Vaquita: A Portable Four Stage Pipeline RISC-V Vector Co-Processor

Muhammad Latif[1], Shahzaib Kashif[1], Dr. Farhan Ahmed Karim[1] and Dr. Ali Ahmed[2]

[1]Department of Computer Science, Usman Institute of Technology (UIT)
[2]Department of Electrical Engineering, Usman Institute of Technology (UIT)

Abstract

This paper introduces a CHISEL based RISC-V Vector (RVV) v1.0 coprocessor, named Vaquita, designed to enhance vector processing performance. The architecture features a meticulously optimized 4-stage pipeline that maximizes computational throughput and minimizes latency, enabling efficient handling of complex vector operations. Built on a portable coprocessor paradigm, The solution facilitates easy integration with a variety of RISC-V-based systems using its plug and play compatible interface, ensuring reliable interoperability with RISC-V cores.

Introduction

The RISC-V[1] Vector (RVV) extension[2] represents a transformative advancement in modern processor design, offering scalable and energy-efficient vector processing for data-parallel workloads such as AI, ML, and embedded signal processing. By enabling Single Instruction, Multiple Data (SIMD) operations with configurable vector lengths (VLEN) and register grouping (LMUL), RVV reduces instruction overhead while maintaining flexibility across diverse computational domains.

The Vaquita co-processor, developed under the MERL (Micro Electronics Research Lab) initiative, is an open-source project designed to enhance vector computation capabilities within RISC-V based architectures. Hosted on GitHub (*https://github.com/merledu/vaquita*), Vaquita provides a modular and configurable framework for integrating vector processing units into RISC-V systems.

Vaquita is integrated with the NucleusRV core and tested for performance. The architecture is validated using the RISC-V Imperas test suite, ensuring compliance and reliability. With evaluations conducted at VLEN=256, Vaquita demonstrates its ability to efficiently execute complex vector operations. Its portable design allows easy integration across diverse RISC-V systems, making it suitable for a wide range of applications.

To achieve optimal performance, this work introduces a 4-stage pipeline architecture within the Vaquita co-processor, specifically tailored for the RISC-V Vector extension. The pipeline is designed to address common challenges such as data hazards and structural hazards, ensuring efficient execution of vector operations. By increasing throughput and reducing latency, the proposed architecture achieves maximum performance. Advanced techniques such as operand forwarding and hazard detection are employed to resolve dependencies and maintain pipeline integrity. This design not only enhances the co-processor's performance but also ensures scalability, enabling it to handle increasingly complex workloads with ease.

This paper presents a brief exploration of the Vaquita co-processor architecture. We discuss the design and implementation of a 4-stage pipeline, highlighting its ability to resolve hazards and achieve high throughput. The portable and configurable nature of Vaquita makes it an ideal platform for developing vector processing units tailored to specific applications. By advancing the capabilities of the Vaquita co-processor, this work contributes to the growing RISC-V ecosystem, providing a robust and scalable solution for next-generation vector processing.

Vaquita Micro Architecture

The Vaquita vector co-processor is designed to accelerate vector operations and improve computational efficiency. It adheres to an in-order execution model, ensuring deterministic and predictable instruction flow. It efficiently manages data and control dependencies across the four pipeline stages while interfacing with the memory subsystem for data access and processing, as described in Figure1.

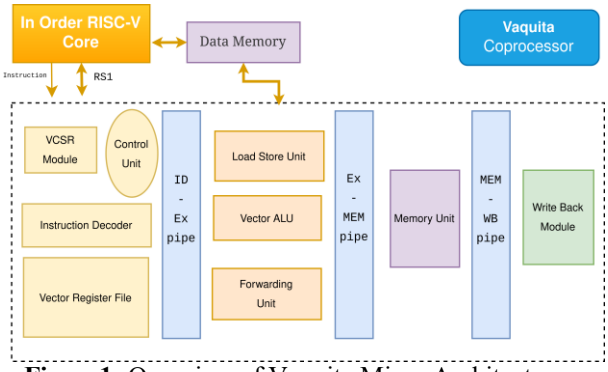

Figure1: Overview of Vaquita Micro Architecture

Instruction Decode (ID):

The Instruction Decode (ID) stage is responsible for interpreting the fetched instructions. During this stage, the instruction decoder takes the instruction and the value of register rs1 from the core. It breaks down the instruction into its constituent parts, such as the opcode and operands (i.e. rs1). If an instruction (i.e. vsetvli) returns the rs1 value, it is sent back to the core during the Execute stage by the coprocessor.

