High-Speed VLSI Architectures for Modular Polynomial Multiplication via Fast Filtering and Applications to Lattice-Based Cryptography

Weihang Tan, *Graduate Student Member, IEEE*, Antian Wang, *Graduate Student Member, IEEE*, Xinmiao Zhang, *Senior Member, IEEE*, Yingjie Lao, *Senior Member, IEEE*, and Keshab K. Parhi, *Fellow, IEEE*

***Abstract*—This paper presents a low-latency hardware accelerator for modular polynomial multiplication for lattice-based post-quantum cryptography and homomorphic encryption applications. The proposed novel modular polynomial multiplier exploits the fast finite impulse response (FIR) filter architecture to reduce the computational complexity of the schoolbook modular polynomial multiplication. We also extend this structure to fast M-parallel architectures while achieving low-latency, high-speed, and full hardware utilization. We comprehensively evaluate the performance of the proposed architectures under various polynomial settings as well as in the Saber scheme for post-quantum cryptography as a case study. The experimental results show that our proposed modular polynomial multiplier reduces the computation time and area-time product, respectively, compared to the state-of-the-art designs.**

***Index Terms*—Fast filtering, homomorphic encryption, high-speed, lattice-based cryptography, polyphase decomposition, Parallel modular polynomial multiplication, post-quantum cryptography, systolic array, saber cryptosystem.**

I. Introduction

MODULAR polynomial multiplication is commonly used in lattice-based post-quantum cryptography (PQC) and homomorphic encryption applications. While homomorphic encryption aims at allowing computations to be directly carried out in the encrypted domain without decryption [1], lattice-based cryptographic algorithms are also designed to be resistant against attacks from both traditional and quantum computers and are thus well suited for PQC. Three out of the four finalists for the NIST PQC standardization in round-3 fell into the category of lattice-based cryptography [2]. In prior works, the modular polynomial multiplication for the lattice-based cryptography scheme has mostly been implemented by schoolbook polynomial multiplication [3], number theoretic transform (NTT) [4] or the Karatsuba multiplication [5]. Different from the prior works, this paper proposes *novel* high-speed architectures by exploiting the fast finite impulse response (FIR) parallel filter architecture [6], [7], [8], [9], [10]. The paper also proposes a novel *weight-stationary* systolic array for modular polynomial multiplication; these are used as building blocks for the fast parallel architecture. The proposed architecture is feed-forward and can be pipelined at arbitrary levels to achieve the desired speed. To the best of our knowledge, this is the first paper to utilize the fast parallel filter architecture to accelerate the modular polynomial multiplication for lattice-based schemes.

Exploiting the fast parallel filter approach to modular polynomial multiplication is neither straightforward nor trivial. Since the fast parallel filters contain several subfilters and merging operations, the modular operations must be incorporated at the subfilter level and merging level. No prior work has addressed these design aspects. The subfilters should correspond to single-input single-output architectures that should integrate the modular operation and should operate in real-time with no hardware under-utilization. These should also require simpler control circuits. Such designs have not been presented before.

The contributions of this paper are four-fold. First, using systolic mapping methodology [6], [11], [12], we derive a sequential weight-stationary systolic array for modular polynomial operation. This structure is *partly* similar to the *transpose-form* FIR digital filter [6] and is the main building block of the proposed architecture. The low-latency systolic array achieves full hardware utilization. Second, we propose a low-latency fast modular polynomial multiplication architecture that integrates the modular reduction at the merging level, achieves full hardware utilization and minimizes latency. Third, using *iterated* fast parallel filter design approach, we propose highly parallel architectures where the level of parallelism is the product of short lengths. The modular operation is also carried out at the merging step of each iteration to reduce overall latency and achieve full hardware utilization. Fourth, the advantages of the proposed architecture are demonstrated using the Saber scheme as a PQC benchmark.

The rest of the paper is organized as follows: Section II reviews the mathematical background and the prior works on

Manuscript received 11 October 2021; revised 13 February 2023; accepted 26 February 2023. Date of publication 3 March 2023; date of current version 9 August 2023. This paper was supported in part by the Semiconductor Research Corporation under Contract 2020-HW-2998. Recommended for acceptance by J. Hormigo. *(Corresponding author: Keshab K. Parhi.)*

Weihang Tan and Keshab K. Parhi are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455 USA (e-mail: wtan@umn.edu; parhi@umn.edu).

Antian Wang and Yingjie Lao are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634 USA (e-mail: antianw@clemson.edu; ylao@clemson.edu).

Xinmiao Zhang is with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: zhang.8952@osu.edu).

Digital Object Identifier 10.1109/TC.2023.3251847

modular polynomial multiplication. Sections III and IV present the details of the proposed hardware architecture, including the modular polynomial multiplier and fast M-parallel architecture. Section V describes the experimental results and comparisons with the state-of-the-art designs. Finally, Section VI concludes the paper.

II. Background and Related Work

In this section, we briefly review the essential notations, mathematical background, and related works.

A. Lattice-Based Cryptography

Lattice-based cryptography relies on the NP-hard lattice problems that even quantum computers cannot solve efficiently. One example of the lattice problems is the shortest vector problem (SVP), whose security relies on the hardness of approximating SVP in the euclidean norm [1].

There are several representative schemes for both homomorphic encryption and PQC based on the lattice-based cryptography primitive. For example, in lattice-based homomorphic encryption, BFV scheme [13] and CKKS scheme [14] support a limited number of homomorphic computations, which are also categorized as somewhat homomorphic encryption (SHE) schemes. The fully homomorphic encryption (FHE) schemes such as [1], [15] allow an unlimited number of homomorphic computations by using bootstrapping algorithm.

On the other hand, the NIST finalist lattice-based PQC schemes can be classified as either NTRU-based (e.g., NTRU [16]), and learning with errors-based (LWE) (e.g., Crystals-Kyber [17] and Saber [18]) in general.

In this paper, we evaluate and compare the performance of our proposed architectures used in the Saber scheme [18] as a case study. Saber is indistinguishability under chosen-ciphertext attack secure Key Encapsulation Mechanism (KEM), which consists of three algorithms: key generation (KeyGen), encapsulation (Encaps), and decapsulation (Decaps) [18]. More specifically, the primitive of the Saber scheme is based on the hardness of the module-learning with rounding (M-LWR) problem and the use of the Fujisaki-Okamoto transform [19].

Among all the steps in the Saber scheme, the most widely used functions are the matrix-vector multiplication and the inner product of two vectors. For the medium-security level of Saber (post-quantum security level similar to AES-192), there are 9, 12, and 15 polynomial multiplications in the key generation, encapsulation, and decapsulation, respectively. In this paper, we only consider the medium-security level.

Modular polynomial multiplication is a fundamental and yet the most computationally intensive operation of lattice-based cryptography. For the lattice-based PQC, modular polynomial multiplication dominates the computations across key-generation, encryption, and decryption steps in the prior works [3], [20]. Similarly, the most expensive operation for homomorphic encryption schemes is also modular polynomial multiplication. Therefore, improving the efficiency of modular polynomial multiplication is critical to the practical deployment of lattice-based PQC schemes and homomorphic encryption.

B. Schoolbook Modular Polynomial Multiplication

For the product $P(x)$ of two polynomials

$$A(x) = a[0] + a[1]x + a[2]x^2 + \cdots + a[n-1]x^{n-1}, \quad (1)$$

$$B(x) = b[0] + b[1]x + b[2]x^2 + \cdots + b[n-1]x^{n-1}, \quad (2)$$

over R_q, all the coefficients of $P(x)$ need to be less than q but non-negative integers, while the degree of $P(x)$ should be less than n, where $R_q = \mathbb{Z}_q/(x^n+1)$ is ring of the polynomial, and $\mathbb{Z}_q$ is the ring of integers modulo an integer q. The schoolbook modular polynomial multiplication between $A(x)$ and $B(x)$ modulo (x^n+1, q) can be described as

$$\begin{aligned} &A(x)\cdot B(x) \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} a[i]b[j]x^{i+j} \bmod (x^n+1, q) \\ &= \sum_{i=0}^{n-1}\left(\sum_{j=0}^{n-1}(-1)^{\lfloor (i+j)/n \rfloor} a[i]b[j] \bmod q\right)\cdot x^{(i+j) \bmod n}. \end{aligned} \quad (3)$$

For the schoolbook modular polynomial multiplication, the moduli are not required to be prime, which is different from the NTT-based polynomial multiplication. Consequently, the polynomial multiplication used in the M-LWR problem [21] and ring-learning with errors (R-LWE) problem [22] can benefit from these moduli. In these cases, since all the moduli can be selected as power-of-two integers, the modular reduction for the coefficients on the schoolbook polynomial multiplication can be simply performed by keeping the least significant ϵ bits (ϵ is the bit-length of the modulus q, i.e., $\epsilon = \lceil \log_2(q) \rceil$) instead of using the expensive Barrett reduction [23] or Montgomery modular multiplication [24]. Meanwhile, schemes based on the M-LWR problem (such as the Saber scheme) obtain the error term by rounding, while naturally aligning with the power-of-two modulus. Besides, recent work shows that a power-of-two modulus can simplify and improve the polynomial multiplication for the R-LWE based homomorphic encryption schemes without affecting the computational hardness [25]. The modulus in this format has been applied in some popular schemes such as BFV scheme [13]. Based on this advantage, shortening the word-length of the operand and eliminating the modular reduction for the coefficients can increase the resource available, which can then enable the designer to increase the level of parallelism to achieve a high-speed modular polynomial multiplier.

