The Impact of Hash Primitives and Communication Overhead for Hardware-Accelerated SPHINCS+

Patrick Karl
Technical University of Munich, Germany
TUM School of Computation, Information and Technology
patrick.karl@tum.de

Jonas Schupp
Technical University of Munich, Germany
TUM School of Computation, Information and Technology
jonas.schupp@tum.de

Georg Sigl
Technical University of Munich, Germany
TUM School of Computation, Information and Technology,
Fraunhofer Institute for Applied and Integrated Security, Garching, Germany
sigl@tum.de

***Abstract*—SPHINCS+ is a signature scheme included in the first NIST post-quantum standard, that bases its security on the underlying hash primitive. As most of the runtime of SPHINCS+ is caused by the evaluation of several hash- and pseudo-random functions, instantiated via the hash primitive, offloading this computation to dedicated hardware accelerators is a natural step. In this work, we evaluate different architectures for hardware acceleration of such a hash primitive with respect to its use-case and evaluate them in the context of SPHINCS+. We attach hardware accelerators for different hash primitives (SHAKE256 and Asconxof for both full and round-reduced versions) to CPU interfaces having different transfer speeds. We show, that for most use-cases, data transfer determines the overall performance if accelerators are equipped with FIFOs.**

***Index Terms*—SPHINCS+, PQC, post-quantum cryptography, hardware acceleration, Ascon**

I. Introduction

Post-Quantum Cryptography (PQC) schemes spend a substantial computation time to hash data or to generate pseudo-randomness. One of the post-quantum secure signature schemes to be standardized by the National Institute of Standards and Technology (NIST) is SPHINCS+ [1], a hash-based PQC framework that can be instantiated with different hash primitives. One of the proposed instances uses the SHA-3 Extendable Output Function (XOF) SHAKE256. For this choice, the algorithm spends more than $95\,\%$ of its runtime inside the hash function when running on a commonly used microcontroller [2]. Moreover, using the SHA-3 functions for hashing and generation of pseudo-randomness is a usual choice for PQC algorithms. Thus, hardware acceleration for the underlying hash primitive is a natural step to improve the performance and energy consumption of PQC schemes, and in particular SPHINCS+.

Related Work: Previous works investigated hardware implementations for SPHINCS+, but mostly focused on co-processors that compute the entire signature. For instance, the work in [3] presents a standalone, high-throughput co-processor using SHAKE256. In [4], another co-processor with area efficiency as primary design goal is presented using SHA-2 as hash primitive. These standalone designs yield high performance, but lack flexibility – they speed-up a single algorithm, but their re-use for other schemes is limited. Hardware/software co-designs provide a trade-off between performance and flexibility by offloading only computationally intensive operations to dedicated hardware, but run the main algorithm in software. The work presented in [5] investigates the use of SPHINCS+ in the context of secure boot within a hardware-software co-design, and extends the platform's SHA-2 core to improve the performance for hash-based signatures in general. To the best of our knowledge, no hardware/software co-design evaluating SPHINCS+ using the SHA-3 standard has been presented so far.

Contribution: We explore different design architectures for accelerators and evaluate the advantages and disadvantages of these options with respect to resource cost and performance. We show that the choice for the most performant design option is application dependent. We use SPHINCS+ as a case study and provide performance metrics for the resulting hardware/software co-design. Furthermore, we replace the SHAKE256 primitive with other alternatives like TurboSHAKE256 [6], a round-reduced SHAKE256 version, and functions from the Ascon [7] suite, the winner of the NIST Lightweight Cryptography (LWC) competition. Instantiating Ascon in the SPHINCS+ framework has been submitted to NIST's additional call for signature schemes[1] under the name of Ascon-Sign[2]. In short, our contributions are as follows:

- Design-space exploration of hash-accelerator architectures on a $32\,\text{-bit}$ RISC-V platform, considering the corresponding requirements of data transfer.
- Performance evaluation for SPHINCS+ using different primitives, i.e. SHAKE256, TurboSHAKE256, Asconxof and Asconxofa.

Organization: Section II provides a brief overview of SPHINCS+, the SHA-3 standard and our evaluation setup. In Section III we explain different architectures for the investigated hardware accelerators and their parameterization

[1]https://csrc.nist.gov/Projects/pqc-dig-sig/standardization
[2]https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures

and cost. A performance analysis of these architectures and their application to SPHINCS+ is conducted in Section IV. Section V investigates the usage of cryptographic primitives for SPHINCS+ that are not part of the NIST submission. Finally, Section VI concludes this work.

