Towards High-speed ASIC Implementations of Post-Quantum Cryptography

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

***Abstract*—In this brief, we realize different architectural techniques towards 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 buffer 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 compared to 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. For these applications, 8920 and 8955 families of Intel chipsets can process 5k and 40k RSA decryption operations in one second [1]. Moreover, IBM 4769 hardware security module offers different security services such as 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. Hence, high-speed quantum-resistant cryptographic hardware accelerators are mandated to supersede ECC- and RSA-based devices.

This work was partially supported by the EC through the European Social Fund in the context of the project "ICT programme". It was also partially supported by European Union's Horizon 2020 research and innovation programme under grant agreement No 952252 (SAFEST). It is also partially supported by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

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

A. Aikata and S. S. Roy are affiliated to Institute of Applied Information Processing and Communications, Graz University of Technology, Graz, Austria. (e-mail: {aikata, sujoy.sinharoy}@iaik.tugraz.at.)

Existing architectures for post-quantum cryptography (PQC) algorithms on field-programmable gate array (FPGA) and on application-specific integrated circuit (ASIC) platforms are demonstrated in [4]–[14]. 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), i.e., 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, i.e., $theta$, 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 [4]–[14]. 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 [15], i.e., LightSABER, SABER, and FireSABER. There are several possibilities on how to organize this memory; one choice is to use one single 1024×64 memory as in [9]. This choice does not allow for parallel read/write operations, resulting in a higher cycle count for the overall PQC algorithm. Another solution is to use multiple smaller memories like those employed in SABER designs of [10]–[14]. 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 that accompany 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 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 further improve the performance of the studied accelerator.

II. Proposed Crypto Accelerator

Fig. 1 shows the block diagram of our proposed crypto accelerator architecture. It consists of a data memory, an address decoder unit, and a SABER crypto core. The data

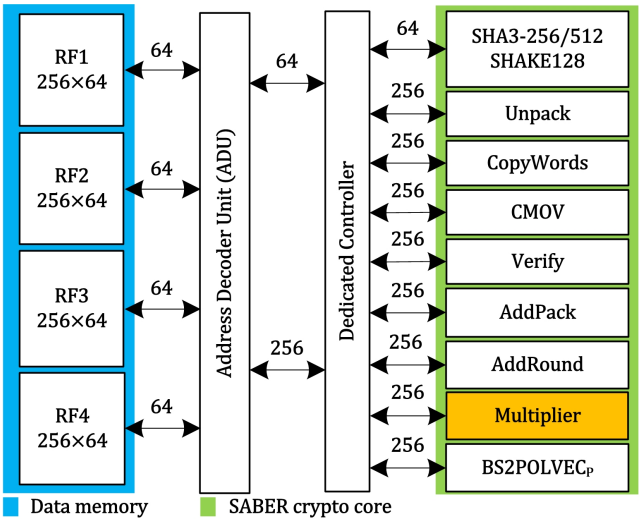

Fig. 1. Block diagram of our proposed crypto accelerator architecture.

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 clock 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 and SHAKE128*

Since all of SHA3 variants utilize the KECCAK sponge function [16], 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.

1) KECCAK with single-sponge function: The high-speed KECCAK core of [17] requires an instance of (i) buffers to hold the initial vectors and to keep the intermediate and the final results, (ii) KECCAK round constants block to generate the round vectors and (iii) KECCAK sponge function to operate the KECCAK building-blocks ($theta$, pi, rho, chi and $iota$) based on the round constants. The high-speed KECCAK core of [17] needs 28 clock cycles to operate 24 rounds in an iterative fashion: 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 [17] has been utilized in [9], [13], [14] for SABER hardware accelerators.

2) KECCAK with double-sponge function: The number of clock cycles of the KECCAK core of [17] can be reduced by half using (i) an instance of a KECCAK buffer, (ii) two instances of KECCAK round constants and (iii) two instances of KECCAK sponge functions. The KECCAK buffer holds the initial, intermediate, and final outputs. Each instance of a round constant block takes a 5-bit counter value as input and generates a 64-bit constant vector as an output. Moreover, each instance of the 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 output of the second sponge function is connected as an input to the KECCAK buffer to accumulate the results. With this strategy, 14 clock cycles are required to operate 24 rounds of KECCAK. Hence, the clock cycle count is halved as compared to [9], [13], [14].

