Work in Progress: Enabling Deterministic User-Level Interrupts in Microcontrollers via Hardware Extension

Authors: Hongbin Yang, Huanle Zhang, Tuo Wu, Runyu Pan, Runyu Pan

Date: 2026-03-18T17:21:24+00:00

Abstract

The growing complexity of microcontroller-based embedded systems demands strong isolation of software components into separate protection domains to reduce attack surfaces and limit fault propagation.However, application-supplied device interrupt handlers—even untrusted—have to remain in the kernel to minimize interrupt latency, undermining security and burdening manual certifications.Current hardware extensions accelerate interrupts only when the target protection domain is scheduled by the kernel; otherwise, a protection domain switch is required. Consequently, they are limited to improving average-case performance but not worst-case latency, and do not meet the requirements of safety-critical real-time applications such as autonomous vehicles or medical devices.To overcome this limitation, we propose a novel hardware extension that enables direct, deterministic switching to the appropriate protection domain upon user-level interrupt arrival—without kernel intervention—even when that domain is dormant. Our hardware extension reduces worst-case latency by more than 50x with a 19% increase in core area (7% of total die area) and 4.1% increase in dynamic power.

Full Text

Preamble

Work in Progress: Enabling Deterministic User-Level Interrupts in Microcontrollers via Hardware Extension

Pipeline Interrupt Controller

Register

Timer

Budget Table

Interrupt ID Unit Identification Data CAM

RV Core

are in yellow, and the intra-core extension are in blue.

Clock Interrupt Pipeline Register

exec_{old}

fetch_{int}

regbank_{old}

regbank_{int}

pmp_{old}

Table

Timer

PMP UPDATE

Budget

loading

pmp_{int}

on hold

countdown

BUDGET UPDATE

interrupt activation, with a latency of 11 cycles (2 more cycles are needed for the first fetched interrupt vector instruction to reach the execute stage).

Hongbin Yang and Runyu Pan are with the Shandong University, China. Tuo Wu is with the City University of Hong Kong, China. (e-mail: {hongbinyang}@mail.sdu.edu.cn, {dtczhang,rypan}@sdu.edu.cn, {tuowu2}@cityu.edu.hk). Runyu Pan is the corresponding author.

This work was supported by National Natural Science Foundation of China (62402291, 62302265, U23A20332), and Shandong Provincial Natural Science Foundation (ZR2023QF172, 2024HWYQ-020). We would also like to thank Kexin Li who aided in building an initial prototype.

Fetch

Table

I. I NTRODUCTION Microcontrollers in modern Cyber-Physical Systems (CPS) execute complex binaries that include components from various sources,

possibly resulting from code reuse, multistakeholder development, and cost-saving consolidation. When they run in a single address space, faults or compromised components can serve as trampolines that hijack the system.

To ensure security and reliability, protection domain isolation strategies like microkernels [1], [2] confine software execution. Ideally, interrupt handlers for untrusted components should remain at the user level. However, passing interrupts from the kernel to user processes bloats latencies by tens of times, forcing developers to keep untrusted interrupt handlers in the kernel, or to carefully certify the handler code.

Recent hardware extensions like Intel' s UIPI [3] and RISCV' s N extension [4] allow direct user-level interrupt delivery, but they share a critical limitation: the target protection domain must be currently scheduled. If the target is dormant, the kernel must still intervene to switch to that protection domain, failing to improve worst-case latency. Moreover, these mechanisms cannot preempt the kernel itself.

External MUX

SRAM/ Periph

Core MUX

Flash

Abstract

—The growing complexity of microcontroller-based embedded systems demands strong isolation of software components into separate protection domains to reduce attack surfaces and limit fault propagation. However, application-supplied device interrupt handlers—even untrusted—have to remain in the kernel to minimize interrupt latency, undermining security and burdening manual certifications. Current hardware extensions accelerate interrupts only when the target protection domain is scheduled by the kernel; otherwise, a protection domain switch is required. Consequently, they are limited to improving averagecase performance but not worst-case latency, and do not meet the requirements of safety-critical real-time applications such as autonomous vehicles or medical devices.

To overcome this limitation, we propose a novel hardware extension that enables direct, deterministic switching to the appropriate protection domain upon user-level interrupt arrival—without kernel intervention—even when that domain is dormant.

