A Deep Dive into Integration Methodologies in RISC-V

Alessadra Dolmeta[1], Valeria Piscopo[1], Maurizio Martina[1], Guido Masera[1]

[1]DET, Politecnico di Torino, Torino, Italy

Abstract

The integration methodology can significantly affect the performance of dedicated accelerators. This work undertakes an exploration of this aspect, considering Keccak, a pivotal hashing standard in Post-Quantum Cryptography (PQC), as a case of study. The paper presents three versions of KRONOS (Keccak RISC-V Optimized eNgine fOr haShing): a loosely-coupled memory-mapped accelerator, a tightly-coupled approach, and a coprocessor. The latter two versions leverage the CV-X-IF interface, with and without, respectively, an additional register file to store the Keccak state. Results show that the tightly approach is the most efficient integration method, achieving a balance between resource consumption and throughput.

Introduction

In embedded systems design, the growing computational demands have driven the integration of specialized accelerators. The chosen integration methodology significantly impacts system performance and efficiency, with three primary approaches: loosely coupled, tightly coupled, and coprocessors.

Loosely coupled accelerators are generally memory-mapped and operate independently, interfacing with the CPU via a system bus (e.g., AXI). Conversely, tightly coupled accelerators are integrated within the CPU microarchitecture and directly access internal registers. This minimizes latency but requires core modifications and instruction set extensions. Whilst the former methodology offers greater versatility, the latter delivers low-latency execution, albeit at the cost of core and toolchain modification. Coprocessors, though also interfaced with the CPU, differ by utilizing external register files, allowing for more complex computations and multi-core scalability. Unlike tightly coupled accelerators, they provide greater flexibility without strict internal register constraints.

RISC-V's open-source ecosystem has further advanced accelerator research, particularly with the Core-V eXtension InterFace (CV-X-IF)[1], which streamlines tightly coupled accelerator integration without modifying the CPU toolchain and pipeline.

This study explores different integration strategies for KRONOS, a Keccak RISC-V Optimized eNgine for haShing. Since Keccak is the core hash function in many PQC schemes [1], dedicated acceleration can significantly enhance performance. While the tightly coupled approach achieves the lowest speed-up among the three methods, it offers the best *Throughput/Area* trade-off with minimal resource overhead. The implementations are available open-source. [2]

Integration Methodologies

Keccak, the winner of the SHA-3 Cryptographic Hash Algorithm Competition, offers superior robustness and security over other hash standards. Its sponge structure relies on the Keccak-f permutation, which operates on a 1600-bit state, structured as a 5×5 matrix of 64-bit words. This permutation runs 24 rounds, each comprising five steps: θ, ρ, π, χ, and ι.

The three versions of KRONOS, shown in Fig. 1, are integrated into X-HEEP[2], a RISC-V-based microcontroller, using different approaches. All three versions use the CV32E40PX core[3].

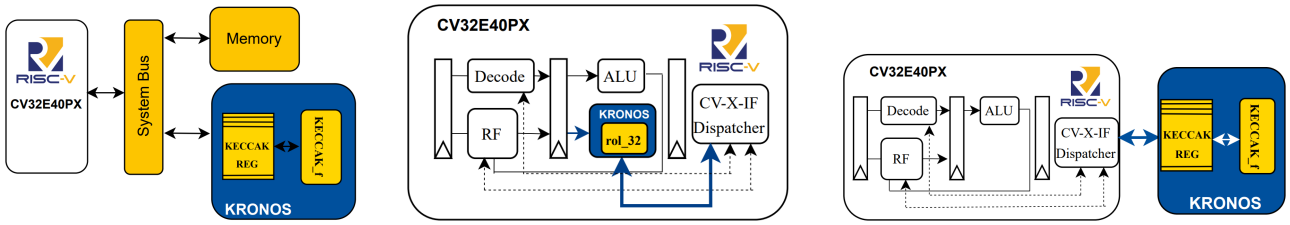
Loosely coupled. The first approach consists of a loosely coupled, memory-mapped component, accelerating the complete Keccak-f permutation (Fig. 1a). Fifty additional 32-bit registers (*Keccak Reg*) are used to store the state, reducing the load/store operations to and from the memory. A dedicated driver is used to call KRONOS when needed.

