Proceedings of CONF-FMCE 2026 Symposium: Smart City and Infrastructure Engineering
DOI: 10.54254/2755-2721/2026.CH37413

Embedded Microprocessor Technology Based on the RISC-V Instruction Set Architecture

Jinghui Yi

Wuhan Britian-China School, Wuhan, China
1070590389@qq.com

Abstract. In recent years, edge computing has been developing very fast, and it is being used in smart homes, wearable devices, and self-driving cars, so these applications are putting higher and higher demands on embedded microprocessors, especially in terms of power saving, quick response, and customizing for specific uses. This paper mainly gives a general review of embedded microprocessor technologies that are based on the open RISC-V instruction set architecture. First of all, it introduces the basic structure of embedded microprocessors and the main modular design ideas of the RISC-V instruction set architecture (ISA); then, it summarizes recent research progress in three important technical areas, which are processor core design, interrupt handling methods, and lightweight branch prediction techniques. The main findings show that hardware-optimized interrupt schemes have cut down latency to a level similar to proprietary ARM cores, and at the same time, area-efficient branch predictors can also effectively reduce pipeline stalls in situations where resources are limited. However, there are still two big problems, namely ecosystem immaturity and architectural fragmentation, and these remain important obstacles that prevent wider use. In the end, the paper talks about possible solutions, including the ratification of standardized extensions and the development of unified software toolchains, and it is expected that these efforts will help RISC-V become a more stable foundation for next-generation embedded computing systems.

Keywords: Embedded microprocessor, RSISC-V, Edge-computing hardware

1. Introduction

Over the past ten years or so, edge computing has been expanding rapidly, and this has caused a huge growth in the use of embedded microprocessors in many different application areas. Smart home controllers, wearable health monitors, and advanced driver-assistance systems, for example, all have to work under strict limits on power use, chip size, heat dissipation, and cost per unit, and these limits put very high demands on the instruction set architecture (ISA) that lies underneath [1]. In the past, the embedded processor market was almost completely controlled by proprietary ISAs; the ARM architecture was basically the industry standard, while MIPS and PowerPC took up some smaller market segments. These commercial architectures have the advantage of mature software ecosystems, lots of peripheral support, and well-optimized compiler toolchains. But their licensing models, which usually involve large upfront fees and royalties for each unit, actually limit product

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.CH37413

innovation and also make it take longer to bring customized solutions to market [2]. What is more, as Internet of Things (IoT) applications become more and more diverse, the fact that these proprietary ISAs are closed and cannot be easily extended has turned into a serious limitation [3].

The RISC-V ISA was first developed at the University of California, Berkeley, and it offers a basically new way to solve these problems that have existed for a long time [1]. As an open standard, RISC-V uses a modular design idea; its base integer instruction set is deliberately kept very small and simple. It also supports quite a few standard optional extensions, which can configure the processor core in a flexible way to meet the needs of their target applications [1]. This design approach fits very well with the varied demands of embedded systems, where different use cases need very different balances between performance, area, and power. According to a market analysis report from 2026, the global RISC-V market is expected to keep growing at a compound annual rate of over 30% between 2026 and 2035, and the main growth is coming from automotive electronics, industrial IoT infrastructure, and consumer edge devices [4].

In both industry and academia, progress in RISC-V-based embedded processors has moved forward quite quickly during the past eight years. Back in 2018, SiFive launched the E31 Coreplex, which was one of the first commercial RISC-V microcontroller cores made specifically for embedded use, and this showed that the open ISA model could work commercially. After this important step, SiFive then released the E76/E77 series, which used a dual-issue superscalar architecture that could provide higher instruction throughput; they also came out with the 64-bit S7 series, which had memory protection units for better system security [5]. Then in 2019, Western Digital made its SweRV core series open source, and this provided real evidence that in-order RISC-V cores could reach performance and power efficiency similar to existing ARM Cortex-M series products in actual storage controller applications [6]. More recently, the CV32E40P core, which was developed together by the PULP Platform and the OpenHW Group, has become a kind of standard reference design for RISC-V microcontroller research, and it offers a fairly balanced mix of performance, area efficiency, and ecosystem support [7]. The RISC-V ecosystem has developed from just a conceptual experimental platform into a competitive alternative in the embedded processor field.

