Efficient Hardware Implementation of Finite Field Arithmetic $AB + C$ for Binary Ring-LWE based Post-Quantum Cryptography

Jiafeng Xie, *Senior Member, IEEE,* Pengzhou He, Xiaofang Wang, and José L. Imaña

Abstract—Post-quantum cryptography (PQC) has gained significant attention from the community recently as it is proven that the existing public-key cryptosystems are vulnerable to the attacks launched from the well-developed quantum computers. The finite field arithmetic $AB + C$, where A and C are integer polynomials and B is a binary polynomial, is the key component for the binary Ring-learning-with-errors (BRLWE)-based encryption scheme (a low-complexity PQC suitable for emerging lightweight applications). In this paper, we propose a novel hardware implementation of the finite field arithmetic $AB + C$ through three stages of interdependent efforts: (i) a rigorous mathematical formulation process is presented first; (ii) an efficient hardware architecture is then presented with detailed description; (iii) a thorough implementation has also been given along with the comparison. Overall, (i) the proposed basic structure ($u = 1$) outperforms the existing designs, e.g., it involves 55.9% less area-delay product (ADP) than [13] for $n = 512$; (ii) the proposed design also offers very efficient performance in time-complexity and can be used in many future applications.

Index Terms—Binary Ring-Learning-with-Errors, finite field arithmetic, FPGA platform, hardware design, post-quantum cryptography.

1 Introduction

Finite field arithmetic is widely used in various types of cryptosystems [1]-[5]. Recently, along with the rapid advancement in quantum technology, the attention on the post-quantum cryptography (PQC) has reached an all-time high [6]. As finite field arithmetic still constitutes the main component of the PQC, the research on efficient implementation of the related arithmetic components for specific cryptosystem is highly demanded.

Overall, the lattice-based cryptography is widely recognized as one of the most promising PQC candidates due to its small implementation complexity and strong security proof [7]. The lattice-based cryptography is usually built on the learning-with-errors (LWE) or Ring-LWE (a variant of LWE) problem [8], [9], where the Ring-LWE-based PQC uses polynomial multiplication over ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle f(x)\rangle$ ($f(x) = x^n + 1$) and hence is more widely used [9]. Recently, a new variant of Ring-LWE, binary Ring-LWE (BRLWE)-based encryption scheme, has been proposed specifically for various emerging lightweight applications [10], [11], where the binary errors are used for low-complexity computation.

The major arithmetic component of the BRLWE-based encryption scheme is the finite field operation $AB + C$, i.e., one polynomial multiplication followed by one polynomial addition, where A and C are polynomials with integer coefficients and B is a binary polynomial. So far, there is limited research in this area: the very recent efforts [12], [13] are focusing on the field-programmable gate array (FPGA) based platform (the main efforts in the field). Noticing the recent report of [1] has proposed a serial-in parallel-out design format for the finite field multiplication over $GF(2^m)$, we follow this trend and have proposed efficient hardware implementation of the $AB + C$ in BRLWE-based PQC, as summarized below (main contributions):

- We have formulated the main finite field arithmetic into the desired form suitable for deriving the proposed hardware architecture.
- We have then presented the desired hardware structure with a detailed internal structural description through a series of algorithm-architecture co-implementation techniques.
- We have demonstrated the efficiency of the proposed design, i.e., it has better area-time complexities than the existing ones and offers small time complexity.

In particular, the proposed hardware architecture has three main unique features: (i) has smaller logic usage with simple control signals, which is beneficial to the low critical-path operation; (ii) offers choices on processing speed and hence is more flexible than the existing structures; (iii) fits well for the practical operations in both encryption and decryption phases and thus is more complete than the existing designs.

The rest of the paper is organized as follows. The preliminary is introduced in Section 2. The algorithmic process is formulated in Section 3. The hardware structure is presented in Section 4. Complexity and comparison are presented in Section 5. The conclusion is given in Section 6.

2 Preliminary

BRLWE-based Encryption Scheme. BRLWE is a new variant of Ring-LWE, which uses binary errors to replace the

- *Corresponding Author: Jiafeng Xie (e-mail: jiafeng.xie@villanova.edu).*
- *J. Xie, P. He, and X. Wang are with the Department of Electrical and Computer Engineering, Villanova University, Villanova, PA, 19085 USA (e-mail: jiafeng.xie@villanova.edu).*
- *J.L. Imaña is with the Department of Computer Architecture and Systems Engineering, Faculty of Physics, Complutense University, Madrid 28040, Spain. (e-mail: jluimana@ucm.es).*

Manuscript received October 21, 2020, revised April 22, 2020.