II. Preliminaries

A. The SPHINCS+ Framework

SPHINCS+ [1] uses different families of hash- and pseudo-random functions and solely relies on their (second-) preimage resistance. The core idea of SPHINCS+ is to combine several layers of Merkle trees with One-Time Signature (OTS) key pairs on its leafs. In the last layer of the resulting hypertree, a Few-Time Signature (FTS) scheme called Forest Of Random Subsets (FORS) is used. For details, we refer to the original work [1]. Within SPHINCS+, the following pseudo-random and message-digest functions are used, where $\mathbb{B}$ denotes the set of bytes:

$$\boldsymbol{PRF} : \mathbb{B}^n \times \mathbb{B}^{32} \rightarrow \mathbb{B}^n$$
$$\boldsymbol{PRF_{msg}} : \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^* \rightarrow \mathbb{B}^n$$
$$\boldsymbol{H_{msg}} : \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^n \times \mathbb{B}^* \rightarrow \mathbb{B}^m$$

Furthermore, a hash function $\boldsymbol{T}_l$ with its two special cases $\boldsymbol{F} = \boldsymbol{T}_1$ and $\boldsymbol{H} = \boldsymbol{T}_2$ is defined as follows:

$$\boldsymbol{T}_l : \mathbb{B}^n \times \mathbb{B}^{32} \times \mathbb{B}^{ln} \rightarrow \mathbb{B}^n$$

The parameter $n \in \{16, 24, 32\}$ defines the output length (in bytes) of all the functions being used except for $\boldsymbol{H_{msg}}$, that outputs an m-byte string. Likewise, the size of n determines the NIST security levels I, III and V. In this work, we consider parameter sets using the XOF SHAKE256 from the SHA-3 standard [8] as the underlying primitive. Therefore, we briefly explain its concept in Section II-B.

B. Keccak and the SHA-3 Standard

SHAKE256 is based on the Keccak primitive and consists of a 1600-bit state on which the permutation function `keccak-f1600` is applied for 24 rounds. For all the functions defined in the SHA-3 standard [8], Keccak is used in a sponge construction. The 1600-bit state is divided into a rate r and a capacity c and is initialized with zeroes. The input data is split into several blocks b_i of size r, that are consecutively absorbed into the state. Between the absorption of two blocks, the permutation function is applied. After absorbing all blocks, the output blocks b_o (of size r) are squeezed and if more than one block is requested, the state is permuted again after each output block. SHA-3 defines several hash functions with fixed output lengths and two XOF functions with outputs of arbitrary length, i.e. SHAKE128 and SHAKE256.

C. Evaluation Platform

We use the PULPino microcontroller platform[3] that instantiates the cv32e40p, a 4-stage pipelined 32-bit RISC-V core supporting the RV32IMC and optional F Instruction Set Architecture (ISA) [9]. We evaluate our experiments on a Zynq UltraScale+ FPGA (xczu9eg-ffvb1156-2) using Vivado 2020.2 as synthesis tool. On this platform, the PULPino baseline implementation consumes 16,038 Look-Up Tables (LUTs) and 10,050 Flip-Flops (FFs) after synthesis and runs at a frequency of 150 MHz. It is noteworthy, that none of the hardware extensions in this work lead to a decrease in frequency and thus, the reduction in cycle counts directly translates to performance gains. For software compilation, the corresponding PULP compiler[4] with `-O3` flag has been used.

III. Accelerator Architectures

Figure 1 shows the schematic of the PULPino microcontroller and different accelerator options. A designer can integrate custom functionality *tightly* into the processor and extend the ISA with corresponding instructions. One can also connect a co-processor *loosely* to the system bus. Finally, the last option is to connect the co-processor to the Load-Store Unit (LSU) before bridging onto the system bus. For evaluation,

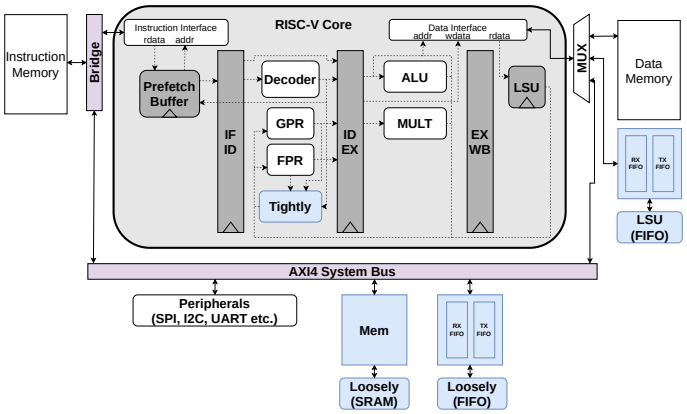
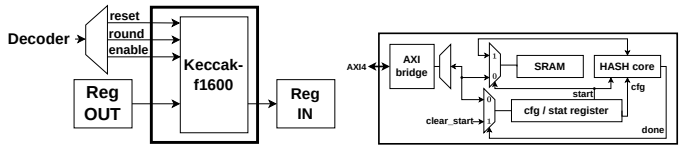


