{mcause} register and memory-mapped, centralized priority registers, the Platform-Level Interrupt Controller (PLIC) features high servicing latency (10–12 cycles). Similarly, CLINT, which is most commonly employed for timer and software interrupts, employs trap handling and has a moderate latency (6–8 cycles) but does not enable dynamic priority or vectoring. In contrast, the proposed CLIC design employs hardware-level vectoring to provide much lower latency (1–3 clock cycles). Software decoding latency is eliminated when SHV is enabled since the processor employs the precalculated address {mtvec. BASE + 4 × irq_id} to branch directly to the ISR.

Refrences

[1] R. Balas, A. Ottaviano, and L. Benini, “CV32RT: Enabling Fast Interrupt and Context Switching for RISC-V Microcontrollers,” IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 2024.

[2] Y. Cui, Z. Yang, J. Xiong, and L. Zhang, “A Tiny Processor with Rapid Interrupt Response for AI Accelerator Control,” in Proc. 2024 4th Int. Conf. Electronic Information Engineering and Computer Communication (EIECC), Wuhan, China

[3] C. Wang, C. Fang, X. Wu, Z. Wang, and J. Lin, “A Scalable RISCV Vector Processor Enabling Efficient Multi-Precision DNN Inference,” in Proc. 2024 4th

Int. Conf. Electronic Information Engineering and Computer Communication (EIECC), Nanjing, China.

[4] E. Cui, T. Li, and Q. Wei, "RISC-V Instruction Set Architecture Extensions: A Survey," IEEE Access, 2023.

[5] P. Lindgren, P. Dzialo, H. Lunnikivi, and J. Ericsson, "ENEST - Efficient Interrupt Nesting for RISC-V based CPUs," in Proc. 2023 IEEE 2nd Industrial Electronics Society Annual On-Line Conference (ONCON). F. Marques, M. Rodr´ıguez, and S. Pinto, "Interrupting' the Status Quo: A First Glance at the RISC-V Advanced Interrupt Architecture (AIA)," 2024.

[6] F.Conti et al., "MARSELLUS: A Heterogeneous RISC-V AI-IoT End Node SoC With 2–8 b DNN Acceleration and 30%-Boost Adaptive Body Biasing," IEEE Journal of Solid-State Circuits, 2024.

[7] M. Rogenmoser, A. Ottaviano, T. Benz, R. Balas, M. Perotti, A. Garofalo, and L. Benini, "SentryCore: A RISC-V Co-Processor System for Safe, Real-Time Control Applications," 2024.

[8] V. N. Chander et al., "A Soft RISC-V Vector Processor for Edge-AI," in Proc. 2022 35th Int. Conf. VLSI Design and 2022 21st Int. Conf. Embedded Systems (VLSID).

[9] Z. Su, Q. Li, H. Kaneko, H. Li, and L. Meng, "Optimization and Deployment of DNNs for RISC-V-based Edge AI," in Proc. IEEE Int. Conf. Real-time Comput. Robot. (RCAR), Kusatsu, Japan, Jun. 2024.

[10] S. Pandian Annamalai, "Interrupt Handling in RISC-V Architecture," Embien Blog, 2024.

[11] M. Arteaga Castillo, "Interrupt Controller for RISC-V," Master's Thesis, Dept. Computer Architecture, Universitat Politecnica de Catalunya, ` 2020.

[12] M. Nagar, A. Rahman, and V. Garg, "Design of Nested Vectored Interrupt Controller for 32-bit RISC Processor," International Journal of Computer Applications, vol. 182, no. 1, pp. 1–6, Jul. 2018.

[13] A. Olofsson et al., "A Heterogeneous RISC-V Based Open Hardware Platform for Research and Education," in Proc. 2022 Design, Automation and Test Europe Conf. (DATE), pp. 1–6. P. Kumar, R. Kumar, and D. Singh, "Efficient Hardware Architecture for Low Latency Interrupt Handling in Embedded Systems," in Proc. 2023 IEEE Int. Conf. VLSI Design and Embedded Systems (VLSID), Jan. 2023.

[14] S. Wang, J. Liang, and L. Liu, "Design of Low Latency Programmable Interrupt Controller for RISC-V," in Proc. 2022 IEEE Int. Conf. Electron Devices and Signal Processing (ICEDSP), Dec. 2022.

[15] M. M. Shur, D. Nevrly, and K. M. McNeill, "Hardware Support for Nested and Preemptive Interrupt Handling," IEEE Trans. Computers, vol. 72, no. 1, pp. 99–111, Jan. 2023.

[16] L. An, F. Li, and H. Wang, "A Configurable Interrupt Vector Table Mechanism for Real-Time RISC-V Systems," in Proc. 2023 IEEE Int. Conf. Real-Time and Embedded Systems (RTES), Apr. 2023.

[17] D. Yadav and S. Parameswaran, "Reducing Interrupt Latency Through Software-Controlled Interrupt Prioritization in RISC Architectures," in Proc. 2021 IEEE Asia and South Pacific Design Automation Conf. (ASPDAC), pp. 624–629, Jan. 2021.

[18] A. Siddiqui, N. Chatterjee, and M. Shukla, "Efficient Real-Time Interrupt Management in IoT Microcontrollers Using Enhanced NVIC," IEEE Internet of Things Journal, vol. 9, no. 8, pp. 5841–5850, Apr. 2022.