Hardware-Implemented Lightweight Accelerator for Large Integer Polynomial Multiplication

Pengzhou He, Yazheng Tu, Çetin Kaya Koç, *Fellow, IEEE*, and Jiafeng Xie, *Senior Member, IEEE*

***Abstract*—Large integer polynomial multiplication is frequently used as a key component in post-quantum cryptography (PQC) algorithms. Following the trend that efficient hardware implementation for PQC is emphasized, in this letter, we propose a new hardware-implemented lightweight accelerator for the large integer polynomial multiplication of Saber (one of the National Institute of Standards and Technology third-round finalists). First, we provided a derivation process to obtain the algorithm for the targeted polynomial multiplication. Then, the proposed algorithm is mapped into an optimized hardware accelerator. Finally, we demonstrated the efficiency of the proposed design, e.g., this accelerator with $v = 32$ has at least 48.37% less area-delay product (ADP) than the existing designs. The outcome of this work is expected to provide useful references for efficient implementation of other PQC.**

***Index Terms*—Hardware implementation, large integer polynomial multiplication, lightweight accelerator, post-quantum cryptography.**

I. Introduction

IT IS known that large integer polynomial multiplication is the critical component for many post-quantum cryptography (PQC) algorithms. In particular, polynomial multiplication over ring $\mathbb{Z}_l/(x^N+1)$ (l is q/p [1]) is the bottleneck arithmetic operation for key encapsulation mechanism (KEM) Saber (the National Institute of Standards and Technology third-round PQC standardization finalist) as it involves unequal-sized coefficients between two input polynomials and it is challenging to be implemented. Following the recent research trend [2], efficient hardware acceleration for this polynomial multiplication is of great importance.

Current Efforts. There are two types of hardware implementations for the targeted polynomial multiplication: the high-speed one and the lightweight design. The former type includes two recent designs in [3] and [4] and three other optimized high-performance architectures in [5], [6], and [7], respectively. While for the latter type: the first design was presented in [3]; two designs were then respectively given in [5] and [8]. These works are the major efforts in the field.

Manuscript received 7 February 2023; accepted 4 April 2023. Date of publication 10 May 2023; date of current version 30 May 2023. This work of Çetin Kaya Koç was supported in part by TUBITAK Project under Grant 1001-121F348. This work of Jiafeng Xie was supported in part by NIST-60NANB20D203 and in part by NSF under Grant SaTC-2020625. *(Corresponding author: Jiafeng Xie.)*

Pengzhou He, Yazheng Tu, and Jiafeng Xie are with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085 USA (e-mail: phe@villanova.edu; ytu1@villanova.edu; jiafeng.xie@villanova.edu).

Çetin Kaya Koç is with the University of California, Santa Barbara, Santa Barbara, CA 93106 USA, and also with the Iğdir University, NUAA, 76000 Merkez/Iğdir, Turkey (e-mail: etinkoc@ucsb.edu).

Digital Object Identifier 10.1109/LCA.2023.3274931

Our observation is that most of the existing designs were tailored for high-performance operations while lightweight designs were less released [5], [9], [10]. Meanwhile, we notice that polynomial multiplications for different PQC have been investigated recently [11], [12], [13], [14]. Following this trend, we propose an efficient hardware-implemented lightweight polynomial multiplication accelerator (LPMA) for Saber on the field-programmable gate array (FPGA).

Main Contribution. The challenge is that the targeted large integer polynomial multiplication involves unequal-sized coefficients of the input polynomials, and thus, standard fast algorithms are not possible to deploy. For instance, the pre-addition related operations in the Karatsuba method will increase the bit-width in the subsequent point-wise multiplications, and thus, the increased overhead may offset the gain from deploying Karatsuba. Meanwhile, the targeted polynomial multiplication uses number theoretic transform (NTT) unfavored parameter sets. To obtain a lightweight design, we have derived the schoolbook algorithm into a new format, i.e., a novel multi-channel processing technique, for efficient computation and flexible processing. In summary,

- We have given a detailed derivation process to obtain the proposed polynomial multiplication algorithm.
- We have presented a design process to transfer the proposed algorithm into desired LPMA.
- We have provided complexity analysis and comparison to show the efficiency of the proposed design.

This paper is organized as follows. The preliminary is introduced in Section II. The proposed algorithm is presented in Section III. The hardware accelerator is described in Section IV. Complexity and comparison are provided in Section V. Finally, conclusions are given in Section VI.

