Marian: An Open Source RISC-V Processor with Zvk Vector Cryptography Extensions

Thomas Szymkowiak
thomas.szymkowiak@tuni.fi
Tampere University
Tampere, Finland

Endrit Isufi
endrit.isufi@tuni.fi
Tampere University
Tampere, Finland

Markku-Juhani Saarinen
markku-juhani.saarinen@tuni.fi
Tampere University
Tampere, Finland

Abstract

The RISC-V Vector Cryptography Extensions (Zvk) were ratified in 2023 and integrated into the main ISA manuals in 2024. These extensions support high-speed symmetric cryptography (AES, SHA2, SM3, SM4) operating on the vector register file and offer significant performance improvements over scalar cryptography extensions (Zk) due to data parallelism. As a ratified extension, Zvk is supported by compiler toolchains and is already being integrated into popular cryptographic middleware such as OpenSSL. We report on Marian, the first open-source hardware implementation of a vector processor with the Zvk extensions. The design is based on the PULP "Ara" vector unit, which itself is an extension of the popular CVA6 processor. The implementation is in SystemVerilog and has been tested using Virtex Ultrascale+ FPGA prototyping, with a planned tapeout targeting a 22nm process node. We offer an analysis of the architectural requirements that vector cryptography imposes on a processor, as well as the initial estimates of performance and area for our implementation.

CCS Concepts

• **Computer systems organization → Single instruction, multiple data**; **Embedded hardware**; • **Security and privacy → Cryptography**.

Keywords

RISC-V, Vector, Cryptography

1 Introduction

Modern operating systems and applications make extensive use of cryptography to guarantee data confidentiality and integrity of communications and data storage. According to CloudFlare Radar, 97% of web requests are now made with encrypted protocols such as TLS, and only 3% with unencrypted HTTP.[1] Other key areas include the encryption of data storage media.

Application-class processors must be designed to meet the ever-increasing computational demands of the domains in which they are most heavily utilized, e.g., mobile and high-performance computing. Multiple ISAs have addressed the need to efficiently perform cryptographic operations by defining ISEs that target cryptographic workloads (e.g., x86, SPARC, ARM, and Power ISAs define extensions for cryptographic operations).

The RISC-V cryptography extensions come in two variants: scalar cryptography (*"Zk"*, ratified in 2021), which operates on the general register file (limited to 32 or 64 bits), and vector cryptography (*"Zvk"*, ratified in 2023), which uses the scalable vector register file (32 registers, usually between 128 and 2048 bits each), allowing for improved performance via data-level parallelism. The vector cryptography specification builds upon the instructions and functionality specified within the RISC-V Vector (RVV) Extension. Both Scalar and Vector Cryptography extensions have recently been integrated into the RISC-V unprivileged ISA Manual [4].

Vector cryptography extensions have gained commercial significance since they will be required in prominent RISC-V application processor profiles, and in RISC-V-based devices running Google's Android operating system[2]. While there is existing research work covering the implementation of the RISC-V scalar cryptography specification [2], to our knowledge, there has yet to be an equivalent analysis of the RISC-V vector cryptography specification.

Our Contributions. This work presents Marian [1], the first open-source implementation of the RISC-V Vector Cryptography extensions. Marian is implemented in SystemVerilog and extends the existing RISC-V Vector extension implementation provided by Pulp Ara to fully support the NIST Algorithm Suite with GCM extension (*Zvkng*), ShangMi Algorithm Suite with GCM extension (*Zvksg*), and the Entropy Source extension (*Zkr*). All implemented instructions conform with the "constant time" or Data Independent Execution Latency (DIEL / *Zvkt* [4]) requirements.

The design targets ASIC using a 22nm process and has been thoroughly verified in simulation and using FPGA prototyping. The performance of the design has been evaluated using FPGA prototyping as the ASIC physical design is currently underway.

The focus of this work is to investigate and assess the potential development challenges and performance benefits when implementing the RISC-V Vector Cryptography extensions on top of an existing implementation of the RVV extension.

2 Development

2.1 Vector Unit Selection

A review of the available open-source implementations of RVV 1.0 vector extension was performed to determine which of the available implementations would be most suitable for using a baseline for extension. Criteria such as implementation maturity, verification artefact availability, technology compatibility, and documentation quality were used to guide the selection. "Ara" is an RVV 1.0 co-processor developed in the PULP project [3]. Ara has been successfully taped out, targeting a 22nm node at 1.35GHz, supports Questasim and Verilator-based flows, and the repository contains a

[1]CloudFlare Radar, Adoption and Usage HTTP vs HTTPS, July 2024: https://radar.cloudflare.com/adoption-and-usage?dateStart=2024-07-01&dateEnd=2024-07-31

[2]https://opensource.googleblog.com/2023/10/android-and-risc-v-what-you-need-to-know.html

