# High-Speed Design of Post Quantum Cryptography With Optimized Hashing and Multiplication

Malik Imran, *Graduate Student Member, IEEE*, Aikata Aikata, Sujoy Sinha Roy, *Senior Member, IEEE*, and Samuel Pagliarini, *Member, IEEE*

***Abstract*—In this brief, we realize different architectural techniques for improving the performance of post-quantum cryptography (PQC) algorithms when implemented as hardware accelerators on an application-specific integrated circuit (ASIC) platform. Having SABER as a case study, we designed a 256-bit wide architecture geared for high-speed cryptographic applications that incorporates smaller and distributed SRAM memory blocks. Moreover, we have adapted the building blocks of SABER to process 256-bit words. We have also used a buffering technique for efficient polynomial coefficient multiplications to reduce the clock cycle count. Finally, double-sponge functions are combined serially (one after another) in a high-speed KECCAK core to improve the hash operations of SHA/SHAKE. For key-generation, encapsulation, and decapsulation operations of SABER, our 256-bit wide accelerator with a single sponge function is 1.71x, 1.45x, and 1.78x faster than the raw clock cycle count of a serialized SABER design. Similarly, our 256-bit implementation with double-sponge functions takes 1.08x, 1.07x & 1.06x fewer clock cycles compared to its single-sponge counterpart. The studied optimization techniques are not specific to SABER – they can be utilized for improving the performance of other lattice-based PQC accelerators.**

***Index Terms*—PQC, ASIC design, hardware accelerator, cryptocore, SABER.**

## I. Introduction

HIGH-PERFORMANCE hardware-based cryptographic accelerators are essential for wireless, telecom, cloud, data centers, and enterprise systems. As examples, the 8920 and 8955 Intel chipsets can process 5k and 40k RSA decryption operations per second [1]. The IBM 4769 hardware security module offers key exchange and signature generation/verification using Elliptic Curve Cryptography (ECC) and RSA standards [2]. Even if these remarkable chips deliver thousands of operations per second, they might become compromised since the security strength of ECC and RSA can be broken using Shor's algorithm [3] on a quantum computer. Recently, Google's Sycamore [4] delivered a 53-qubit quantum computer that can do in 200 seconds a task that would take a classical computer 10,000 years. Different labs worldwide have developed even more powerful quantum computers [5]. Hence, high-speed quantum-resistant cryptographic hardware accelerators are mandated to supersede ECC- and RSA-based devices.

Existing architectures for post-quantum cryptography (PQC) algorithms on field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) platforms are demonstrated in [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. These accelerators reveal that the PQC algorithms need secure hash functions for different purposes, e.g., binomial sampling. For instance, the recently standardized CRYSTALS-Kyber algorithm requires variants of SHA3 and an extended output function (EoF), that is, SHAKE. The execution of variants of SHA3 and an EoF depends on a KECCAK sponge function to compute state permutations. The building blocks of the KECCAK sponge function, that is, *theta*, *pi*, *rho*, *chi*, and *iota*, can operate (only) on 64-bit words. This encourages designers to select 64 bits for memory width and for datapaths in their PQC accelerators. This is the case for different PQC accelerators in [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16]. Moreover, PQC algorithms require relatively large storage elements to keep initial, intermediate, and final results. For example, a memory size of 1024×64 is needed to implement different variants of SABER [17], that is, LightSABER, SABER, and FireSABER. There are several possibilities for organizing this memory; one choice is to use one single 1024×64 memory as in [10]. This choice does not allow for parallel read/write operations, resulting in a higher cycle count. Another solution is to use multiple smaller memories like those employed in SABER designs of [11], [12], [13], [14], [15]. These implementations, however, are not taking full benefit of the smaller memories because the read/write operations are performed in a serial way instead of a parallel fashion – even if the memories have different purposes.