Even though there has been quite a lot of research progress as mentioned above, several important technical bottlenecks still have not been solved. For example, the interrupt latency of basic RISC-V microcontrollers, which use a single-entry software-dispatch model, is slower than that of ARM Cortex-M series chips that have dedicated hardware interrupt controllers, and this limits their use in hard real-time control situations. Branch prediction technology, which is widely used in high-performance cores to reduce pipeline stalls, has not been studied enough for embedded RISC-V processors that have limited resources, where both area and power budgets are very tight. In addition, the modularity and extensibility that are RISC-V's main advantage also bring the risk of architectural fragmentation, which could hurt software portability and make it harder to build a unified ecosystem.

Based on this background, this paper gives a comprehensive review of embedded microprocessor technology that is based on the RISC-V ISA.

2. Fundamentals of embedded microprocessors

2.1. Basic architecture of embedded microprocessors

When memory and peripherals are put together on one single chip, people usually call an embedded microprocessor a microcontroller unit, or MCU for short, and this kind of processor is mainly used

DOI: 10.54254/2755-2721/2026.CH37413

to do some specific control jobs inside a bigger electronic system. The processors used in personal computers or servers are different, they are general-purpose ones, but embedded processors do not care that much about how fast they can compute; instead, they pay more attention to things like whether the behavior is predictable, whether the power use is low, and whether the chip size is small enough [2]. The typical embedded processor core works with several parts. There is an instruction fetch unit, its job is to get instructions from the memory. Then there is a decode stage, which changes those instructions into control signals. The execution stage comes next, and it has arithmetic logic units, also called ALUs, and sometimes multiplier and divider units too. A register file is also there, it stores operands quickly so that the processor can get them fast. And finally, there is a load/store unit, this one takes care of accessing the data memory.

Embedded systems organize their memory in a way that is different from application processors. As shown in Figure 1, most embedded microcontrollers go with a Harvard architecture, under this architecture, instruction memory and data memory are put in separate address spaces, so the processor can read both at the same time. Because of this separate design, there will not be structural conflicts when the processor fetches instructions and loads data at the same moment, and the execution becomes easier to predict, this matters a lot for real-time systems. The memory on the chip is usually either static random-access memory, that is SRAM, or embedded flash; it does not have a very large capacity, but the good thing is that you can access it in just one cycle. Besides that, the memory map also has some memory-mapped I/O regions, the processor uses these regions to talk to peripherals, for example, universal asynchronous receiver-transmitters (UARTs), inter-integrated circuit (I2C) controllers, and analog-to-digital converters (ADCs) [2].

