High-Speed FPGA Implementation of the SIKE Based on An Ultra-Low-Latency Modular Multiplier

Jing Tian, Bo Wu, and Zhongfeng Wang, *Fellow, IEEE*

***Abstract*—The supersingular isogeny key encapsulation (SIKE) protocol, as one of the post-quantum protocol candidates, is widely regarded as the best alternative for curve-based cryptography. However, the long latency, caused by the serial large-degree isogeny computation which is dominated by modular multiplications, has made it hard for practical applications. In this paper, we present a fast FPGA implementation for the SIKE by incorporating algorithmic transformations and architectural optimizations. Firstly, we introduce a novel data representation, which can facilitate faster and higher-parallel field arithmetic computing than prior arts. Secondly, an extremely low-latency modular multiplier is devised based on the new algorithm by fully parallelizing and highly optimizing the small-size multipliers and reduction modules. Thirdly, a compact control logic is developed based on the benchmark provided in the newest SIKE library, well fitting our arithmetic logic unit (ALU). Finally, we code the proposed architectures using the Verilog language and integrate them into the SIKE library. The implementation results on a Xilinx Virtex-7 FPGA show that for the SIKEp751, our design only costs 13.2 ms with a frequency of 138.9 MHz, about 2x faster than the state-of-the-art. Particularly, the modular multiplier merely needs 16 clock cycles, reducing the delay by nearly one order of magnitude with a small factor of increase in hardware resource.**

***Index Terms*—Modular multiplication, supersingular isogeny key encapsulation (SIKE), elliptic curve cryptography (ECC), post-quantum cryptography (PQC), hardware implementation, FPGA.**

I. Introduction

In recent years, much progress has been made in quantum computers [1], [2]. Many cryptography systems are threatened by quantum computers. Notably, the commonly used public-key cryptographic algorithms like the Rivest-Shamir-Adleman (RSA) [3] and elliptic curve cryptography (ECC) [4] which are protected by the difficulty to factor extremely large integers and to perform elliptic curve discrete logarithms, respectively, could be easily solved by using the Shor's algorithm [5] with a powerful quantum computer. Although it is unclear when such computers will be invented, these achievements have indeed promoted the development of post-quantum cryptography (PQC) which are resistant to classical and quantum computers' attacks. From 2017, the National Insititute of Standards and Technology (NIST) [6] has hosted three rounds of PQC standardization process and the supersingular isogeny key encapsulation (SIKE) protocol [7] still exists in the latest announced candidates.

The first two authors contributed equally to this work. (Corresponding authors: Zhongfeng Wang.) The authors all are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210023, China (E-mail: tianjing@nju.edu.cn, qaqwubo@foxmail.com, zfwang@nju.edu.cn)

The SIKE is developed from the Supersingular Isogeny Diffie-Hellman(SIDH) key exchange protocol. In 2011, Jao and De Feo proposed an isogeny key exchange algorithm, the SIDH, based on a supersingular elliptic curve to resist the quantum attack based on the difficulty to find isogenies between supersingular elliptic curves [8]. This protocol has the characteristics of ECC's public key and secret key with small sizes and the advantages of perfect forward secrecy. However, the high computational complexity forms the bottleneck in practical applications. Usually, large-degree isogeny computations are needed to meet the security requirement. The long latency is caused by the considerable and serial field computations. Additionally, several works have recently reported that the SIDH is threatened by some side-channel attacks [9]–[11]. As an improved version of the SIDH, the SIKE protocol is proposed to provide reliable security not only in the post-quantum era but also in the current environment. Similarly, the SIKE also suffers from large computational complexity.

In order to alleviate this problem, many researches have been done to speed up the SIDH/SIKE protocol based on software platform [12]–[21] or hardware platform [22]–[29]. On the software side, the first version of software implementation for SIDH was done by Jao using the GMP library in 2011 [12]. The latest version provided in [18] is recognized as the fastest software implementation. On the hardware side, many improvements have been made based on FPGA. Koziel *et al.* have done much progress for the SIDH and SIKE protocols using the high-radix Montgomery multiplication algorithm [30]. In 2016, they proposed the first architecture of the SIDH protocol in [24]. In 2020, their latest version in [29] dropped the time of SIKE protocol from 33.4 ms to 25.5 ms over the NIST security level 5.

It should be noted that in general cases, the Montgomery reduction algorithm [31] has better performance than others and its variants are widely used in SIDH/SIKE protocol. In fact, the special form of the supersingular elliptic curve can be utilized for accelerating the modular multiplication. The first of such work was proposed by Karmarkar *et al.* in [32], in which an efficient modular multiplication (EFFM) algorithm using an unconventional radix is presented with the modulus form of $p = 2 \cdot 2^{e_A} b^{e_B} - 1$ where e_A and e_B are even integers. Based on the EFFM, many studies [22], [33]–[37] have been made to extend the limitation of the form and simplify the field multiplication algorithm. In our latest work [37], the limitation for the prime form is removed and a fast modular multiplication algorithm is provided based on a new data representation. Especially, for the parameters of the SIKE provided in the documentation [18], the proposed modular multiplication performs faster than the state-of-the-art. Similar conclusions are also drawn in other field arithmetic operations. Notably, these new algorithms are very easily to be processed in parallel over an FPGA platform.

In this paper, we firstly review and conclude the promising field arithmetic algorithms based on the new data representation. Then, according to these algorithms, we propose an ultra-low-latency field arithmetic logic unit (FALU), including a modular multiplier, a modular adder, a modular subtractor, and an inverse domain conversion module. Especially the modular multiplier is greatly improved compared to the previous works, by fully parallelizing and highly optimizing the small-size multipliers and reduction modules. It should be noted that those three modular arithmetic modules all are feed-forward architectures and can be extensively pipelined, which greatly simplifies the control logic. As the data conversion happens only a few times, the conversion module is designed in an iteratively computing way to save the area. Additionally, a compact control logic is developed based on the FPGA implementation provided in the newest SIKE library [18], well fitting the proposed arithmetic logic unit (ALU). We code the new ALU in Verilog language, generate the consistent instruction for the control logic in MATLAB script, and integrate them into the SIKE library. Then, a complete constant-time SIKE design is obtained. The correctness is verified by using the original testbench over the Xilinx Vivado 2018.2 EDA platform. The implementation results on a Xilinx Virtex-7 FPGA show that for the SIKEp751, our SIKE design only needs 13.2 ms with a frequency of 138.9 MHz, about 2x faster than the state-of-the-art. Particularly, the modular multiplier merely costs 16 clock cycles, reducing the latency by close to one order of magnitude in that case. When being implemented on a Kintex UltraScale+, the new SIKE only costs 8.9 ms at a frequency of 200.0 MHz.