TABLE 1: Details of The BRLWE-based Encryption Scheme

scheme stages	operations
key generation	a_p: public parameter known by Alice and Bob r_1, r_2: binary polynomials (r_2 is the secret key) Alice: $p = r_1 - a_p \cdot r_2 \rightarrow$ Bob (public key)
encryption	e_1, e_2, e_3: three binary errors m: message; $\widetilde{m}$: encoded message; Bob: $c_{t1} = a_p \cdot e_1 + e_2 \rightarrow$ Alice Bob: $c_{t2} = p \cdot e_1 + e_3 + \widetilde{m} \rightarrow$ Alice
decryption	Alice: $m = \text{decode}(c_{t1} \cdot r_2 + c_{t2})$

TABLE 2: Arithmetic Operations of The BRLWE-based PQC

scheme stages	arithmetic operations
key generation	$\text{P}_{\otimes}$ and $\text{P}_{\oplus}$: produce p (public key)
encryption	$\text{P}_{\otimes}$ and $\text{P}_{\oplus}$: produce c_{t1} $\text{P}_{\otimes}$ and $\text{P}_{\oplus}$ and $\text{P}_{\oplus}$: produce c_{t2}
decryption	$\text{P}_{\otimes}$ and $\text{P}_{\oplus}$: produce output

$\text{P}_{\otimes}$: polynomial multiplication; $\text{P}_{\oplus}$: polynomial addition.

Gaussian distributed errors [10]. The BRLWE-based encryption scheme is firstly formulated and implemented in [10] with detailed and rigorous security analysis. An efficient hardware BRLWE-based PQC is then presented in [12]. Very recently, a pair of low-complexity and high-speed BRLWE-based cryptoprocessors are given in [13].

The BRLWE-based encryption scheme is built on the operations over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ (Table 1):

Key generation. Firstly, a_p is a integer polynomial, known as the public parameter, for both Alice and Bob. Selecting two binary polynomials r_1 and r_2, where r_2 is the secret key, Alice then calculates the public key $p = r_1 - a_p \cdot r_2$ and sends it to Bob (r_1 is discarded after this). The bit-lengths of the public and secret keys are $n\log_2 q$ and n, respectively.

Encryption. A n-bit message m is coded to $\widetilde{m}$ following that each coefficient of m (binary polynomial) is multiplied with $q/2$. After that, three binary errors e_1, e_2, and e_3 are used to generate the ciphertext c_{t1} and c_{t2} to be sent to Alice. The length of the ciphertext is $2n\log_2 q$.

Decryption. Alice uses the secret key and the ciphertext c_{t1} and c_{t2} to obtain the original message through $c_{t1}r_2 + c_{t2}$ as well as the threshold decoder function (decode(·)), which will return a binary value of '1' if the coefficient lies in the range of $(q/4, 3q/4)$, else it will yield '0'.

The polynomial multiplication (Table 1), needs to deal with the degree overflow as determined by $f(x) = x^n + 1$. The inverted range of $(-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor - 1)$ is then proposed in [13] for the integer coefficients such that there is no reduction involved with the modular addition/subtraction under the two's complement representation. All the major operations of Table 1 are almost the same except minor changes on the encode and final decode functions [13]. Note that in this paper, we also employ this strategy.

3 Mathematical Formulation

Main Arithmetic Operation. Without loss of generality, one can conclude that (from Table 1) the major operation involved within the BRLWE-based encryption scheme is the polynomial multiplication (one integer polynomial and the other one with binary coefficients) followed by a polynomial addition, as depicted in Table 2.

Definition 1. Let A, C, and D be polynomials over R_q and B is a binary polynomial as: $A = \sum_{i=0}^{n-1} a_i x^i$, $B = \sum_{i=0}^{n-1} b_i x^i$, $C = \sum_{i=0}^{n-1} c_i x^i$, and $D = AB \bmod f(x) = \sum_{i=0}^{n-1} d_i x^i$ (a_i, c_i, and d_i are integers in $\mathbb{Z}_q$, $b_i \in \{0, 1\}$, and $f(x) = x^n + 1$).

Meanwhile, define $W = AB + C = \sum_{i=0}^{n-1} w_i x^i$, where w_i is an integer. Note that n refers to the security size and the integer coefficients in the polynomials are $\log_2 q$-bit.

The common operation of the three stages of the BRLWE-based encryption scheme according to Table 2 can thus be

$$W = AB \bmod f(x) + C = D + C. \tag{1}$$

Algorithmic Derivation. Equation (1) can be rewritten as

$$\begin{aligned} W =& A \times (b_0 + \cdots + b_{n-1}x^{n-1}) \bmod f(x) + C \\ =& (a_0 + \cdots + a_{n-1}x^{n-1})b_0 \bmod f(x) + \cdots \\ &+ (a_0 x^{n-1} + \cdots + a_{n-1}x^{2n-2})b_{n-1} \bmod f(x) + C, \end{aligned} \tag{2}$$

where we can use $x^n \equiv -1$ to remove the mod operation.

Thus, (2) can be expressed in the form of

$$W = A_0 + A_1 + \cdots + A_{n-1} + C = \sum_{j=0}^{n-1} A_j + C, \tag{3}$$

where $A_0 = a_0 b_0 + a_1 b_0 x + \cdots + a_{n-1} b_0 x^{n-1}$, $A_1 = a_0 b_1 x + \cdots - a_{n-1} b_1, \cdots$, $A_{n-1} = a_0 b_{n-1} x^{n-1} - \cdots - a_{n-1} b_{n-1} x^{n-2}$. And the involved operations are just point-wise additions. Besides, for $n = uv$ (u and v are integers and u is typically selected as a small integer such as 1 or 2), one can then have

$$W = \sum_{l=0}^{u-1} \sum_{k=0}^{v-1} A_{lv+k} + C, \tag{4}$$

where u groups of A_{lv+k} can be processed in parallel to speed up the computation process.