II. Preliminary Knowledge

Polynomial Multiplication for Saber: Large integer polynomial multiplication over ring $\mathbb{Z}_l/(x^N+1)$ (l is q or p) is the major arithmetic operation of Saber [1]. Based on [1], we have $N = 256$ and the two moduli as $q = 2^{13}$ and $p = 2^{10}$, respectively. Meanwhile, in the polynomial multiplication of Saber, one polynomial has coefficients in the range [-5,5], [-4,4], and [-3,3] (according to the three security ranks), respectively; while another polynomial has coefficients of 10-bit/13-bit (13-bit design covers the 10-bit one).

III. Algorithmic Derivation

Notation 1. Define $D = \sum_{i=0}^{N-1} d_i x^i$, $G = \sum_{i=0}^{N-1} g_i x^i$, and $W = \sum_{i=0}^{N-1} w_i x^i$, where d_i, g_i, and w_i are 13-bit, 4-bit, and 13-bit coefficients over $\mathbb{Z}_q$, respectively. Meanwhile, we define W

as the product of D and G, and thus, we have $(f(x) = x^N + 1)$

$$W = DG \bmod f(x), \tag{1}$$

which is substituted with $x^N \equiv -1$ $(x^N + 1 \equiv 0)$ to have

$$w_0 = g_0 d_0 - g_{N-1} d_1 - \cdots - g_1 d_{N-1},$$
$$\cdots\cdots\cdots,$$
$$w_{N-1} = g_{N-1} d_0 + g_{N-2} d_1 + \cdots + g_0 d_{N-1}. \tag{2}$$

Limitations of the Existing Processing Strategies: The first lightweight accelerator for the targeted polynomial multiplication was based on the Toom-Cook algorithm, however, it involves the usage of too many DSPs [3]. Another accelerator of [5] has small area usage, but it involves long latency and requires frequent assistance with the external memory. Design of [8] has used the strategy of sharing DSPs to speed up the computation at the cost of two memory usage. For practical application, it is desirable that: (i) the major computation can be executed within the accelerator (even without the support of external memory); (ii) low resource usage; (iii) flexible processing choices.

Proposed Algorithmic Processing Strategy: It is observed that all the coefficients of G in w_{N-1} have positive signs, and thus we can consider w_{N-1} and w_{N-2} of (2) first

$$w_{N-1} = g_{N-1} d_0 + g_{N-2} d_1 + \cdots + g_0 d_{N-1},$$
$$w_{N-2} = g_{N-2} d_0 + g_{N-3} d_1 + \cdots - g_{N-1} d_{N-1}, \tag{3}$$

where the positions of coefficients of G only shift by one position in a circular format (not including the sign) between w_{N-1} and w_{N-2}, while the positions of coefficients of D remain the same. If we adjust w_{N-2} by one position, i.e.,

$$w_{N-1} = g_{N-1} d_0 + g_{N-2} d_1 + g_{N-3} d_2 + \cdots + g_0 d_{N-1},$$
$$w_{N-2} = -g_{N-1} d_{N-1} + g_{N-2} d_0 + \cdots + g_0 d_{N-2}, \tag{4}$$

where the same data sequence of coefficients of G can be shared between w_{N-1} and w_{N-1} (only the sign of g_{N-1} is inverted), i.e., only one identical input processing component is needed (Section IV). Note the coefficients of D are serially fed in for related multiplication and accumulation.

This property exists in all neighboring w_j of (2), and from which we can propose a new *multi-channel processing technique* to improve the computation efficiency and the implemented architecture, i.e., the coefficients of G within neighboring W_j (not including the signs), can be shared between multiple processing channels.

Notation 2: Define $N = uv$, where u and v are integers. We then divide w_j of (2) into u groups as W_0, …, W_{u-1}, where $W_0 = \{w_{v-1}, w_{v-2}, \ldots, w_0\}$, $W_1 = \{w_{2v-1}, \ldots, w_v\}$, …, $W_{u-1} = \{w_{N-1}, , w_{N-2}, \ldots, w_{N-v}\}$. We also define G^0, G^1, …, G^{u-1} represent the corresponding coefficients (including the signs) of G for W_0, …, W_{u-1}, respectively. Moreover, G_0^{u-1} denotes all the coefficients of G for w_{N-1} and the same to $G_1^{u-1}, \ldots, G_{v-1}^{u-1}$. Finally, we have $G_{0,0}^{u-1} = g_{N-1}$, …, $G_{0,N-1}^{u-1} = g_0$, which apply to G^0, …, G^{u-2}.