It may be noted that the use of the power-of-two moduli, as needed in the Saber scheme, cannot leverage the acceleration from the NTT-based polynomial multiplication without further expensive transformation. However, NTT-based polynomial multiplication has been widely applied in many lattice-based cryptography schemes when the moduli are not power-of-two [4], [26], [27], [28], [29], [30], [31], [32].

C. Karatsuba Polynomial Multiplication

To improve the efficiency and reduce the complexity of schoolbook polynomial multiplication, methods based on the

divide-and-conquer strategy to increase parallelism are of great interest. One of the examples is the Karatsuba algorithm [33], which has been utilized in some prior modular polynomial multiplier designs for Saber scheme [20], [34]. The 2-level Karatsuba polynomial multiplication first decomposes the input polynomials into higher-degree and lower-degree parts as $A(x) = A_0(x) + A_1(x) \cdot x^{n/2}$ and $B(x) = B_0(x) + B_1(x) \cdot x^{n/2}$ and computes

$$\begin{aligned} C_0(x) &= A_0(x) \cdot B_0(x) \\ C_1(x) &= (A_0(x) + A_1(x)) \cdot (B_0(x) + B_1(x)) \\ C_2(x) &= A_1(x) \cdot B_1(x). \end{aligned} \tag{4}$$

Then the above products are summed up, and polynomial modular reduction is carried out to derive the product $P(x)$ over the ring as

$$P(x) = C_0(x) + C_3(x) \cdot x^{n/2} + C_2(x) \cdot x^n \mod (x^n + 1), \tag{5}$$

where

$$C_3(x) = (C_1(x) - C_0(x) - C_2(x)). \tag{6}$$

Note that the degrees of $C_3(x) \cdot x^{n/2}$ and $C_2(x) \cdot x^n$ are $\frac{3}{2}n$ and $2n$, respectively. Hence polynomial subtractions are needed to perform the modular reduction by $x^n + 1$. Based on this divide-and-conquer strategy of the Karatsuba algorithm, the number of coefficient multiplications is reduced from n^2 to $3(n/2)^2$.

D. Prior Hardware Implementations

Several hardware accelerators for lattice-based cryptography without using the NTT algorithm have been proposed recently [3], [20], [34], [35], [36], [37], [38], [39]. As expected, optimizing the polynomial multiplier is the main focus of these works, since it is the bottleneck. The hardware/software co-design for the modular polynomial multiplication accelerator in [35] shows a significant acceleration compared with the software implementation. Subsequently, the work in [20] introduced the compact hardware/software interfacing design, which applies a hybrid method of Toom-Cook multiplication [40] (a generalized form of Karatsuba algorithm) and a length-64 schoolbook polynomial multiplier to optimize the modular polynomial multiplication. A full hardware implementation is proposed in [3], which utilizes a memory-based schoolbook polynomial multiplier. This design achieves a higher speed where each length-256 polynomial multiplication only consumes 256 clock cycles. Later, an extended work of [3] is presented in [41], which is based on a design called centralized multiplier architecture. This optimized design retains the same timing performance but requires fewer hardware resources since each multiply-and-accumulate (MAC) is replaced by one multiplexer (MUX) and one adder. Furthermore, an 8-level recursive split hierarchical Karatsuba algorithm-based implementation is introduced in [34], which reduces a length-256 polynomial multiplication to only 81 clock cycles without considering the pipelining startup time.

Besides, several architectures of modular polynomial multipliers for the R-LWE schemes are introduced in [5], [38], [42]. The works in [38], [42] investigate the low-area design for the schoolbook modular polynomial multiplication, which only consumes fewer LUTs and DSPs. Meanwhile, the design in [5] proposes a modular polynomial multiplier using the Karatsuba algorithm and reduces the complexity by merging the polynomial modular reduction on the post-processing stage of the Karatsuba algorithm.

However, these designs cannot consider an architecture using the fast filtering technique to reduce the latency. Also, the architectures based on the Karatsuba algorithm generally consider the polynomial modular reduction after the multiplication. These designs do not reduce the number of additions. Therefore, it is possible to further reduce the number of additions/subtractions at the post-processing stage, thereby reducing the total number of addition/subtraction operations. Since our objective is to improve the speed under a given hardware budget, we define the following two metrics in evaluating the performance from the speed perspective:

- Response time: clock cycles between the first input and the first output sample.
- Total latency: clock cycles between the first input and the last output sample.

III. Modular Polynomial Multiplier Based on Weight-Stationary Systolic Array

Consider the design of a length-n modular polynomial multiplier described by (3). In this section, we use $n = 4$ as an example to illustrate our proposed novel modular polynomial multiplier. The modular polynomial multiplication is described by

$$\begin{aligned} P(x) &= A(x) \cdot B(x) \mod (x^4 + 1, q) \\ &= p[0] + p[1]x + p[2]x^2 + p[3]x^3, \end{aligned} \tag{7}$$

where

$$\begin{aligned} A(x) &= a[0] + a[1]x + a[2]x^2 + a[3]x^3, \\ B(x) &= b[0] + b[1]x + b[2]x^2 + b[3]x^3. \end{aligned}$$

The polynomial multiplication of $A(x)$ and $B(x)$ leads to

$$\begin{aligned} P'(x) = {} & p'[0] + p'[1]x + p'[2]x^2 + p'[3]x^3 \\ & + p'[4]x^4 + p'[5]x^5 + p'[6]x^6. \end{aligned} \tag{8}$$

Since the polynomial multiplication has a degree higher than three, the terms x^4, x^5, and x^6 are replaced by -1, $-x$, and $-x^2$, respectively, to perform the modular reduction. Thus, the coefficients of the modular polynomial multiplication are

$$\begin{aligned} p[3] &= a[3]b[0] + a[2]b[1] + a[1]b[2] + a[0]b[3], \\ p[2] &= a[2]b[0] + a[1]b[1] + a[0]b[2] - a[3]b[3], \\ p[1] &= a[1]b[0] + a[0]b[1] - a[3]b[2] - a[2]b[3], \\ p[0] &= a[0]b[0] - a[3]b[1] - a[2]b[2] - a[1]b[3]. \end{aligned} \tag{9}$$

A dependence graph (DG) of the modular polynomial multiplication for the $n = 4$ example is shown in Fig. 1.

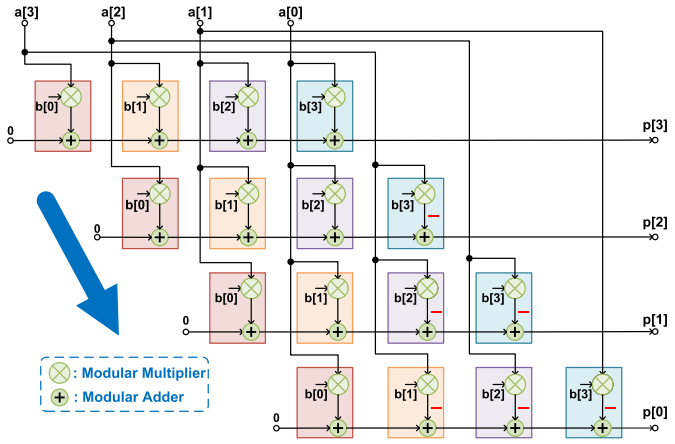


Fig. 1. DG of the modular polynomial multiplication when $n = 4$. The DG is mapped to a systolic array using the projection vector shown in blue.

A. Architecture of Modular Polynomial Multiplier Using Transpose-Form FIR Filter

Given the similarity between modular polynomial multiplication and FIR filter, it is useful to consider three common types of FIR filter structures [6], i.e., direct-form, transpose-form, and hybrid-form, respectively, as shown in Fig. 2.

FIR filter is one of the digital filters that is used to modify the frequency properties of the input signal to achieve specific design requirements [6]. It can also be mathematically expressed as a discrete convolution of two signals, which can be defined as

$$p[n] = \sum_{j=0}^{n-1} b[j]a[n-j], \tag{10}$$

where n is the number of taps, $a[n]$ is the input signal, $b[j]$ is value of the impulse response at the jth instant ($j \in [0, n-1]$), and $p[n]$ is the output signal. Though using any of the FIR filter structures in Fig. 2 can sufficiently instantiate (10) and show a negligible difference in the overall performance in most of the digital signal processing applications, the FIR structure for modular polynomial multiplication needs to be carefully selected.

The direct-form FIR filter shown in Fig. 2(a) leads to a long critical path, which consists of one multiplier and n adders. It can also be observed from Fig. 2(c) that the hybrid-form architecture generates its first output immediately after loading the first input, and requires additional registers to store the intermediate results; however, the architecture is not feed-forward and has slightly longer critical path than the transpose-form. Thus, the best choice for implementing polynomial multiplication in lattice-based schemes is the transpose-form as shown in Fig. 2(b) as it has the least critical path and a feed-forward datapath. Fortunately, the DG in Fig. 1 can be mapped to a weight-stationary systolic array using the projection vector shown in blue in the figure. Alternatively, the systolic array can also be derived using the folding algorithm [43]. Note that all multiplications with coefficient $b[j]$ are mapped to the same hardware multiplier.