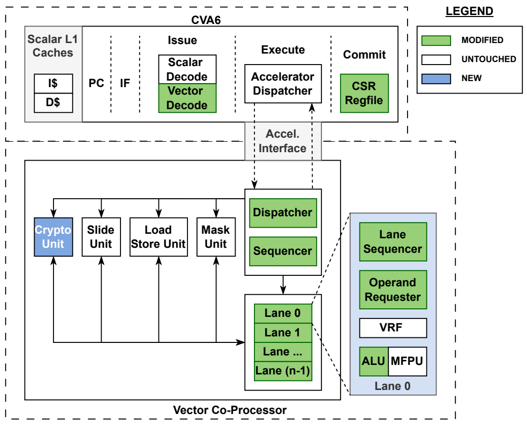

Figure 1: CVA6 and Vector Unit Architecture (Ext. Interfaces Omitted)

large set of software tests targeting the vector processor, making it a suitable choice for extension. Ara is a parametrizable design, allowing users to statically define the architectural configuration values, such as the number of lanes and the vector register length (VLEN). Ara also possesses a number of architectural constraints, including the maximum element width (ELEN) and the bit width of the lane data paths, which are both fixed at 64b.

2.2 Architecture

Many instructions defined within the RISC-V Vector Cryptography specification perform operations on Element Groups (EG) with an Element Group Width (EGW) greater than 64b. To reduce the design complexity of extending Ara, we decided to separate the arithmetic operations of the cryptographic operations from the lane logic. The lanes would only be used for accessing the operands in the Vector Register File (VRF), and a decoupled processing unit (Crypto Unit) would be created to execute the cryptographic operations. In addition to the insertion of the Crypto Unit, existing components within Ara were modified to support the additional cryptography instructions (see Figure 1). For tapeout, a *vlen* of 512b and four lanes was selected to balance performance and area, as the vector unit area increases significantly with the number of lanes. [3].

Marian is designed to be integrated as an IP within the next SoC being developed by the SoC Hub (Tampere University). Therefore, in addition to the modifications made to the vector unit of Ara, several peripherals and supporting components were added to enable the IP's use within an SoC context. These include PLL, CLINT, PLIC, QSPI, Timer, RISC-V Debug module, and internal memories.

2.3 Crypto Unit Design

The Crypto Unit is a three-stage, latency-insensitive pipeline consisting of an operand collection stage, an execution unit stage, and a write-back stage (see Figure 2). When a cryptography instruction is issued after decode, the instruction request is broadcast to both the lane sequencing logic (to request the operands from the VRF) and to the Crypto Unit. Once the request is received, the Crypto

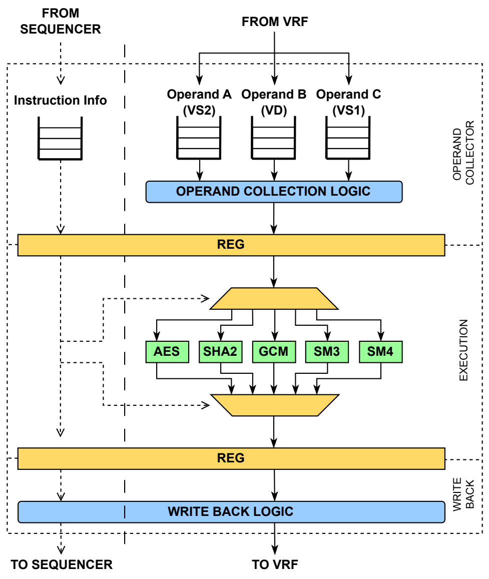

Figure 2: Crypto Unit Architecture

Unit then awaits the operand data from the VRF and, upon receipt, assembles the 64b data components from each lane into the operand format required by the cryptographic operation. After the operands have been fully populated, they are propagated to the execution stage of the pipeline. The execution unit stage contains the individual arithmetic units for each operation and the routing logic to access them. When valid results from the arithmetic units become available, they are forwarded to the write-back stage. Within this stage, the VRF addresses are calculated, and the data is organized for storage within the destination vector register, which is then written to the VRF. After all of the vector elements of the current operation have been written, a notification is sent to the vector sequencer to indicate that the instruction can be retired.

2.4 CVA6 Modifications

In RISC-V, Random Bit Generators (RBGs) are intended to be constructed using the Entropy Source Extension (*Zkr*). This was implemented by extending the CVA6 CSR registers to support the *seed* CSR that interfaces to an SP 800-90B Entropy Source placeholder.

2.5 Verification

A combination of randomized testbenches and software-driven testing was used to verify Marian in both simulation and FPGA prototyping platform environments. Within the randomized testbenches, the scoreboards were implemented using behavioral models based on the Sail code contained within the RISC-V Cryptography Specification. For software-based testing, reference result vectors were initially taken from the corresponding NIST specifications. Once the design was verified with these values, additional result vectors were generated using Spike ISA simulator with randomized inputs. The same vectors were tested on Marian and the results were compared. The software tests were run in both simulation and FPGA-based emulation environments.