The rest of the paper is organized as follows. Section II firstly gives a brief introduction of SIDH protocol and SIKE protocol. Subsequently, the basic field arithmetic operations are summarized based on a novel data representation, where the modular addition, modular subtraction, modular multiplication, and inverse domain conversion algorithms are detailed. Section III shows the proposed hardware architectures for those algorithms. The used top-level architecture and the modified instruction scheduling are presented in Section IV. In Section V, the FPGA implementation results are provided and compared with previous works. Finally, Section VI concludes this paper.

II. Preliminaries

In this section, we will firstly review the SIDH and SIKE protocols, and then detail the basic arithmetic operations over field $\mathbb{F}_p$ based on a novel data representation.

A. Supersingular Isogeny Diffe-Hellman

The supersingular isogeny Diffie-Hellman (SIDH) key-exchange [8] is designed for two parties (saying, Alice and Bob), who want to communicate with each other secretly over a public communication environment. Alice and Bob both obtain their shared secret key by using their own secret key and the other party's public key. The shared secret key is the j-invariant of two isomorphic supersingular elliptic curves generated based on a public supersingular elliptic curve E. Such curve is usually set as the Montgomery curve with the form of $E/\mathbb{F}_{p^2} : Dy^2 = x^3 + Cx^2 + x$, where $C, D \in \mathbb{F}_{p^2}$, $D(C^2 - 4) \neq 0$, and the prime $p = f \cdot a^{e_A} b^{e_B} \pm 1$.

Algorithm 1: The SIDH key-exchange protocol [8].

```
Input: Public parameters: $E/\mathbb{F}_{p^2}$, $P_A$, $Q_A$, $P_B$, and $Q_B$.
1: Key Generation: Bob generates his secret key and
   public key and sends the public key to Alice.
   $sk_B = random\{0, 1, ..., 2^{\lfloor \log_2 3^{e_B} \rfloor} - 1\}$
   $pk_B = isogen_B(sk_B)$
2: Encryption: Alice encrypts the plaintext with the shared
   secret key and sends her public key and the ciphertext to
   Bob.
   $sk_A = random\{0, 1, ..., 2^{e_A} - 1\}$
   $pk_A = isogen_A(sk_A)$
   $j = isoex_B(pk_B, sk_A)$
   $ss = H(j, M)$
   $c_A = ss \oplus m_A$, where $m_A \in \{0, 1\}^M$
3: Decryption: Bob decrypts the ciphertext with the shared
   secret key.
   $j = isoex_B(pk_A, sk_B)$
   $ss = H(j, M)$
   $m_A = ss \oplus c_A$
Output: Bob's received message $m_A$.
```

The main process of the SIDH is shown in Alg. 1. Assume that Bob receives messages from Alice. $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ are two independent points on the public curve $E/\mathbb{F}_{p^2}$, and satisfy $< P_A, Q_A >= E[a^{e_A}]$ and $< P_B, Q_B >= E[a^{e_B}]$. Firstly, Bob generates his secret key and public key with the corresponding parameters. Bob's secret key sk_B is chosen from the keyspace $\{0, 1, ..., 2^{\lfloor \log_2 3^{e_B} \rfloor} - 1\}$ and his public key is gotten by using the $isogen_B$ function which can be referred to in the documentation of [18]. And then Bob sends his public key to Alice. Secondly, Alice generates her secret key and public key in the same way as Bob. With her own secret key sk_A and Bob's public key pk_B, she can calculate their shared secret key j by function $isoex_A$. Assuming the dimension of the plaintext space is M, the j-invariant is encrypted by the Hash function with a bit width of M. The plaintext m_A is encrypted as c_A by using the output of the Hash function. Finally, to decrypt the c_A, Bob gets their shared secret key j with the $isoex_B$ function. After encrypting j by the Hash function, he obtains the plaintext m_A by using ss.

B. Supersingular Isogeny Key Encapsulation

We know that the SIDH can defend from the quantum computer's attack. However, this protocol is proved unable to resist some side-channel attacks [9]–[11]. The SIKE is just proposed to make up this flaw, by using the encapsulation mechanism.

Algorithm 2: The SIKE protocol [18].

```
Input: Public parameters: $E/\mathbb{F}_{p^2}$, $P_A$, $Q_A$, $P_B$, and $Q_B$.
1: Key Generation: Bob generates his secret key and public key and sends the public key to Alice.
   $sk_B = random\{0, 1, ..., 2^{\lfloor \log_2 3^{e_B} \rfloor} - 1\}$
   $pk_B = isogen_B(sk_B)$
2: Encapsulation: Alice encrypts her plaintext as $c_A$ and $em$. And $em$ becomes an another shared secret key.
   $sk_A = H(\{m_A, pk_B\}, e_A)$
   $pk_A = isogen_A(sk_A)$
   $j = isoex_A(pk_B, sk_A)$
   $ss = H(j, M)$
   $c_A = ss \oplus m_A$, where $m_A \in \{0, 1\}^M$
   $em = H(\{m_A, pk_A, c_A\}, K)$
3: Decapsulation: Bob decrypts the ciphertext and judges whether the message is $em$ or $em'$.
   $j = isoex_B(pk_A, sk_B)$
   $ss = H(j, M)$
   $m'_A = ss \oplus c_A$
   $sk'_A = H(\{m'_A, pk_B\}, e_A)$
   $pk'_A = isogen_B(sk'_A)$
   $fm_B \in \{0, 1\}^M$
   $em_A = \begin{cases} H(\{m'_A, pk_A, c_A\}, K) \ (pk'_A = pk_A) \\ H(\{fm_B, pk_A, c_A\}, K) \ (pk'_A \neq pk_A) \end{cases}$
Output: Bob's calculated message $em_A$.
```

Similar to the SIDH, we divide the SIKE protocol into three steps: key generation, encapsulation, and decapsulation, shown in Alg. 2. In the first step, Bob generates his secret key and public key by using the function $isogen_B$. Then he sends out the public key pk_B to Alice. In the encapsulation step, Alice gets her secret key sk_A by hashing her plaintext cascaded with Bob's public key. Her public key and shared secret key are computed in the same way as those in the SIDH. Then she sends to Bob her public key and her ciphertexts c_A. At the same time, she computes a new shared secret key as em with a bit width of K which corresponds to the number of bits of classical security. In the decapsulation step, Bob computes the original shared secret key and then calculates the plaintext m'_A. With the computed plaintext and his public key, Bob can recover Alice's secret key and public key. Meanwhile, he generates a random fake message of fm_B. Finally, he chooses the output by judging whether pk'_A is equal to pk_A.

In the SIKE library, four sets of parameters have been provided, namely, SIKEp434, SIKEp503, SIKEp610, and SIKEp751. The corresponding NIST security levels are 1 (AES128), 2 (SHA256), 3 (AES192), and 5 (AES256), respectively, which are the newest judgments corrected by Costello *et al.* in [38]. All of those primes have the form of $p = 2^{e_A}3^{e_B} - 1$, which is considered in our field arithmetic computing in this paper.