Our hardware extension reduces worst-case latency by more than 50x with a 19% increase in core area (7% of total die area) and 4.1% increase in dynamic power.

CPU cycles

for Cortex-M33, Cortex-M3, IBEX and our RISC-V core respectively. (b) shows user-level interrupt latencies for the vanilla microkernel software path, plus our core with extension C1, C2 and C3, respectively. It can be observed that our extension performs comparable to kernel-level latencies and is far less than microkernel latencies.

To this end, we introduce a hardware extension that (1) delivers interrupts to user-level without kernel intervention under any circumstances, (2) enforces spatial and temporal isolation, and (3) achieves latencies that rival kernel-level interrupts.

Contributions. This paper presents the first hardware extension enabling efficient and deterministic user-level interrupt delivery without kernel intervention. We explore the hardware design space (§III) and analyze implementation options in a custom RISC-V SoC (§IV), discussing trade-offs between Power, Performance and Area (PPA). Our preliminary evaluation (§V) demonstrates that this approach has the potential to achieve deterministic and low interrupt latencies while maintaining security and reliability.

a balance that meets real-time responsiveness requirements without excessive overheads.

A. Microcontroller-oriented Processor Core To explore the design space, a flexible microcontroller CPU backbone is needed. In this light, we aim to design a RISCV core with a 3-stage pipeline, static branch prediction, a FPU, and a performance around 2.50 CoreMark/MHz (which is common) as our baseline core.

II. BACKGROUND AND R ELATED W ORK User-level Interrupts. Architectural extensions like Intel UIPI [3] and the RISC-V N-extension draft [4] enable userlevel interrupt delivery. However, they optimize for average-case throughput rather than deterministic latency because kernel remains on the slow path.

Hardware Scheduling. While hardware schedulers [5] reduce dispatch latency, they lack accelerated protection domain switching. Existing mechanisms like x86 task gates [3] suffer from high overheads, and TrustZone-based approaches [6] are limited to two protection domains, as well as lacking budget enforcement. Besides, we do note that numerous work have addressed the latency problem by approaches such as register banking, improved controller design or optimized stacking schemes, however they do not consider user-space.

Limitations of Current Isolation. Microkernel-based approaches [7] and language-based sandboxes [8], [9] provide isolation but incur software switching overheads reaching hundreds of cycles. These latencies prevent sub-microsecond response, necessitating hardware acceleration.

Threat Model. We assume mutually distrusting user-level handlers attempting

spatial (illegal access) or temporal (CPU hogging) violations.

Task Model. Run-to-completion aperiodic user-level interrupt handlers are hosted on deferrable servers with strict budget constraints, which are enforced by the hardware extension.

III. S YSTEM D ESIGN The primary objective of this extension is to ensure deterministic user-level interrupt delivery without kernel intervention. It strives to achieve the following goals:

G1: Deterministic and low latency. User-level interrupt handlers must start execution with a deterministic and low latency upon activation, regardless of their target protection domain.

G2: Spatial confinement. User-level interrupt handlers must have their untrusted memory and device accesses strictly confined.

G3: Temporal confinement. User-level interrupt handlers must execute within a rigidly enforced budget.

G4: Minimal hardware overhead. The extension must incur minimal silicon area and power overheads, respecting the Size, Weight, and Power plus Cost (SWaP-C) constraints.

Latency vs. Area vs. Power Trade-off. Lower interrupt latency typically comes at the cost of increased hardware complexity and power consumption to support faster operation. We aim to characterize the latency–area–power trade-off and strike

B. Hardware Extension Synopsis The extension allows confining specific user-level interrupt sources to (1) a Spatial Protection Domain (SPD) described by a set of Physical Memory Protection (PMP) configurations (G2), and (2) a Temporal Protection Domain (TPD) enforced by a hardware budget countdown timer (G3), which will be switched to when the interrupt source is activated, then the whole register set will be saved by the hardware. We showcase the operation of the extension's internal mechanisms with a concrete implementation that we call the C3 variant (see §IV.C). Its block diagram and timing waveform are shown in Upon interrupt entry, the Interrupt Controller (IC) calls upon the Interrupt Identification Data Unit (IIDU) to distinguish 1 whether this is a user-level interrupt. If it is, IIDU identifies 2 the address of the PMP configuration and budget value from the data structure it manages, which is a Content Addressable Memory (CAM) in this variant, and notifies the IC 3 . Subsequently, the IC triggers 4 parallelizable hardware operations where (1) the execution budget is loaded 5 , in this variant, from Tightly Coupled Memory (TCM), (2) the PMP configuration is loaded 6 from its TCM as well, (3) the registers are saved by switching banks or spilling to memory 7 then clearing. Once all completed, a finish signal is sent 8 to the IC, after which it resumes 9 the interrupt vector and the timer countdown. When the interrupt vector executes, if (1) the timer counts down to zero, or (2) the PMP detects access violation, a return is forced to maintain isolation.