Hence, in this brief, we present an **ASIC 256-bit accelerator for SABER** to showcase the advantages of wider datapaths and the memory decisions accompanying it. These advantages also apply to other PQC algorithms. For reducing the clock cycle count, we employed four high-speed SRAM memories of sizes 256×64 each and described their control logic to

Manuscript received 10 February 2023; accepted 3 May 2023. Date of publication 8 May 2023; date of current version 7 February 2024. This work was supported in part by the EC through the European Social Fund in the context of the Project "ICT Programme"; in part by the European Union's H2020 Research and Innovation Programme under Grant 952252 (SAFEST); and in part by the State Government of Styria, Austria—Department Zukunftsfonds Steiermark. This brief was recommended by Associate Editor S. Taneja. *(Corresponding author: Malik Imran.)*

Malik Imran and Samuel Pagliarini are with the Centre for Hardware Security, Department of Computer Systems, Tallinn University of Technology, 12618 Tallinn, Estonia (e-mail: malik.imran@taltech.ee; samuel.pagliarini@taltech.ee).

Aikata Aikata and Sujoy Sinha Roy are with the Institute of Applied Information Processing and Communications, Graz University of Technology, 8010 Graz, Austria (e-mail: aikata@iaik.tugraz.at; sujoy.sinharoy@iaik.tugraz.at).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSII.2023.3273821.

Digital Object Identifier 10.1109/TCSII.2023.3273821

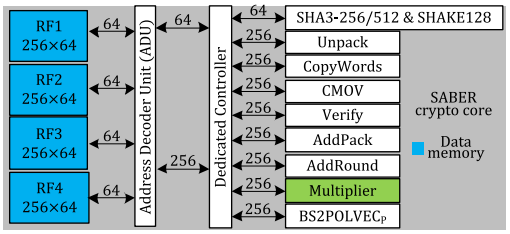


Fig. 1. Block diagram of our proposed crypto accelerator architecture.

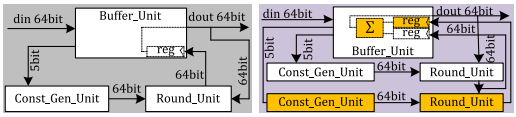


Fig. 2. KECCAK optimizations. The gray diagram corresponds to the high-speed KECCAK core of [19]. The purple diagram indicates our optimized KECCAK where additional blocks appear in orange.

allow for parallel read/write operations. The building blocks of SABER are implemented to process 256-bit words. We have also used a long buffer approach for multiplying polynomial coefficients in parallel. Finally, double-sponge functions are connected serially (one after another) in a high-speed KECCAK core to improve further the studied accelerator's performance.

## II. Proposed Crypto Accelerator

Fig. 1 shows the block diagram of our proposed crypto accelerator architecture. It includes data memory, an address decoder unit, and a SABER crypto core. The data memory holds initial, intermediate, and final results. Each memory can read/write one 64-bit word in one clock cycle. So, four memory instances in parallel can read/write one 256-bit word in one cycle. The address decoder unit selects an appropriate memory for reading/writing a 64-bit word. Also, it communicates to the SABER controller to pass/collect 64-bit (for SHA3 variants) or 256-bit (for other SABER blocks) data as input/output to/from the SABER core. The SABER crypto core includes the required building blocks and is wrapped by a dedicated controller that handles 64-bit or 256-bit data for write/read operations. The controller generates the control signals for the corresponding SABER building blocks. Additionally, it allows one SABER block to operate at a time. Next, we have described the implementation of the SABER blocks.

### A. Optimization of SHA3-256/512 & SHAKE128

Since all of the SHA3 variants utilize the KECCAK sponge function [18], we operate the SHA3-256, SHA3-512, and SHAKE-128 like a wrapper in our proposed architecture. Moreover, details about the utilized KECCAK cores with single- and double-sponge functions are described below.