Fig. 3(a) shows an example systolic architecture for modular polynomial multiplication for length-4 where the components in each tap (node) are illustrated in Fig. 3(b). The systolic array contains additional switches and a shift register of size-n (see the top of Fig. 3(a)) for continuous processing of input polynomials and polynomial modular reduction. Note that using a conventional transpose-form like structure to perform the polynomial multiplication would require padding zeros until the entire operation finishes; otherwise, it will lead to conflicts and produce wrong results. Furthermore, to perform polynomial modular reduction, the shift register and switches can control the signals (coefficients of polynomial $A(x)$) properly based on the expression in (9). Specifically, the coefficients of polynomial $A(x)$ with the negative signs are extracted from the shift register in its negative form. Then, the switches select either the negative form or the original form coefficients from polynomial $A(x)$ in different clock cycles. As shown in Fig. 3, the proposed length-4 modular polynomial multiplier consists of four modular multipliers, three modular adders, three delay elements, three switches, and one shift register. Specifically, the shift register consists of four delay elements, and the switches are constructed using MUXs. The design in Fig. 3 can be easily extended to length-n. A length-n modular polynomial multiplier requires n modular multipliers, $(n-1)$ modular adders, $(n-1)$ delay elements, $(n-1)$ switches at the lower data paths and one shift register (consisting of n delay elements). For one modular polynomial multiplication, the response time is n clock cycles, while the total latency is $(2n-1)$ clock cycles. For L polynomial multiplications, the response time remains the same, while the total latency in clock cycles is given by

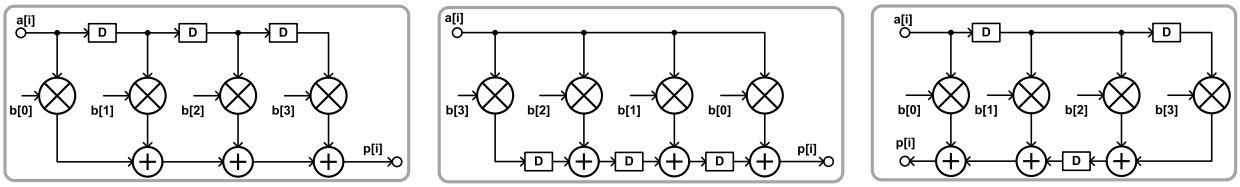
$$T_{lat} = n \cdot (L+1) - 1. \tag{11}$$

This architecture also has a full hardware utilization after the first output is computed. Hardware utilization is the percentage of the components inside this circuit that are performing useful operations, and full hardware utilization means no component is performing null operations.

The modular reduction can be performed by simply keeping the least ϵ bits for a 2^{ϵ} modulus. For the lattice-based cryptography schemes, the degrees of the polynomial are relatively large, i.e., n can be up to hundreds or thousands, which could cause a high fan-out issue on the output of the shift register and the input node. To overcome this, buffers (registers) are inserted after the switches, as shown as the green dashed line in Fig. 3(a). As a result, the critical path is one modular multiplier and one modular adder.

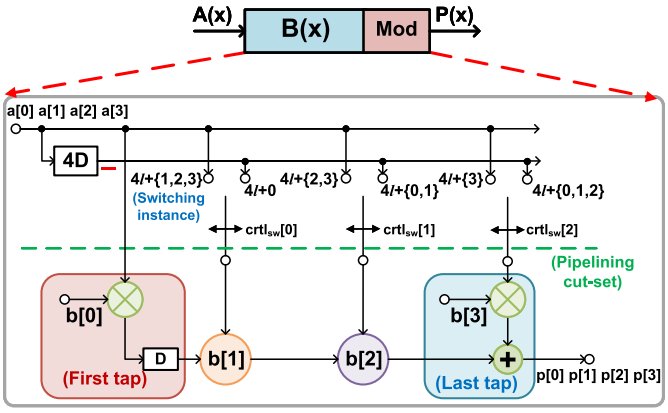
B. Scheduling for the Modular Polynomial Multiplier

The scheduling and control logic in the proposed architecture are very simple and efficient. The coefficients of polynomial $A(x)$ are loaded sequentially from the most significant (highest degree) coefficient to the least significant (lowest degree) coefficient while the coefficients of polynomial $B(x)$ are stored starting with the least significant coefficient to the most significant coefficient from left to right. Finally, the result coefficients are

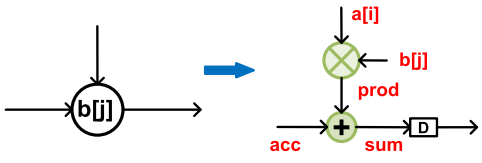


(a) Direct-form (b) Transpose-form (c) Hybrid-form

Fig. 2. Three different forms of FIR filter architecture when $n = 4$.



(a) Top-level architecture with transpose-form like structure.



(b) Details of each tap.

Fig. 3. A degree-4 weight-stationary systolic modular polynomial multiplier.

output in the same order as $A(x)$ (i.e., from the most significant coefficient to the least significant coefficient).

The notation $4l + \{0, 1, 2, 3\}$ represents the switch instances with a switch period of 4 clock cycles. Hence, l can be interpreted as the lth period (iteration). For example, the left node will be connected when the switch instances are $4l + \{1, 2, 3\}$, while the right node will be connected at the switch instance of $4l + 0$. Each switch is controlled by a one-bit signal from the $(n-1)$-bit controller $ctrl_{sw}$: if this bit is equal to 1, the operand from polynomial $A(x)$ of the modular multiplier is loaded from the input node; otherwise, it is loaded from the shift register. These control signals $ctrl_{sw}$ can be simply generated by a counter (ranging from 0 to $(n-2)$), as

$$ctrl_{sw} = \begin{cases} \{0, \ldots, 0, 0\}, & \text{if counter} = 0, \\ \{ctrl_{sw}[n-3:0], 1\}, & \text{otherwise.} \end{cases} \quad (12)$$

After resetting the counter, all $(n-1)$-bit control signals $ctrl_{sw}$ are zeros. Then, in every subsequent clock cycle, $ctrl_{sw}$ shifts left by padding a "1" to the least significant bit (LSB).

IV. Fast Polynomial Multiplier Using Fast M-Parallel Filter Architecture

In this section, we derive a highly parallel hardware architecture for the polynomial multiplication based on the fast parallel filter algorithm [6], [7], [8], [9]. The proposed design requires less resource overhead than prior Karatsuba-based polynomial multipliers in the post-processing stage. Parallel structures for modular polynomial multiplication for small lengths are first derived. These can then be *iterated* to obtain architectures for larger levels of parallelism. For example, a fast 2-parallel (i.e., $M = 2$) modular polynomial multiplier can be iterated twice (or thrice) to design a 4-parallel (or 8-parallel) multiplier.

A. Fast 2-Parallel Architecture

The fast 2-parallel modular polynomial multiplication, referred to as Fast.2.PolyMult, is described in Algorithm 1, which mainly consists of three stages: pre-processing (Step 1), intermediate polynomial multiplication (Step 2), and post-processing (Steps 3 and 4).

We first decompose the polynomials $A(x)$ and $B(x)$ based on the even and odd indices, as shown in Step 1, also called polyphase decomposition [6]. We denote $y = x^2$, and the polynomial $A(x)$ is expressed as

$$A(x) = A_0(x^2) + A_1(x^2) \cdot x = A_0(y) + A_1(y) \cdot x, \quad (13)$$

where the even indexed polynomial $A_0(y)$ and the odd indexed polynomial $A_1(y)$ are expressed as

$$A_0(y) = a[0] + a[2]y + a[4]y^2 + \ldots + a[n-2]y^{n/2-1} \mod (y^{n/2}+1), \quad (14)$$

$$A_1(y) = a[1] + a[3]y + a[5]y^2 + \ldots + a[n-1]y^{n/2-1} \mod (y^{n/2}+1). \quad (15)$$

A similar decomposition is applied to $B(x)$ to obtain its even indexed and odd indexed polynomials $B_0(y)$ and $B_1(y)$. The product $P(x)$ can be computed as

$$\begin{aligned} P(x) &= P_0(y) + P_1(y) \cdot x \\ &= (A_0(y) + A_1(y) \cdot x) \cdot (B_0(y) + B_1(y) \cdot x) \\ &= A_0(y)B_0(y) + [A_0(y)B_1(y) + A_1(y)B_0(y)] \cdot x \\ &\quad + [A_1(y)B_1(y)] \cdot y. \end{aligned} \quad (16)$$

Algorithm 1: Fast.2.PolyMult($A(x), B(x)$).