C. Field Arithmetic Operations for SIKE Based on a New Data Representation

By breaking down the computations of the SIKE protocol, we can find that the five large-degree isogeny operations, including the $isogen$ and $isoex$ functions, dominate the total computation. According to the Vélu's formula [39], in practical computing, a large-degree isogeny is required to be divided into many data-dependent small-degree isogenies which are made up of finite-field arithmetic operations. A supersingular isogeny elliptic curve is usually considered over a quadratic finite field $\mathbb{F}_{p^2}$. The arithmetic operations over $\mathbb{F}_{p^2}$ can be decomposed into operations over $\mathbb{F}_p$, including the modular addition, modular subtraction, modular negation, modular multiplication, modular division, and modular inversion.

As introduced in the fourth paragraph of Introduction Section, our previous work [37] presents very efficient field arithmetic algorithms for the SIKE over a new data representation. The key idea of that work is to replace a large modulus p with a small modulus R, which can benefit the modular multiplication a lot. The prime p of the SIKE is rewritten as:

$$\begin{aligned} p &= 2^{e_A}3^{e_B} - 1 \\ &= 2^{-\alpha}3^{-\beta}2^{e_A+\alpha}3^{e_B+\beta} - 1 \\ &= f' \cdot R^n - 1, \end{aligned} \tag{1}$$

where $f' = 2^{-\alpha}3^{-\beta}$, $R = 2^{\frac{e_A+\alpha}{n}}3^{\frac{e_B+\beta}{n}}$, and the parameters α and β are used to make $e^A + \alpha$ and $e_B + \beta$ divisible by a larger n. With this form, a field number, saying $A \in \mathbb{F}_p$, can be directly represented as:

$$A = \sum_{i=0}^{n-1} a^i \cdot R^i, \tag{2}$$

where $a_i \in [0, R-1]$ for $0 \leq i < n-1$ and $a_{n-1} \in [0, f'R-1]$.

Generally, the basic field arithmetic operations can be completed by three operations, namely, the modular addition, subtraction, and multiplication. Note that the first two operations are much simpler than the last one. Therefore, Optimizing the modular multiplication can do much favor to the efficiency of the SIKE protocol. We will detail the modular addition, subtraction, and multiplication operations based on the new data representation in the following. The inverse domain conversion algorithm named U2N in [37] will also be presented.

1) Modular Addition: Consider two field operands $A, B \in \mathbb{F}_p$ represented in Eq. (2). The modular addition computing is split into two steps shown in Alg. 3. In the first step, the coefficients a_i and b_i for $0 \leq i < n$ are added as $c_i = a_i + b_i$. In this way, one N-bit addition is converted into n w-bit additions, where N is the bit width of p, w is the bit width of R, and $w = \lceil N/n \rceil$. Since there are no carries in the adjacent terms, the n additions can be computed in parallel, which can reduce the critical path and improve the parallelism in hardware. The second step is to make the coefficients c_i in the standard range as shown in Steps 4-13. If c_i for $0 \leq i < n-1$ are larger than $R-1$, they will be reduced by R and c_{i+1} will be added by one. If $a_{n-1} + b_{n-1} > f' \cdot R - 1$, c_{n-1} will be reduced by $f' \cdot R$ and c_0 is added by one. It should be noted that a lazy reduction is used for c_0 to simplify the reduction, where c_0 ranges in $[0, R]$.

Algorithm 3: Modular Addition.

```
Input: $A = \sum_{i=0}^{n-1} a_i \cdot R^i$, $B = \sum_{i=0}^{n-1} b_i \cdot R^i$, where
    $a_i, b_i \in [0, R-1]$ for $i = 0, ..., n-2$ and
    $a_{n-1}, b_{n-1} \in [0, f' \cdot R - 1]$.
1: The first step:
2: $C = A + B = \sum_{i=0}^{n-1} (a_i + b_i) \cdot R^i = \sum_{i=0}^{n-1} c_i \cdot R^i$
3: The second step:
4: for $i = 0 \to n-2$ do
5:     if $c_i \geq R$ then
6:         $c_i = c_i - R$
7:         $c_{i+1} = c_{i+1} + 1$
8:     end if
9: end for
10: if $c_{n-1} \geq f' \cdot R$ then
11:     $c_{n-1} = c_{n-1} - f' \cdot R$
12:     $c_0 = c_0 + 1$
13: end if
Output: $C = \sum_{i=0}^{n-1} c_i \cdot R^i \equiv A + B \bmod p$
```

2) Modular Subtraction: Similar to the modular addition, the modular subtraction is also split into two steps. The first step directly uses n w-bit subtractions for coefficients. The second step is to make $c_0, ..., c_{n-1}$ lie in the right ranges. The detail for the modular subtraction is showed in Alg. 4, where the output c_0 is also applied with the lazy reduction.

Algorithm 4: Modular Subtraction.

```
Input: $A = \sum_{i=0}^{n-1} a_i \cdot R^i$, $B = \sum_{i=0}^{n-1} b_i \cdot R^i$,$a_i, b_i \in [0, R-1]$
    where $i = 0, ..., n-2$ and $a_{n-1}, b_{n-1} \in [0, f' \cdot R - 1]$.
1: The first step:
2: $C = A - B = \sum_{i=0}^{n-1} (a_i - b_i) \cdot R^i = \sum_{i=0}^{n-1} c_i \cdot R^i$
3: The second step:
4: if $c_0 \leq 0$ then
5:     $c_0 = c_0 + R$
6:     $c_1 = c_1 - 1$
7: end if
8: for $i = 1 \to n-2$ do
9:     if $c_i < 0$ then
10:        $c_i = c_i + R$
11:        $c_{i+1} = c_{i+1} - 1$
12:    end if
13: end for
14: if $c_{n-1} < 0$ then
15:     $c_{n-1} = c_{n-1} + f' \cdot R$
16:     $c_0 = c_0 - 1$
17: end if
Output: $C = \sum_{i=0}^{n-1} c_i \cdot R^i \equiv A - B \bmod p$
```

3) Modular Multiplication: Assume two field operands $A, B \in \mathbb{F}_p$ represented in Eq. (2). According to [37], the modular multiplication can be computed as:

$$\begin{aligned} C &\equiv A \times B = \sum_{i=0}^{n-1} a_i \cdot R^i \times \sum_{i=0}^{n-1} b_i \cdot R^i \qquad (3) \\ &\equiv (\sum_{j=0}^{n-1} a_j b_{n-j-1}) \cdot R^{n-1} + \\ &\quad \sum_{i=0}^{n-2} (\sum_{j=0}^{i} a_j b_{i-j} + \sum_{j=i+1}^{n-1} a_j b_{i-j+n} \cdot 2^\alpha 3^\beta) \cdot R^i \\ &\equiv \sum_{i=0}^{n-1} c_i \cdot R^i \bmod p, \end{aligned}$$

where $c_j \in [0, R-1]$ for $0 < j < n-1$, $c_0 \in [0, R]$, and $c_{n-1} \in [0, f'R-1]$. the modular multiplication is also divided into two parts: the integer multiplication part and the reduction part. In the equation, the second step is defined as the integer multiplication part and the last step is the standard output after the reduction part.