The above whole process can thus be written as shown in Algorithm 1.

Algorithm 1: Algorithmic process for $AB + C$.

Input : A and C are polynomials with integer coefficients and B is a binary polynomial.
Output: $W = AB \bmod f(x) + C = D + C$.

Initialization step
$T = 0$;
Main step
$T = C$ // serially load in C;
for $l = 0$ *to* $u - 1$ **do**
  for $k = 0$ *to* $v - 1$ **do**
    $T = T + A_{lv+k}$. // following (2)-(4)
  end
end
Final step
$W = T$ and serially deliver the output W;

Note that we also use XORs to the w_i for the decryption stage of the BRLWE-based PQC, following [12], [13].

4 Proposed Structure for $AB + C$

Strategy. Let us first consider Algorithm 1 and all the coefficients of each A_j ($0 \le j \le n - 1$), as shown in Table 3.

TABLE 3: Coefficients of A_j $(0 \leq j \leq n-1)$

polynomials	coefficients						
	x^0	x^1	x^2	x^3	$\cdots$	x^{n-2}	x^{n-1}
A_0	a_0b_0	a_1b_0	a_2b_0	a_3b_0	$\cdots$	$a_{n-2}b_0$	$a_{n-1}b_0$
A_1	$-a_{n-1}b_1$	a_0b_1	a_1b_1	a_2b_1	$\cdots$	$a_{n-3}b_1$	$a_{n-2}b_1$
A_2	$-a_{n-2}b_2$	$-a_{n-1}b_2$	a_0b_2	a_1b_2	$\cdots$	$a_{n-4}b_2$	$a_{n-3}b_2$
A_3	$-a_{n-3}b_3$	$-a_{n-2}b_3$	$-a_{n-1}b_3$	a_0b_3	$\cdots$	$a_{n-5}b_3$	$a_{n-4}b_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$A_{n/2}$	$-a_{n/2}b_{n/2}$	$-a_{n/2+1}b_{n/2}$	$-a_{n/2+2}b_{n/2}$	$-a_{n/2+3}b_{n/2}$	$\cdots$	$a_{n/2-2}b_{n/2}$	$a_{n/2-1}b_{n/2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
A_{n-v}	$-a_vb_{n-v}$	$-a_{v+1}b_{n-v}$	$-a_{v+2}b_{n-v}$	$-a_{v+3}b_{n-v}$	$\cdots$	$a_{v-2}b_{n-v}$	$a_{v-1}b_{n-v}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
A_{n-2}	$-a_2b_{n-2}$	$-a_3b_{n-2}$	$-a_4b_{n-2}$	$-a_5b_{n-2}$	$\cdots$	a_0b_{n-2}	a_1b_{n-2}
A_{n-1}	$-a_1b_{n-1}$	$-a_2b_{n-1}$	$-a_3b_{n-1}$	$-a_4b_{n-1}$	$\cdots$	$-a_{n-1}b_{n-1}$	a_0b_{n-1}