Fig. 1: Extended PULPino with accelerator architectures (blue).

we implemented a SHA-3 hash core that is integrated in the different architectures as depicted in Figure 2.



(a) Tightly-coupled as in [10] (b) Loosely-coupled with SRAM

(c) Loosely-coupled with FIFOs (d) LSU-coupled with FIFOs

Fig. 2: Overview of accelerator architectures.

A. Tightly-coupled into Register Set

In [10], the `keccak-f1600` function has been integrated tightly-coupled into the register set of a RISC-V microcontroller. This concept is depicted in Figure 2a. The advantage

[3] https://github.com/pulp-platform/pulpino

[4] https://github.com/pulp-platform/pulp-riscv-gnu-toolchain

of this strategy is, that it prevents load-store overheads that are required for loosely-coupled solutions connected to a system bus. It also provides a flexible solution that can be re-used for any function relying on the Keccak permutation, because only the round function itself is implemented in hardware. Consequently, different functions (e.g., SHA-3 hash functions, SHAKE128 or SHAKE256) are run in software and only their costly permutation routine is replaced by a single custom RISC-V instruction per round. A drawback, however, is that for $32\,\text{-bit}$ platforms, it requires the presence of both the General-Purpose Registers (GPRs) and the Floating-Point Registers (FPRs) to store the entire $1600\,\text{-bit}$ Keccak state. Besides that, compilers must be adapted accordingly to support the custom instruction.

B. Loosely-coupled with SRAM

The simplest way of integrating a hardware co-processor is attaching it to the platforms's system bus as shown in Figure 2b. In this architecture, the address space is divided into a data section that enables writing into and reading from a dedicated SRAM, and a configuration/status register. For a hash co-processor, the procedure using this architecture is as follows: The user writes the input data to the SRAM via the system bus and configures the hash core. This includes for instance the desired mode of operation (e.g., SHA-3-256, SHAKE256, etc.) or the input and output length of the data. Finally, a signal *start* triggers the core to process the input data and the produced hash output is written into the SRAM. Afterwards, the produced data can be read again via the AXI4 interface.

The advantage of this approach is the easy integration into the platform and its accessibility in software. The drawback is, that the core starts the computation after the data is fully written into the memory and the produced digest is read after the computation is finished. As we show in Section IV, this has a severe impact on performance for large data sizes.

C. Loosely-coupled with FIFOs

Using First-In-First-Outs (FIFOs) as depicted in Figure 2c instead of a dedicated SRAM enables to hide the computation inside the hash core in the data transfer. A write-only *RX FIFO* buffers the input data and a read-only *TX FIFO* buffers the output data. This enables the co-processor to start absorbing the input data as soon as the first chunk is written into the *RX FIFO*. The output data is written into the *TX FIFO* and the microcontroller can immediately start reading the output. This also reduces the complexity in the module, as no control bits are required anymore. How to properly dimension the FIFOs depends on the underlying hash function (the number of permutation rounds), as during permutation, no input data can be absorbed and no output data is produced.

D. LSU-coupled using FIFOs

Although the usage of FIFOs benefits the performance in general, writing and reading the data via the AXI4 system bus can still pose a notable overhead due to the bus latency. Therefore, the last design option is to integrate the co-processor closer to the RISC-V core and to connect it directly to the core's LSU, as shown in Figure 2d. This yields two advantages: First, the latency caused by the system bus is reduced as no bridging is required anymore – it only requires multiplexing between the accelerator's and the remaining address space. Second, the co-processor does not require a complex AXI4 interface anymore which reduces the area cost. The drawback of this approach is, however, the increased integration complexity. Connecting the module to the LSU requires knowledge of the microcontroller implementation.

