RISC-V Core with ISA Extensions and In-Memory Computing: Design and Verification

Maurizio Capra, Valerio Venceslai, Maria Elena D'Agostino, Francesco De Maldé, Andrei Roibu, Alessandro Aimar
e-mail format: name.surname@synthara.ai

Introduction and Project Objectives

RISC-V has gained significant attention due to its flexibility, scalability, and community-driven development model. The growing demand for application-specific optimizations in today's diverse computational landscape, including edge AI and IoT applications, necessitates the creation of custom ISA extensions [1].

	Conventional Microcontrollers ARM, RISC-V, others	NPU cores, AI/DSP accelerators	ComputeRAM™ Microcontrollers
Ease of use	✓		✓
Compute performance		✓	✓
Low silicon cost	✓		✓
Extensibility to new workloads	✓	?	✓

Table 1. *Comparison of different solutions in terms of ease of use, performance, cost, and extensibility*

Using a coprocessor to implement these custom extensions can offload specialized tasks from the main CPU, allowing for parallel processing and freeing the primary core to handle general-purpose workloads. This approach boosts overall system performance and helps in managing power consumption more effectively.

In addition, by performing computations directly within the memory where data resides, in-memory computing (ComputeRAM™ in our implementation) reduces the data transfer bottleneck, leading to faster and more energy-efficient processing.

Current solutions often struggle with either usability or performance (Table 1), yet research highlights the efficiency gains from customized RISC-V extensions and in-memory computing for advanced processing architectures.

This poster presents a **RISC-V system enhanced with Synthara's C** ComputeRAM™, which provides compute-in-memory capabilities and custom ISA extensions, aiming to boost performance in MCU-based edge systems with low energy consumption. The **RISC-V core**, the **co-processor**, and the **verification environment**, explained in the following sections, are released **as open-source**.

System Architecture Overview: from Design to Verification

The system architecture (Figure 1) showcases the interaction between various components within the RISC-V based system. At the core is the **RISC-V CPU** (CV32E20 [2]), which communicates with a specialized **co-processor** (CO-PROC) through the **CV-X-IF** interface. The memory subsystem features several tiles of ComputeRAM™, an advanced in-memory computing technology that performs computations directly within the memory array, significantly reducing data transfer bottlenecks and improving efficiency.

The architecture is interconnected via the AMBA/Peripheral Interconnect, facilitating communication between the CPU, co-processor, memory subsystem, and various peripheral units such as the Event Unit, JTAG, uDMA, Timer, Debug Unit, and I/O interfaces. This interconnect ensures efficient data flow and coordination among all components, enabling the system to achieve high performance and low power consumption.

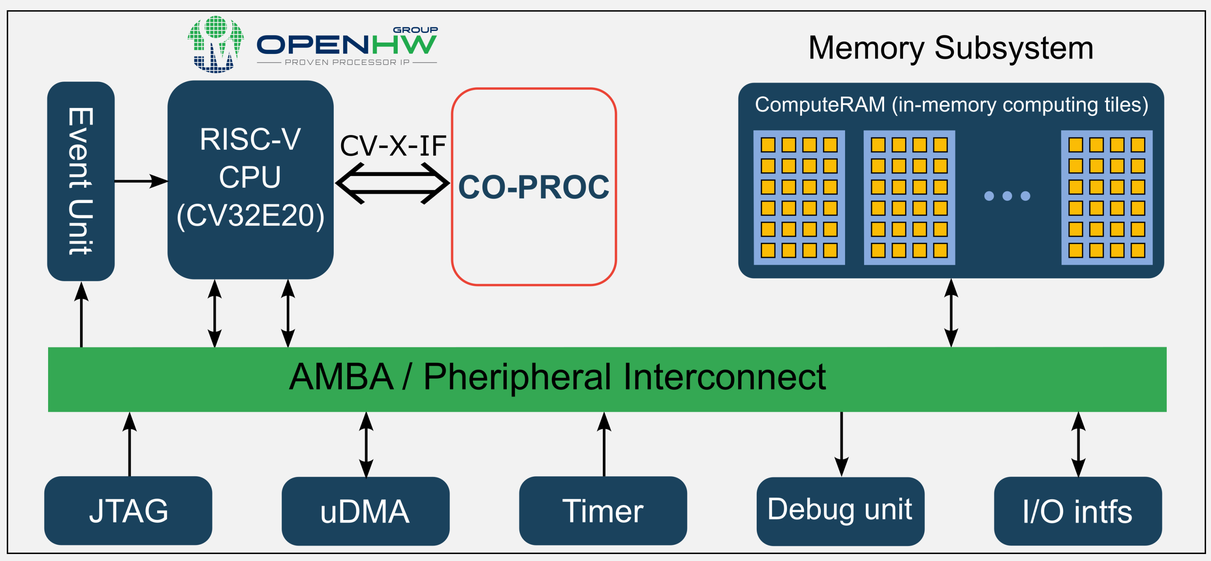

Figure 1. *Generic system overview of a microcontroller system embedding a RISC-V CPU, a coprocessor and one or more in-memory computing tiles*

ISA extension

Our co-processor is equipped with a custom Instruction Set Architecture (ISA) that includes several non-standard operations not ratified by any official RISC-V organization. These operations support different data parallelism modes, such as half-word (16-bit), byte (8-bit), and mixed precision (16-bit and 8-bit).