The gray diagram in Fig. 2 (left) depicts the high-speed KECCAK core of [19]. As can be seen, it needs an instance each of (i) Buffer_Unit, (ii) Const_Gen_Unit and (iii) Round_Unit. The Buffer_Unit holds the initial vectors and keeps the intermediate and final results. Const_Gen_Unit generates the round vectors based on a 5-bit counter value (coming from Buffer_Unit). The Round_Unit is the KECCAK sponge function and operates the KECCAK building blocks (*theta*, *pi*, *rho*, *chi* and *iota*) based on the round constants and a 64-bit buffered value from the Buffer_Unit. Moreover, it generates a 64-bit vector as output which is further connected as an input to a register inside the Buffer_Unit. This strategy requires 28 cycles to operate 24 rounds iteratively: 24 cycles are for 24 KECCAK rounds and an additional 4 cycles specify the 'wait' until the registers in the datapath are free. Previously, the KECCAK core of [19] has been utilized in [10], [14], [15] for SABER hardware accelerators.

The purple-colored diagram in Fig. 2 details how the number of clock cycles of the KECCAK core can be reduced by half using additional orange-colored boxes. We modify the Buffer_Unit by including a register and an accumulator. Moreover, we used additional instances of Const_Gen_Unit and Round_Unit. Each instance of a Const_Gen_Unit takes a 5-bit counter value as input and generates a 64-bit constant vector as an output. Moreover, each instance of the Round_Unit (or sponge function) takes two 64-bit inputs and produces a single 64-bit output. The first 64-bit input to the corresponding sponge function is from the round constants block. The second 64-bit input to the first sponge function is from the KECCAK buffer and its output goes as an input to the second sponge function. This means the sponge functions are connected serially one after another. The outputs of the first and second sponge functions are connected as inputs to the KECCAK buffer to accumulate the results. With this strategy, 14 clock cycles are required to operate 24 rounds of KECCAK. Hence, the cycle count is halved compared to [10], [14], [15].

### B. Fully Parallel Schoolbook Multiplier

We have utilized long public and secret polynomial buffers to load coefficients of public and secret polynomials at once. This one-time data loading from memory helps to reduce the cycle count. For multiplications computation, the long poly buffers need an $m$-bit shift towards left/right. We shift left with 256-bit as our accelerator deals with 256-bit data for reading/writing operations to/from data memory. SABER requires a matrix multiplication for multiplying polynomial coefficients, as presented in Eq. (1). The matrix $P$, $S$ and $R$ hold the public, secret and resultant polynomial coefficients.

$$\begin{bmatrix} P_{(0,0)} & A_{(0,1)} & \dots & P_{(0,255)} \\ P_{(1,0)} & A_{(1,1)} & \dots & P_{(1,255)} \\ P_{(2,0)} & A_{(2,1)} & \dots & P_{(2,255)} \end{bmatrix} \cdot \begin{bmatrix} S_0 \\ S_1 \\ S_2 \end{bmatrix} = \begin{bmatrix} R_0 \\ R_1 \\ R_2 \end{bmatrix} \quad (1)$$

As shown in Fig. 3, our fully parallelized polynomial multiplication architecture consists of two long polynomial buffers (LPPB and LSPB) and three copies of a schoolbook multiplier, that is, SBM1, SBM2, and SBM3. The length of LPPB and LSPB is proportional to the size of the matrix $P$ and matrix $S$, respectively. Each row of matrix $P$ contains 256 13-bit polynomial coefficients. Each row of the matrix $S$ contains 256 4-bit polynomial coefficients. Therefore, 768 coefficients are in three rows of a matrix $P$ and a matrix $S$. Then, the length of LPPB is 9984 bits ($768\times13$) and the length of LSPB is 3072 bits ($768\times4$). Multiplication starts with loading 768 polynomial coefficients into LPPB and LSPB buffers.

After loading all the 768 polynomial coefficients into LPPB and LSPB buffers, the corresponding 256 public and secret

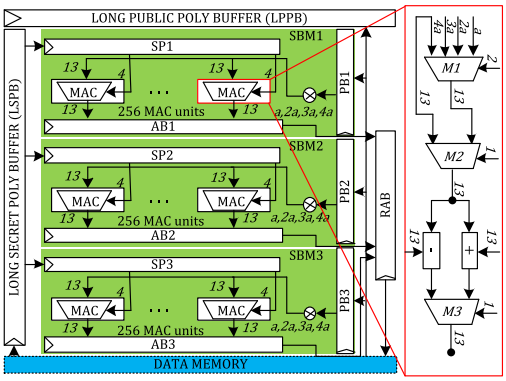


Fig. 3. Fully parallelized schoolbook multiplier for SABER.