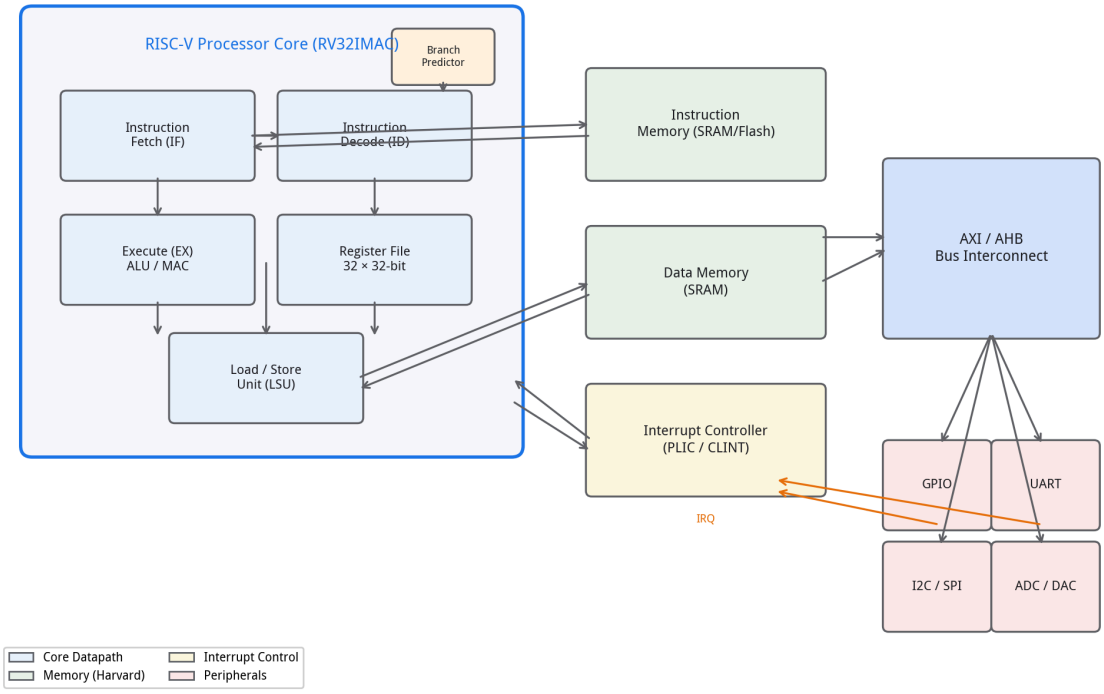

Figure 1. Architecture of a RISC-V-based embedded microcontroller, illustrating the Harvard memory organization, interrupt controller, and peripheral interconnect (picture credit: original)

Another thing that makes embedded processors special is how they handle interrupts. There are always external events happening, like when sensor data comes in, when a timer runs out, or when there is a communication request, and the processor has to react to all these things within a certain

DOI: 10.54254/2755-2721/2026.CH37413

amount of time. To make this work, there needs to be a special interrupt controller; it has to do several things, first it decides which request is more important, then it saves what the processor was doing before, and after that it jumps to the correct interrupt service routine. Whether this whole mechanism works efficiently or not has a direct influence on if the processor can be used in real-time applications.

2.2. Overview of the RISC-V instruction set architecture

What RISC-V is most known for is probably that it keeps the minimal base instruction set separate from those optional standardized extensions [1]. People use the letter "I" to denote the base integer instruction set, and there are only 47 instructions in it, which might sound very few, but actually these 47 instructions are already enough to run a complete general-purpose compiler toolchain. This base set can work with 32-bit, that is RV32I, 64-bit which is RV64I, and even 128-bit called RV128I address spaces; because of this, people can use one unified programming model for everything, from those ultra-low-power microcontrollers to the high-performance processors used in servers.

Standard extensions use a letter-based naming convention; the "M" extension adds integer multiplication and division; the "A" extension supports atomic memory operations; the "F" and "D" extensions provide single- and double-precision floating-point arithmetic; the "C" extension introduces 16-bit compressed instructions, which can cut code size by about 25 to 30 percent, and this effectively saves on-chip memory space in embedded situations. The RV32IMAC configuration is the most commonly used solution for microcontroller-class cores [1, 3].

Besides standard extensions, RISC-V also keeps some opcode space for vendor-specific custom extensions, which allows designers to add acceleration instructions for specific applications, such as digital signal processing or cryptographic algorithms. This flexibility is a main advantage of RISC-V, but at the same time, it also brings some potential compatibility risks. The privileged architecture specification of RISC-V defines machine mode (M-mode), supervisor mode (S-mode), and user mode (U-mode). Embedded microcontrollers usually only implement M-mode, and they may also have a physical memory protection (PMP) unit for isolating firmware. The interrupt system is usually implemented with a core-local interrupt controller (CLINT) and a platform-level interrupt controller (PLIC) [1].