Input: $A(x)$ and $B(x) \in R_q$
Output: $P(x) = (P_0(x^2), P_1(x^2))$
//$P(x) = A(x) \cdot B(x) \mod (x^n + 1, q)$
1: $A(x) = A_0(x^2) + A_1(x^2) \cdot x$
//Split $A(x)$ as two parts based on odd and even indices
$B(x) = B_0(x^2) + B_1(x^2) \cdot x$
//Split $B(x)$ as two parts based on odd and even indices
2: $U(y) = A_0(y)B_0(y) \mod (y^{n/2} + 1, q)$, where $y = x^2$
$V(y) = A_1(y)B_1(y) \mod (y^{n/2} + 1, q)$
$W(y) = (A_0(y) + A_1(y))(B_0(y) + B_1(y))$
$\mod (y^{n/2} + 1, q)$
// Intermediate modular polynomial multiplication
3: $P_0(y) = U(y) + V(y) \cdot y \mod (y^{n/2} + 1, q)$
$P_1(y) = W(y) - (U(y) + V(y)) \mod (y^{n/2} + 1, q)$
4: $P(x) = P_0(x^2) + P_1(x^2) \cdot x$, where $y = x^2$
5: **return** $P(x)$

The polyphase decomposition describes one polynomial multiplication of length-n in terms of four polynomial multiplications of length-$n/2$. While this step in itself does not reduce the computation complexity, it is an essential first step. In Step 2, the fast filter algorithm describes the modular polynomial multiplication in terms of three polynomial multiplications of half length; this reduces the complexity by 25%. Denote the three intermediate modular polynomial multiplication outputs as $U(y), V(y)$, and $W(y)$. In the fast algorithm, $P_1(y)$ is computed as

$$\begin{aligned} P_1(y) &= A_0(y)B_1(y) + A_1(y)B_0(y) \\ &= (A_0(y) + A_1(y))(B_0(y) + B_1(y)) \\ &\quad - A_0(y)B_0(y) - A_1(y)B_1(y) \qquad (17) \\ &= W(y) - (U(y) + V(y)), \qquad (18) \end{aligned}$$

where

$$U(y) = A_0(y)B_0(y), \qquad (19)$$

$$V(y) = A_1(y)B_1(y), \qquad (20)$$

$$W(y) = (A_0(y) + A_1(y))(B_0(y) + B_1(y)). \qquad (21)$$

Note that unlike $P_1(y)$, $P_0(y) = U(y) + V(y) \cdot y \mod (y^{n/2} + 1)$ requires further modular polynomial reduction, which is achieved in the post-processing step. Since $V(y)$ needs to be multiplied by y before adding the coefficients of $U(y)$, the highest degree of coefficient exceeds the range of the ring $(y^{n/2} + 1)$, (i.e., $U(y) + V(y) \cdot y = u[0] + p_0[1]y + p_0[2]y^2 + \cdots + v[n/2-1]y^{n/2}$). As a result, the even polynomial $P_0(y)$ requires an additional subtraction and is computed as

$$\begin{aligned} P_0(y) &= (u[0] - v[n/2-1]) + p_0[1]y + p_0[2]y^2 \\ &\quad + \cdots + p_0[n/2-1]y^{n/2-1}. \qquad (22) \end{aligned}$$

The data-flow of the proposed fast parallel architecture is shown in Fig. 4.

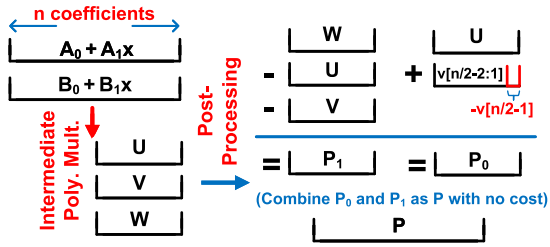


Fig. 4. Data-flow of the Fast.2.PolyMult algorithm.

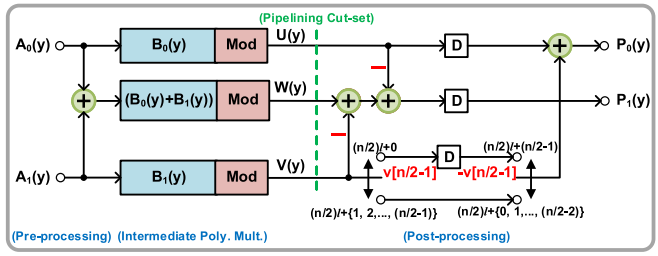


Fig. 5. Fast 2-parallel modular polynomial multiplier.

Different from the traditional methods that execute the polynomial modular reduction during or after post-processing (i.e., combining the intermediate polynomials back to a single polynomial) [5], [20], we integrate polynomial modular reduction into the three intermediate polynomial multiplications. This is achieved by using the sequential systolic modular polynomial multiplication described in the previous section. A 2-level Karatsuba polynomial multiplication requires at least $(n-1)$ clock cycles to output n coefficients sequentially for the three intermediate polynomials and $(\frac{7}{2}n - 4)$ or $(3n-3)$ modular additions/subtractions for post-processing [5]. In contrast, by employing the sequential weight-stationary systolic polynomial modular multiplier as shown in Fig. 3, $\frac{n}{2}$ coefficients of $U(y)$, $V(y)$, and $W(y)$ are output in the same $(n-1)$ clock cycles without requiring additional elements. As these three intermediate polynomials are already in the ring R_q, the post-processing stage has a lower cost, which only needs $\frac{3}{2}n$ modular additions/subtractions.

Fig. 5 depicts the proposed hardware architecture for Algorithm 1 for a length-n modular polynomial multiplication. It mainly consists of four adders/subtractors, three registers, and three length-$\frac{n}{2}$ modular polynomial multipliers that also include three shift registers of size-$\frac{n}{2}$ (as described in Fig. 3). Besides, the bottom path can store $v[n/2-1]$ (coefficient from $V(y)$) for $\frac{n}{2}$ clock cycles and feed its negative form $(-v[n/2-1])$ to the adder at the upper path in each iteration l. This operation is controlled by two switches. When the left switch's instance is at $(n/2)l + 0$, the output coefficient of $V(y)$ is loaded into a register, while the right switch will release the stored data to the next operation at $(n/2)l + (n/2-1)$.

The coefficients of $P_1(y)$ can be simply obtained by using two subtractors, while the coefficients for $P_0(y)$ are more complicated to generate. The addition between $U(y)$ and $V(y) \cdot y$ is explained using the timing diagrams for $n = 8$ shown in Fig.

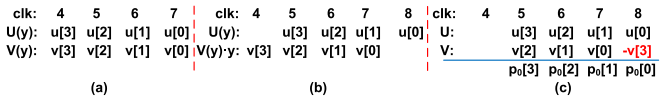


Fig. 6. Timing diagram for $P_0[y]$ at post-processing stage when $n = 8$.

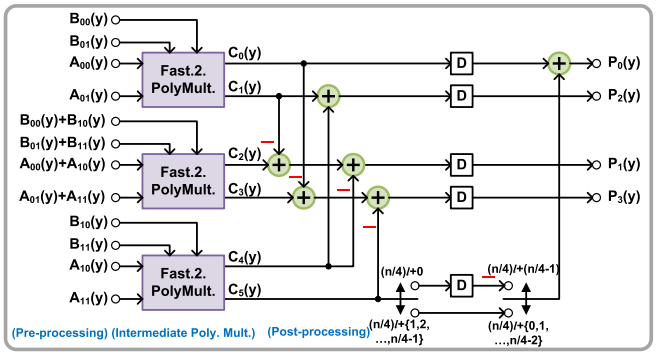


Fig. 7. Fast 4-parallel modular polynomial multiplier.

6. As the coefficients of $U(y)$ and $V(y)$ are generated in the same pattern as shown in Fig. 6(a), directly calculating $P_0(y)$ is infeasible without multiplying y for $V(y)$. However, delaying $U(y)$ by one cycle can enable the addition operation as shown in Fig. 6(b). Furthermore, to perform the polynomial modular reduction as in (22), as described in Fig. 6(c), two switches and two delay elements are required. For the subtraction of $v[3]$ from $u[0]$, the first switch passes $v[3]$ to the delay element and the second switch releases its negative after four clock cycles ($\frac{n}{2}$ clock cycles for general case), as $u[0]$ is output four clock cycles ($\frac{n}{2}$ clock cycles for general case) after $v[3]$. Note that no additional adder/subtractor is needed and full hardware utilization is retained for all the components in the circuit. Moreover, this optimization technique still allows continuous processing of modular polynomial multiplications without requiring any null operations. To align the coefficients of $P_1(y)$ with $P_0(y)$, one delay element is placed at the end of $P_1(y)$'s output.

While the fast modular polynomial multiplier structure is similar to the fast parallel filter, there is one fundamental difference. Unlike the fast parallel filter where all computations are causal, the computation $V(y) \cdot y$ is inherently a *non-causal* operation. This is transformed into a causal operation by introducing a latency of one clock cycle; this can be achieved by placing delays at one feed-forward cut-set in the post-processing step. The proposed *novel* approach of computing $V(y) \cdot y$ does not increase the latency beyond one clock cycle and preserves the feed-forward property of the architecture and continuous data-flow property.

B. Fast 4-Parallel Architecture

A fast 4-parallel architecture can be derived by iterating the fast 2-parallel architecture twice [6], [7], [8], [9]. The fast 4-parallel schoolbook modular polynomial multiplication algorithm (also denoted as Fast.4.PolyMult) is presented in Algorithm 2, while Fig. 7 shows the corresponding architecture.

Algorithm 2: Fast.4.PolyMult($A(x), B(x)$).