E. FIFO Dimensions

For the architectures discussed in Section III-C and Section III-D the FIFOs sizes influences the resource cost as well as the accelerator's performance. Both FIFOs have a width of $32\,\text{-bit}$ to comply with the word width of the RISC-V core. The *RX FIFOs* should be large enough to buffer incoming data while the hash core is permuting without running full, so that the data transfer is not blocked. That is, the FIFO's size (D_{RX}) depends on the number of the hash function's permutation rounds T_{perm} and the bus latency $t_{bus,wr}$ to perform a write operation. For the *TX FIFO*, it is desirable that whenever the microcontroller wants to read data, the FIFO is not empty. However, it should be able to store enough data that the hash core can start the next permutation while the FIFO is still being read without running empty. As a consequence, the required depths (in $32\,\text{-bit}$ words) of the FIFOs are as follows:

$$D_{RX} \geq \lceil \frac{T_{perm}}{t_{bus,wr}} \rceil \qquad D_{TX} \geq \lceil \frac{T_{perm}}{t_{bus,rd}} \rceil \tag{1}$$

In our design, we have $t_{axi,wr} = 6$, $t_{axi,rd} = 7$, $t_{lsu,wr} = 2$ and $t_{lsu,rd} = 3$, until the data is inside the corresponding FIFO or read from it. With $T_{perm} = 24$ for SHAKE256, we end up with FIFO depths of $D_{RX} = D_{TX} = 4$ for the loosely-coupled case and $D_{RX} = 16$ and $D_{TX} = 8$ for the LSU-coupled version. Note, that although for the LSU-coupled version, $D_{RX} = 12$ would meet the requirements, we slightly increased it to stick to a power-of-two.

F. Resource Overhead

A comparison of the different architectures with respect to resource overhead compared with the plain PULPino design is given in Table I. For a discussion on the Ascon cost we refer to Section V-A. The results highlight the advantage of the tightly-coupled approach: Because this architecture only implements the round function in hardware but re-uses the GPR and FPR to store the state, it shows the least overhead. It is worth noting that the tightly-coupled's overhead in FFs mainly stems from the $32 \times 32 = 1024\,\text{-bit}$ of the additional FPR. The loosely- and LSU-coupled architectures require more logic resources due to bus interfaces, control logic and the local memory in terms of FIFOs or SRAM. The SRAM overhead for the loosely-coupled version is omitted for two reasons. First, the accelerator's SRAM can later be re-used for normal operation of the RISC-V core and thus, does not need to be accelerator-specific. Second, the SRAM size depends on the specific application – it needs to be big enough to store the maximum amount of input or output data. In contrast to that,

the FIFO dimensions are application independent. Application independent in the sense that the FIFO dimensions do not depend on the amount of input or output data, but only on the small amount of data that needs to be buffered while the hash core is permuting. For a discussion on the cost for Ascon as hash primitive, we refer to Section V.

TABLE I: Resource overhead including the hash core.

Architecture	SHA-3		Ascon	
	LUTs	FFs	LUTs	FFs
loosely-coupled with SRAM	+8,731	+2,511	+1,984	+1,189
loosely-coupled with FIFOs	+8,963	+2,485	+1,968	+1,158
lsu-coupled	+8,215	+1,890	+1,388	+547
tightly-coupled	+4,494	+1,087	+246	+12

IV. Performance Evaluation

A. Architecture Benchmark

In the following, we compare the performance of the architectures described above. To do so, we consider two scenarios:

In the **Hash** scenario, we benchmark the performance for a fixed output length and vary the input length. Concretely, we set the output length to $16\,\mathrm{B}$ and vary the input lengths up to $4\,\mathrm{kB}$. In the **Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)** scenario, we do the opposite, i.e. the input length is fixed to $16\,\mathrm{B}$, the output length varies up to $4\,\mathrm{kB}$. For both scenarios we consider the SHAKE256, that can be used for arbitrary input and output lengths.

The results for both scenarios are shown in Figure 3. The LSU- and tightly-coupled versions perform best depending on the scenario and amount of data to be processed. To explain this, the data transfer has to be considered. In both cases, the input data is first loaded from memory into the register set and after computation, the hash output is written back to memory. Yet, the computation is different depending on the architecture: In the tightly-coupled case, input blocks b_i must first be loaded into some free GPR registers. For the parts of the state's rate that lie within the FPR, the corresponding state word must be moved to the GPR, then absorption (xor) is performed, and afterwards, the updated word is written back to the FPR. This requires 3 instructions for the rate part in the FPR (in our implementation 16 regs $\times$ 3 instr), whereas the rate part in the GPR can be absorbed within one single xor instruction (18 regs $\times$ 1 instr). Therefore, the absorption takes 66 clock cycles per input block, as for SHAKE256, the rate consists of 34 words of $4\,\mathrm{B}$ each. After absorption of each input block b_i and squeezing of output blocks b_o, the state must be permuted for T_{perm} cycles. Thus, the computation can be modeled as:

$$t_{tightly} = b_i(66 + T_{perm}) + b_o T_{perm} + t_{sw} \tag{2}$$

where t_{sw} denotes a software overhead e.g., for calling functions. It should be noted that the absorption of the last input block might take less than 66 cycles if it is only partially filled.