3. Current research status of RISC-V

3.1. Processor core technology

When developing embedded systems, choosing a suitable RISC-V core is the first and most important design decision, and the main trade-off is between performance, chip area, and power consumption. The RISC-V ecosystem has now developed to the point where it offers a range of open-source and commercial core implementations that can meet different application needs.

Kim et al. did a comprehensive comparative study of five mainstream open-source RISC-V cores, including lowRISC Ibex, PULPino RI5CY, and SiFive E31 [4]. Their evaluation was based on synthesis frequency, resource utilization, and power consumption, which were measured on both field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) platforms, and it provides useful quantitative guidance for designers. The results show that single-issue in-order cores, with Ibex as a representative, offer the best area-performance ratio for simple control and sensor node applications; in a 28 nm process, the core uses less than 15k gate equivalents while still running at a frequency of over 200 MHz. For mid-range situations like motor control and audio

DOI: 10.54254/2755-2721/2026.CH37413

processing, dual-issue cores such as CV32E40P can provide about 1.6 times the instruction throughput, although this comes at the cost of an area increase of about 2.3 times. It should be mentioned that this study is based on synthesis simulation results, so the actual performance after tape-out may show some differences due to physical implementation effects.

For multi-core RISC-V embedded systems that are meant for more computationally heavy workloads, the on-chip interconnection has become the main performance bottleneck. Emil et al. designed a layered AXI interconnect unit for a dual-core RISC-V processor, and it includes separate read and write address channels to support concurrent transactions from multiple cores [8]. Tests on a Xilinx Zynq FPGA platform showed that this structure can improve interconnect throughput by up to 40% compared with the traditional shared bus architecture, while only causing a small increase in resource utilization. This work proves that multi-core RISC-V embedded architectures are feasible, although it is still limited to FPGA verification and has not yet been tested through actual ASIC implementation.

Overall, the RISC-V ecosystem has basically covered core configurations from ultra-low-power single-issue designs to mid-range dual-issue implementations, and the research focus is slowly moving from core design itself toward optimizing the surrounding memory subsystems and interconnection networks.

3.2. Interrupt handling technology

Having a deterministic, low-latency interrupt response is a basic requirement for embedded processors that are used in industrial control, automotive electronics, and other real-time application areas. The basic RISC-V privileged specification uses a single-entry interrupt model; all interrupts, no matter where they come from, jump to the same handler address, and then the software has to figure out which specific interrupt it is and dispatch the right service routine. Although this design is simple in structure, it causes non-deterministic latency and also brings a lot of software overhead when implementing nested interrupts.

Existing optimization research in this area can be divided into three different technical approaches, and each one offers a different balance between how much modification is needed and how much performance gain you get. The first approach involves changing the core pipeline to achieve extremely low latency. Rossi et al. proposed the CV32RT core, which adds a hardware interrupt vectoring mechanism and dedicated shadow register files to the basic design [7]. When an interrupt comes in, the hardware directly looks up the vector table and switches to the dedicated shadow register set, and this gets rid of the software dispatch and stack pushing processes. Tested on a 28 nm ASIC implementation, this design cuts interrupt entry latency from about 24 cycles to less than 6 cycles, achieving a 3.8 times improvement in context switching speed and reaching a level similar to ARM Cortex-M4/M7 cores. However, this method requires deep changes to the core pipeline, which naturally reduces compatibility with standard RISC-V core designs.