In the integer multiplication part, the original $N \times N$ multiplication is replaced by n^2 $a_i \cdot b_j$ where $i, j \in 0, ..., n-1$. Since there are no carries in the adjacent orders, the n^2 small multiplications can be computed in parallel and the one-level Karatsuba-like optimization can be easily applied to the coefficient multiplication combinations, both of which are very friendly in fast hardware implementation.

The reduction part is to make the raw coefficients into standard ranges. According to [37], $n+1$ improved Barrett reduction (IBR) functions are needed with a modulus of R. The IBR function is shown in Alg. 5, The parameter γ is an arbitrary integer to make the range of c covered and usually satisfies $\gamma \ll w$ in the adopted $n+1$ IBR functions. The quotient q and remainder r can be obtained with about 1.75 $w \times w$ multiplications. Since the quotient is added to the next order, after using the IBR function, the updated coefficients are finally reduced by using some additions and subtractions, which can be referred to as the second step of the modular addition.

Algorithm 5: The improved Barrett reduction (IBR).

```
Input: An coefficient $c \in [0, 2^{2w+\gamma})$; the modulus
    $R = 2^{\frac{e_A+\alpha}{n}} 3^{\frac{e_B+\beta}{n}}$, where $w_1 = \frac{e_A+\alpha}{n}$,
    $w_2 = \lceil \log_2(3^{\frac{e_B+\beta}{n}}) \rceil$, $w_1 + w_2 = w$, $R' = 3^{\frac{e_B+\beta}{n}}$;
    the pre-computed constant $\lambda = \lfloor 2^{2w+\gamma+1}/R \rfloor$.
1: $t = \lfloor c/2^{w_1} \rfloor$, $s = c \bmod 2^{w_1}$
2: $q = \lfloor \frac{\lfloor \frac{t}{2^{w_2-2}} \rfloor \cdot \lambda}{2^{w+\gamma+3}} \rfloor$
3: $t_1 = (q \bmod 2^{w_2+1}) \cdot R'$
4: $r = ((t \bmod 2^{w_2+1}) - (t_1 \bmod 2^{w_2+1})) \bmod 2^{w_2+1}$
5: if $r \geq R'$ then
6:     $r = r - R'$, $q = q + 1$
7: end if
8: $r = r \cdot 2^{w_1} + s$
Output: The quotient $q = \lfloor c/R \rfloor$, and the remainder
    $r = c \bmod R$.
```

4) Inverse Domain Conversion Algorithm: Since in the hardware design, the input data are transformed into the new representation in advance and the coefficients are saved in the storage, there is no need to design a forward converter. Therefore, we only provide the inverse domain conversion algorithm here, also named U2N as shown in Alg. 6. The number $A = \sum_{i=0}^{n-1} a_i \cdot R^i$ with the standard ranges (except a_0 which has a lazy reduction) is the output from a field algorithm aforementioned. In the U2N algorithm, recursive multiplication and addition operations from higher orders to lower orders are adopted to calculate the output. The final result is adjusted in terms of the used lazy reduction.

Algorithm 6: From unconventional radix back to normal (U2N).

```
Input: An operand $A = \sum_{j=0}^{n-1} a_j \cdot R^j$, the radix $R$, and the
    modulus $p = f'R^n - 1$.
1: $C = a_{n-1}$
2: for $j \leftarrow n-2$ to $0$ do
3:     $C \leftarrow C \cdot R + a_j$
4: end for
5: If $C = p$, set $C$ to 0.
6: If $C = p + 1$, set $C$ to 1.
Output: The result $C \in \mathbb{F}_p = A \bmod p$.
```

III. Proposed Field Arithmetic Logic Unit

The proposed field arithmetic logic unit (FALU) is shown in Fig. 1, including four submodules: Inverse Domain Conversion Module (IDCM), Modular Subtractor (MS), Modular Adder (MA), and Modular Multiplier (MM). A multiplexer in the right is to select the output from one of the four submodules in different conditions controlled by the signal sel_strl. More details about these submodules are shown below.

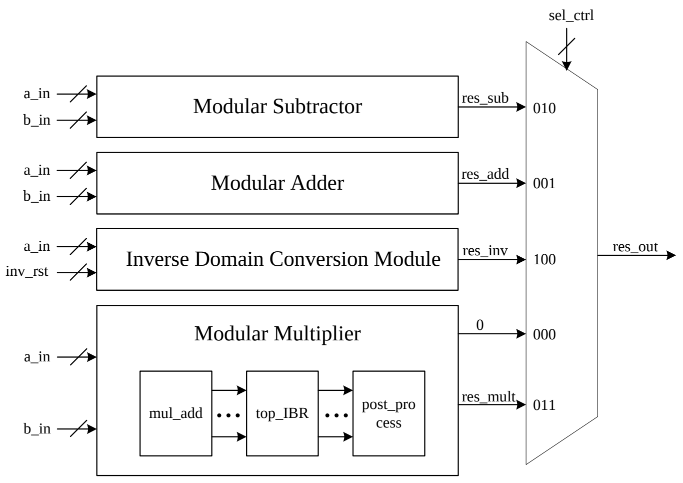


Fig. 1. The proposed *FALU* architecture.

A. Inverse Domain Conversion Module

The IDCM is to convert the data from the new format back to the original format (*i.e.*, the field elements), which is usually used for a Hash input after an isogeny computation. We have carefully analyzed the algorithms of the modular addition, subtraction, and multiplication and found that the case where the input number A of Alg. 6 equals $p+1$ would never happen though a_0 might equal R. Therefore, we only need to consider the situation of $A = p$ in the IDCM.