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 polynomial 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. Note that SABER requires a matrix multiplication for multiplying polynomial coefficients, as presented in Eq. 1. Matrix P, S and R hold the public, secret and resultant polynomial coefficients.

$$\begin{bmatrix} \mathrm{P}_{(0,0)} & \mathrm{A}_{(0,1)} \ldots & \mathrm{P}_{(0,255)} \\ \mathrm{P}_{(1,0)} & \mathrm{A}_{(1,1)} \ldots & \mathrm{P}_{(1,255)} \\ \mathrm{P}_{(2,0)} & \mathrm{A}_{(2,1)} \ldots & \mathrm{P}_{(2,255)} \end{bmatrix} \begin{bmatrix} \mathrm{S}_0 \\ \mathrm{S}_1 \\ \mathrm{S}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{R}_0 \\ \mathrm{R}_1 \\ \mathrm{R}_2 \end{bmatrix} \quad (1)$$

As shown in Fig. 2, our fully parallelized polynomial multiplication architecture consists of two long polynomial buffers (LPPB and LSPB) and three copies of a schoolbook multiplier, i.e., SBM1, SBM2, and SBM3. The length of LPPB and LSPB is proportional to the size of matrix P and matrix S, respectively. Each row of matrix P contains 256 13-bit polynomial coefficients. Each row of matrix S contains 256 4-bit polynomial coefficients. Therefore, 768 coefficients are in three rows of a matrix A and a matrix S. Then, the length of LPPB is 9984 bits (768×13) and the length of LSPB is 3072 bits (768×4). 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 coefficients of public and secret polynomials are forwarded to multipliers SBM1, SBM2 and SBM3. As detailed in Fig. 2, 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 matrix A and matrix S for multiplication. Then, the multiplication using the MAC units takes 256 cycles to complete. 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. 2. 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,

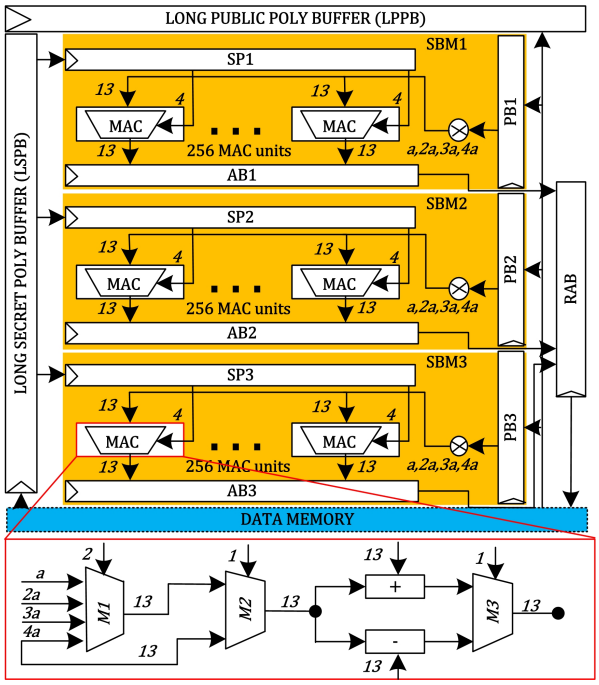

Fig. 2. Fully parallelized schoolbook multiplier for SABER.

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. 2. In the SBM2 multiplier, PB2 and SP2 keep the public and secret polynomial coefficients of the second row of 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. 2, 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.

The architectures described in [9], [13], [14] need 768 clock cycles for multiplication. Our fully-parallelized multiplier takes 256 cycles. Also, our buffer approach is beneficial to avoid frequent memory access for reading/write operations as we have 256-bit data bus instead the typical 64-bit size found in the literature. More precisely, the total clock cycle cost of loading public and secret polynomials from data memory is 156 and 48 for schoolbook designs of [9], [13], [14]. The fully-parallelized architecture of this work reduces these costs to 39 and 12 cycles. As implied by the block diagram of Fig. 2, the area of our multiplier is approximately 3 times that 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

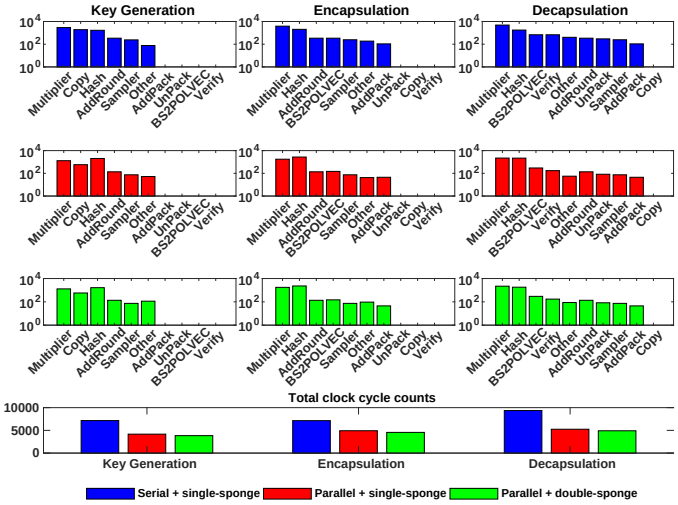

Fig. 3. Clock cycle distribution of SABER for serial and parallel architectures.