For the second way of doing things, people mainly try to make the interrupt controller work better, and they do not touch the core pipeline at all. Gao and his team came up with an optimized interrupt vector table mechanism, this mechanism can handle regular interrupts and also non-maskable interrupts, which are called NMIs for short [9]. In their design, they added an independent NMI vector entry, this entry does not go through the normal priority logic, so for some really serious fault situations, like when a watchdog timer times out or when the clock fails, the latency can be kept very small and predictable. After testing, they found that this method can cut the worst-case interrupt response time by 35%, and it almost does not affect the maximum operating frequency at all; another good point is that it is not hard to port this method to different RISC-V cores. There is

DOI: 10.54254/2755-2721/2026.CH37413

also Helten and his colleagues, they proposed something called the ENEST architecture; in this architecture, a hardware-managed interrupt stack is used, and it can handle as many as 16 levels of nested interrupts, compared with software-based methods, the overhead of nested interrupts goes down by about 60% [10].

To sum up this part, thanks to those hardware-based interrupt optimization methods, the real-time performance gap between RISC-V and those proprietary architectures has become much smaller than before. Right now there are solutions that require different levels of modification, so designers can pick the one that fits their own situation best.

3.3. Branch prediction technology

When there are conditional branches in the code, the pipeline might have to stop and wait, and branch prediction technology is there to reduce this kind of waiting; the basic idea is to guess what the next instructions will be and fetch them before the branch actually gives a result. Those high-performance general-purpose processors often use fairly complex predictors, one example is the tagged geometric-history-length predictor, also known as TAGE, but these predictors need several kilobytes of memory on the chip, which is too much for embedded systems because they do not have much area to spare. So for now, most embedded RISC-V cores either just use static prediction, or they do not have any prediction mechanism at all.

Quite recently, people have come up with some lightweight branch prediction methods, and these are made especially for embedded RISC-V processors; each method tries to find a different balance between how much storage it uses and how accurate its predictions are. Katsaros and Papadopoulou proposed one called the LXOR predictor, how it works is that it takes the XOR result of the global history register and the program counter, and then uses that to index a small pattern history table [11]. This predictor only has 256 entries, taking up 512 bytes of storage, but on the CoreMark benchmark its misprediction rate is just 6.8%, while static prediction has a rate of 12.3%, so it is clearly much better; and in a 40 nm process, it uses less than 3% of the core area. Because it uses so little area, this method works really well for sensor nodes and wearable devices, where the area budget is really tight.

Liu and Wang tried a different approach, they designed a hybrid prediction architecture that puts together a bimodal predictor and a global history predictor, and there is a confidence counter that decides which of the two results to use [12]. On CoreMark, this design gets a misprediction rate of 5.2%, which is more accurate, but you have to pay for it with a little more hardware complexity. Zhang and Chen also did some work in this area, they proposed a trend-based branch predictor; what it does is to put branches into different groups based on how they tend to jump, and then each group gets its own prediction strategy [13]. If you compare this with the traditional two-bit saturating counter predictor, their method can reduce mispredictions by 18%, and the extra resources it needs are almost negligible.

One thing should be mentioned here, most of the studies talked about above were done through simulation or FPGA verification, which means that nobody has really measured the actual power use and timing performance when these designs are made into real ASIC chips, so this still needs more testing later on. In the future, embedded RISC-V cores will probably use deeper pipelines and run at higher clock frequencies to satisfy the increasing demand for performance, and when that happens, lightweight branch prediction may gradually become something that everyone does as a standard optimization.

DOI: 10.54254/2755-2721/2026.CH37413

4. Limitations and prospects

4.1. Ecosystem immaturity

Compared with mature proprietary architectures that have been improved over decades, the software and hardware ecosystem around RISC-V embedded processors is still relatively underdeveloped. Although the RISC-V ISA specification itself is stable and has reached a mature stage, the supporting tools, including compiler optimization backends, real-time operating system (RTOS) ports, debug and trace toolchains, and pre-verified peripheral IP blocks, are much less complete than the ARM ecosystem, which is built around the Keil development environment and the CMSIS hardware abstraction framework [3]. For example, while GCC and LLVM provide basic support for RISC-V, target-specific optimizations like auto-vectorization for custom extensions have not reached the same level of maturity as the ARM Thumb-2 backend. Similarly, although mainstream RTOS such as FreeRTOS and Zephyr have added RISC-V support, the number of complete board support packages and validated driver libraries is still limited, and this significantly increases development costs and time-to-market for teams that are switching to RISC-V for the first time.