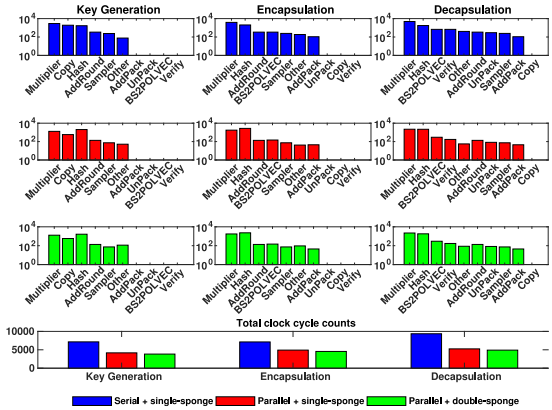


Fig. 4. Clock cycle distribution of SABER for serial and parallel architectures. Serial to parallel designs with single-sponge function results in an average 39% cycle reduction. In parallel designs, moving from single- to double-sponge functions, we obtained a 7% reduction in clock cycles.

polynomial coefficients are forwarded to multipliers SBM1, SBM2 and SBM3. As detailed in Fig. 3, the SBM1 multiplier consists of three buffers (i.e., PB1, SP1, and AB1) and 256 MAC (multiply-and-accumulate) units. PB1 and SP1 contain the 256 coefficients of the first row of the matrix $P$ and matrix $S$ for multiplication. Then, the execution of multiplication using the MAC units takes 256 cycles. Each MAC unit takes 13- and 4-bit public and secret polynomial coefficients as inputs and results in a 13-bit polynomial as output, as presented in Fig. 3. A 13-bit output polynomial from each MAC depends on the 4-bit secret polynomial. Two bits from the LSB side of a secret polynomial decide between shifted 13-bit public polynomial coefficients ($a$, $2a$, $3a$, $4a$) using a multiplexer $M1$. A third bit from the LSB side is a sign bit. Finally, the last bit of a secret polynomial coefficient determines the modular addition or subtraction operation to execute for a 13-bit multiplication result. Moreover, AB1 accumulates the multiplication results. The same multiplication strategy is applied in SBM2 and SBM3 multipliers of Fig. 3. In the SBM2 multiplier, PB2 and SP2 keep the public and secret polynomial coefficients of the second row of the matrix $P$ and matrix $S$. Similarly, PB3 and SP3 hold the public and secret polynomial coefficients for the third row of matrices $P$ and $S$. As presented in Fig. 3, an additional RAB buffer accumulates the multiplication results from SBM1, SBM2, and SBM3 multipliers before writing back on the data memory. Since all three multipliers (SBM1, SBM2 and SBM3) operate in parallel, 256 clock cycles are required to multiply SABER polynomial coefficients.

In our previously implemented schoolbook multipliers of [10], [14], [15], we utilized 256 MAC units, and these MACs are operated serially to compute the polynomial multiplications in 768 clock cycles. In this brief, our fully-parallel multiplier utilizes 768 MAC units and takes 256 cycles. Also, our buffer approach is beneficial to avoid frequent memory access for read/write operations as we have a 256-bit data bus instead of the typical 64-bit size found in the literature. The total cycle cost of loading public and secret polynomials from data memory is 156 and 48 for the schoolbook designs of [10], [14], [15]. The fully-parallelized architecture of this brief reduces these costs to 39 and 12 cycles. As implied by the block diagram of Fig. 3, the area of our multiplier is approximately 3 times of a serialized schoolbook multiplier.