Based on Notation 2, we can then consider W_{u-1} first as

$$w_{N-v} = -g_{N-1} d_{N-v+1} - \cdots + g_0 d_{N-v},$$
$$\cdots\cdots\cdots$$
$$w_{N-1} = g_{N-1} d_0 + g_{N-2} d_1 + g_{N-3} d_2 \cdots + g_0 d_{N-1}, \tag{5}$$

where the coefficients of G are processed in the same sequence for each w_j $(j = N-1$ to $N-v)$, i.e., from g_{N-1}, …, to g_0. Besides that, the values of D of all w_j (matched with related coefficients of G) are regularly arranged. For instance, for g_{N-1} of all w_j, the matched coefficients of D are $\{d_0, d_{N-1}, \ldots, d_{N-v+1}\}$, which becomes $\{d_1, d_0, \ldots, d_{N-v+2}\}$ for d_1 of all w_j (this feature exists in other values of G of (5), and we can use $S(\cdot)$ to denote it).

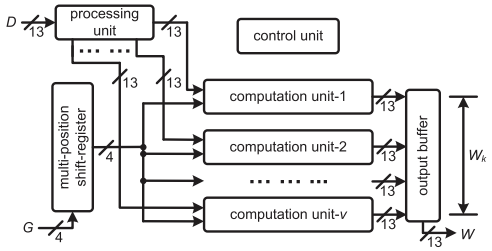

Fig. 1. Proposed hardware accelerator.

Notation 3: Define D_0, …, D_{u-1} as related D coefficients matching W_0, …, W_{u-1}, respectively. Define $S(D_{u-1})_i$ $(0 \leq i \leq N-1)$ as related coefficients for each G within W_{u-1}, e.g., we have $S(D_{u-1})_0 = \{d_0, \ldots, d_{N-v+1}\}$ and $S(D_{u-1})_1 = \{d_1, \ldots, d_{N-v+2}\}$ (similar to $D_0, \ldots, D_{u-2}$).

The proposed algorithm is thus summarized as:

Algorithm 1: Proposed Large Integer Polynomial Multiplication Algorithm.

Input : G and D are integer polynomials. // see Notation 1 for the actual bit-width;
Output: $W = GD \bmod (x^N + 1)$;

Initialization step
1 Make ready the inputs G and D;
2 $\overline{w} = 0$;
Main step
3 **for** $k = u - 1$ *to* 0 **do**
4 **for** $j = 0$ *to* $N - 1$ **do**
5 **for** $i = v - 1$ *to* 0 **do**
6 $\overline{w_{i+kv}} = \overline{w_{i+kv}} + G_{i,j}^k S(D_k)_j$;
7 **end**
8 **end**
9 $W_k = \{\overline{w_{kv+v-1}}, \overline{w_{kv+v-2}}, \cdots, \overline{w_{kv}}\}$;
10 **end**
Final step
11 Obtain all the coefficients of output W from W_k;

Details of Algorithm 1. (i) operations in Line 5 of Algorithm 1 are executed in parallel, i.e., all v number of w_i in each W_k are processed at the same time; (ii) Line 6 executes the N cycles of accumulations, i.e., point-wise multiplications of $G_{i,0}^k S(D_k)_0$, $G_{i,1}^k S(D_k)_1$, …, $G_{i,N-1}^k S(D_k)_{N-1}$ are accumulated to produce one W_k (v output results at the same time, Line 9); (iii) the above operations are repeatedly executed for u times to deliver all W_k (Line 3 and Line 11).

IV. Proposed Hardware Accelerator

The overall layout of the LPMA is shown in Fig. 1. *Note that* we have followed [3], [5] that the coefficients of G are represented in the sign magnitude format ([−5, 5]) while the coefficients of D are denoted as two's complement form.

Multi-Position Shift-Register for G: This multi-position shift-register is designed to to have multiple functions: (i) loading all the coefficients of G into the registers in a serial format; (ii)

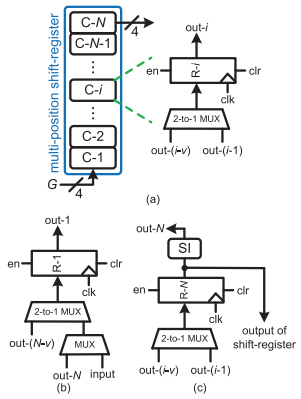

Fig. 2. (a) The novel multi-position shift-register; (b) the C-1; (c) the C-N. SI: sign inverter. R-i refers to the register.