This flexibility allows for optimized processing of various computational tasks, enhancing performance and energy efficiency. Key operations include a selected subset of Xpulp_V2 [3] and pseudo-SIMD operation [4], where pseudo indicates that the data bit width is dynamic and programmed on the fly in the CSR, rather than embedded in the instruction itself.

The enhancements included in the ISA extension are:
- **arithmetic functions** like absolute value, minimum, maximum, and complex multiply-accumulate operations.
- 32, 16 and 8 bits **MAC operations** with quantization and rounding
- **pseudo-SIMD** operations with mixed precision (16x8 bits) for **arithmetic, logic and shift functions**
- **pseudo-SIMD dot-product operations**.
- **Stream Semantic Register (SSR)** [5] for load/store address generation automation and code reduction.

In-memory computing

In this system, the ComputeRAM™ macro, an advanced in-memory computing tile, is employed. ComputeRAM™ enables computations to be performed directly where the data resides, eliminating the need for extensive data movement between the CPU and memory. This integration accelerates processing and enhances energy efficiency, making it ideal for edge AI and IoT applications.

Compute features:
- **Bit-accurate computation**
- **Matrix-vector product** computation primitive
- **Intermediate** and **full-precision** modes
- **Fixed point**, **signed**, and **unsigned** integer data types, as well as **32, 16, 8** bit width and mixed precision **16x8** operation supported

The system-level improvement factors attributed to the use of ComputeRAM™ are presented in Figure 2 (right). When ComputeRAM™ is used instead of a traditional SRAM, improvements between **11.74x and 139.72x** can be achieved in terms of **latency**, and between **12.23x and 158.7x** in terms **of energy efficiency**.

This macro is seamlessly integrated into the existing memory hierarchy, providing a drop-in replacement for conventional SRAM with minimal area overhead and substantial computational benefits. ComputeRAM™ is built with GlobalFoundries 22FDX process, with Memory Compiler and FinFET variants under development.

ComputeRAM™'s SDK includes libraries that support writing target independent code (Figure 2 left), enabling programmers to develop and optimize neural networks for ComputeRAM™ enabled devices in PyTorch and TensorFlow, boosting performance and energy efficiency for linear algebra-intensive applications.

ComputeRAM code for *A * B = C*

```
C/C++

int *matrixA;
int *vectorB;
int *vectorC;

configureComputeRAM(. . .)

// matrixA is kept in-memory
memcpy(matrixA,   . . .)

// Continuosly compute C = A*B
while(device_is_running)
      // Load new vectorB
      memcpy(vectorB,   . . .)
      // Compute is self-triggered
      memcpy(. . . ,   vectorB)
```

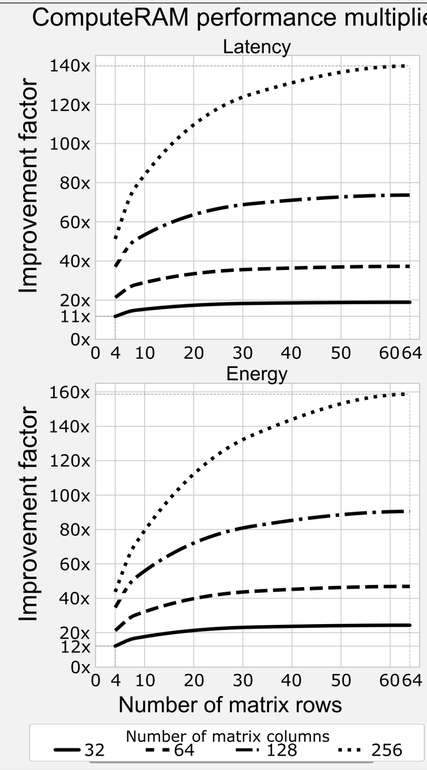

Figure 2. *Left: ComputeRAM™ code example for matrix-vector multiplication using the provided libraries. Right: Gate-level simulation MVM improvement factors achieved by a microcontroller-based system with ComputeRAM™ vs regular SRAM configuration, calculated as the ratios between the two.*

Verification Methodology

The verification process leverages SystemVerilog and UVM methodologies. Starting with Google's RISC-V-DV random instruction generator [6] or specific test cases, the source code is compiled using an LLVM toolchain. This toolchain accommodates both the ISA extension and the in-memory compute tiling to produce an executable .elf file. This executable is used to set up the Device Under Test (DUT) simulation and the instruction set simulator (ISS), which in this case is Spike. The simulation runs in step-and-compare mode to verify the correct functioning of the core. Concurrently, a **UVM-based RISC-V model**, fully written in SystemVerilog, is developed to replace the ISS in a unified UVM-closed simulation environment.