### C. Other Implemented SABER Building Blocks

A sampler is needed to compute the sample from a pseudo-random input string. The binomial sampler in our proposed architecture is a combinational block. It maps 256-bit pseudo-random bits to a 256-bit sample value in one clock cycle. The transformation from a byte into a bit string is the task of the Unpack unit. A copy block is only needed during the KEM key-generation process. It transforms the rows and columns to determine a transpose of a matrix generated using SHAKE128. The verify block is only required during the decapsulation operation. It provides a word-by-word comparison between the received ciphertext and the re-encrypted ciphertext. The result of verify block is stored in a register that is used by CMOV to either copy the decrypted session key or a pseudo-random string at a specified memory address. The AddPack performs coefficient-wise addition with a constant followed by the generated message and packs the resultant bits into a byte string. Like the AddPack block, AddRound computes coefficient-wise addition of a constant followed by coefficient-wise rounding. The BS2POLVEC$_p$ block converts the byte string into a polynomial vector.

## III. Results and Comparisons

In Fig. 4, we show the clock cycle count for serial and parallel SABER architectures. From left to right, the first row with three panels in Fig. 4 specifies the key generation, encapsulation and decapsulation operations for a serial SABER architecture. Similarly, the second row includes three panels for the same three operations on a parallel SABER architecture with a single sponge in its KECCAK block. The considered SABER architecture has double-sponge functions in the third row of Fig. 4. The bottom panel of Fig. 4 provides the total cycle counts for key generation, encapsulation, and decapsulation operations of all three considered designs. Moreover, in Fig. 4, hash determines the SHA3-256/512 and SHAKE128 functions. Notably, the multiplier and hash operations dominate the computation time, so they are prime targets for optimizations.

As expected, Fig. 4 shows a decrease in clock cycles for key generation, encapsulation, and decapsulation operations when moving from a serial to a parallel design with a

TABLE I
RESULTS OF PROPOSED CRYPTO ACCELERATOR ON 28NM TECHNOLOGY

| Implementation details | single-sponge | double-sponge |
|---|---|---|
| Maximum Frequency ($MHz$) | 2500 | 2500 |
| Latency (KG/ENC/DEC) ($\mu s$) | 1.66/1.96/2.09 | 1.53/1.82/1.96 |
| Utilized Area ($mm^2$) | 0.251 | 0.255 |
| Power (Lkg/Dyn) ($mW$) | 10.96/556.25 | 11.49/597.05 |
| Energy ($\mu J$) | 0.923/1.090/1.162 | 0.913/1.086/1.170 |

single-sponge function (see blue and red bars). Similarly, we have a decrease in clock cycles for hash operation when comparing two parallel SABER designs with single- and double-sponge functions in the KECCAK (see red and green bars). The last panel in Fig. 4 highlights the total cycle count for each operation on all architectures. On average, the number of clock cycles required to execute key generation, encapsulation, and decapsulation operations using a parallel accelerator with one sponge function is 1.65× lower compared to the serial SABER architectures of [14], [15]. The use of double-sponge functions in our parallel accelerator further reduces the clock cycle requirement by 1.07× when compared to a parallel implementation with one sponge function. Therefore, a significant decrease in the clock cycle count when moving from a serialized design to parallel architectures, reveals that the realized approaches in this brief can be utilized in other PQC algorithms for performance improvements.

On a commercial 28nm ASIC technology, the frequency, latency, area, power, and energy results (after synthesis) of our proposed parallel SABER architectures are given in Table I. KG, ENC and DEC in Table I define the SABER key-generation, encapsulation, and decapsulation operations. Similarly, Lkg and Dyn are the leakage and dynamic power consumption. We have utilized the Vivado IDE tool for simulations and Cadence Genus for logic synthesis. Both implementations operate at 2500$MHz$. The use of a double-sponge function allows us to minimize the computation time (i.e., latency, calculated as clock cycles over frequency) at a modest increase in power (+4.63% and +6.84% for leakage and dynamic power, respectively) and area (+1.57%). The max frequency is obtained by pushing the timing constraint until the slack is close to zero. Despite the small area and power increase, the double-sponge function has higher merit as it consumes nearly the same energy (product of dynamic power and computation time) than the single-sponge version.