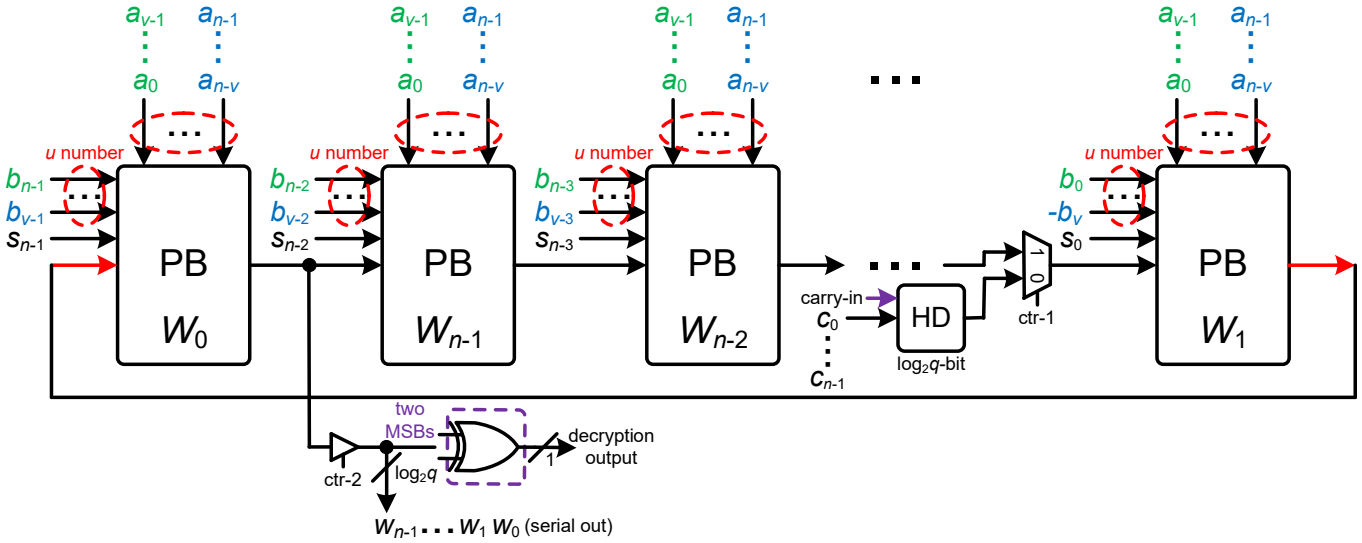

Fig. 1: Proposed structure for computing $AB + C$. PB: processing block. HD: half adder. MSB: most significant bit.

One can observe that the actual value of the corresponding coefficients of each A_j $(0 \leq j \leq n-1)$ can be obtained in an orderly format, e.g., each coefficient of A_0 can be obtained through the multiplication of every individual coefficient of A with the b_0 of B while each coefficient of A_1 can be obtained through the multiplication of circularly-shifted coefficients of A (i.e., a_{n-1} is inverted as $-a_{n-1}$) with the b_1 of B. The other A_j $(2 \leq j \leq n-1)$ follow the same rule.

Proposed strategy. Following the computation process of Algorithm 1 and the observation of Table 3, we can thus design a hardware structure for finite field arithmetic $AB + C$: (i) The coefficients of C can be serially loaded into the related storage places, respectively. (ii) All the coefficients of B, as well as the corresponding ones for A_{lv}, are stably located with one coefficient being inverted per every cycle and meanwhile multiplied with the serially delivered coefficient of A. (iii) The corresponding multiplication results are then circularly-shifted to the right position (one position-shifting per cycle, see Table 3) to be added to obtain the result of $C + \sum_{l=0}^{u-1} \sum_{k=0}^{v-1} A_{lv+k}$. There are u groups processed in parallel to speed up the whole process, based on Algorithm 1, which actually offers flexible speed choices. (iv) The output is delivered out in a serial format because of the large bit-width. Note that the corresponding coefficient of polynomial C has already been added to the related multiplication results in Step (iii) to yield the final result, including both encryption and decryption results.

Hardware Structure. The corresponding hardware structure is thus shown in Fig. 1. The proposed structure consists of mainly n processing blocks (PBs), where each PB has the same internal structure. According to Algorithm 1 and Table 3, each coefficient of polynomial C is firstly delivered to the related PBs, respectively, through the MUX attached to the far-right PB of the structure. Then, all the bits of polynomial A are decomposed into u number of groups (denoted by the red dotted circle), where each group has v coefficients, i.e., $\{a_0, a_1, \cdots, a_{v-1}\}, \{a_v, a_{v+1}, \cdots, a_{2v-1}\}, \cdots$, and $\{a_{n-v}, a_{n-v+1}, \cdots, a_{n-1}\}$, to be fed in a serial format to each PB (group-wise parallel). Accordingly, we have also prepared the u parallel groups of coefficients of polynomial B (some with a negative sign), based on the constitution of the coefficients of A_j (as seen from Table 3), to be fed to each PB, i.e., A_j is decomposed into u groups of $\{A_0, A_1, \cdots, A_{v-1}\}, \{A_v, \cdots, A_{2v-1}\}, \cdots$, and $\{A_{n-v}, A_{n-v+1}, \cdots, A_{n-1}\}$ based on Algorithm 1. For a clear presentation, we have used different color signals to denote the corresponding relationships, e.g., A_0 is related with $\{a_0b_0, a_1b_0, \cdots, a_{n-1}b_0\}$ (so do the following

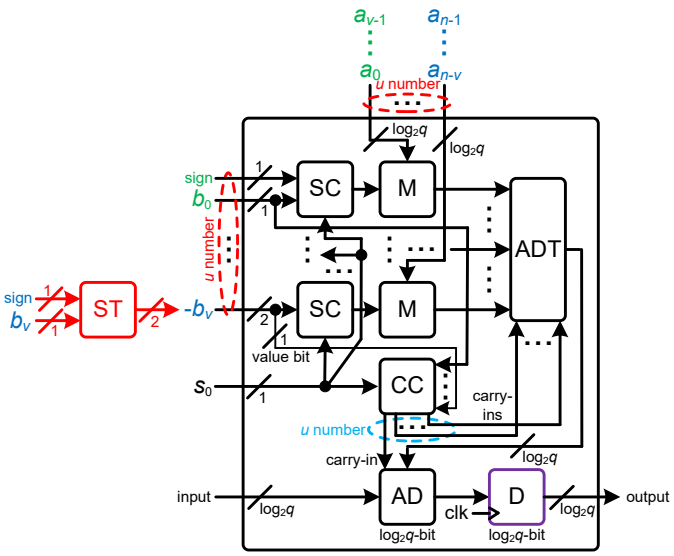

Fig. 2: The details of the PB, the far right one of Fig. 1.