The gap in the ecosystem does not just affect technology, it also has a direct impact on business. Even though you do not have to pay for ISA licensing fees, if you add up all the costs, including the effort of developing software, buying or getting the toolchain, and the cost of validation, the total cost of owning RISC-V microcontrollers is often still higher than using similar ARM Cortex-M products [4]. Because of this economic reason, people in cost-sensitive consumer markets have been slow to adopt RISC-V; in those markets, whether the ecosystem is mature usually matters more than what the architecture itself can do.

If people want to fix the problem that the ecosystem is not mature enough, they cannot do it alone, it needs the whole industry to work together. RISC-V International has already created several technical working groups, these groups focus on making standards for things like debug interfaces, trace specifications, boot firmware protocols, and security extension frameworks. The OpenHW Group also has their CORE-V open-source core series, and these cores come with full software development kits and support for development boards, so it has become much easier for developers to get started, and at the same time it gives people a reference platform to test and validate the ecosystem. As big semiconductor companies and software vendors keep putting money into this area, people expect that the ecosystem gap will become much smaller in the next 3 to 5 years [4]. Also, when more and more large-volume applications come in, like automotive electronics and industrial automation, the third-party software and tools will mature faster too, and this creates a good cycle where more adoption leads to a better ecosystem, which then leads to even more adoption.

4.2. Architectural fragmentation

RISC-V can be extended very easily, and this gives people a lot of freedom to customize it in ways that were not possible before, but this same feature also brings a risk, that is, the architecture might become fragmented. Different semiconductor vendors have made their own proprietary custom extensions for different parts of the market; some vendors add AI acceleration, some focus on making cryptographic workloads run faster, and there are also those who add special DSP operations. Because there are so many different extensions out there, binary compatibility between different RISC-V cores becomes a problem; if a program is compiled for one vendor's core and that program uses some custom extensions, it might not run properly on another vendor's core. On top of

DOI: 10.54254/2755-2721/2026.CH37413

that, before a standard extension gets officially ratified, there are often several different draft versions competing with each other at the same time, and even the privileged architecture specification can be implemented a little differently from one core to another; all these things added together make the fragmentation problem even more serious [3].

People should understand that fragmentation and an immature ecosystem are not two separate problems; they actually make each other worse. When the hardware is fragmented, it becomes harder and more expensive to adapt software to different platforms, and this slows down the process of building one unified ecosystem; because the ecosystem is not good enough, fewer companies want to use RISC-V, and this creates a vicious cycle that is hard to break.

If people want to deal with architectural fragmentation, the most important thing is to push extension specifications through the official ratification process at RISC-V International. Once an extension is ratified as a standard, like Vector which is V, Bit Manipulation which is B, and Scalar Cryptography which is K, software developers get stable and unified interfaces to work with, and then programs can run on different implementations without compatibility issues. For situations where people need specific configurations, the industry is also trying to promote something called standardized "profiles"; these profiles basically tell you what combination of the base ISA and standard extensions is recommended for a certain kind of application, for example, there are profiles for embedded use, for application-class processors, and for servers. If vendors are encouraged to follow these standard profiles, and custom extensions are only used when you really need application-specific acceleration, then the ecosystem can have both customization flexibility and software portability at the same time. There are also some industry alliances working on this, one example is the RISC-V Software Ecosystem project, also known as RISE; they help coordinate software adaptation at the upstream level and try hard to make sure that different hardware implementations can work with the same software.