Input: $A(x)$ and $B(x) \in R_q$
Output: $P(x) = (P_0(x^4), P_1(x^4), P_2(x^4), P_3(x^4))$,
$//P(x) = A(x) \cdot B(x) \mod (x^n + 1, q)$
1: $A(x) = A_0(x^2) + A_1(x^2) \cdot x^2$
//Split $A(x)$ as two parts based on odd and even indices
$B(x) = B_0(x^2) + B_1(x^2) \cdot x^2$
//Split $B(x)$ as two parts based on odd and even indices
2: $(C_0(y), C_1(y)) =$ Fast.2.PolyMult$(A_0(x^2), B_0(x^2))$, where $y = x^4$
$(C_2(y), C_3(y)) =$ Fast.2.PolyMult$((A_0(x^2) + A_1(x^2)), (B_0(x^2) + B_1(x^2)))$
$(C_4(y), C_5(y)) =$ Fast.2.PolyMult$(A_1(x^2), B_1(x^2))$
3: $P_0(y) = C_0(y) + C_5(y) \cdot y \mod (y^{n/4} + 1, q)$
$P_1(y) = C_2(y) - C_1(y) - C_4(y) \mod (y^{n/4} + 1, q)$
$P_2(y) = C_1(y) + C_4(y) \mod (y^{n/4} + 1, q)$
$P_3(y) = C_3(y) - C_0(y) - C_5(y) \mod (y^{n/4} + 1, q)$
4: $P(x) = P_0(x^4) + P_1(x^4) \cdot x + P_2(x^4) \cdot x^2 + P_3(x^4) \cdot x^3$, where $y = x^4$
5: **return** $P(x)$

The Fast.4.PolyMult algorithm has four steps. In Step 1 of Algorithm 2, $A(x)$ is split into two parts based on the odd and even indices. Then, $A_0(x^2)$, $A_1(x^2)$, and their sum $(A_0(x^2) + A_1(x^2))$ are further split based on Step 1 in Fast.2.PolyMult (Algorithm 1). $A_0(x^2)$ and $A_1(x^2)$ are decomposed as four polynomials $(A_{00}(x^4), A_{01}(x^4), A_{10}(x^4), A_{11}(x^4))$ which are fed to upper and lower fast 2-parallel modular polynomial multipliers (denoted Fast.2.PolyMult. in Fig. 7), respectively. Meanwhile, as the fast 2-parallel modular polynomial multiplier has two inputs in parallel, $(A_0(x^2) + A_1(x^2))$ in Step 2 is simply implemented as two adders in the middle fast 2-parallel modular polynomial multiplier, i.e., $(A_{00}(x^4) + A_{10}(x^4))$ and $(A_{01}(x^4) + A_{11}(x^4))$. Let y represent x^4; Hence the four polynomials decomposed from $A_0(x^2)$ and $A_1(x^2)$ can be expressed as

$$A_{00}(y) = a[0] + a[4]y + a[8]y^2 + \cdots + a[n-4]y^{n/4-1} \mod (y^{n/4} + 1), \quad (23)$$

$$A_{10}(y) = a[1] + a[5]y + a[9]y^2 + \cdots + a[n-3]y^{n/4-1} \mod (y^{n/4} + 1), \quad (24)$$

$$A_{01}(y) = a[2] + a[6]y + a[10]y^2 + \cdots + a[n-2]y^{n/4-1} \mod (y^{n/4} + 1), \quad (25)$$

$$A_{11}(y) = a[3] + a[7]y + a[11]y^2 + \cdots + a[n-1]y^{n/4-1} \mod (y^{n/4} + 1), \quad (26)$$

where

$$A(x) = A_{00}(x^4) + A_{10}(x^4) \cdot x + A_{01}(x^4) \cdot x^2 + A_{11}(x^4) \cdot x^3. \quad (27)$$

$B(x)$ can be decomposed in a similar manner.

Algorithm 3: Fast.3.PolyMult($A(x), B(x)$).

Input: $A(x)$ and $B(x) \in R_q$
Output: $P(x) = (P_0(x^3), P_1(x^3), P_2(x^3))$,
$//P(x) = A(x) \cdot B(x) \mod (x^n + 1, q)$

1: $A(x) = A_0(x^3) + A_1(x^3) \cdot x + A_2(x^3) \cdot x^2$
$B(x) = B_0(x^3) + B_1(x^3) \cdot x + B_2(x^3) \cdot x^2$
2: $C_0(y) = A_0(y)B_0(y) \mod (y^{n/3} + 1, q)$
$C_1(y) = A_1(y)B_1(y) \mod (y^{n/3} + 1, q)$
$C_2(y) = A_2(y)B_2(y) \mod (y^{n/3} + 1, q)$
$C_3(y) = (A_0(y) + A_1(y))(B_0(y) + B_1(y))$
$\mod (y^{n/3} + 1, q)$
$C_4(y) = (A_1(y) + A_2(y))(B_1(y) + B_2(y))$
$\mod (y^{n/3} + 1, q)$
$C_5(y) = (A_0(y) + A_1(y) + A_2(y))(B_0(y) + B_1(y)$
$+ B_2(y)) \mod (y^{n/3} + 1, q)$, where $y = x^3$
3: $D_0(y) = C_3(y) - C_1(y) \mod (y^{n/3} + 1, q)$
$D_1(y) = C_4(y) - C_1(y) \mod (y^{n/3} + 1, q)$
$D_2(y) = C_0(y) - C_2(y) \cdot y \mod (y^{n/3} + 1, q)$
$D_3(y) = C_5(y) \mod (y^{n/3} + 1, q)$
4: $P_0(y) = D_2(y) + D_1(y) \cdot y \mod (y^{n/3} + 1, q)$
$P_1(y) = D_0(y) - D_2(y) \mod (y^{n/3} + 1, q)$
$P_2(y) = D_3(y) - D_0(y) - D_1(y) \mod (y^{n/3} + 1, q)$
5: $P(x) = P_0(x^3) + P_1(x^3) \cdot x + P_2(x^3) \cdot x^2$, where
$y = x^3$
6: **return** $P(x)$

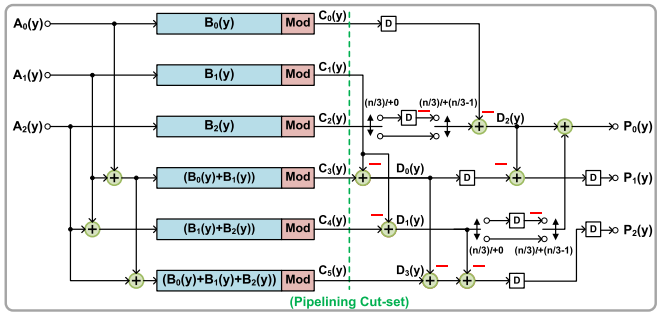


Fig. 8. Fast 3-parallel modular polynomial multiplier.

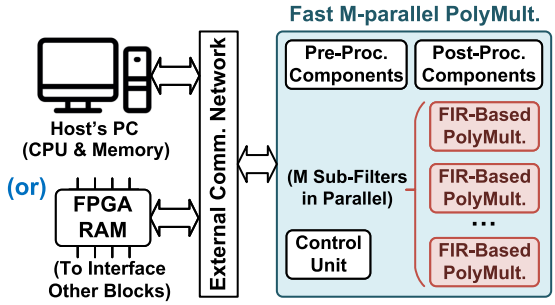


Fig. 9. High-level overview of generalized fast M-parallel modular polynomial multiplier.

In the intermediate polynomial multiplication stage, three length-$\frac{n}{4}$ fast 2-parallel modular polynomial multipliers (Fig. 5) generate six length-$\frac{n}{4}$ polynomials, $C_0(y), C_1(y), \ldots, C_5(y)$. As shown in Fig. 7, $\frac{3}{2}n$ additions/subtractions are carried out by six adders/subtractors, where each adder/subtractor performs $\frac{n}{4}$ additions/subtractions. Finally, polynomial modular reduction for $C_5(y)$ and $C_0(y)$ are performed in a manner similar to the fast 2-parallel architecture (Fig. 5).

C. Fast 3-Parallel Architecture

We also present the design for a fast 3-parallel schoolbook modular polynomial multiplication algorithm (denoted as Fast.3.PolyMult), allowing M to be a multiple of 3, enabling various levels of parallelism. Fast.3.PolyMult algorithm also consists of three stages, which is illustrated in Algorithm 3. During the polyphase decomposition (pre-processing stage), polynomial $A(x)$ is decomposed as

$$A(x) = A_0(x^3) + A_1(x^3) \cdot x + A_2(x^3) \cdot x^2. \quad (28)$$

The modular multiplication result $P(x)$ can be defined as

$$P(x) = P_0(y) + P_1(y) \cdot x + P_2(y) \cdot x^2, \quad (29)$$

where $y = x^3$, and these three sub-polynomials are presented in Step 4 in Algorithm 3. The derivation of the fast 3-parallel modular multiplier is similar to the fast parallel filter derivation; the reader is referred to [6], [7], [9].

The architecture for the Fast.3.PolyMult algorithm is shown in Fig. 8, which consists of six length-$\frac{n}{3}$ modular polynomial multipliers, thirteen modular adders/subtractors with additional delay elements. These six length-$\frac{n}{3}$ modular multipliers compute the intermediate polynomials $C_0(y)$ to $C_5(y)$ with an additional pipelining stage at the end of the modular multipliers' output.

In the post-processing stage, six intermediate polynomials are used to generate four new intermediate polynomials $D_0(y)$ to $D_3(y)$ before computing the outputs $P_0(y)$, $P_1(y)$, and $P_2(y)$ using fewer additions/subtractions.