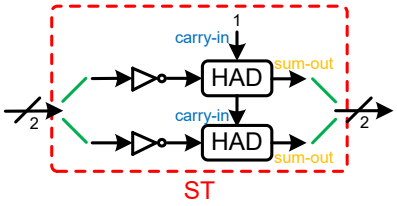

Fig. 3: The details of the ST cell, where the least significant bit's carry-in is set as '1'. HAD: 1-bit half-adder.

$A_1, \cdots, A_{v-1}$, as denoted by the green color), while A_{n-v} is involved with $\{-a_v b_{n-v}, -a_{v+1} b_{n-v}, \cdots, a_{v-1} b_{n-v}\}$ (the same to the following $A_{n-v+1}, \cdots, A_{n-1}$, as represented by the blue color). There are in total n number of control signals (s_i, $0 \leq i \leq n-1$) fed to each individual PB, respectively (one per PB, and the corresponding function is covered later). The output of the structure (namely the coefficients of polynomial W) becomes available after n cycles to be serially delivered out through the buffer attached to the output of the far right PB, after all the initial input signals are loaded into the PBs. The internal structure of the PB, as well as the related functions, are described as follows.

Internal structure of the PB. The internal structure of the PB (far-right one) is shown in Fig. 2. Based on Table 3 and Algorithm 1, u groups of coefficients from A and B, for A_{lv+k}, are fed to the PB, respectively, as shown in the dotted red circle area. Details of the internal cells are as follows.

- **ST cell**. Note that the negative value of a certain bit is obtained through the sign-transfer (ST) cell, e.g., b_v is fed to the ST cell to produce -b_v. The ST cell is designed to follow the two's complement representation, i.e., the sign bit and value bit of b_v are inverted and then go through the 1-bit half-adder (HAD), as shown in Fig. 3, where the least significant bit's carry-in is set as '1'.
- **SC and M cells**. There are in total u groups of paired signal-selection (SC) cells and multiplication (M) cells, as shown in Fig. 4. The SC cell has a 2-bit MUX connected with the input and is controlled by the control signal s_0. When the input signal needs to be inverted, the control signal switches to let the inverted input deliver to the M cell (under two's complement denotation, the extra carry-in of '1' is added in the following ADT cell). The value bit from the MUX is connected with the $(\log_2 q - 1)$ AND gates working in parallel, while the two sign bits are ANDed together. Thus, the SC and M cells work together to produce the multiplication results based on Table 3, e.g., the result is $a_0 b_0$ when $s_0 = 0$.
- **CC and ADT cells**. The M cells are then added together through the adder-tree (ADT) cell according to (4). The ADT cell has $(u-1)$ number of full-adders (ADs), where all the carry-ins are generated from the carry-conversion (CC) cell, as shown in Fig. 5. Due to the use of two's complement representation, we have to let the corresponding u value bits of B be multiplied (ANDed) with the control signal of s_0, respectively, and then deliver the u number of output bits (one bit to the AD below and $(u-1)$ bits to the ADT cell, as shown in Figs. 2 and 5). Such that when the control signal moves from '0' to '1' and the input bits from B are not '0', the carry-in of all these ADs turns into '1' to produce the correct output.
- **The remaining cells**. The remaining cells in the PB include one AD and one register cell (D cell, purple box). As shown in Fig. 2, the input from the left is added with the output of the ADT and then yielded out through the register cell to the right. Note that the input and right output signals of all the PBs are circularly connected, as indicated by the red line in Fig. 1) and the register cell in the PB is the storage place of the intermediate value. Thus, after the corresponding coefficient of the polynomial C (c_{n-1} in this case) is firstly loaded in the D cell, it will be added with the multiplication result in the far left PB to be stored in the D cell there. After the required number of cycles for calculation, the desired result W_1 is then available in the D cell (similar to other PBs of the structure of Fig. 1), which can be circularly delivered out through the buffer connected to the far left PB of Fig. 1. One extra XOR gate is needed for the two most significant bits (MSBs), as indicated in Fig. 2 (following [12], [13]), for the output of the decryption phase.

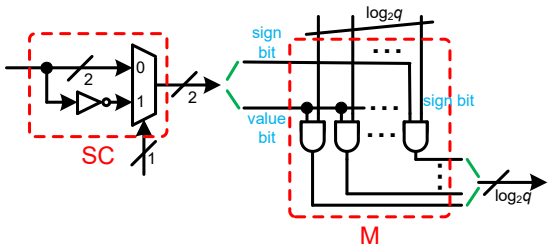

Fig. 4: The internal structure of the SC and M cells.