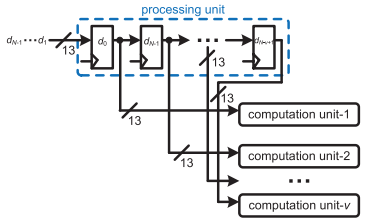

Fig. 3. Processing unit (values in registers are initial loaded).

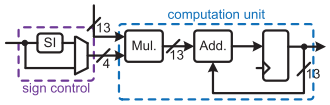

Fig. 4. Computation unit (Mul.: multiplier; Add.: adder).

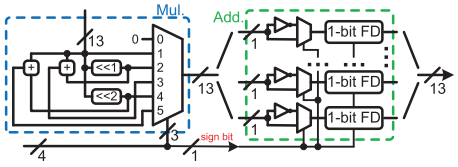

Fig. 5. Multiplier and adder (FD: full adder).

circularly shifting the positions of all coefficients by v positions, after all the coefficients of G are loaded, to obtain G^{k-1} from G^k ($k = u-1$ to 1) once per every N cycles including the related signs; (iii) circularly shifting the coefficients of G^k that these N coefficients can be delivered out once per cycle. As shown in Fig. 2, the proposed multi-position shift-register contains N cells, i.e., "C-1" to "C-N". The regular cell ("C-i" in Fig. 2(a)), contains one 4-bit register and one 4-bit 2-to-1 MUX. Define that the output of the register as "out-i", then the two inputs of the MUX are "out-$(i-v)$" and "out-$(i-1)$", respectively, to execute the mentioned two types of shifting, namely the 1-position based shifting (or with circular) and the v-position circular-shifting. Apart from that regular setup, there is also a need of specific designing for those cells in the bottom and top positions, as shown in Fig. 2(b) and (c), respectively. Basically, there is a need of another 2-to-1 MUX in "C-1" such that the shift-register can work in a circular-shifting format and serially loading the input coefficients. While "C-N" needs a sign inverter (SI) to realize the sign inverting according to Algorithm 1. Note that because of the required v-position shifting, "C-1" to "C-v" have similar setup as Fig. 2(b), which applies to "C-$(N-1)$" to "C-$(N-v+1)$".

During the loading phase, with the function of the MUXes in respective cells, the input values are serially loaded in the related registers. During the accumulation phase, all the loaded coefficients in the registers are serially shifted to produce the required coefficients to the accumulation units. During the group switching phase, e.g., from W_k to W_{k-1}, all the coefficients in the shift-register are circularly shifted by v positions, i.e., obtaining G^{k-1} from G^k.

Processing Unit for D: The values of $S(D_k)_i$ for each W_k, connecting Algorithm 1, are exactly the same, and hence the needed values can be repeated per every N cycles. Besides that, we notice that the values of $S(D_k)_i$ and $S(D_k)_{i+1}$, for a specific W_k, are sequentially shifted by one position with a new value being added to $S(D_k)_{i+1}$. We can thus use these two properties to design the *processing unit for* D, as shown in Fig. 3. It is seen that in total v number of registers are involved within the processing unit and the initially loaded values are d_0, d_{N-1}, …, d_{N-v-1}, respectively. Then, in the following $N-1$ cycles, values of $d_1, \ldots, d_{N-1}$ are fed to the first register from left such that the outputs of these registers produce the exact $S(D_k)_i$. This process repeats u times until all W_k are produced.

Computation Unit: As shown in Fig. 4, two inputs are multiplied and delivered to an accumulator; while the accumulator functions to accumulate all the obtained multiplication results (N cycles) and then output the result according to Line 9 of Algorithm 1. Note that the signs for values of $G^k_{j,i}$ within each W_k are regularly switching from $i = 0$ to $N-1$. For instance, for W_{u-1}, the values of $G^{u-1}_{j,0}$ within each w_j have $v-1$ negative signs, which becomes $v-2$ negative signs for $G^{u-1}_{j,1}$ (similar to others). We have thus used an v-length shift-register (1-bit) loaded with all '1' but fed with '0' and the output of all v registers are used as control signals attaching to the corresponding MUXes in the sign control cells of these v computation units, respectively. A MUX-based method is used to realize the multiplier function (Fig. 5), following the strategy of [5]. Since the 4-bit input lies in the range of $[-5, 5]$, we can pre-compute the multiplication results and attach them to the MUX. Considering that the 4-bit input is represented in the sign magnitude format, we just need to cover the range of [0, 5] and the rest can be covered by the sign bit connecting with the 2-to-1 MUXes in the adder. In this way, all the multiplication and addition results are correctly obtained.