Algorithm 7: The inverse domain conversion algorithm (IDCA) for hardware efficiency.

```
Input: An operand $A = \sum_{i=0}^{n-1} a_i \cdot R^i$, where $0 \le a_0 \le R$,
    $0 \le a_1, ..., a_{n-2} < R$, $0 \le a_{n-1} < f' \cdot R$, where
    $R = 2^{\frac{e_A+\alpha}{n}} 3^{\frac{e_B+\beta}{n}}$; $w = \log_2 R$
1: $S_1 = a_{n-1}$, $f_1 = \begin{cases} 1, & (a_{n-1} = f'R - 1) \\ 0, & (a_{n-1} \neq f'R - 1) \end{cases}$
2: for $i = 1 \to n-1$ do
3:     for $j = 1 \to i$ do
4:         if $j == 1$ then
5:             $t = a_{n-1-i}$
6:         else
7:             $t = D$
8:         end if
9:         $P = S_j \times R + t$, $S_j = P \bmod 2^w$, $D = P \,\%\, 2^w$
10:        if $j == i$ then
11:            $S_{j+1} = D$
12:        end if
13:    end for
14:    $f_2 = \begin{cases} 1, & (a_{n-1-i} = R - 1) \\ 0, & (a_{n-1-i} \neq R - 1) \end{cases}$, $f_1 = f_1 \& f_2$
15: end for
16: $C = \begin{cases} \{S_n, S_{n-1}, ..., S_1\}, & (f_1 = 0) \\ 0, & (f_1 = 1) \end{cases}$
Output: Field element $C \in \mathbb{F}_p$.
```

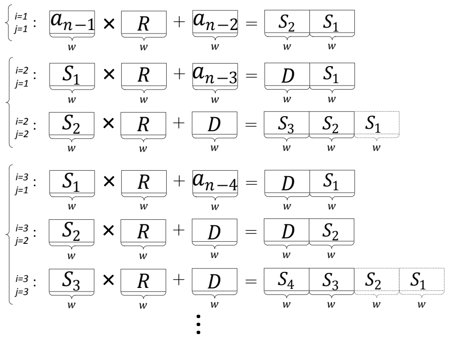

i=1 j=1 : a_{n-1} × R + a_{n-2} = S_2 S_1
i=2 j=1 : S_1 × R + a_{n-3} = D S_1
i=2 j=2 : S_2 × R + D = S_3 S_2 S_1
i=3 j=1 : S_1 × R + a_{n-4} = D S_1
i=3 j=2 : S_2 × R + D = D S_2
i=3 j=3 : S_3 × R + D = S_4 S_3 S_2 S_1
w

Fig. 2. A step-by-step illustration for the IDCA.

In fact, the original U2N algorithm is unfriendly to hardware design because the data width is increasing when iteratively computing. Therefore, we propose a new conversion algorithm called IDCA for hardware efficiency, as shown in Alg. 7. The key idea of this algorithm is to save the lower bits in memory S_j dynamically and only use the higher bits to compute in each iteration. Two loops are adopted to make the multiplication and addition smaller. The outer loop is to skip to the lower-order coefficients and the inner loop is to refine the multiplication and addition in each order. To make it clearer, a step-by-step illustration for the IDCA is shown in Fig. 2. We can see that this algorithm costs $\frac{n(n-1)}{2}$ $w \times w$ multiplications and $2w+w$ additions, which can greatly reduce the hardware resource. It should be noted that since the upper bound of a_{n-1} is the smallest among the coefficients and the output C is smaller than 2^{nw}, the data width of these variables is ensured no bigger than w. Meanwhile, the flag signals f_1 and f_2 are used to record whether the result equals the modulus p or not. If yes, the output would be set to zero.

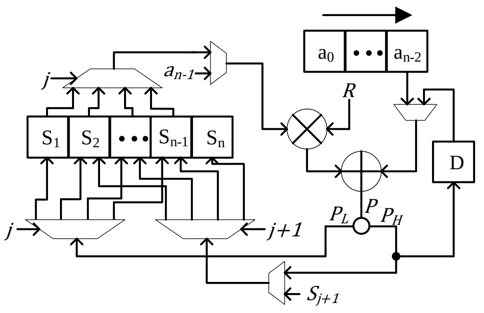


Fig. 3. The proposed *IDCM* architecture.

The corresponding hardware architecture of the main body is shown in Fig. 3, where the judgment for equal to p is omitted for brevity. The input coefficients $a_0, ..., a_{n-2}$, intermediate variables $S_0, ..., S_n$ and D, are saved in $2n$ w-bit registers. One $w \times w$ multiplier and one $2w + w$ adder are used. In each outer iteration, the register group of the input coefficients shifts w bits to the right. The registers for the intermediate variables are updated in each cycle. At the beginning, the input coefficient a_{n-1} is selected and put into the multiplier. After added by a_{n-2}, the $2w$-bit sum P is divided into two parts: w-bit P_L and w-bit P_H. P_L is saved back to S_j with a demux. P_H is saved back to S_{j+1} with an another demux only when $j = i$. Meanwhile, P_H is also sent to register D. In the following iterations, the input of the multiplier is selected from the registers $S_0, ..., S_n$ and that of the adder is selected from the register D and the rightmost register of the coefficients. After $\frac{n(n-1)}{2}$ iterations, the output is obtained as $\{S_n, S_{n-1}, ..., S_1\}$. When k levels of the pipeline are inserted in the iteration, this module needs $\frac{n(n-1)k}{2}$ clock cycles. In our following implementation, k is set to 1.

B. Modular Adder

The architecture for the modular addition of Alg. 3 is shown in Fig. 4. The adders for the coefficients are directly computed in parallel. According to the reduction step of Alg. 3, serial computations are needed, which would cause long latency. In order to deal with this problem, we have investigated this algorithm carefully and found that the candidate data can be computed in advance and selected by some control logic. As shown in Fig. 4, the sums are subtracted by R or $f'R$ in parallel and the leftmost column of multiplexers are used to select the remainders $r_0, ..., r_{n-2} \in [0, R-1]$, $r_{n-1} \in [0, f'R-1]$, and the quotients $q_0, ..., q_{n-1} \in \{0, 1\}$, controlled by the sign bit of the subtractors' outputs. It should be noted that the multiplexers of the remainders and quotients here are combined together for brevity. The following step is to deal with the quotients. Since the quotients are whether 0 or 1, the final outputs of c_i for $0 < i < n$ are equal to r_i, $r_i + 1$, or 0. And that for c_0 only equals r_0 or $r_0 + 1$ because of the lazy reduction. We prepare those candidates in parallel and only update the 1-bit quotients based on their dependency, which can largely reduce the latency. For example, considering the SIKEp751 and $n = 12$, only one pipeline is used to reach a frequency of about 200 MHz.

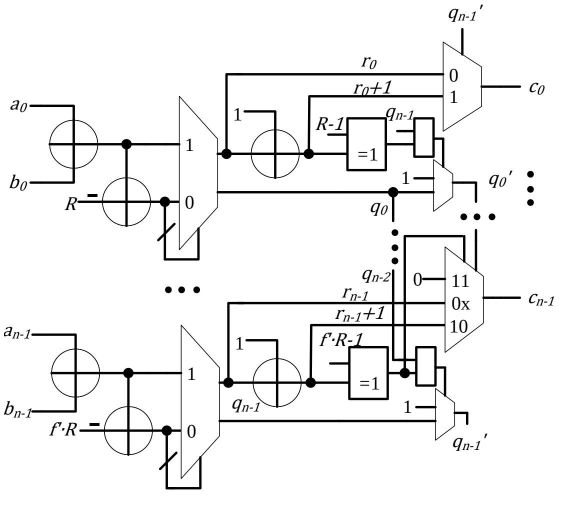


Fig. 4. The proposed *modular adder* architecture.