The shift-register for the control signals. The control signals $s_{n-1}, s_{n-2}, \cdots, s_0$ are generated by a shift-register as shown in Fig. 6, where it contains n 1-bit registers. For the first cycle, all the registers are initiated as '0's. Then in the following cycles, the input of the shift-register will be set as '1', which is according to the feature that there is one

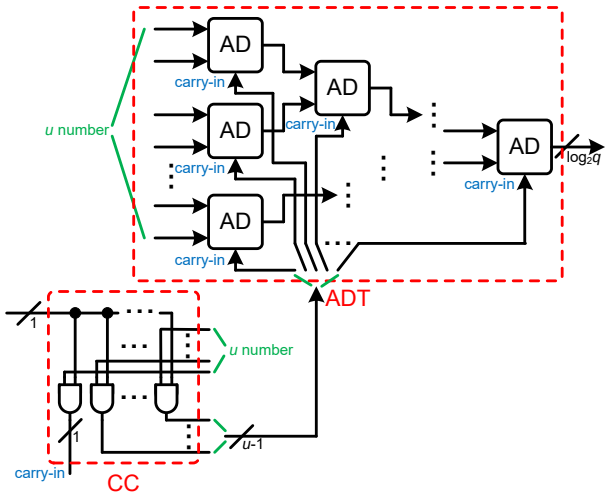

Fig. 5: The internal structure of the ADT cell (AD:full-adder).

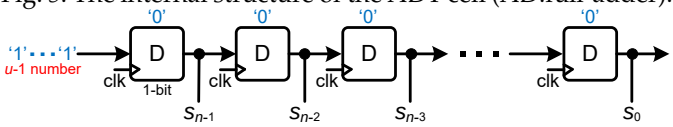

Fig. 6: The setup of the shift-register for control signals.

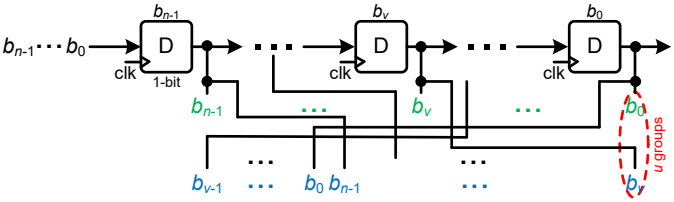

Fig. 7: The shift-register (for B) produces u parallel groups.

coefficient's (of B) sign change between A_j and A_{j+1} (see Table 3). The output of each register is connected with the corresponding control signals s_i ($0 \leq i \leq n-1$), respectively.

The shift-register for the input B. All the coefficients of the polynomial B are firstly serially loaded to the shift-register (n 1-bit registers), as shown in Fig. 7. Then, in the following cycles, the values of the corresponding registers will be shared accordingly to produce the u number parallel groups of n-bit outputs, respectively, following Fig. 2.

The overall operation of the structure. The operation of the structure can be illustrated by an example below.

Example. For $n = 4$ and $u = 2$, we can have (from Table 3)

$$\begin{aligned} w_0 &= a_0b_0 - a_3b_1 - a_2b_2 - a_1b_3 + c_0, \\ w_1 &= a_1b_0 + a_0b_1 - a_3b_2 - a_2b_3 + c_1, \\ w_2 &= a_2b_0 + a_1b_1 + a_0b_2 - a_3b_3 + c_2, \\ w_3 &= a_3b_0 + a_2b_1 + a_1b_2 + a_0b_3 + c_3, \end{aligned} \tag{5}$$

which exactly match the values shown in Table 4.

Note that the proposed structure fits for both encryption and decryption operations of the BRLWE-based PQC, i.e., the carry-in of the HD (connecting the far-right PB) in Fig. 1 can be connected with the related coefficient of the binary polynomial e_3 for the operation in the encryption phase.

Besides that, in the initial shift-register's loading time for operand B, the corresponding coefficients are serially loaded into the D cells within the related PBs, respectively, with the help of the MUX (ctr-1 is set as '0'). Then, during the actual computation time, the ctr-1 is switched to '1' for the sake of circular accumulation. Finally, when the desired result is available in the related D cells within the PBs, the control signal of the buffer (ctr-2 is set as '1') functions to deliver the output in a serial format.

TABLE 4: An example of $n = 4$ ($u = 2$ and $v = 2$)

C*	PB-1	PB-2	PB-3	PB-4
control signals fed to each PB				
1	$s_3 = 0$	$s_2 = 0$	$s_1 = 0$	$s_0 = 0$
2	$s_3 = 1$	$s_2 = 0$	$s_1 = 0$	$s_0 = 0$
values initiated within the D cell of each PB				
-	c_2	c_1	c_0	c_3
operations involved within each PB (values in the D cell)				
1	$a_0b_3 + a_2b_1$ $+c_3$	$a_0b_2 + a_2b_0$ $+c_2$	$a_0b_1 - a_2b_3$ $+c_1$	$a_0b_0 - a_2b_2$ $+c_0$
2	$-a_1b_3 - a_3b_1$ $+c_0+$ $a_0b_0 - a_2b_2$	$a_1b_2 + a_3b_0$ $+c_3+$ $a_0b_3 + a_2b_1$	$a_1b_1 - a_3b_3$ $+c_2+$ $a_0b_2 + a_2b_0$	$a_1b_0 - a_3b_2$ $+c_1+$ $a_0b_1 - a_2b_3$
values stored within the D cell of each PB				
-	w_0	w_3	w_2	w_1

C*: cycle. The PBs are ordered from left to right (follow Fig. 1).