Output Buffer: A buffer is used to deliver all parallel v outputs in serial, i.e., insert a MUX & register-based buffer, where one input of the MUX is connected from the output of register from the corresponding computation unit.

Control Unit: The control unit is based a finite state machine (FSM) with five stages, i.e., "reset", "load", "computation", "switch", and "done". The control unit enters the "reset" stage after receiving a reset signal. Then, all the registers in the shift-register and processing unit are enabled to load the corresponding coefficients in the "load" stage. After that, during the "computation" stage, all the point-wise multipliers are executed for accumulation. Meanwhile, the input shift-register is activated that the polynomial coefficients of G start to rotate. The "computation" stage lasts $N-1$ cycles, and then the control unit goes to the "switch" stage that the output buffer receives the accumulated results (v number) and then delivers them out serially. The control unit then jumps back to the "computation" stage to finish the remaining $(u-1)$ rounds (sign control shift-register also repeats $(u-1)$ rounds) and finally goes to the "done" stage.

TABLE I
IMPLEMENTATION RESULTS FOR THE PROPOSED LPMA

design	LUT	FF	Slice	Fmax	latency	power[1]
Artix-7 XC7A12TLCSG325-2L FPGA device						
$v = 2$	1,403	1,404	457	138	32,768	0.017
$v = 4$	1,608	1,486	544	127	16,384	0.020
$v = 8$	2,032	1,648	678	123	8,192	0.050
$v = 16$	2,912	1,979	949	120	4,096	0.042
$v = 32$	4,191	2,572	1,315	104	2,048	0.052
$v = 64$	7,174	3,824	2,103	98	1,024	0.089

Unit for Fmax: MHz. [1]: Dynamic power consumption (W).

TABLE II
COMPARISON WITH THE EXISTING DESIGNS

design	LUT	FF	DSP	ELUT[1]	Fmax	latency	ADP[2]
Artix-7 XC7A12TLCSG325-2L FPGA device							
[3]	2,927	1,279	38	22,383	125	8,176	1,464,027
[5]	541	301	0	821*	100	19,471	159,857
[8]	561	302	2	2,145*	130	16,384	270,336
$v = 4$	1,608	1,486	0	1,608	127	16,384	207,445
$v = 8$	2,032	1,648	0	2,032	123	8,192	135,334
$v = 16$	2,912	1,979	0	2,912	120	4,096	99,396
$v = 32$	4,191	2,572	0	4,191	104	2,048	82,530
$v = 64$	7,174	3,824	0	7,174	98	1,024	74,961

[1]: ELUT (equivalent LUT, reported LUTs added with the transferred equivalent LUTs from DSPs and BRAMs). As [3], [5] did not report the slice usage, we use LUT to calculate ELUT. 1 BRAM(8k) equals 70 slices, 1 DSP equals 128 slices, and 1 slice contains 4 LUTs (from [11]).
[2]: Area-delay product=#ELUT×(latency/Fmax). Unit for Fmax: MHz.
∗: [5] needs at least one BRAM; while [8] needs two BRAMs.

V. IMPLEMENTATION AND COMPARISON

The area-time complexities of LPMA are briefly stated as follows. The multi-position shift-register for G has $(N + v)$ 4-bit 2-to-1 MUXes, N 4-bit registers, and v 1-bit inverters. The processing unit for D contains v 13-bit registers. Each computation unit has one multiplier, one adder, and one register (all 13-bit). Finally, an output buffer of size-v with 13-bit is needed. The computation time is uN cycles.

Implementation: The experimental setup is as follows.

- The proposed LPMA is coded by VHDL with function validated on ModelSim (source code will be released upon the paper's acceptance). The implementation results are obtained by AMD-Xilinx Vivado 2019.2 on Artix-7 XC7A12TLCSG325-2 L (follow [5]).
- The design is implemented with $v = 2$, $v = 4$, $v = 8$, $v = 16$, $v = 32$, and $v = 64$ ($N = 256$). We have used $[-5, 5]$ for G (covers all three security ranks of Saber).