This model features:
- **Modular Design**: The CPU model is structured in a modular fashion, facilitating easy integration and improvement.
- **Customizability**: It allows for extensive customization, enabling tailored adjustments to meet specific project requirements or design variations.
- **Full SystemVerilog and UVM Compliance**: Ensures compatibility and standardization across verification processes.
- **Multiple Model Views**: The scoreboard provides multiple views of the CPU model, offering different levels of detail based on verification needs.
- **Stage-Specific Detailing**: Detailed information from each stage of the CPU model is accessible during verification, enhancing the depth and accuracy of the verification process.

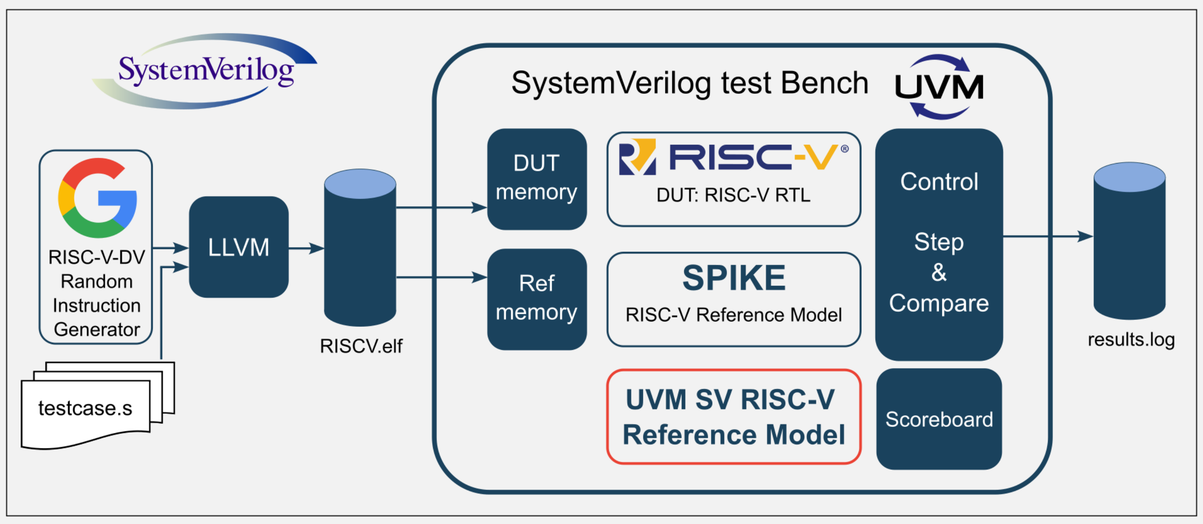

Figure 3. *Testbench overview and verification methodology for the RISC-V core using Google RISC-V-DV, LLVM, SPIKE, and a custom UVM SystemVerilog model..*

Conclusions

This poster outlines the ongoing efforts and future goals in integrating custom ISA extensions and in-memory computing into a RISC-V-based system architecture. These innovations aim to significantly enhance processing speed and energy efficiency, crucial for meeting the demands of modern computational tasks, particularly in edge AI and IoT applications.

The ComputeRAM™ macro represents a promising in-memory computing solution designed to reduce latency and energy consumption for matrix-vector multiplications. Early benchmarks indicate that ComputeRAM™ could drastically improve performance, positioning it as an ideal component for boosting the capabilities of microcontroller-based systems without extensive modifications to existing memory architectures. Its potential for seamless integration as a drop-in replacement for conventional SRAM highlights its practicality and effectiveness.

It be noted that the RISC-V core, the co-processor, and the verification environment will be released **open-source** as part of the TRISTAN project [7].

References

[1] Benini, L. et al. (2017). "PULP: A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision." IEEE Journal of Solid-State Circuits, 52(1), 150-164.
[2] https://docs.openhwgroup.org/projects/cve2-user-manual/en/latest/
[3] Gautschi, M., Schiavone, P. D., Traber, A., Loi, I., Pullini, A., Rossi, D., Flamand, E., Gurkaynak, F. K., & Benini, L. (2016). A near-threshold RISC-V core with DSP extensions for scalable IoT Endpoint Devices. ArXiv. /abs/1608.08376
[4] G. Ottavi, A. Garofalo, G. Tagliavini, F. Conti, L. Benini and D. Rossi, "A Mixed-Precision RISC-V Processor for Extreme-Edge DNN Inference," 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Limassol, Cyprus, 2020, pp. 512-517, doi: 10.1109/ISVLSI49217.2020.000-5
[5] Schuiki, F., Zaruba, F., Hoefler, T., & Benini, L. (2019). Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores. ArXiv. /abs/1911.08356
[6] https://github.com/chipsalliance/riscv-dv
[7] https://tristan-project.eu/

TRISTAN Project has received funding from the Chips Joint Undertaking (Chips-JU) under the grant agreement nr. 101095947. Chips-JU receives support from the European Union's Horizon Europe's research and innovation programme and Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia and Turkey.