D. Fast M-Parallel Architecture

Using the *iterated* approach, we can use fast 2-parallel architecture and/or fast 3-parallel architecture to achieve higher levels of parallelism. Therefore, we can implement various fast M-parallel architectures, where the level of parallelism M can be a power-of-two integer, power-of-three integer, or product of any power-of-two and power-of-three.

The high-level overview of the generalized fast M-parallel architecture is shown in Fig. 9. This architecture mainly has M sub-modular polynomial multipliers of length-$\frac{n}{M}$ operating in parallel to generate M sub-polynomials of $P(x)$. In addition, the components for post-processing as well as the control unit, are used to align the coefficients from all the output sub-polynomials of $P(x)$. This is similar to inserting a pipelining cut-set to transform non-causal operations into causal operations, at the expense of an increase in latency by one cycle. During the computation, the data can be either accessed from the host's personal computer (PC) to get the input polynomials' coefficients directly or communicated to the FPGA's RAM.

The timing performance can be theoretically derived as follows. The fast M-parallel design can reduce the response time to

TABLE I
COMPARISON OF AREA CONSUMPTION AND FREQUENCY FOR MODULAR POLYNOMIAL MULTIPLIER WHEN $n = 256$

Design	Device	LUTs	FFs	DSPs	BRAM	Freq. [MHz]
Roy (1 Mult.) [3]	Ultrascale+	17406	5083	0	0	250
Roy (2 Mult.) [3]	Ultrascale+	31853	8844	0	0	250
Zhu [34]	Ultrascale+	13954	3943	85	6	100
Basso (HS-I 256) [41]	Ultrascale+	10844	5150	0	0	250
Basso (HS-I 512) [41]	Ultrascale+	22118	4920	0	0	250
FIR.PolyMult	Ultrascale+	16971	8755	0	0	250
Fast.2.PolyMult	Ultrascale+	25831	12850	0	0	250
Fast.4.PolyMult	Ultrascale+	35306	19143	64	0	250
Mera [20]	Artix-7	7400	7331	38	2	125
FIR.PolyMult	Artix-7	16902	8755	0	0	133
Fast.2.PolyMult	Artix-7	25854	12850	0	0	133
Fast.4.PolyMult	Artix-7	35396	19143	64	0	133

approximately n/M clock cycles. In general, the total latency of an M-parallel modular polynomial multiplier for L polynomial multiplications can be expressed as

$$T_{lat} = n(1 + L)/M + \lceil \log_2(M) \rceil. \tag{30}$$

V. EXPERIMENTAL RESULTS

The performance of the proposed modular polynomial multiplication is demonstrated for the Saber scheme using Verilog HDL implementation. Several changes have been adopted specifically for the Saber scheme. Due to the Saber scheme's advantages, the basic components do not consume a large amount of hardware resources. In particular, the modular multiplier discussed in Section III can be replaced by a few adders since the coefficients of polynomial $B(x)$ are in the range of $[-4, 4]$ [18]. As the moduli are power-of-two integers, the modular reduction can again be performed by simply keeping the lower bits. Note that, the coefficients in both polynomials $A(x)$ and $B(x)$ are represented in the sign-magnitude form, and the word-lengths of the magnitudes of these two polynomials are 13-bit and 3-bit, respectively. The modular adder calculates the 13-bit sum (sum) by adding the product ($prod$) of the corresponding $a[i]$ and $b[j]$, and the output from the register acc as shown in Fig. 3(b), which can also be mathematically expressed as

$$sum = \begin{cases} acc - prod, & \text{if } a_{sign} \bigoplus b_{sign} = 1, \\ acc + prod, & \text{otherwise,} \end{cases} \tag{31}$$

where a_{sign} and b_{sign} are the sign bits of the two operands $a[i]$ and $b[j]$, respectively. Note that all the modular polynomial multiplications correspond to degree $n = 256$.

The experiment is performed on the Xilinx Artix-7 AC701 FPGA since Artix-7 family FPGAs are recommended by NIST for PQC hardware implementation. In addition, since several prior works also used the high-performance Xilinx UltraScale+ FPGA, we also demonstrate the performance of our architecture on this FPGA for more comparisons. The communication and data transmission between FPGA and PC use the universal asynchronous receiver transmitter (UART) module provided by the AC701 device for functionality verification.

A. Evaluation of Modular Polynomial Multiplier

We first examine the performance of our proposed modular polynomial multipliers, including the FIR filter-based (Fig. 3), fast 2-parallel architecture, and fast 4-parallel architecture.

The experimental results and comparison with prior works [3], [20], [34], [41] are summarized in Table I. A further comparison of the timing performance is presented in Table II. The clock frequencies are set as 250 MHz and 133 MHz for UltraScale+ and Artix-7, respectively.

Table II summarizes the number of clock cycles and actual latency for one modular polynomial multiplication (PolyMult.), and all the modular polynomial multiplications in KeyGen, Encaps, and Decaps steps of Saber scheme with the medium security level. Note that the number of modular polynomial multiplications in encryption (decryption) is the same as in Encaps (Decaps). It can be seen from Table I that our design has a shorter critical path than those of the designs in [20], [34] and the same as the work in [3], [41].

For a fair comparison, we focus on the evaluation against the architecture [3], and its extended work [41] since both works use the same clock frequency and target high-speed design.

Note that the high-speed designs in [41] do not provide the timing performance in the KeyGen, Encaps, and Decaps steps for Saber scheme but only the result for one modular polynomial multiplication; so we adopt the number of clock cycles from their previous work [3] and present in Table II. In particular, the framework and the timing performance (including the number of clock cycles and frequency) for one modular polynomial multiplication of [41] are maintained to be the same as their previous work [3], while their optimized centralized multiplier design for the MAC unit in [41] significantly reduces LUTs. Besides, these two works present the general design and parallel design. In Tables I and II, the general designs correspond to Roy (1 Mult.) as well as Basso (HS-I 256), and parallel designs correspond to Roy (2 Mult.) as well as Basso (HS-I 512).

Compared to the general design in [3], our proposed FIR filter-based modular polynomial multiplier (i.e., FIR.PolyMult) has slightly fewer LUTs and a smaller total latency in the modular polynomial multiplications used in three steps of the

TABLE II
TIMING PERFORMANCE OF MODULAR POLYNOMIAL MULTIPLIER WHEN $n = 256$ IN MEDIUM SECURITY LEVEL OF SABER

Design	Device	1 PolyMult.[a]	KeyGen[a]	Encaps[a]	Decaps[a]	ATP-LUT[b]
Roy (1 Mult.) [3][c]	Ultrascale+	256 (1.02)	2685 (10.74)	3592 (14.37)	4484 (17.94)	7.49×10^5
Roy (2 Mult.) [3][c]	Ultrascale+	128 (0.51)	1552 (6.21)	2205 (8.82)	2911 (11.64)	8.50×10^5
Zhu [34]	Ultrascale+	81 (0.81)	(Not Reported)	978 (9.78)	1227 (12.27)	-
Basso (HS-I 256) [41][c]	Ultrascale+	256(1.02)	2685 (10.74)	3592 (14.37)	4484 (17.94)	4.67×10^5
Basso (HS-I 512) [41][c]	Ultrascale+	128 (0.51)	1552 (6.21)	2205 (8.82)	2911 (11.64)	5.89×10^5
FIR.PolyMult	Ultrascale+	511 (2.04)	2560 (10.24)	3328 (13.31)	4096 (16.38)	6.77×10^5
Fast.2.PolyMult	Ultrascale+	255 (1.02)	1281 (5.12)	1665 (6.66)	2049 (8.20)	5.16×10^5
Fast.4.PolyMult	Ultrascale+	127 (0.51)	642 (2.57)	834 (3.34)	1026 (4.10)	3.50×10^5
Mera [20]	Artix-7	1299 (10.30)	11592 (92.74)	15456 (123.65)	19320 (154.56)	27.45×10^5
FIR.PolyMult	Artix-7	511 (3.83)	2560 (19.25)	3328 (25.02)	4096 (30.80)	12.66×10^5
Fast.2.PolyMult	Artix-7	255 (1.92)	1281 (9.63)	1665 (12.52)	2049 (15.41)	9.71×10^5
Fast.4.PolyMult	Artix-7	127 (0.95)	642 (4.83)	834 (6.27)	1026 (7.71)	6.65×10^5

[a]: Total latency in the unit of clock cycle (actual latency in the unit of μs) of one modular polynomial multiplication, or all the modular polynomial multiplications in Saber's specific step
[b]: ATP-LUT (area-time product of LUTs) is calculated from the number of LUTs times the sum of actual latency (μs) of the total number of modular polynomial multiplications in KeyGen, Encaps, and Decaps steps
[c]: Clock cycles for reading and writing operations are not counted

Saber scheme; however, a higher number of flip-flops (FFs) is needed due to the additional shift registers. Note that the clock cycles used in one modular polynomial multiplication in Roy (1 Mult.) are only 256 clock cycles due to the fact that their result does not count the number of clock cycles used for reading and writing operations [41], which is same as the response time in our proposed FIR.PolyMult design. When compared to Roy (2 Mult.), our fast 2-parallel architecture achieves 18.91%, 25.08%, and 39.88% reduction on the number of LUTs, latency, and area-time product of LUTs (ATP-LUT), respectively.