Overall summary. The proposed structure offers flexible processing speeds: (i) with $u = 1$, it produces the output after n cycles of computation; (ii) with a slightly larger u, it offers a latency of n/u computational cycles.

5 Complexity and Comparison

The area-time complexities of the proposed structure along with the design of [13] are shown in Table 5 (the design of [13] has shown its superior performance than [12], here we just list [13]). One can see that the proposed design has smaller area usage than the existing one because of the use of smaller-size MUXes, similar to the time-complexity.

For further detailed comparison, we have obtained the FPGA implementation results (after place & route) along with the existing designs [12], [13], [14].

Experimental setup. The experiment is set as follows: (i) we noticed that the existing report of [13] does not cover the overall area usage and hence for a fair comparison, we have re-code the existing one of [13] (have verified its functionality through ModelSim) with VHDL and implemented them using Xilinx Vivado 2019.2 on the Virtex-7 (XC7V2000) and Kintex-7 (XC7K325) devices; (ii) following [12], [13], we have chosen $n = 256$, $n = 512$, $q = 128$, $u = 1$, and $u = 2$ (according the recent analysis of [11], $n = 512$ and $n = 256$ can provide equivalent 190/140 and 84/73 bits of class and quantum securities, respectively); (iii) we have used the same type of AD for all the coded designs, i.e., 8-bit ripple carry adder; (iv) we have used the same type of shift-register for the binary polynomial, as shown in Fig. 7; (v) for a successful implementation on the targeted FPGA device, the input integer polynomial in the existing design of [13] is delivered to the shift-register first and then all the corresponding values are fed to the hardware structure in a parallel format; (vi) the actual area usage of the design of [13] listed in Table 6 does not include the input integer polynomial related shift-register, but the power is reported based on the whole structure; (vii) a state-of-the-art design of regular Ring-LWE based structure [14] is also listed.

Comparison. The area-time complexities of the proposed and the existing designs, namely #LUT, #FF, #slice,

TABLE 5: Main Complexities of Various Designs for $AB + C$ for BRLWE-based Encryption Scheme ($n = uv$)

design	AND	adder ($\log_2 q$-bit)	register (1-bit)	MUX ($\log_2 q$-bit)	critical-path	latency
[13]*	$n\log_2 q$	n	$n + n\log_2 q$	$n + 1$	$\geq T_A + T_{MUX} + T_{AD}$	$n + 1$
Fig. 1 ($u = 1$)**	$n\log_2 q$	n	$n + n\log_2 q$	1	$\simeq T_A + T_{MUX_2} + T_{AD}$	n
Fig. 1 ($u = 2$)**	$2n\log_2 q$	$2n$	$n + n\log_2 q$	1	$\simeq T_A + T_{MUX_2} + 2T_{AD}$	$n/2$

T_A: delay of the AND gate. T_{MUX}: delay of the $\log_2 q$-bit MUX. T_{AD}: delay of the $\log_2 q$-bit adder. T_{MUX_2}: delay of the 2-bit MUX.
*: The area-time complexities are only for the decryption phase (Fig. 4 of [13]). The structure for the encryption phase is not clearly provided.
**: The area-time complexities listed here are for the proposed structure that can operate in both the encryption and decryption phases. There are also n and $2n$ 2-bit MUXes for the proposed structure of $u = 1$ and $u = 2$, respectively.

TABLE 6: Comparison of the Complexities (FPGA platform)

design	n	phase	device	LUT	FF	Slice	Fmax	latency[1]	delay	ADP[2]	power	EPC[3]	Thr.
[12]	256	Dec	Spartan-6	6,728	6,813	1,874	101	262	2,594	4,861	-	-	99
[13]*	256	Dec	Virtex-7	5,153	2,151	1,701	261	257	985	1,675	921	3.529	260
Fig. 1[4]	256	Enc/Dec	Virtex-7	3,600	2,568	1,146	415	512/256	1,234/617	707	233	0.561	415
Fig. 1[5]	256	Enc/Dec	Virtex-7	6,237	2,568	1,881	314	256/128	815/408	767	336	0.535	627
[13]*	512	Dec	Virtex-7	10,285	4,249	3,289	263	513	1,951	6,417	1,871	7.114	262
Fig. 1[4]	512	Enc/Dec	Virtex-7	7,184	5,128	2,208	399	1,024/512	2,566/1,283	2,833	456	1.143	399
Fig. 1[5]	512	Enc/Dec	Virtex-7	13,100	5,133	3,796	286	512/256	1,790/895	3,397	670	1.171	572
[14]	256	Enc/Dec	Kintex-7	1,381	1,179	479	275	35,478/17,732	129k/64k	30,656	-	-	4
Fig. 1[4]	256	Enc/Dec	Kintex-7	3,600	2,568	1,134	394	512/256	1,299/650	737	237	0.602	394
Fig. 1[5]	256	Enc/Dec	Kintex-7	6,244	2,568	1,932	318	256/128	805/403	779	348	0.547	635