C. Modular Subtractor

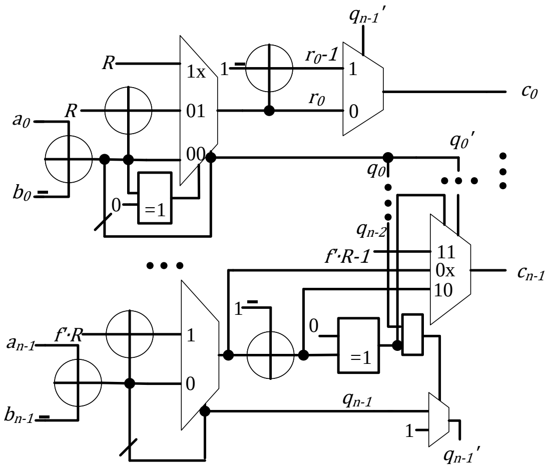


Fig. 5. The proposed *modular subtractor* architecture.

Similar to the modular adder, the modular subtractor is also devised with a high degree of parallelism, as shown in Fig. 5. After the subtractors for the coefficients, the differences are added by R or $f'R$ in parallel. The leftmost column of multiplexers are used to select the remainders $r_0 \in [0, R]$, $r_1, ..., r_{n-2} \in [0, R-1]$, and $r_{n-1} \in [0, f'R-1]$, controlled by the sign bit of the subtractors' outputs. The quotients

$q_0, ..., q_{n-1} \in \{0, 1\}$ are directly set as these sign bits. The candidates of c_i for $0 < i < n-1$ are r_i, $r_i - 1$, and $R-1$, those of c_{n-1} are r_{n-1}, $r_{n-1}-1$, and $f'R-1$, and those of c_0 only are r_0 or $r_0 - 1$ because of the lazy reduction. The candidates are computed in parallel and the final 1-bit quotients are calculated in terms of the dependency between the adjacent orders. The output are selected from the candidates controlled by the updated quotients. Similarly, the experiment result for the SIKEp751 shows that this architecture can achieve nearly 200 MHz with only one stage of the pipeline.

D. Modular Multiplier

The modular multiplier is used to compute the product $C \equiv A \times B$, where $A, B, C \in \mathbb{F}_p$. Since the modular multiplication takes up a large proportion of the computations in SIKE protocol, accelerating this operation can efficiently speed up the entire protocol. As shown in Fig. 1, the proposed modular multiplier has three submodules: 1) *mul_add*; 2) *top_IBR*; and 3) *post-process*. They will be detailed in the following.

***mul_add*:** This module is used to calculate the integer multiplication as raw coefficients for the following reduction operations, as shown in Eq. (3). To reduce the latency, we devise binary-tree adders to compute those multiply-accumulation-like operations. This module can be divided into two parts: the coefficient multipliers and the accumulators. In the first part, by using the one-level Karatsuba-like optimization method, the coefficient multipliers are designed with n $a_i b_i$ $(0 \le i < n)$ and $\frac{n(n-1)}{2}$ $(a_i + b_j)(a_j + b_i)$ $(i \neq j)$ multipliers. We also use one or two levels of Karatsuba optimization [40] for the coefficient multipliers to reduce the resources. In the second part, those products are accumulated for the corresponding orders. In each order, two adder trees are used for the two accumulations in Step 2 of Eq. (3) since the second term has an extra small multiplication factor. The two terms are added as the final output after the second sum is shifted and added. We also use the carry-save adder [41] before sending it into the adder trees, to further reduce the critical path and resource. The critical path includes one coefficient multiplier, one shifter, one 1-bit full adder, and normal adders (the number equal to the maximum depth of the tree plus one). To achieve a high clock speed, several stages of the pipeline are needed, the number of which depends on the data width of the input and the depth of the tree. For the SIKEp751, three pipelines are good enough.

***top_IBR*:** This module is used to compute the main course of the reduction. Our goal is still to reduce the latency. Thanks to the complexity of IBR function strongly being correlated with the maximum input data width, we can design different sizes of IBR architectures to speed up the reduction process and meanwhile maintain a low complexity. Therefore, we propose a two-level reduction architecture with two sizes of IBR modules as shown in Fig. 6. The IBR architecture is referred to in our previous work [34]. The first level composed of n *IBR* modules is to reduce the raw coefficients $c'_0, ..., c'_{n-1}$. The outputs of IBR$_i$ are q'_i and r'_i. The size of those IBR modules is determined by the maximum of the raw coefficients. As presented in [37], the maximum data width

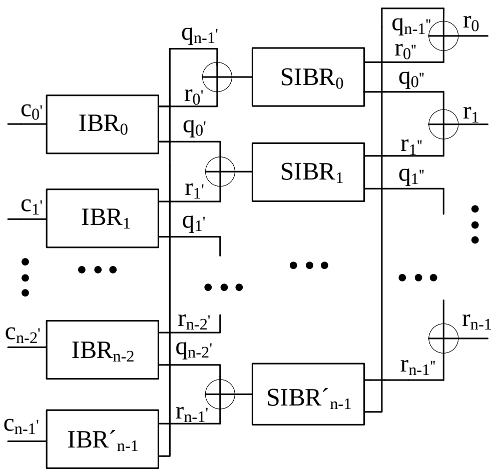


Fig. 6. The proposed *top_IBR* architecture.

equals about $\log_2\lceil((n-1)2^\alpha 3^\beta + 1)\rceil + 2 \cdot \log_2\lceil(R-1)\rceil$. Since r'_i is small than R or $f'R$, the data with of q'_i can reach about $\log_2\lceil((n-1)2^\alpha 3^\beta + 1)\rceil + \cdot \log_2\lceil(R-1)\rceil$. Though the first term is usually very small relative to the second term, it is not convenient to directly reduce those quotients with some subtractors yet. For instance, for the SIKEp751, if $n = 12$, $\alpha = 0$, and $\beta = 1$, the first and second terms will equal 6 and 63, respectively. Thus, we use n adders to compute $q'_{i-1} + r'_i$ for $0 < i < n$ and $q'_{n-1} + r'_0$, and then send their sums into the n small *IBR* (*SIBR*) modules as the second reduction level. The input data width is almost equal to the maximum size of the quotients. The outputs of SIBR$_i$ are q''_i and r''_i. Similarly, these quotients and remainders are added up respectively and output as $r_0, ..., r_{n-1}$, whose upper bounds are larger than R but far smaller than $2R$. They are reduced with some subtractors in the *post-process* module introduced in the following.