ASIC implementations of recent PQC accelerators are compared in Table II. The clock cycles (CC) and latency (Lat) values are reported for KG, ENC, and DEC operations. Moreover, the architectures marked with the blue checkmark in Table II give the best-in-class results.

Due to the parallel use of smaller SRAM memories, our architecture with a single-sponge function requires 1.73, 1.44 and 1.78 times lower clock cycles for SABER key-generation, encapsulation and decapsulation operations when compared to [14]. Our SABER design with double-sponge functions results in 1.86, 1.57 and 1.89 times lower cycle count. Our single-sponge SABER design requires 1.73, 1.44 and 1.78 times lower computation time (i.e., latency). Similarly, when using a double-sponge function, the latency values are 1.77, 1.47 and 1.89 times lower. As seen in the last two columns of Table II, the area and power values of our designs are higher than [14] as we are utilizing a parallelized 256-bit architecture.

TABLE II
ASIC COMPARISON WITH EXISTING ACCELERATORS FROM LITERATURE

| Ref. | Cycles (K) KG/ENC/DEC | Latency ($\mu s$) KG/ENC/DEC | Freq $MHz$ | Area $mm^2$ | Pow $mW$ |
|---|---|---|---|---|---|
| **65nm technology** | | | | | |
| [14] | 7.1/7.1/9.3 | 7.1/7.1/9.3 | 1002 | 0.314 | 142.5 |
| [12] | 14.3/18.7/23.3 | 89.6/116.9/146.1 | 160 | 0.158 | – |
| [15] | 7.1/7.1/9.3 | 10.0/9.9/13.0 | 715 | 1 | 153.6 |
| [16] | 350/405/425 | 7740/9011/9437 | 45 | 0.840 | 2.6 |
| TW† | 4.1/4.9/5.2 | 4.1/4.9/5.2 ✔ | 1002 | 0.944 | 647.2 |
| TW‡ | 3.8/4.5/4.9 | 4.0/4.8/5.2 ✔ | 936 | 1.026 | 860.9 |
| **40nm technology** | | | | | |
| [11] | 1.0/1.4/1.6 | 2.6/3.6/4.2 | 400 | 0.380 | – |
| TW† | 4.1/4.9/5.2 | 2.45/2.90/3.09 ✔ | 1694 | 0.846 | 163.2 |
| TW‡ | 3.8/4.5/4.9 | 3.47/4.10/4.47 | 1095 | 0.767 | 137.0 |
| **28nm technology** | | | | | |
| [9] | 9/11/13 | 4.54/5.67/6.95 | 2000 | 0.263 | 367.1 |
| [13] | –/–/– | –/–/– | 500 | 3.6 | 39–368 |
| TW† | 4.1/4.9/5.2 | 1.6/1.9/2.0 | 2500 | 0.251 | 567.1 |
| TW‡ | 3.8/4.5/4.9 | 1.5/1.8/1.9 ✔ | 2500 | 0.255 | 608.4 |

TW† & TW‡: our designs with single- & double-sponge, Fabricated: [12], [13], [15], Technology mapped: [9], [11], [14], [16], TW† & TW‡, Area (chip size): [13], [15], CRYSTALS-Kyber: [9], [16], SABER: Others.

A 64-bit SABER chip fabricated in [12] requires 3.48, 3.81, and 4.48 times higher clock cycles compared to our parallel SABER design with a single-sponge function. With double-sponge functions, the cycle requirement of our design is 3.76, 4.15, and 4.48 times lower than [12]. Our 256-bit implemented SABER design with single-sponge and double-sponge functions show 6.26 and 5.85 times speedup in clock frequency. Moreover, our single-sponge SABER design displays 21.85, 23.85, and 28.09 times lower latency. For double-sponge functions, the required computation time is 22.4, 24.35, and 28.09 times lower. We utilize 5.97 and 6.49 times more hardware resources with single and double-sponge functions. Two different operating frequencies, 160$MHz$, and 10$MHz$ are reported in [12]. For 160$MHz$, the consumed power is not reported in the reference design. However, for 10$MHz$, the consumed power is 0.3339$mW$. Our parallel SABER architectures with single- and double-sponge functions consume 647.2 and 860.9$mW$ power at 1002 and 936$MHz$ clock frequency. This increase is expected given that our frequency of operation is 1-2 orders of magnitude higher.