The data in Table I serves 3 purposes: (i) the proposed accelerator offers flexibility in processing speed; (ii) the proposed LPMA maintains efficiency in implementation though its area-complexity increases with v while the maximum frequency slightly decreases; (iii) if the application environment allows, we would recommend using a slightly larger v to obtain better overall area-time efficiency.

Comparison: The comparison is listed in Table II with recent lightweight designs of [3], [5], [8]. Note that these designs did not report the slice and power, We thus calculated the equivalent LUT (ELUT, reported LUTs added with the transferred equivalent LUTs from DSPs and BRAMs, where 1 BRAM(8 k) equals 70 slices, 1 DSP equals 128 slices, and 1 slice contains 4 LUTs [11]). Meanwhile, the area-delay product (ADP) is obtained from ADP=#ELUT×(latency/Fmax).

As shown in Table II, the proposed accelerator involves significant efficiency over the existing ones. For example, the proposed LPMA of $v = 32$ has at least 48.37% less ADP than the existing designs (53.11% less when $v = 64$). Meanwhile, LPMA also offers better choices in processing flexibility than [3], [5], [8]. Lastly, the proposed LPMA covers all security ranks of Saber, i.e., more complete than the existing designs.

Discussion: While this paper aims to design lightweight architecture, we do not compare it with the existing high-speed designs (e.g., [5], [6], [7], [9], [10]) as different types of structures have different priorities in architectural setup. Nevertheless, the proposed algorithm and architecture can be extended to other NIST PQC such as Falcon [15] (where the PQC scheme is not bound with fast algorithms) to obtain similar efficiency under lightweight applications.

VI. CONCLUSION

In this letter, a novel hardware implementation of large integer polynomial multiplication is proposed. We first derive the proposed algorithm for lightweight implementation of polynomial multiplication. Then, we have presented the proposed accelerator in a detailed format. Finally, implementation and comparison have been carried out to demonstrate the effectiveness of the proposed design strategy.

REFERENCES

[1] J.-P. D'Anvers et al., "SABER: Mod-LWR based KEM (round 3)," 2020. [Online]. Available: https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[2] J. Xie, K. Basu, K. Gaj, and U. Guin, "Special session: The recent advance in hardware implementation of post-quantum cryptography," in *Proc. IEEE 38th VLSI Test Symp.*, 2020, pp. 1–10.

[3] J. B. Mera, F. Turan, A. Karmakar, S. Sinha Roy, and I. Verbauwhede, "Compact domain-specific co-processor for accelerating module lattice-based KEM," in *Proc. IEEE/ACM 57th Des. Autom. Conf.*, 2020, pp. 1–6.

[4] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," in *Proc. Int. Conf. Field-Programmable Technol.*, 2019, pp. 206–214.

[5] A. Basso and S. S. Roy, "Optimized polynomial multiplier architectures for post-quantum KEM saber," in *Proc. IEEE/ACM 58th Des. Autom. Conf.*, 2021, pp. 1285–1290.

[6] J. Xie, P. He, and C.-Y. Lee, "CROP: FPGA implementation of high-performance polynomial multiplication in saber KEM based on novel cyclic-row oriented processing strategy," in *Proc. IEEE 39th Int. Conf. Comput. Des.*, 2021, pp. 130–137.

[7] W. Tan et al., "Low-latency VLSI architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography," 2021, *arXiv:2110.12127*.

[8] Y. Zhang, Y. Cui, Z. Ni, D. -E. -S. Kundi, D. Liu, and W. Liu, "A lightweight and efficient schoolbook polynomial multiplier for saber," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2022, pp. 2251–2255.

[9] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 4, pp. 443–466, 2020.

[10] 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.

[11] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2459–2463, Oct. 2019.

[12] D. D. Chen et al., "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.

[13] B. Lucas et al., "Lightweight hardware implementation of binary ring-LWE PQC accelerator," *IEEE Comput. Architecture Lett.*, vol. 21, no. 1, pp. 17–20, Jan.–Jun. 2022.

[14] S. Bian, M. Hiromoto, and T. Sato, "Filianore: Better multiplier architectures for LWE-based post-quantum key exchange," in *Proc. IEEE/ACM 56th Des. Autom. Conf.*, 2019, pp. 1–6.

[15] P.-A. Fouque et al., "FALCON," 2020. [Online]. Available: https://falcon-sign.info/