3 Implementation and Benchmarking

An initial performance evaluation of Marian was performed against C-language reference implementations of cryptographic primitives taken from OpenSSL 3.3.1.[3] OpenSSL contains open-source implementations of AES, SHA-2, SM3, and SM4 operations, targeting numerous platforms with varying support for cryptographic ISEs.

The equivalent operations were subsequently executed using code with *Zvk* instructions. The RISC-V `instret` and `cycle` performance CSRs were used to measure the number of instructions retired and CPU cycles elapsed during execution, respectively. As the implemented *Zvk* extensions satisfy the data-independent execution latency requirements of the *Zvkt* extension, constant time reference implementations of AES operations were selected for use in benchmarking. A message length of 1kB was used to benchmark SHA256, SHA512 and SM3 operations. The cryptography code was mostly based on the "sample code" contained within the RISC-V Crypto Github repository.[4] A minimum *vl* value was used for each operation (equivalent to the instruction EGS) to determine the worst-case performance of the vector operations. Averages of the instructions retired and cycle count values attained through benchmarking are presented within Table 1. More comprehensive benchmarks, including a comparison against an implementation of the RISC-V scalar cryptography extension, are out of the scope of this work and are scheduled to be completed as a part of future research.

Table 1: Initial Benchmark Results

Operation		Reference		Zvk	
		Cycles	Instret	Cycles	Instret
AES128	Enc.	18,794	12,482	343	53
	Dec.	23,731	15,077	226	53
AES256	Enc.	24,493	17,478	441	65
	Dec.	32,677	21,213	278	65
SHA256	Hash	156,205	82,179	12,106	3,802
SHA512	Hash	109,905	45,903	9,140	2,712
SM3	Hash	304,031	70,075	8,134	1,410
SM4	Enc.	4,187	1,423	272	39
	Dec.	2,564	1,425	178	39

Marian has been successfully prototyped on an AMD-Xilinx VCU118 (Virtex Ultrascale+) FPGA running at 75MHz. The ASIC physical design flow for Marian is currently underway, targeting a 22nm low-power process and an F_{max} of 1GHz. FPGA component utilization and ASIC post-synthesis logic area values for Marian are listed within Table 2 and Table 3. The Gate Equivalent (GE) values are calculated using a two-input NAND gate in the target technology. Note that the ASIC flow is in-progress and therefore the presented area numbers are not final.

[3]OpenSSL Github https://github.com/openssl/openssl/tree/openssl-3.3.1
[4]RISC-V Crypto Github Repository https://github.com/riscv/riscv-crypto/tree/v20230823

Table 2: FPGA Resource Utilisation

Top Module	Registers	LUT (logic)	LUT (RAM)	BRAM (kB)	DSP
Marian	115,767	420,056	1908	360	225
CVA6	24,924	40,900	884	117	28
Vector Unit	67,680	322,474	1,024	0	197
Lane (single)	14,513	57,175	256	0	49
Crypto Unit	2,800	33,465	0	0	0

Table 3: Initial ASIC Resource Report

Top Module	Logic Cell Area (mm^2)	Area (kGE)	% of Total Area
Marian	2.08	1834.263	100%
CVA6	0.28	242.973	13.25%
Vector Unit	1.04	915.536	49.91%
Lane (single)	0.21	181.040	9.87%
Crypto Unit	0.14	118.131	6.44%

4 Conclusion

We have presented Marian [1], an open-source implementation of the RISC-V vector cryptography extensions, using Pulp Ara as a baseline. To efficiently implement the RISC-V *Zvk* extensions in a traditional vector style (i.e., arithmetic operations performed with the lanes) the data width of each lane should be at least as wide as the largest cryptographic operation (256b). Alternatively, logic external to the lanes can be utilized to implement the extension efficiently. To ease implementation complexity, we have used this alternative approach within Marian, and initial benchmarks indicate that a significant performance gain can still be achieved. Furthermore, both the FPGA and ASIC area estimates for the Crypto Unit have been presented and indicate that its addition does not substantially increase the area of the existing vector unit.

References

[1] SoC Hub. 2024. Marian. https://github.com/soc-hub-fi/Marian.

[2] Ben Marshall, G. Richard Newell, Dan Page, Markku-Juhani O. Saarinen, and Claire Wolf. 2020. The design of scalar AES Instruction Set Extensions for RISC-V. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021, 1 (December 2020), 109–136. https://doi.org/10.46586/tches.v2021.i1.109-136

[3] Matteo Perotti, Matheus Cavalcante, Renzo Andri, Lukas Cavigelli, and Luca Benini. 2024. Ara2: Exploring Single- and Multi-Core Vector Processing with an Efficient RVV 1.0 Compliant Open-Source Processor. *IEEE Trans. Comput.* 73, 7 (2024), 1822–1836.

[4] RISC-V. 2024. *The RISC-V Instruction Set Manual Volume I: Unprivileged Architecture.* Ratified ISA Release 20240411. RISC-V International. https://github.com/riscv/riscv-isa-manual/releases/download/20240411/unpriv-isa-asciidoc.pdf