Besides, their extended work [41] is also taken into comparison. Our FIR.PolyMult and fast 2-parallel architectures require a larger number of LUTs compared to their general design (Basso (HS-I 256)) and parallel design (Basso (HS-I 512)), but the latency of our two proposed designs is smaller than theirs. In this case, the performance of ATP-LUT is utilized for a fair comparison. The Basso (HS-I 256) design has a 31.02% reduction compared to our FIR.PolyMult architecture. However, the ATP-LUT product of our fast 2-parallel architecture is reduced by 13.24% compared to the Basso (HS-I 512). This result implies that our fast M-parallel design has a superior performance compared to conventional parallel processing techniques.

Even though our design requires more FFs in the data-path and shift registers, we argue that it has a small influence on the overall performance of UltraScale+ and Artix-7 FPGAs since both devices have a much higher resource budget for FFs than LUTs. For example, our proposed FIR.PolyMult design consumes 12% of LUTs (16971/134600), but only 3% of FFs (8755/269200) in AC701 FPGA. Even for the fast 4-parallel architecture, only 7% FFs are utilized.

Furthermore, both modular polynomial multiplier in LWR-pro [34] and the compact modular polynomial multiplier in [20], [34] use the Toom-Cook/Karatsuba algorithm with 8-level and 4-level, respectively. The compact polynomial multiplier in [20] has a long critical path of five adders/subtractors and two multipliers in the interpolation part, which requires two pipelining stages to reduce the critical path for maintaining a high frequency. This design targets the low-area performance, which only requires limited numbers of LUTs, FFs, and only 38 DSP units, as shown in Table I. While this design has lower LUT usage than our architecture, it suffers from a low speed since their length-64 polynomial multipliers require 1,168 clock cycles for each computation, which causes the actual latency in such a compact design to be around 19 times of the latency in our fast 4-parallel architecture as presented in Table II. If we consider the ATP-LUT as the performance metric to compare our proposed fast 4-parallel architecture and this prior low-area design, it shows that our design achieves a 75.77% reduction.

Besides, the modular polynomial multiplier in [34] requires the lowest number of clock cycles among all the prior works, while having the lower clock frequency as illustrated in Table II. In comparison, our fast 4-parallel architecture requires 15.65% fewer clock cycles and achieves a 66.26% reduction in the actual latency for all the modular polynomial multiplications in the Encaps and Decaps steps. Considering the area performance of this work, their modular polynomial multiplier uses 60.48% fewer LUTs but 24.71% more DSPs compared to the proposed fast 4-parallel architecture. Thus, the ATP products of DSP and LUT need to be considered separately. Since the clock cycles used for KeyGen are not reported in [34], the ATP-LUT (ATP-DSP) for the comparison with this work is defined as the number of LUTs (DSPs) times the sum of actual latency (μs) of the total number of modular polynomial multiplications in two steps only, Encaps and Decaps. Specifically, the ATP-LUT and ATP-DSP in their modular polynomial multiplier are 3.08×10^5 and 1874.20, respectively. The ATP-LUT and ATP-DSP in the fast 4-parallel architecture are 2.63×10^5 and 476.16, respectively.

TABLE III
PERFORMANCE OF MODULAR POLYNOMIAL MULTIPLIER USING FAST M-PARALLEL ARCHITECTURE WHEN $n = 180$ BASED ON ARTIX-7 FPGA FAMILY

Design	LUTs	FFs	DSPs	Freq. [MHz]	1 PolyMult.[a]	9 PolyMult.[a]	ATP-LUT[b]	Throughput[c]
Fast.2.PolyMult	17902	9096	0	133	181 (1.36)	901 (6.77)	1.21×10^5	2
Fast.3.PolyMult	21729	11996	60	133	122 (0.91)	602 (4.53)	0.98×10^5	3
Fast.4.PolyMult	25110	13633	45	133	92 (0.69)	452 (3.40)	0.85×10^5	4

[a]: Total latency in the unit of clock cycle (actual latency in the unit of μs)
[b]: ATP-LUT (area-time product of LUTs) is calculated from the number of LUTs times the sum of actual latency (μs) of nine modular polynomial multiplications
[c]: Throughput in the unit of samples per clock cycle

TABLE IV
COMPARISON WITH RECENT SABER SCHEME IMPLEMENTATION IN MEDIUM SECURITY LEVEL

	Platform	Time in (μs): KeyGen/Encaps/Decaps	Freq. [MHz]	Area: LUTs/FFs/DSPs/BRAM	ATP-LUT[a]
Roy [3]	UltraScale+	21.8/26.5/32.1	250	23.6k/9.8k/0/2	1.9×10^6
Ours	UltraScale+	10.2/12.6/15.6	250	41.5k/22.3k/64/2	1.6×10^6
Ours	Artix-7	19.2/23.6/29.2	133	41.5k/22.3k/64/2	3.0×10^6

[a]: ATP-LUT (area-time product of LUTs) is calculated from the number of LUTs times the sum of actual latency (μs) of KeyGen, Encaps, and Decaps steps

Under this comparison, ATP-LUT and ATP-DSP are reduced by 14.61% and 75.07%, respectively, in our design.

Thus, we can conclude that our design achieves a significant reduction in the latency or the delay (critical path), which leads to reductions in ATP compared to the two prior works that employ the Karatsuba/Toom-Cook algorithm-based modular polynomial multiplication.

Our proposed modular polynomial multipliers can be sufficient to support different security levels of Saber without any change of the area consumption and the frequency presented in Table I, but only requires a different number of clock cycles.

B. Parallel Architectures

The works in [3] and [41] also present a parallel architecture, which is a scaled version. The parallel design in [3] (Roy (2 Mult.)) uses two multipliers in one MAC unit, and the parallel design in [41] (Basso (HS-I 512)) doubles their MAC units (each MAC unit has one MUX and one adder). When compared to the Roy (2 Mult.) design, our fast 2-parallel architecture achieves a significant reduction in the area overhead and latency. In particular, our fast 2-parallel architecture consumes only about 34% higher area consumption than the FIR filter polynomial multiplier while reducing latency by 50%, while their scaled version of the parallel modular polynomial multiplier has 45% overhead compared with the general design architecture (Roy (1 Mult.)). Along with parallelization, the delay also increases, as the critical path will change from one multiplication and one addition to one multiplication and two additions. In this case, an additional pipeline is added in the design of Roy (2 Mult.) [3] to maintain the same high frequency. Under the same number of pipelining stages, our fast 2-parallel architecture achieves a lower critical path and hence can be driven by a clock with a higher frequency.

We also compare different fast M-parallel architectures for $n = 180$ in Table III. It can be noticed that when the level of parallelism increases (M becomes larger), the actual latency is reduced at the expense of higher area consumption. Besides, the throughput of the designs is increased when M becomes larger. The ATP-LUT product for the fast 2-parallel, 3-parallel, and 4-parallel architectures are listed for computing nine modular polynomial multiplications, which indicates that a higher level of parallelism can provide a more efficient design if sufficient resource budget is available.

C. Comparison With Saber PQC Scheme Implementations

For the implementation of the entire Saber scheme, the modular polynomial multiplication is implemented by the proposed fast 4-parallel architecture, while other simple functional blocks are modified from the open-source codes provided in [18] and [3].

Table IV presents the comparison of the FPGA performance with recent hardware implementation [3] for the Saber PQC scheme at a medium security level. The latency in our design is 52% less than the latency in [3] with the cost of more LUTs and DSPs consumed. In fact, the reduction is mainly from our optimized low-latency modular polynomial multiplier. Besides, instead of directly adopting the open-source SHA3 hash function block as in [3], we also implement the hash function block when implementing the entire scheme. For example, the total latency of SHA3-256 (needs to process 32-byte, 64-byte, 992-byte, and 1088-byte seeds) operating in the hash function block is reduced from 585 clock cycles to 526 clock cycles in the Saber Encaps. The rationale behind this latency reduction is as follows. Most open-source packages add stages of pipelining to achieve a high frequency (low critical path) design in order to adapt to general applications [44]. However, the critical path of the modular polynomial multiplier that requires addition or multiplication from prior work is much higher than the Keccak core provided in the open-source packages, thus implying that some pipelines are redundant. Different from the prior work, we implement our

own hash function block as we aim to reduce the total latency for computing the hash functions by eliminating unnecessary pipelining stages.

For the area performance, although we have increased hardware costs, both Artix-7 and UltraScale+ FPGAs still have sufficient resources to accommodate our fast 4-parallel design. In other words, our proposed fast 4-parallel architecture is under the constraint of hardware complexity specified by NIST (Artix-7).

VI. Conclusion

This paper has presented a novel modular polynomial multiplier and demonstrated its applications for lattice-based cryptography. The proposed hardware design exploits the fast filtering technique to achieve low latency, high scalability, and full hardware utilization. We proposed efficient parallel architectures with much lower hardware overhead and latency than prior works. Our design can be easily generalized across different levels of parallelism. Comprehensive experimental results are presented. We show that our design achieves superior performance than the state-of-the-art modular polynomial multipliers based on schoolbook polynomial multiplication or the Karatsuba algorithm. A case study of the implementation of the Saber scheme shows that our proposed design can accelerate the computation and reduce the actual latency of the cryptosystem compared with the prior work.