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 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 it 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. 3, we show the clock cycle count for serial and parallel SABER architectures. The first row with three panels in Fig. 3, from left to right, 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. In the third row of Fig. 3, the considered SABER architecture has double-sponge functions. The bottom panel of Fig. 3 provides the total cycle counts for key generation, encapsulation, and decapsulation operations of all three considered designs. Moreover, in Fig. 3, hash determines the SHA3-256/512 and SHAKE128 functions. It is noteworthy that the multiplier and hash operations dominate the computation time, so they are prime targets for optimizations.

As expected, Fig. 3 shows a decrease in clock cycles for key generation, encapsulation, and decapsulation operations when moving from a serial to a parallel design with a 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. 3 highlights the total cycle count for each operation on all architectures. On average, the number of

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) (μs)	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

TABLE II
COMPARISON TO EXISTING ASIC IMPLEMENTATIONS OF SABER.

Ref.	Cycles (K) KG/ENC/DEC	Latency (μs) KG/ENC/DEC	Freq MHz	Area mm^2	Pow mW
65nm technology					
[13]	7.1/7.1/9.3	7.1/7.1/9.3	1002	0.314	142.5
[11]	14.3/18.7/23.3	89.6/116.9/146.1	160	0.158	–
[14]	7.1/7.1/9.3	10.0/9.9/13.0	715	1	153.6
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					
[10]	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					
[12]	–/–/–	–/–/–	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† and TW‡ are our designs with single- & double-sponge functions.
Fabricated: [11], [12], [14], Technology mapped: [10], [13], TW† & TW‡
Area is in chip size for [12], [14].

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 [13], [14]. 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 work can be utilized in other PQC algorithms for performance improvements.

On a commercial 28nm ASIC technology, the frequency, latency, area, and power results 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. Both implementations operate at $2500MHz$. 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%).

Known ASIC implementations of SABER 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.

1) Comparison on 65nm: 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 [13]. Our SABER design with double-sponge functions results in 1.86, 1.57 and 1.89 times lower clock 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 as compared to [13] as we are utilizing a parallelized 256-bit architecture.

A 64-bit SABER chip fabricated in [11] requires 3.48, 3.81 and 4.48 times higher clock cycles compared to our parallel SABER design having a single-sponge function. With double-sponge functions, the clock cycle requirement of our design is 3.76, 4.15, and 4.48 times lower than [11]. 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 are utilizing 5.97 and 6.49 times more hardware resources with single and double-sponge functions. Two different operating frequencies, $160MHz$, and $10MHz$, are reported in [11]. For $160MHz$, the consumed power is not reported in the reference design. However, for $10MHz$, the consumed power is $0.3339mW$. Our parallel SABER architectures with single- and double-sponge functions consume 647.2 and $860.9mW$ power at 1002 and $936MHz$ 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 [14], our proposed design with single-sponge function takes 1.73, 1.44, and 1.78 times lower clock cycles. SABER 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 parallelized multiplier in our SABER design. On the other hand, smaller memories are accessed serially in [14]. Moreover, an iterative schoolbook multiplier is utilized in [14]. 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 [14]. Column five shows that our SABER core utilizes an area (almost) equivalent to the chip size of [14]. Due to parallel computations in this work, our single-sponge and double-sponge functions consume 4.21 and 5.60 times higher power than [14]. There is always a trade-off between processing speed and area/power parameters, naturally.

2) Comparison on 40nm: 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 as compared to 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). 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 [10], 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 [10]. 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 compared to [10] because our focus was to reduce the computation time and improve the circuit frequency. The comparison with power results is not possible as they are not available in the reference design.

3) Comparison on 28nm: A flexible design [12] for SABER, NTRU, Dilithium, Rainbow, Kyber and McEliece PQC algorithms is five times slower in clock frequency as compared to 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 not possible. Similarly, a reasonable comparison with consumed power is not possible 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 [12]. Therefore, this comparison is (also) not possible.