C. User-level Interrupt Identification Identifying which interrupt source should be directed to the user-level and what protection domain it targets requires querying a hardware-managed Interrupt Identification Data (IID) structure managed by the IIDU. The design must balance lookup latency (G1) against silicon area (G4), as this step is on the critical path.

Location: CPU vs. RAM. Placing IID in CPU registers ensures deterministic, single-cycle lookup for its target protection domain information, but increases the core area. Conversely, storing IID in RAM saves logic area but introduces additional bus latencies, undermining G1.

Organization: Linear table vs. CAM. A linear table scales with the total interrupt vector size, wasting space for sparse user interrupts. A Content-Addressable Memory (CAM) compresses storage effectively but requires parallel comparator logic, which can only be implemented within the CPU.

Combining these factors, we dismiss the area-prohibitive CPU-Linear approach and the latency-prohibitive RAM-CAM approach. Instead, we propose the hybrid architecture illustrated in the Interrupt Identification Unit of Fig. 1 [Figure 1: see original paper], narrowing the design space to two viable options: (1) CPU-resident CAM: prioritizes latency and determinism (G1) via parallel lookup. (2) RAM-resident Linear table: prioritizes area and power efficiency (G4) at the cost of potential memory latencies. To maintain compatibility with RISC-V standard, we choose to separate this from the interrupt vector table.

D. PMP and Budget Table The PMP table stores spatial configurations for each domain (G2), and the budget table enforces temporal isolation (G3).

Due to their size, they must reside in memory, necessitating a trade-off between bus interface complexity and bus contention possibility.

Main SRAM. Mapping the tables to main SRAM via the standard interconnects simplifies the system design. However, since the hardware parallelizes context saving and protection domain data loading, sharing the SRAM bus creates contention, introducing latency spikes (G1).

Tightly-Coupled Memory. To eliminate bus contention, the PMP and budget table could be placed in one shared or two dedicated SRAM blocks. However, this requires additional memory ports, complicating the bus design (G4). Fortunately, many processors come architecturally with optional instruction or data TCM ports.

E. Register Context Saving To prevent untrusted user-level handlers from peeking the register set of threads or handlers that belong to other processes, the hardware must save the full register set and zeroize it, which is different from the kernel-level ARM FIQ approach that only partially banks the registers. Given the size of the register set, this is the primary bottleneck for latency (G1), and conflicts with other activities must be avoided.

Shadow Banking. Employing dedicated register banks for interrupt use enables

single-cycle context switching. While this eliminates bus accesses, it incurs significant area overhead (G4) and limits interrupt nesting depth.

Hybrid Spilling. To support unbounded nesting without unbounded area, we implement a fallback mechanism. When shadow banks are exhausted (or no bank at all), the hardware automatically spills the context to memory. The spill destination presents a secondary trade-off: (1) Main SRAM: preserves standard interfaces (G5) but suffers from potential bus contention. (2) Dedicated TCM: ensures determinism (G1) but increases bus interface complexity.

IV. I MPLEMENTATION A. Implementation Primer In the exploration, we synthesize and lay out our core on the Nangate45 platform using OpenROAD [10] to derive its silicon area, and on Xilinx XC7K410T Field-Programmable Gate Array (FPGA) to obtain runnable instances. Then, we derive the interrupt latency of the implementation, and use both silicon area and latency to guide our search.