We have re-implemented Fig. 4 of [13] and listed results here. Enc: encryption; Dec: decryption. Thr: Throughput (Dec, coefficient/second$\times 10^6$).
*: The area usage for [13] does not include the shift-register for integer polynomial, but the power refers to the whole structure.
Unit for Fmax: MHz. Unit for delay: ns. Unit for power (dynamic): mW. Delay=critical-path$\times$latency. [1]: Shift-register's loading is not included.
[2]: ADP=#Slice$\times$delay (Dec) $\times 10^3$. [3]: EPC: energy per computation=power/(Fmax $\times$ #output coefficient per cycle (Dec)). [4]: $u = 1$. [5]: $u = 2$.

maximum frequency, latency, delay (critical-path$\times$latency cycles), area-delay product (ADP), power, energy per computation (EPC), and throughput, are listed in Table 6.

It is clear that the proposed design significantly outperforms the existing ones in various metrics aspects, e.g., the proposed design of $u = 1$ (the same latency style as [13]) has 57.8% and 55.9% less ADP than [13] for $n = 256$ and $n = 512$, respectively. Meanwhile, the proposed architecture involves flexible and efficient time-complexity, which is very suitable for high-speed applications (such as a server) with abundant resources, i.e., the delay time drops significantly when u turns from 1 to 2 (at the cost of reasonable extra area occupation). Lastly, one has to mention that the proposed structure takes care of the input/output in a practical way, while the existing design of [13] has to employ an extra shift-register for integer polynomial loading.

Discussion. While the main focus of this paper is to obtain an efficient implementation of $AB + C$ for the BRLWE-based PQC, we still believe the strategy proposed in [12], [13], [15] against side-channel attack is also applicable to the proposed design, which is part of our future work.

6 Conclusion

This paper proposes a novel implementation of finite field arithmetic $AB + C$ for the BRLWE-based encryption scheme. We firstly derive the proposed algorithmic operation through mathematical formulation. Then, the proposed hardware structure is presented with detailed descriptions. Finally, the comparison has shown the superior performance of the proposed design. The proposed structure is highly efficient and can be extended for future applications.

7 Acknowledgement

The work of J. Xie was supported by the NSF Award No. 2020625. The work of J.L. Imaña was supported by the Spanish MINECO and CM under grants S2018/TCS-4423 and RTI2018-093684-B-I00.

References

[1] A. Namin, et al., "High-speed architectures for multiplication using reordered normal basis," *IEEE TC* vol. 61, no. 2, pp. 164-172, 2012.
[2] J. Xie et al., "Novel bit-parallel and digit-serial systolic finite field multipliers over $GF(2^m)$ based on reordered normal basis," *IEEE Trans. VLSI Systems*, vol. 27, no. 9, pp. 2119-2130, 2019.
[3] J. Xie et al., "Low register-complexity systolic digit-serial multiplier over $GF(2^m)$ based on trinomials," *IEEE Trans. Multiscale Computing Systems*, vol. 4, no. 4, pp. 773-783, 2018.
[4] P. Meher and X. Lou, "Low-latency, low-area, and scalable systolic-like modular multipliers for $GF(2^m)$ based on irreducible all-one polynomials," *IEEE TCAS-I* vol. 64, no. 2, pp. 399-408, 2017.
[5] J. L. Imaña, "LFSR-based bit-serial $GF(2^m)$ multipliers using irreducible trinomials" *IEEE Trans. Computers*, 2020 (early access).
[6] W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. *Symp. Founda. of Computer Science*, pp. 124-134, 1994.
[7] D. Micciancio. Lattice-based cryptography. *Encyclopedia of Cryptography & Security*, 2011.
[8] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, 34, 2009.
[9] V. Lyubashevsky et al., "On ideal lattices and learning with errors over rings," *Int. Conf. Theory & Appl. of Crypto. Tech.*, pp. 1-23, 2010.
[10] J. Buchmann et al., "High-performance and lightweight lattice-based public-key encryption," *ACM IoTPTS*, pp. 1-8, 2016.
[11] J. Buchmann et al., "On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack," *Int. Conf. on Cryptology in Africa*, pp. 24-43, 2016.
[12] A. Aysu et al., "Binary Ring-LWE hardware with power side-channel countermeasures, *DATE*, pp. 1253-1258, 2018.
[13] S. Ebrahimi et al., "Post-quantum cryptoprocessors optimized for edge and resource-constrained devices in IoT," *IEEE IoT Journal*, vol. 6, no. 3, pp. 5500-5507, 2019.
[14] Y. Zhang et al., "An efficient and parallel R-LWE cryptoprocessors," *IEEE TCAS-II*, vol. 67, no. 5, pp. 886-890, 2020.
[15] T. Schneider et al., "Part I Towards combined hardware countermeasures against side-channel and fault-injection attacks," *Proc. Annu. Cryptol. Conf.*, pp. 302-332, 2016.