VII. Disclosure

Part of this paper is covered in the patent application [45].

References

[1] C. Gentry, Ph.D. dissertation, Stanford Univ., Stanford, CA, Sep. 2009.

[2] National Institute of Standards and Technology (NIST), "Post-quantum cryptography standardization," 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[3] 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, pp. 443–466, 2020.

[4] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of NewHope-NIST on FPGA using low-complexity NTT/INTT," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, pp. 49–72, 2020.

[5] X. Zhang, Z. Huai, and K. K. Parhi, "Polynomial multiplication architecture with integrated modular reduction for R-LWE cryptosystems," *J. Signal Process. Syst.*, vol. 94, no. 8, pp. 799–809, 2022.

[6] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. Hoboken, NJ, USA: Wiley, 1999.

[7] D. A. Parker and K. K. Parhi, "Low-area/power parallel FIR digital filter implementations," *J. VLSI Signal Process. Syst. Signal Image Video Technol.*, vol. 17, no. 1, pp. 75–92, 1997.

[8] C. Cheng and K. K. Parhi, "Hardware efficient fast parallel FIR filter structures based on iterated short convolution," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 51, no. 8, pp. 1492–1500, Aug. 2004.

[9] D. A. Parker and K. K. Parhi, "Area-efficient parallel FIR digital filter implementations," in *Proc. IEEE Int. Conf. Appl. Specific Syst., Architectures Processors*, 1996, pp. 93–111.

[10] J. Tian, B. Wu, and Z. Wang, "High-speed FPGA implementation of SIKE based on an ultra-low-latency modular multiplier," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 68, no. 9, pp. 3719–3731, Sep. 2021.

[11] S. Y. Kung, "VLSI array processors," *IEEE ASSP Mag.*, vol. 2, no. 3, pp. 4–22, Jul. 1985.

[12] H. V. Jagadish, S. K. Rao, and T. Kailath, "Array architectures for iterative algorithms," *Proc. IEEE*, vol. 75, no. 9, pp. 1304–1321, Sep. 1987.

[13] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, pp. 144, 2012, Paper no. 2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[14] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, Springer, 2017, pp. 409–437.

[15] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf.*, Springer, 2013, pp. 75–92.

[16] C. Chen et al., "NTRU: Algorithm specifications and supporting documentation," Mar. 30, 2019. [Online]. Available: https://ntru.org/f/ntru-20190330.pdf

[17] J. Bos et al., "CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2018, pp. 353–367.

[18] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *Proc. Int. Conf. Cryptol. Afr.*, Springer, 2018, pp. 282–305.

[19] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Proc. Annu. Int. Cryptol. Conf.*, Springer, 1999, pp. 537–554.

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

[21] J. Alwen, S. Krenn, K. Pietrzak, and D. Wichs, "Learning with rounding, revisited: New reduction, properties and applications," in *Proc. Annu. Cryptol. Conf.*, Springer, 2013, pp. 57–74.

[22] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM*, vol. 60, no. 6, pp. 1–35, 2013.

[23] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Proc. Conf. Theory Appl. Cryptogr. Techn.*, Springer, 1986, pp. 311–323.

[24] P. L. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.

[25] Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D. Stehlé, "Classical hardness of learning with errors," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, 2013, pp. 575–584.

[26] G. Xin et al., "VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 67, no. 8, pp. 2672–2684, Aug. 2020.

[27] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2021, pp. 328–356, 2021.

[28] D. T. Nguyen, V. B. Dang, and K. Gaj, "A high-level synthesis approach to the software/hardware codesign of NTT-based post-quantum cryptography algorithms," in *Proc. Int. Conf. Field-Programmable Technol.*, 2019, pp. 371–374.

[29] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Trans. Cryptogr. Hardware Embedded Syst.*, vol. 2019, pp. 17–61, 2019.

[30] D. T. Nguyen, V. B. Dang, and K. Gaj, "High-level synthesis in implementing and benchmarking number theoretic transform in lattice-based post-quantum cryptography using software/hardware codesign," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2020, pp. 247–257.

[31] S. S. Roy, F. Vercauteren, J. Vliegen, and I. Verbauwhede, "Hardware assisted fully homomorphic function evaluation and encrypted search," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1562–1572, Sep. 2017.

[32] W. Tan, A. Wang, Y. Lao, X. Zhang, and K. K. Parhi, "Pipelined high-throughput NTT architecture for lattice-based cryptography," in *Proc. IEEE Asian Hardware Oriented Secur. Trust Symp.*, 2021, pp. 1–4.

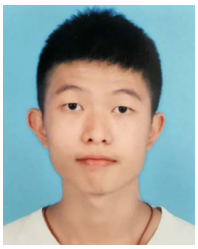
[33] A. Karatsuba, "Multiplication of multidigit numbers on automata," *Sov. Phys. Doklady*, vol. 7, pp. 595–596, 1963.

[34] 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.

[35] 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.

[36] 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.

[37] S. Fan, W. Liu, J. Howe, A. Khalid, and M. O'Neill, "Lightweight hardware implementation of R-LWE lattice-based cryptography," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, 2018, pp. 403–406.
[38] T. Pöppelmann and T. Güneysu, "Area optimization of lightweight lattice-based encryption on reconfigurable hardware," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2014, pp. 2796–2799.
[39] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using karatsuba algorithm," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 335–347, Mar. 2018.
[40] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Sov. Math. Doklady*, vol. 3, no. 4, pp. 714–716, 1963.
[41] A. Basso and S. S. Roy, "Optimized polynomial multiplier architectures for post-quantum KEM saber," in *Proc. IEEE/ACM 58th Des. Automat. Conf.*, 2021, pp. 1285–1290.
[42] Y. Zhang, C. Wang, D. E. S. Kundi, A. Khalid, M. O'Neill, and W. Liu, "An efficient and parallel R-LWE cryptoprocessor," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 67, no. 5, pp. 886–890, May 2020.
[43] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE J. Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, Jan. 1992.
[44] M. Sundal and R. Chaves, "Efficient FPGA implementation of the SHA-3 hash function," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, 2017, pp. 86–91.
[45] K. K. Parhi, X. Zhang, W. Tan, A. Wang, and Y. Lao, "Low latency polynomial modulo multiplication over ring," 2022, US Patent 17,582,560, Filed: Jan. 24, 2022.



Weihang Tan (Graduate Student Member, IEEE) received the BS, MS, and PhD degrees in electrical engineering from Clemson University, Clemson, South Carolina,, in 2018, 2020, and 2022, respectively. He is a postdoctoral research associate with the Department of Electrical and Computer Engineering, University of Minnesota. His research interests include hardware security and VLSI architecture design for fully homomorphic encryption, post-quantum cryptography, and digital signal processing systems. He is the recipient of the best PhD forum presentation award with the Asian Hardware Oriented Security and Trust Symposium (AsianHOST).

Antian Wang (Graduate Student Member, IEEE) received the BE degree in communication engineering from Shanghai Maritime University, in 2017. He is currently working toward the PhD degree with Clemson University. His research interests include hardware security and VLSI architecture design, and design automation.

Xinmiao Zhang (Senior Member, IEEE) received the PhD degree from the University of Minnesota. She joined The Ohio State University as an Associate Professor, in 2017. Prior to that, she was a senior technologist with Western Digital, an Associate Professor with Case Western Reserve University. Her research interests include VLSI architecture design, digital storage and communications, security, and signal processing. She has published more than 100 papers and authored the book "VLSI Architectures for Modern Error-Correcting Codes" (CRC Press, 2015). She is a recipient of the NSF CAREER Award, in 2009 and the Best Paper Award with 2004 ACM Great Lakes Symposium on VLSI. She was elected to serve on the BoG (2019–2021) of the IEEE CASS. She is also the Chair (2021–2022) of the Data Storage Technical Committee. She served on the committees of many conferences, including ISCAS, SiPS, ICC, and GLOBECOM.

Yingjie Lao (Senior Member, IEEE) received the BS degree from Zhejiang University, China, in 2009, and the PhD degree from the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, in 2015. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Clemson University. He is the recipient of an NSF CAREER Award, a Best Paper Award with the International Symposium on Low Power Electronics and Design (ISLPED), and an IEEE Circuits and Systems Society Very Large Scale Integration Systems Best Paper Award.

Keshab K. Parhi (Fellow, IEEE) received the PhD degree in EECS from the University of California, Berkeley, in 1988. He is a Distinguished McKnight University professor and Erwin A. Kelen chair professor with the Department of Electrical and Computer Engineering. He has published more than 700 papers, is the inventor of 34 patents, and has authored the textbook VLSI Digital Signal Processing Systems (Wiley, 1999). His current research addresses VLSI architectures for machine learning, hardware security, and data-driven neuroscience. He is the recipient of numerous awards including the 2003 IEEE Kiyo Tomiyasu Technical Field Award, and the 2017 Mac Van Valkenburg award and the 2012 Charles A. Desoer Technical Achievement award from the IEEE Circuits and Systems Society. He served as the editor-in-chief of the *IEEE Trans. Circuits and Systems, Part-I: Regular Papers* during 2004 and 2005. He is a fellow of the ACM, AIMBE, AAAS, and NAI.