B. Baseline RISC-V Core To evaluate the extension, we must first evaluate the baseline CPU core to (1) represent the microcontroller cores on the market in terms of silicon area and performance, and (2) derive a baseline silicon area and power footprint that we can use to evaluate the overheads of our extension. We implemented a RISC-V processor supporting the RV32IM instruction set, and it measured 2.84 CoreMark/MHz when realized with the FPGA, which is in accordance with our expectations in §III. The core also integrates a FPU that is connected but not powered, clocked or supported at the instruction level, which is for silicon area accounting only. As shown in Fig. 3 Figure 3: see original paper, its kernel-level interrupt latency is less than Cortex-M3 [11] and Cortex-M33 [12], and is on par with IBEX, which is an open-source RISC-V core.

C. Curated Implementation Variants TABLE I: Characteristics of different processor configurations, including those with different extension variants. Floorplan area is in mm2 , interrupt latency is in CPU cycles, and power is in static (mW) / dynamic (µW/MHz) format. "Base" refers to our RV32IM core, and "Vanilla" refers to the microkernel software path in [7].

Configuration Base+C1 Base+C2 Base+C3 Vanilla

Interrupt latency 5 (kernel-level)

Floorplan area 0.0928 (100%) 0.1112 (119%) 0.1111 (119%) 0.1398 (151%)

Power 0.934/30.48 1.27/31.75 1.27/32.60 1.78/33.75

Among all possible variants, we implemented and preliminarily tested the following variants on FPGA after analyzing the critical paths deciding interrupt latencies.

C1: IID as table, register context on stack, no extra TCM ports. This variant has the minimum area and requires no changes to the memory interface; however, it has the longest latency.

C2: IID as table, register context on stack, dedicated TCM for stack. This variant offers significantly decreased interrupt latency at the cost of one additional TCM port, which changes the memory interface.

C3: IID as CAM, banked register context, dedicated TCM for protection tables, optional dedicated TCM for register spills. This variant offers the shortest interrupt latency yet has the largest area and power penalty, as well as two additional TCM ports.

Table I lists the latency, area and power of these variants, with the user-level interrupt latencies depicted in Fig. 3(b). We also port the open-source microkernel in [7] to our CPU, and list its interrupt latency as "vanilla". The latency of all three variants are well below 50 cycles (250ns @ 200MHz), far below what is achievable by software. Notably, the latency of C3 is even comparable to the kernel-level interrupt latency of Cortex-M3 and Cortex-M33. The power metrics were measured using OpenROAD' s built-in flow, simulating a computation of a single-layer convolutional neural network on the processor. The results show that static power increases with silicon area, while dynamic power, which is the primary factor, grows moderately, consistent with the fact that our extension only activates when an interrupt occurs.

V. E VALUATION P LAN Since this is a work-in-progress, our current focus is on implementing working prototypes and validating the functional and timing correctness of them.

A. Research Questions We aim to answer the following questions:

Q1: Does the extension exhibit consistent low interrupt latencies regardless of the currently scheduled process?

Q2: Does the extension terminate the handlers after they break spatial or temporal isolation?

Q3: Is the area cost acceptable, given the fact that CPU is a portion of the entire chip?

B. Experiment Setup The aforementioned RISC-V processor with the three extension variants are implemented on a XC7K410T FPGA with 128KiB of RAM and 256KiB of Flash, which is simulated with RAM. Tests are compiled by GCC 15.2 with -O3.

Interrupt Latency (Q1; ongoing). We port an existing open-source microkernel to our CPU core, and compare our extension variants against existing approaches: (1) the vanilla user-level interrupt passing through typical microkernel facilities [7], (2) the Intel style which precludes kernel intervention only when the current process is active [3], and (3) alternative approaches that deprivilege interrupts via software that do not assume a microkernel abstraction [9]. All extension variants should exhibit a low and constant latency regardless of the currently scheduled protection domain.

End-to-End Applications (Q1; ongoing). We use two common industry control tasks to evaluate the effectiveness of all extension variants against those aforementioned approaches. (1) Pulse Train Output (PTO): A stepper motor control task requiring timely square wave output. We expect higher attainable wave frequency as well as near-zero jitter under heavy background task load. (2) Modbus-RTU: Evaluating the maximum sustainable baud rate and response time while running background image recognition tasks. We expect our extension to maintain a higher baudrate while having lower response jitter; when the baud rate is given, we expect our extension to spare more CPU for the background task, resulting in a higher framerate.