Tightly coupled. The second approach leverages the CV-X-IF interface (Fig. 1b) to implement RISC-V-compliant custom instructions. These instructions adhere to the standard two-source, one-destination register format, ensuring compatibility with the scalar register file. Since Keccak was originally designed for 64-bit architectures, an initial adaptation involves modifying the code to operate efficiently on 32-bit registers. A key component of KRONOS in this context is bitwise rotation, for which the *rol_32* instruction has been introduced. This dedicated operation enables efficient 64-bit rotations by working directly on two 32-bit registers.

[1] https://github.com/openhwgroup/core-v-xif/tree/main

[2] https://github.com/vlsi-lab/KRONOS

[3] https://github.com/esl-epfl/cv32e40px



(a) *Loosely coupled approach.* **(b)** *Tightly coupled approach.* **(c)** *Coprocessor approach.*

Figure 1: *Comparison of different integration approaches. The controllers of (b) and (c) are not shown for simplicity.*

Coprocessor. The third version is a coprocessor (Fig. 1c). As in the first case, the complete Keccak-f permutation is performed, but the CV-X-IF facilitates communication with the core. Three R-type custom instructions have been added to store/load the state into *Keccak Reg* and to start the 24-round permutation. Unlike the previous version, this implementation is not fully compliant with RISC-V instruction set extensions (ISE).

Results and conclusions

In this section, we report our implementation results. Tab. 1 illustrates the cycle counts for executing the SHA3-384 function on X-HEEP platform using the three different methods, and their relative speed-ups and throughput. The `-O2` flag is used for reference and accelerated simulations.

Table 1: *Cycle counts (SHA3-384).*

Method	Ref.	Accel.	Speed-up factor	Thr. [Mb/s]
Loosely	56,529	4,169	13.56×	4.61
Tightly		31,527	1.79×	0.61
Coproc		7,553	7.48×	2.54

Tab. 2 presents a comparison of the FPGA area results for the three versions, implemented on the Zynq UltraScale+ ZCU104 board (xczu7ev-ffvc1156-2-e). The synthesis and implementation are performed using Xilinx Vivado, with a global clock of 50 MHz.

The last column of Tab. 2 also presents the *Throughput/Area* results, allowing for a final comparison among the three approaches. The optimal trade-off between area and throughput is provided by the tightly coupled version. Although it offers the lowest throughput and speed-up among the three cases, it provides a lower number of LUTs with respect to the loosely and coprocessor versions of, respectively, almost 9 and 11 times. Additionally, the absence of a dedicated Keccak register file results in a substantially more compact solution when compared to the other integration methodologies.

Table 2: *Resource utilizations (ZCU104, 50 MHz).*

Method	LUT	Registers	Thr./Area [kb/(s·LUTs)]
Loosely			
◇ KRONOS	4,915	3,252	
○ Controller	7	35	0.937
○ Keccak	4,908	1,617	
○ Register File	0	1,600	
Tightly			
◇ KRONOS	569	139	
○ Controller	569	102	1.070
○ rol_32	0	32	
Coproc			
◇ KRONOS	6,591	3,372	
○ Controller	1,181	125	0.386
○ Keccak	5,409	1,615	
○ Register File	0	1,600	

Acknowledgement

This work was funded by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. This work received funding from the Key Digital Technologies Joint Undertaking (KDT-JU) under grant agreement No 101095947.

References

[1] G. Bertoni et al. "FIPS PUB 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions". In: (2015). URL: `https://keccak.team/hardware.html`.

[2] Simone Machetti et al. *X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators*. 2024. arXiv: `2401.05548 [cs.AR]`.