For the LSU-coupled and loosely-coupled architectures, the input data must be written to and the output data read from the accelerator. However, when using FIFOs, the absorption, squeezing and subsequent permutations are hidden in the data transfer assuming the FIFOs do not run full or empty. With this assumption, the behavior of the architectures can be modelled as follows:

$$t_{loosely,sram} = t_{store} + (b_i + b_o)T_{perm} + t_{load} + t_{hw} \tag{3}$$

$$t_{loosely,fifo} = t_{store} + t_{load} + t_{hw} \tag{4}$$

$$t_{lsu} = t_{store} + t_{load} + t_{hw} \tag{5}$$

The variables t_{store} and t_{load} denote the cycles to write data to and read data from the accelerator, t_{hw} the constant overhead of the hardware drivers.

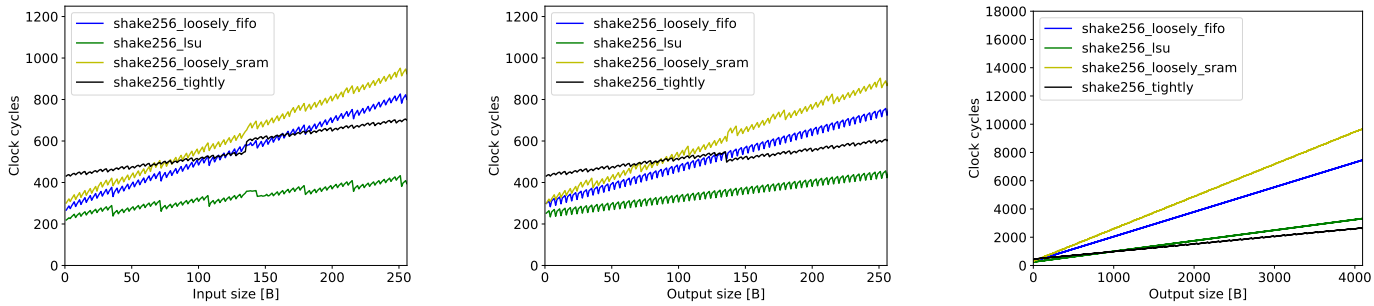
a) Implications for Hash scenario: For the hash scenario, the t_{load} as well as b_o is negligible, as only a fixed, typically small amount of data is read back from the hash core. For the LSU-coupled approach, Equation (5) yields $t_{lsu} = t_{store} + t_{hw}$. In fact, as the co-processor is connected directly to the LSU, writing to the co-processor only takes a single cycle and thus, $t_{store} \approx \frac{d_{in}}{4}$ is equal to the amount of input data words where d_{in} is the input data in bytes. If we neglect the terms t_{sw} and t_{hw}, Equation (2) yields $t_{tightly} \approx b_i(66 + T_{perm}) = \lceil \frac{d_{in}}{34\times 4} \rceil \times (66 + T_{perm}) \approx d_{in}\frac{90}{136} > \frac{d_{in}}{4}$, such that $t_{tightly} > t_{lsu}$, which confirms the observation of Figure 3a.

For both loosely-coupled scenarios, t_{load} and t_{store} is larger than in the LSU-coupled case. In fact, writing takes 6 instead of 1 clock cycle, hence $t_{loosely,fifo} > t_{lsu}$ according to Equations (4) and (5). Furthermore, as the loosely-coupled solution using a SRAM interface cannot hide the absorption and squeezing phase, it is $t_{loosely,sram} > t_{loosely,fifo} > t_{lsu}$. Nevertheless, as the data transfer time for small data sizes is negligible, the computational overhead of the tightly-coupled approach exceeds the loosely-coupled variants.