Isolation Effectiveness (Q2; complete). We constructed user-level interrupt handlers with infinite loops and illegal accesses, and tested them with the FPGA implementation. All variants confined the execution within their protection domains by forcing the handler to return upon violation.

Microcontroller Decap (Q3; complete). We decapped four microcontrollers implemented using 40nm technology (more compact than our 45nm) and obtained their die area, to evaluate the area impact of our extension on the entire die.

It could be observed from Table II that CPU core accounts for a tiny portion of the total die area, particularly noting that our core uses 45nm process technology. As a result, our extensions account for less, and is negligible in some cases.

Even if we assume a much smaller die such as 1mm2 , the C3 extension would account for 7% of extra area.

TABLE II: Total die area of typical microcontrollers. Length and

width are in mm, while area are in mm2 . Core% are the portion of our base core on the same assumed die area, C1%, C2% and C3% is the extra area that the C1, C2 and C3 extension variant would require.

The reason for a very small Core% is because microcontrollers integrate RAM, Flash and various (possibly analog) peripherals.

STM32H743 LPC55S69 ESP32C3 TC387

L×W = Area 5.06$×4.70 = 23.782.90×2.78 = 8.062.28×2.27 = 5.187.64×$7.39 = 56.46

Core%

C1% / C2% / C3% 0.08% / 0.08% / 0.1% 0.23% / 0.23% / 0.8% 0.36% / 0.36% / 0.9% 0.03% / 0.03% / 0.08%

VI. T ENTATIVE C ONCLUSIONS This paper proposes a hardware extension that aims to achieve deterministic user-level interrupt delivery. Through preliminary design and implementation, we confirm the feasibility of the proposal and present three variants. In future investigations, we plan to evaluate the performance of these variants through benchmarking on synthetic tests and realworld applications.

We expect the extension to provide a comprehensive solution for secure, reliable and deterministic microcontrollerbased cyber-physical systems, and welcome feedback on the design trade-offs and evaluation methodology. We also note that similar extensions might be possible for microprocessor-based systems as future research.

R EFERENCES [1] H. Karlsson and R. Guanciale, "Partitioning kernel with capability controlled temporal and spatial partitioning," in 2025 IEEE 46th RealTime Systems Symposium (RTSS), 2025. [2] J. Li, R. Hou, G. Shang, H. Zhang, X. Cheng, and R. Pan, "FVM: Practical Feather-Weight Virtualization on Commodity Microcontrollers," IEEE Transactions on Computers, 2025. [3] Intel, "Volume 3: Full system programming guide," Intel 64 and IA-32 Architectures Software Developer' s Manuals, 2025. [4] A. Waterman, K. Asanovic, and J. Hauser, "Volume II: Privileged architecture," The RISC-V Instruction Set Manual, 2025. [5] Y. Tang and N. W. Bergmann, "A Hardware Scheduler Based on Task Queues for FPGA-Based Embedded Real-Time Systems," IEEE Transactions on Computers, vol. 64, 2015. [6] S. Pinto, H. Araujo, D. Oliveira, J. Martins, and A. Tavares, "Virtualization on TrustZone-enabled microcontrollers? Voilà!" in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2019. [7] R. Pan, G. Peach, Y. Ren, and G. Parmer, "Predictable virtualization on memory protection unit-based microcontrollers," in 24th IEEE RealTime and Embedded Technology and Applications Symposium (RTAS), [8] A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis, "Multiprogramming a 64 kB computer safely and efficiently," in Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), 2017. [9] Z. Ma, G. Chen, Z. Chen, and L. Zhong, "Hopter: a safe, robust, and responsive embedded operating system," in Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services, 2025. [10] T. Ajayi, D. Blaauw, T. Chan, C. Cheng, V. Chhabria, D. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça et al., "OpenROAD:

Toward a self-driving, open-source digital layout implementation tool chain," Proc. GOMACTECH, 2019. [11] ARM Limited, Cortex-M3 Technical Reference Manual, ARM Limited, Cambridge, UK, 2007, available at https://developer.arm.com/ documentation/ddi0337/e. [12] J. Yiu, "Chapter 8 - exceptions and interrupts—architecture overview," in Definitive Guide to Arm® Cortex®-M23 and Cortex-M33 Processors, J. Yiu, Ed. Newnes, 2021, pp. 299–340. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128207352000081

Note: Figure translations are in progress. See original paper for figures.