Execute (Ex):

In the execution (Ex) stage, the actual computation is performed by the Vector Arithmetic Logic Unit (Vec ALU), which is designed to execute parallelized vector operations. The dataflow within the Vec ALU is organized into lanes, where each lane processes a subset of vector elements according to the Selected Element Width (SEW). Each lane operates on 32-bit data, as defined by the Element Length (ELEN = 32). The number of lanes is dynamically determined based on the Vector Length (VLEN), allowing the system to adapt to varying computational demands. Additionally, the Forwarding Unit plays a critical role in this stage by resolving data hazards. It ensures that dependent instructions receive the correct intermediate results without causing pipeline stalls, thereby maintaining throughput of the execution process.

Memory (MEM):

MEM stage handles all memory-related operations, such as loading data from or storing data to memory. This stage is essential for instructions that require data transfer between the processor and memory. The memory unit ensures that data is correctly fetched or stored, and it coordinates with the pipeline to maintain data integrity. Efficient memory access is vital for the overall performance of the pipeline.

Write-Back (WB):

Finally, the Write-Back (WB) stage is where the results of the executed instructions are written back to the appropriate registers. This stage ensures that the computed values are stored in the correct locations, making them available for subsequent instructions. The WB module coordinates with the register file to update the registers, completing the instruction cycle. This stage is crucial for maintaining the correctness and continuity of the instruction flow.

Synthesis Results of Vaquita Design

We evaluated the Vaquita design with VLEN=64 and EEW=32 by synthesizing it using Synopsys Design Compiler with the FreePDK45 process at a target frequency of 1 GHz. The results show that Vaquita uses 50.2517 mW of power and occupies an area of 0.421 mm² on the chip, as shown in Table 1.

Table 1: PPA Metrics for the Proposed Design

Design Compiler	Synopsis
Technology	FreePDK45
Frequency	1GHz
Total Power	50.2517mW
Total Area	0.421mm^2

Conclusion and Future work:

In conclusion, Vaquita has been successfully integrated with the NucleusRV core and validated using the RISC-V Imperas test suite, demonstrating compliance and efficient execution of vector operations. Its portable design makes it adaptable across various RISC-V systems[3,4,5], enabling diverse applications in AI, signal processing, and scientific computing. For future work, Add the remaining part of the vector extension, benchmarking Vaquita against existing vector co-processors will provide insights into its performance, power efficiency, and scalability. Comparative analysis with established vector architectures will help refine its design, optimize execution pipelines, and enhance overall efficiency. Further improvements could focus on reducing latency, optimizing memory access patterns, and expanding instruction support to meet the growing demands of high-performance computing.

References:

[1] Waterman A, Lee Y, Patterson D, Asanovic K, level Isa VI, Waterman A, Lee Y, Patterson D. The RISC-V instruction set manual. Volume I: User-Level ISA', version. 2014 Apr;2.

[2] RISC-V International. RISC-V "V" Vector Extension Version 1.0. https://github.com/riscv/riscv-v-spec/releases/tag/v1.0,2021. [Online; accessed 6-Aug-2024]

[3] Patsidis, Kariofyllis, Chrysostomos Nicopoulos, Georgios Ch Sirakoulis, and Giorgos Dimitrakopoulos. "RISC-V 2: a scalable RISC-V vector processor." In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5. IEEE, 2020.

[4] Johns, Matthew, and Tom J. Kazmierski. "A minimal RISC-V vector processor for embedded systems." In *2020 Forum for Specification and Design Languages (FDL)*, pp. 1-4. IEEE, 2020.

[5] Lee, Joseph KL, Maurice Jamieson, and Nick Brown. "Backporting risc-v vector assembly." In *International Conference on High Performance Computing*, pp. 433-443. Cham: Springer Nature Switzerland, 2023.