***post-process*:** This module is to meet the constraints $c_0 \in$

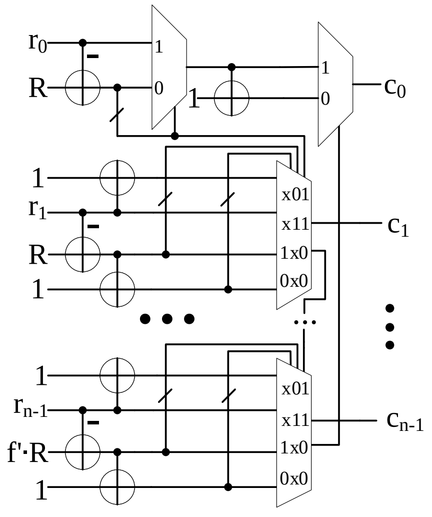


Fig. 7. The proposed *process* architecture.

$[0, R], c_1, ..., c_{n-2} \in [0, R-1]$ and $c_{n-1} \in [0, f \cdot R - 1]$. As the outputs of *top_IBR* $r_0, ..., r_{n-1}$ are in the standard ranges or a little bigger than the upper bound, only one R or $f' \cdot R$

should be subtracted to make them in the right range. If $R \leq r_i (i \in \{0, ..., n-2\})$, r_i will be need to be subtracted by R and r_{i+1} will be required to be added by 1. If $f' \cdot R \leq r_{n-1}$, r_{n-1} will be subtracted by $f' \cdot R$ and r_0 will be added by 1. Based on this analysis, the coefficients need to be processed one after another, which causes a long latency. Similar to the modular adder or subtractor introduced above, we propose a parallel architecture for this module as shown in Fig. 7. All the candidates are parallelly computed in advance and selected by the control logic made up of the sign bits. In this way, the critical path covers two adders and four multiplexers.

IV. Top-Level Architecture and Instruction Scheduling

A. Top-Level Architecture

According to the open VHDL source code provided in the SIKE library [18], the design coded by Koziel *et al.* is divided into five parts: the control unit, ROM, arithmetic logic unit (ALU), RAM, and interface logic. The state transition instruction is generated by other scripts and saved in the ROM for the control logic which is used to schedule the whole design. The ALU includes the Keccak Hash unit and the FALU. The former is referred to as the work of the Keccak team [42] and the latter is elaborately designed by Koziel *et al.* The dual-port RAM is used to store the intermediate data or to cache the output. The interface logic is to make the data standard for exchanging with the outside hardware. The interactive relationship of those five parts for the SIDH with 512-bit prime is illustrated in Fig. 4 of [23] by Koziel *et al.*

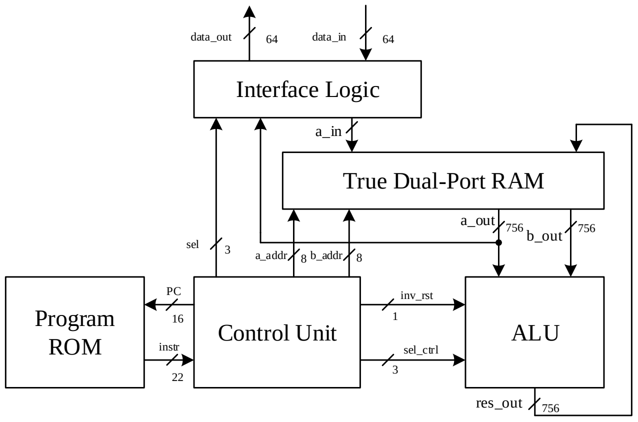


Fig. 8. The top-level architecture of the SIKE.

In this work, our focus is also on the FALU. As presented above, we have proposed a new FALU based on our field arithmetic algorithms to achieve lower latency. Figure 8 is the adopted top-level architecture of the SIKE for the SIKEp751, referred to as Fig. 4 in [23]. It is used to clearly show the data width of the new SIKE architecture. Apart from the transmitted data width between the RAM and the ALU or the interface, the main difference lies in the data widths between the control logic and the ALU and the ROM. Since the fully parallelizing scheme is adopted in the FALU, the control logic to regulate the ALU can be much simpler than previous works [23], [25], [28]. We can see that the bit width of the instruction is reduced from 26 bits to 22 bits and that of the control signals for the field arithmetic computing reduced from 12 bits to 4 bits.

The 22 bits of an instruction are allocated as follows. The first bit is used for starting the inversion unit. The second bit is the reset signal of IDCM. Bits 3-5 are used to select the results of FALU. As shown in Fig. 1, five combinations of the three bits correspond to the five outputs from the four arithmetic modules. Bit 6 indicates whether data should be written to address A of the dual-port RAM. Bits 7-14 are the address of the first port (A) of the RAM and bits 15-22 are the second port's address (B).

B. Instruction Scheduling

Good scheduling can increase the throughput of a design. We have analyzed some of the instructions saved in the ROM provided by the SIKE library and we found that the scheduling flow is fully optimized in serial. Maybe, we could design a new flow with a higher degree of parallelism, but that is not an easy thing. In this work, we still adopt the original flow to schedule the whole design.

As we can only access the instruction data saved in the ROM, we have devised a MATLAB script to transfer those data to adapt to our design. The correctness is verified with the provided testbench in VHDL using the Vivado 2018.2 platform. Take the operating flow of $A \times B$ in $\mathbb{F}_{p^2}$ as an example, where the SIKEp751 parameter is considered.

An element A in $\mathbb{F}_{p^2}$ is represented as $A = A_0 + A_1 i$, where $A_0, A_1 \in \mathbb{F}_p$. The optimized formula to compute $A \times B$ over $\mathbb{F}_{p^2}$ is:

$$\begin{aligned} A \times B &= (A_0 B_0 - A_1 B_1) + \\ &\quad ((A_0 + A_1)(B_1 + B_0) - A_0 B_0 - A_1 B_1) i. \end{aligned} \tag{4}$$

A step-by-step illustration for the operating flow of $A \times B$ over $\mathbb{F}_{p^2}$ is shown in Fig. 9, where the required clock cycles (CCs) in [18] and ours are also listed for a clear comparison. We can see that including the read-write CCs, our design only needs 4 CCs for the modular addition/subtraction and 18 CCs for the modular multiplication, while the design in [18] requires 9 CCs and 151 CCs, respectively. Eight steps are required to compute this operation, *i.e.*, sixteen instructions are needed to deploy the 8 groups of inputs and outputs. A detailed instruction scheduling comparison is shown in Table I. The notations, like "in1_add" and "out1_add", denote the instructions to permit an input or output of the adder/subtractor/multiplier in the corresponding cycles and "nop" means no input or output in that cycle. The omitted cycles all are "nops". Clearly, the latency in CCs of our design is reduced by more than 80% in this example compared with the previous work in [18].

V. Implementation Results and Comparison

To compare with conventional SIKE implementations, the proposed FALU architecture is coded with Verilog language and the required instructions are generated with our MATLAB script. We integrated them into the latest SIKE library [18] and implemented the new SIKE core with the Vivado 2018.2 platform for the SIKEp751 targeting at NIST security level 5.