If we compare our results to [15], our proposed design with a single-sponge function takes 1.73, 1.44, and 1.78 times lower clock cycles. The design with double-sponge functions requires 1.86, 1.57, and 1.89 times fewer clock cycles. The reasons are the parallel use of smaller memories and a fully parallel multiplier in our SABER design. On the other hand, in [15], smaller memories are accessed serially and an iterative schoolbook multiplier is utilized. Our single-sponge and double-sponge implemented SABER designs are 1.40 and 1.30 times faster (in frequency). As shown in column three of Table II, the computational cost of our SABER design is much lower than [15]. Column five shows that our SABER core utilizes an area (almost) equivalent to the chip size of [15]. Due to parallel computations in this brief, our single-sponge and double-sponge functions consume 4.21 and 5.60 times higher power than [15]. There is always a trade-off between processing speed and area/power parameters.

We have achieved very interesting results on 40nm process technology. SABER designs with single-sponge and double-sponge functions utilize 0.079$\mu m^2$ and 0.115$\mu m^2$ area

for SHA3-256/512 and SHAKE128. For identical SABER designs, the hardware utilization of our fully parallelized multiplier is $0.637\mu m^2$ and $0.523\mu m^2$. In Table II, if we see the total utilized area and consumed power of our design with single-sponge and double-sponge functions, the SABER design with double-sponge functions takes lower resources and consumes less power than the SABER design with single-sponge function. This is counterintuitive at first but becomes clear once we notice the significant frequency decrease, from $1694MHz$ to $1095MHz$, in column four of Table II. This happens due to the different critical paths shifting from one design to another (the double-sponge becomes the critical path).In other words, the critical paths of the synthesis of our single- & double-sponge functions lie in the KECCAK but with different start & end points. Additionally, the choice between single- and double-sponge is a function of the technology: the relative speed of logic versus that of the memory dictates where the critical path lies and whether the design can accommodate a double-sponge KECCAK. This consideration applies not only to SABER but also to other PQC accelerators.

Compared to [11], our parallel designs take more clock cycles for KG, ENC, and DEC operations of SABER. The reason is the parallel use of smaller memories in our design while dedicated memories for specific SABER computations are utilized in [11]. Our SABER design with single-sponge and double-sponge functions is 4.23 and 2.73 times faster in clock frequency. Moreover, our implementation with a single-sponge function requires lower computation time (see column three in Table II). We are utilizing more area than [11] because our focus was to reduce the computation time and improve the circuit frequency. The comparison with power results is impossible as they are unavailable in the reference design.

A flexible design [13] for SABER, NTRU, Dilithium, Rainbow, CRYSTALS-Kyber and McEliece PQC algorithms is five times slower in clock frequency than our dedicated SABER design. The utilized area is in chip size ($3.6mm^2$), as seen in Table II, so a fair one-to-one comparison is impossible. Similarly, a reasonable comparison with consumed power is impossible as the power values (are given in a) range from $39mW$ to $368mW$. The information about the clock cycle and latency parameters is not reported in the reference design of [13]. Therefore, this comparison is (also) not possible.

On 65 and 28nm technologies, the flexible accelerators of [9], [16] implement multiple PQC algorithms and, as expected, their area is higher than in our SABER accelerators. If we consider a variant of CRYSTALS-Kyber (i.e., Kyber-1024) from these accelerators for comparison, Table II shows that our accelerators outperform in clock cycles, latency and frequency. The consumed power of our accelerators is high because we operate related operations on a much higher frequency.