b) Implications for CSPRNG scenario: If only a small seed is absorbed (typically lower than one block of the Keccak primitive), we can neglect the impact of the terms containing b_i, as the absorption phase is negligible. In the LSU-coupled case, we can neglect t_{store} for the same reason. Hence, we end up with $t_{tightly} \approx b_o T_{perm} + t_{sw} = \lfloor \frac{d_{out}}{34\times 4} \rfloor\ 24 + t_{sw} \approx d_{out}\frac{3}{17} + t_{sw}$ and $t_{lsu} \approx t_{load} + t_{hw} = 2\frac{d_{out}}{4} + t_{hw} = \frac{d_{out}}{2} + t_{hw}$, where d_{out} denotes the number of output bytes and loading from the LSU-coupled accelerator requires 2 cycles. As $\frac{3}{17} < \frac{1}{2}$ the tightly-coupled architecture will be faster as soon as the lower computational effort for sufficient amounts of data compensates that $t_{sw} > t_{hw}$. Concerning the two loosely coupled approaches, the same consideration as for the hash scenario holds, i.e. the larger penalty for data transfers dominates the smaller initial overhead compared to the tightly-coupled approach from a certain amount of data on.

B. Application to SPHINCS+

The results in Section IV-A have shown, that either the tightly-coupled or the LSU-coupled approach shows the best performance. Therefore, we restrict our evaluations for SPHINCS+ to these architectures and only present results for the *simple* versions of SPHINCS+ that are considered by NIST [11]. We performed measurements for all NIST security levels but only refer to NIST level I for brevity. We consider



(a) SHAKE256 hash scenario short (b) SHAKE256 CSPRNG scenario short (c) SHAKE256 CSPRNG scenario long

Fig. 3: Performance evaluation of the accelerators in the hash (Figure 3a) and CSPRNG (Figures 3b and 3c) scenario.

message sizes of 59 B as in the pqm4 benchmarks [2] and take the C reference code of the NIST submission as baseline.

A comparison of the cycle counts for both architectures is given in Table II, and is compared with the software implementation that does not use the hardware accelerators. The LSU-coupled architecture shows advantages compared to the tightly-coupled alternatives. These results are in line with the results observed in the benchmark in Figure 3. As discussed in Section II-A, the main hash computations within SPHINCS+ have rather short outputs of only n-byte (in the results shown $n = 16$), the LSU-coupled architecture is expected to be faster, as also the previous benchmark showed an advantage of the LSU-coupled version for all input sizes. Compared to the

TABLE II: SPHINCS+ (128f-simple) cycle counts averaged over 50 iterations.

Architecture	Keygen	Sign	Verify
shake256_sw	145,118,540	3,602,834,821	201,208,510
shake256_lsu	1,724,534	42,597,665	2,457,742
shake256_tightly	2,902,191	71,920,640	4,090,820
asconxof_sw	112,542,937	2,793,035,580	159,128,161
asconxof_lsu	1,732,802	42,867,690	2,475,238
asconxof_tightly	2,297,884	56,895,911	3,281,842
turbo_shake256_sw	85,583,963	2,124,535,303	118,878,833
turbo_shake256_lsu	1,684,079	41,593,651	2,400,881
turbo_shake256_tightly	2,847,867	70,573,841	4,013,566
asconxofa_sw	82,715,595	2,052,961,820	116,680,454
asconxofa_lsu	1,570,919	38,851,303	2,242,401
asconxofa_tightly	2,136,072	52,883,808	3,048,805

software reference, the LSU-coupled architecture yields speed-up factors between $\times 81$ and $\times 84$.

V. Deviation from the SPHINCS+ NIST Submission

In this section, we evaluate deviations from the official NIST submission of SPHINCS+. More specifically, we investigate the use of the Ascon suite [7] and evaluate the changes also for round-reduced versions for Ascon and SHAKE256.

A. Ascon Primitive

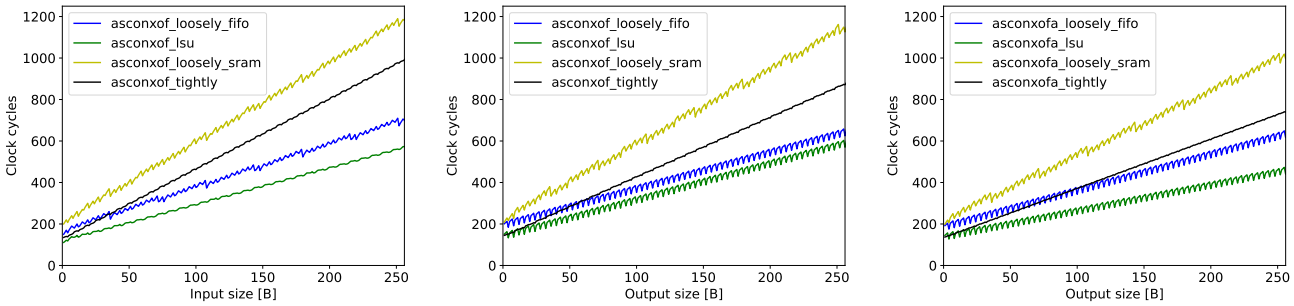
Just as Keccak, Ascon is based on the sponge construction but consists only of a 320-bit state and a round function that is applied for 12 rounds. Due to its smaller state and numbers of rounds, performance gains could be expected. Yet, it has a rate of $r = 64$-bit (or 2 words, compared to SHAKE256's 34 words), such that it requires more permutations when large amounts of data are processed. Furthermore, due to the smaller state, Ascon only provides a (second-) preimage resistance of 128-bit and thus, reaches only NIST security level I if instantiated within SPHINCS+. For evaluation we developed a small hash core that fits into the architectures described above. Moreover, we implemented a tightly-coupled approach as previously presented in [12].