5. Conclusion

In this paper, a systematic review is given of how embedded microprocessor technology has developed, focusing on those that use the RISC-V instruction set architecture. The paper first talked about the basic architecture of an embedded microprocessor and the modular design features of RISC-V. Then, it summarized the research progress in three main technical areas, which are processor core design, interrupt handling mechanisms, and lightweight branch prediction techniques. From what has been found so far, the RISC-V ecosystem already has a fairly complete range of core products, covering everything from ultra-low-power designs to mid-range performance ones; hardware-optimized interrupt schemes have brought latency down to a level that is similar to traditional proprietary architectures, and those area-efficient branch predictors have also made the pipeline work more efficiently without using too many extra resources. Even with all this progress, there are still two big problems that stop RISC-V from being more widely used in commercial products, one is that the ecosystem is not mature enough, and the other is architectural fragmentation. In the future, there are some research directions that look promising, for example, putting lightweight machine learning accelerators into embedded RISC-V cores, making standards for secure boot and trusted execution environments, and developing many-core RISC-V architectures that can do edge inference. The reason why this review matters is that it gives researchers and people working in the industry a structured reference, especially since the RISC-V field is changing so fast. For embedded computing, RISC-V is more than just another choice besides those proprietary ISAs; it is an open and customizable foundation, and with it, the industry can change how embedded processors are designed in a fundamental way.

DOI: 10.54254/2755-2721/2026.CH37413

References

[1] Liu, C., Wu, Y. J., & Wu, J. Z. (2021). A survey of RISC-V instruction set architecture. *Journal of Software, 32*(12), 3992–4024.

[2] Saleh, S., & Al-Haiqi, A. (2024). RISC-V SoC design for IoT: A survey of open-source cores and applications. *TechRxiv*.

[3] Wang, X., Li, Y., & Zhang, H. (2023). RISC-V instruction set architecture extensions: A survey. *IEEE Access, 11*, 12389–12412.

[4] Global Market Insights. (2026, July). RISC-V market size & share report, 2026-2035. *Global Market Insights*. https://www.gminsights.com/industry-analysis/risc-v-market.

[5] Kim, J., Lee, S., & Park, H. (2022). Selecting a synthesizable RISC-V processor core for low-cost hardware devices. *Journal of Information Processing Systems, 18*(4), 521–536.

[6] Western Digital. (2019). SweRV RISC-V core family technical report. *GitHub*. https://github.com/westerndigitalcorporation/swerv-iss.

[7] Rossi, D., Benini, L., & Balas, R. (2023). CV32RT: Enabling fast interrupt and context switching for RISC-V microcontrollers. arXiv:2311.08320

[8] Emil, D., Hamdy, M., & Nagib, G. (2023). Development an efficient AXI-interconnect unit between set of customized peripheral devices and an implemented dual-core RISC-V processor. *The Journal of Supercomputing, 79*, 17000–17019.

[9] Gao, J. X., Wang, C., & Li, T. (2024). Vector-table-based optimization design of regular interrupt and NMI for RISC-V processors. *Microelectronics & Computer, 41*(2), 89-96.

[10] Lindgren, P., Dzialo, P., Lunnikivi, H., & Ericsson, J. (2023). ENEST - Efficient interrupt nesting for RISC-V based CPUs. In *2023 IEEE 2nd Industrial Electronics Society Annual On-Line Conference (ONCON)* (pp. 1-7). SC, USA.

[11] Katsaros, K., & Papadopoulou, M. (2025). Towards memory-efficient and high-performance branch prediction: The LXOR architecture for embedded RISC-V processors. *Journal of Low Power Electronics and Applications, 15*(2), 64.

[12] Liu, M., & Wang, Z. (2025). Design of a hybrid branch prediction architecture for RISC-V processors. *2025 IEEE 14th International Conference on Communications, Circuits and Systems*, 234-239.

[13] Zhang, Y., & Chen, T. Z. (2025). Design of a trend branch predictor for RISC-V processors. *Computer Science, 52*(6), 312-318.