## IV. Lessons Learned & Conclusion

The comparison and discussions reveal that the parallel use of several smaller memories is more beneficial to reduce frequent read/write access from the data memory. One-time data loading from data memory helps to decrease clock cycles. Also, the one-time loading benefits the design of a compact and a parallel NTT (number-theoretic-transform) multiplier for CRYSTALS-Kyber and CRYSTALS-Dilithium PQC standards. The PQC algorithms involve secure hash computations; hence, efficient hash computations allow optimization of the circuit frequency and also help to minimize the cycles.

This brief shows that our SABER design with a single-sponge function performs better in achieving higher clock frequency on 65nm and 40nm process technologies. However, on a 28nm technology, our SABER designs with single- and double-sponge functions outperform the state-of-the-art in frequency and latency. The adopted wider datapath strategy, one-time data loading approach and KECCAK optimizations can be considered in high-speed implementations of CRYSTALS-Kyber and CRYSTALS-Dilithium accelerators.

## References


[1] Intel. "Integrated cryptographic and compression accelerators on Intel architecture platforms." Accessed: Sep. 29, 2022. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/quickassist-adapter-8920-brief.pdf

[2] IBM. "IBM CEX7S/4769 PCIe cryptographic coprocessor (HSM)." Accessed: Oct. 20, 2022. [Online]. Available: https://public.dhe.ibm.com/security/cryptocards/pciecc4/docs/4769_Data_Sheet.pdf

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[4] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, Oct. 2019.

[5] M. Gong et al., "Quantum walks on a programmable two-dimensional 62-qubit superconducting processor," *Science*, vol. 372, no. 6545, pp. 948–952, 2021.

[6] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of crystals-dilithium," *Proc. Int. Conf. Field-Programmabale Technol. (ICFPT)*, Auckland, New Zeland, 2021, pp. 1–10.

[7] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal—Implementing Dilithium on reconfigurable hardware," in *Proc. 20th Int. Conf. Smart Card Res. Adv. Appl. (CARDIS)*, 2021, pp. 210–230.

[8] Z. Zhou, D. He, Z. Liu, M. Luo, and K.-K. R. Choo, "A software/hardware co-design of crystals-Dilithium signature scheme," *ACM Trans. Reconfig. Technol. Syst.*, vol. 14, no. 2, pp. 1–21, 2021.

[9] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "KaLi: A crystal for post-quantum security using kyber and dilithium," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 2, pp. 747–758, Feb. 2023.

[10] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, pp. 443–466, Apr. 2020.

[11] Y. Zhu et al., "LWRpro: An energy-efficient configurable crypto-processor for module-LWR," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 3, pp. 1146–1159, Mar. 2021.

[12] A. Ghosh et al., "A 334$\mu$W 0.158mm$^2$ saber learning with rounding based post-quantum crypto accelerator," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2022, pp. 1–2.

[13] Y. Zhu et al., "A 28nm 48KOPS 3.4$\mu$J/op agile crypto-processor for post-quantum cryptography on multi-mathematical problems," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 65, 2022, pp. 514–516.

[14] M. Imran, F. Almeida, J. Raik, A. Basso, S. S. Roy, and S. Pagliarini, "Design space exploration of SABER in 65nm ASIC," in *Proc. 5th Workshop Attacks Solutions Hardw. Security*, 2021, pp. 85–90.

[15] M. Imran, F. Almeida, A. Basso, S. S. Roy, and S. Pagliarini, "High-speed SABER key encapsulation mechanism in 65nm CMOS," *J. Cryptograph. Eng.*, to be published. [Online]. Available: https://doi.org/10.1007/s13389-023-00316-2

[16] T. Fritzmann, G. Sigl, and J. Sepúlveda, "RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 4, pp. 239–280, Aug. 2020.

[17] A. Basso et al. "SABER: Mod-LWR based KEM (round 3 submission)." Accessed: Mar. 23, 2022. [Online]. Available: https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf

[18] *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, document FIPS PUB 202, NIST, Gaithersburg, MD, USA, Accessed: Mar. 9, 2022. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.202

[19] Keccak Team. "Keccak in VHDL: High-speed core." Accessed: Sep. 16, 2022. [Online]. Available: https://keccak.team/hardware.html