a) FIFO Dimensions: As Ascon permutes for $T_{perm} = 12$ rounds, D_{RX} and D_{TX} of the loosely-coupled architecture are set $D_{RX} = D_{TX} = 2$ according to Equation (1). For the LSU-coupled version, the same dimensions apply although the rationale is different: With a write-delay of only 2 cycles, the microcontroller can provide up to 6 data words while Ascon is permuting. As the rate of Ascon is only 64-bit, i.e. 2 words, data is provided faster than it is processed by the hash core. Thus, setting $D_{RX} = 2$ is sufficient to constantly provide data to the hash core. The same applies for D_{TX}: Data is read faster than the hash core can produce and thus, $D_{TX} = 2$ is sufficient such that the hash core can output a full block and continue permuting the state immediately. As such, the permutation of the Ascon state is the actual bottleneck in the computation.

b) Resource Overhead: The overhead for using the Ascon primitive is also given in Table I. Compared to the SHA-3 core, it shows reductions in both, the LUT and FF count, mostly due to the smaller state. In particular for the tightly-coupled architecture, this overhead is almost neglectable.

c) Architecture Benchmark: Figures 4a and 4b show the architecture benchmark using Asconxof. For the CSPRNG scenario, the loosely-coupled version with FIFOs is similiarly fast as the LSU-coupled architecture (same slope, only a small offset). This indicates that the Ascon core is the actual bottleneck, but not the bus latency. However, this is not the case for the hash scenario, because the data transmission itself hides the computation in the absorption phase. For small amounts of data, the tightly-coupled version is competitive but the arithmetic overhead for a tightly-coupled option poses a notable drawback in both scenarios when the amount of data increases.

d) Application to SPHINCS+: In Table II we also provide the cycle counts using Asconxof for SPHINCS+. It can be seen



(a) Asconxof hash scenario short (b) Asconxof CSPRNG scenario short (c) Asconxofa CSPRNG scenario short

Fig. 4: Performance evaluation of the Ascon accelerators.

that the gap between the two Ascon architectures is smaller than for the SHAKE256 variants. While the LSU-coupled versions of SHAKE256 and Ascon are roughly in the same ballpark, the tightly-coupled Asconxof design yields better results than its SHAKE256 counterpart, as it does not require the FPR. Compared to the software reference implementation, the LSU-coupled architecture yields speed-up factors between $\times 64$ and $\times 65$.

B. Round-reduced Versions

Asconxofa is a round-reduced version of Asconxof. The first permutation and the permutation after absorbing the last input block remain at 12 rounds, all others are reduced to 8 rounds. Reducing the number of rounds within a permutation has also been discussed in the context of the Keccak primitive. As a result, reducing the number of rounds from 24 to 12 has been formalized under the name of TurboSHAKE256 [6]. In the following, we evaluate the advantages of the round-reduced versions for Asconxof and SHAKE256.

a) FIFO Dimensions: For TurboSHAKE256, halving the number of rounds per permutation also enables to halve the FIFO dimensions of the accelerator. For the Asconxofa variant, however, the *RX FIFO* in the loosely-coupled case must be slightly increased to $D_{RX} = 4$.

b) Resource Overhead: As only the FIFO dimensions change slightly, the change in area cost is negligible.

c) Architecture Benchmark: For the architecture benchmark, our results for TurboSHAKE256 look similar as the ones shown in Figure 3, with only minor offsets, which underlines the dominant impact of the data transfer. For Asconxofa, this case is slightly different. As mentioned previously, the Ascon core has been a bottleneck due to its lower rate. This has been visible in Figure 4b, where the slope of the loosely-coupled FIFO version and the LSU-coupled architecture are equal. Yet, Figure 4c shows that this changes for Asconxofa. The cycle counts are reduced and the slope of the LSU- and loosely-coupled FIFO version are different, as the computation within the core is blocking the data transfer less.