IV. Lessons Learned & Conclusions

In a nutshell, comparison and the 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. This aids the design of parallel multipliers for efficient polynomial coefficient multiplications. Also, the one-time loading benefits the design of a compact and a parallel NTT (number-theoretic-transform) multiplier for the present Crystals-Kyber and Crystals-Dilithium PQC standards. In addition, it assists designers to minimize the critical path of their circuits. Almost all the PQC algorithms involve secure hash computations for different purposes. Hence, efficient hash computations allow optimization of the circuit frequency and also help to minimize the clock cycles.

This article shows that our SABER design with a single-sponge function performs better in achieving higher clock frequency on 65nm and 40nm process technologies. On a 28nm technology, however, our SABER designs with single-sponge and double-sponge functions outperform the state of the art in both frequency and latency. The realized approaches in this work are practical to other lattice-based PQC algorithms to improve circuit frequency, reducing computation time and cycle count for high-speed cryptographic applications.

References

[1] Intel, "Integrated cryptographic and compression accelerators on intel architecture platforms," last accessed on September 29, 2022, available at: https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/integrated-cryptographic-compression-accelerators-brief.pdf.

[2] IBM, "Ibm cex7s / 4769 pcie cryptographic coprocessor (hsm)," last accessed on October 20, 2022, available at: 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, p. 1484–1509, 1997.

[4] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of crystals-dilithium," Cryptology ePrint Archive, Paper 2021/1451, 2021, https://eprint.iacr.org/2021/1451. [Online]. Available: https://eprint.iacr.org/2021/1451

[5] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal - implementing dilithium on reconfigurable hardware," in *Smart Card Research and Advanced Applications: 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11–12, 2021, Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 210–230. [Online]. Available: https://doi.org/10.1007/978-3-030-97348-3_12

[6] 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. Reconfigurable Technol. Syst.*, vol. 14, no. 2, jun 2021. [Online]. Available: https://doi.org/10.1145/3447812

[7] A. Jati, N. Gupta, A. Chattopadhyay, and S. K. Sanadhya, "A configurable crystals-kyber hardware implementation with side-channel protection," Cryptology ePrint Archive, Paper 2021/1189, 2021, https://eprint.iacr.org/2021/1189. [Online]. Available: https://eprint.iacr.org/2021/1189

[8] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "Kali: A crystal for post-quantum security using kyber and dilithium," Cryptology ePrint Archive, Paper 2022/1086, 2022, https://eprint.iacr.org/2022/1086. [Online]. Available: https://eprint.iacr.org/2022/1086

[9] S. Sinha Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, p. 443–466, 2020.

[10] Y. Zhu, M. Zhu, B. Yang, W. Zhu, C. Deng, C. Chen, S. Wei, and L. Liu, "Lwrpro: An energy-efficient configurable crypto-processor for module-lwr," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1146–1159, 2021.

[11] A. Ghosh, J. Mera, A. Karmakar, D. Das, S. Ghosh, I. Verbauwhede, and S. Sen, "A $334\mu w$ $0.158mm^2$ saber learning with rounding based post-quantum crypto accelerator," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*, 2022, pp. 1–2.

[12] Y. Zhu, W. Zhu, M. Zhu, C. Li, C. Deng, C. Chen, S. Yin, S. Yin, S. Wei, and L. Liu, "A 28nm 48kops 3.4µj/op agile crypto-processor for post-quantum cryptography on multi-mathematical problems," pp. 514–516, 2022, iEEE International Solid State Circuits Conference (ISSCC), San Francisco, CA, USA, p. 514–516, February 20-26, 2022.

[13] M. Imran, F. Almeida, J. Raik, A. Basso, S. S. Roy, and S. Pagliarini, "Design space exploration of saber in 65nm asic," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 85–90. [Online]. Available: https://doi.org/10.1145/3474376.3487278

[14] M. Imran, F. Almeida, A. Basso, S. S. Roy, and S. Pagliarini, "High-speed saber key encapsulation mechanism in 65nm cmos," Cryptology ePrint Archive, Paper 2022/530, 2022, https://eprint.iacr.org/2022/530. [Online]. Available: https://eprint.iacr.org/2022/530

[15] A. Basso, J. M. B. Mera, J.-P. D'Anvers, A. Karmakar, S. S. Roy, M. V. Beirendonck, and F. Vercauteren, "Saber: Mod-lwr based kem (round 3 submission)," last accessed on March 23, 2022, available at https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf.

[16] NIST, "Sha-3 standard: Permutation-based hash and extendable-output functions," FIPS PUB 202, last accessed on March 9, 2022, available at https://doi.org/10.6028/NIST.FIPS.202.

[17] K. Team, "Keccak in vhdl: High-speed core," last accessed on September 16, 2022, available at: https://keccak.team/hardware.html.