d) Application to SPHINCS+: Table II also lists the cycle counts for the round-reduced versions in the context of SPHINCS+. As expected, it shows performance gains for all cases. It is notable, however, that the reduction for the round-reduced version Asconxofa is larger than for TurboSHAKE256. This is due to the obervation made previously, where the computation inside the hash core posed a bottleneck. Reducing the rounds speeds up this computation and thus, resolves this issue. As a consequence, the Asconxofa design yields the highest performance when instantiated in SPHINCS+ using the LSU-coupled architecture. Compared to the software implementations, the LSU-coupled architecture yields speed-up factors of between $\times 49$ and $\times 51$ for TurboSHAKE256 and between $\times 52$ and $\times 53$ for Asconxofa.

VI. Conclusion

The efficiency of different hardware architectures for hash acceleration depend on their specific use-case – whether they are used to compute a hash digest or generate pseudo-random data. For sponge-based hash functions, we have shown, that tightly-coupled architectures are best suited when large amounts of pseudo-randomness are required. On the contrary, if large amounts of data must be digested, architectures where the absorption phase can be hidden in data transfer are better suited. This evaluation illustrates that not only hardware accelerators itself, but also the application including data transfer must be taken into account to find suitable acceleration solutions. As SPHINCS+ heavily relies on the evaluation of hash- and pseudo-random functions with short outputs, the best performance is obtained by the LSU-coupled architecture. Dedicated hardware acceleration for SHAKE256 shows promising performance results even on resource constrained environments. Furthermore, using the Ascon suite over the proposed SHA-3 standard poses a competitive alternative for the applicable security level, especially when taking the resource cost into account.

Acknowledgment

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of “Souverän. Digital. Vernetzt.”. Joint project 6G-life, project identification number: 16KISK002

REFERENCES

[1] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ Signature Framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, nov 2019.

[2] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4," https://github.com/mupq/pqm4, as of commit 918f379.

[3] D. Amiet, L. Leuenberger, A. Curiger, and P. Zbinden, "Fpga-based sphincs$^+$ implementations: Mind the glitch," in *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020*. IEEE, 2020, pp. 229–237. [Online]. Available: https://doi.org/10.1109/DSD51259.2020.00046

[4] Q. Berthet, A. Upegui, L. Gantel, A. Duc, and G. Traverso, "An area-efficient sphincs$^+$ post-quantum signature coprocessor," in *IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2021, Portland, OR, USA, June 17-21, 2021*. IEEE, 2021, pp. 180–187. [Online]. Available: https://doi.org/10.1109/IPDPSW52791.2021.00034

[5] A. Wagner, F. Oberhansl, and M. Schink, "To be, or not to be stateful: Post-quantum secure boot using hash-based signatures," in *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, ASHES 2022, Los Angeles, CA, USA, 11 November 2022*, C. Chang, U. Rührmair, D. Mukhopadhyay, and D. Forte, Eds. ACM, 2022, pp. 85–94. [Online]. Available: https://doi.org/10.1145/3560834.3563831

[6] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, R. V. Keer, and B. Viguier, "Turboshake," *IACR Cryptol. ePrint Arch.*, p. 342, 2023. [Online]. Available: https://eprint.iacr.org/2023/342

[7] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2," 2021, https://ascon.iaik.tugraz.at/specification.html.

[8] N. I. of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Tech. Rep., jul 2015.

[9] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable iot endpoint devices," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 25, no. 10, pp. 2700–2713, 2017. [Online]. Available: https://doi.org/10.1109/TVLSI.2017.2654506

[10] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 239–280, 2020. [Online]. Available: https://doi.org/10.13154/tches.v2020.i4.239-280

[11] D. Moody, "NIST PQC: Looking into the Future," Nov. 2022, https://csrc.nist.gov/Presentations/2022/nist-pqc-looking-into-the-future. [Online]. Available: https://csrc.nist.gov/Presentations/2022/nist-pqc-looking-into-the-future

[12] S. Steinegger and R. Primas, "A fast and compact RISC-V accelerator for ascon and friends," in *Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers*, ser. Lecture Notes in Computer Science, P. Liardet and N. Mentens, Eds., vol. 12609. Springer, 2020, pp. 53–67. [Online]. Available: https://doi.org/10.1007/978-3-030-68487-7_4