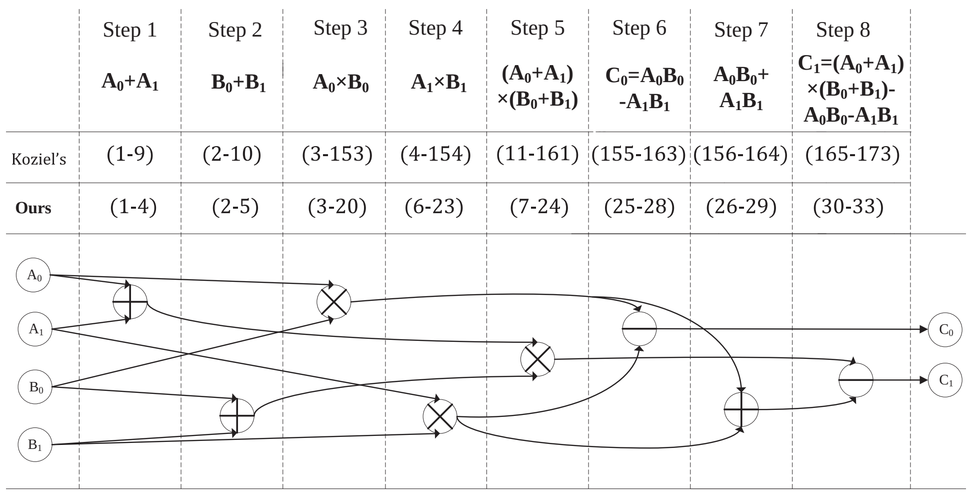


Fig. 9. A step-by-step illustration example for the operating flow of $A \times B$ over $\mathbb{F}_{p^2}$, where the required CCs for the SIKEp751 in [18] and ours are also listed for a clear comparison.

TABLE I
COMPARISON OF INSTRUCTION SCHEDULING BETWEEN KOZIEL'S AND OURS TO CALCULATE $A \times B$ OVER $\mathbb{F}_{p^2}$

Koziel's Instruction Scheduling										
cycle	1	2	3	4	...	9	10	11	...	153
instr	in1_add	in2_add	in3_mul	in4_mul	nop	out1_add	out2_add	in5_mul	nop	out3_mul
154	155	156	...	161	162	163	164	165	...	173
out4_mul	in6_sub	in7_add	nop	out5_mul	nop	out6_sub	out7_add	in8_sub	nop	out8_sub

Our Instruction Scheduling										
cycle	1	2	3	4	5	6	7	...	20	...
instr	in1_add	in2_add	in3_mul	out1_add	out2_add	in4_mul	in5_mul	nop	out3_mul	nop
23	24	25	26	27	28	29	30	31	32	33
out4_mul	out5_mul	in6_sub	in7_add	nop	out6_sub	out7_add	in8_sub	nop	nop	out8_sub

According to [37], the prime of this parameter is divided as $p = 2^{372}3^{239} - 1 = 1/3 \cdot (2^{31}3^{20})^{12} - 1$. The results after place-and-route are shown in the following.

Two advanced FPGA devices are considered in our implementation. The first one is a Xilinx Virtex-7 xc7vx690tffg1157-3 device, which is widely used in previous works [18], [23], [25], [28], [29], [43]. We adopt it in our design to make a fair comparison. The other one is a Xilinx Kintex UltraScale+ xcku13p-ffve900-3-e device, which is manufactured with more advanced technology (from 28nm to 16nm) and therefore owns better performance than the first one. This device is also adopted in [28]. During our design process, we have found that the net delay is the bottleneck for high clock speed. This device can help relieve this problem well.

TABLE II
LATENCY IN CCS OF THE PROPOSED ARITHMETIC MODULES OVER TWO XILINX FPGA DEVICES FOR SIKEP751

Device	MM	MA	MS	IDCM
Virtex-7 (No.1)	16	2	2	135
Kintex UltraScale+ (No.2)	12	2	2	135

Table II shows the required latency in CCs of the proposed arithmetic modules over the two Xilinx FPGA devices. We can see that except the MM, the other modules are designed with the same latency in the two devices. Note that the MM takes up the most portion of the whole design. Due to the high degree of parallelism, the wire congestion would be easily aroused in this module. The net delay gradually becomes the domination in the critical path. To deal with this problem, more registers or register trees are required to enhance the load capacity. When we consider the Virtex-7 device, 15 stages of the pipeline are inserted and the whole SIKE obtains a clock frequency of 138.9MHz. When we implement this design on the Kintex UltraScale+ device, it achieves a frequency of 200.0MHz. When we cut down four pipelines in the MM, the clock frequency is unchanged. Hence, the MM is designed with 11 stages of the pipeline for the second device to reduce the registers.

TABLE III
TIMING COMPARISONS OF THE MODULAR MULTIPLIERS OVER THE VIRTEX 7 FPGA FOR SIKEP751

Work	prime	# Mults	f_{clk} (MHz)	Latency / Interleaved (cc)	Normalized Latency/ Interleaved
[25]	p_{751}	8	193	148 / 101	9.25 / 101
[28]	p_{751}	8	167.4	100 / 69	6.25 / 69
[29]	p_{751}	8	294	138 / 90	8.63 / 90
[34]	p_{771}	1	60	18 / 1	1.13 / 1
this work	p_{751}	1	138.9	16 / 1	1.00 / 1

[1] The maximum throughput = $\frac{N \times f_{clk} \times \#mults}{interleaved\ latency}$.

Table III shows the timing comparisons between our MM and previous works over the first FPGA device. It can be seen that the proposed MM achieves the shortest latency and interleaved latency among the state-of-the-arts thanks to the fully interleaving scheme and the adopted optimized algorithm. Our design has two overwhelming advantages in the SIKE implementation. The first advantage is that for the dependent-data computations, our MM can greatly help accelerate such computations because of the ultra-low latency. The second advantage is that for the independent-data computations, we can only use one multiplier to satisfy any degrees of parallelism because the proposed MM can be utmostly interleaved with only one cycle, nearly two orders of magnitude compared to the designs used in SIKE implementations [25], [28], [29]. However, obtaining a high degree of parallelism in these previous works must parallelize the multipliers. For example, eight multipliers are used in those implementations as listed in the table. The design in [34] also has those two advantages but it cannot support the SIKEp751 parameter. Since the resources of this module are not available in the first three literatures, the area comparisons are not provided here. They can be implicitly shown in the following comparisons of the whole SIKE implementation results.

TABLE IV
ROUND COMPUTATIONS OF SIKEP751

	Alice R1	Bob R1	Alice R2	Bob R2	**Total**
Virtex-7[1]					
Latency (cc)	466,465	512,061	399,926	451,503	1,829,955
Time (ms)	3.36	3.69	2.88	3.25	13.18
Kintex UltraScale+[2]					
Latency (cc)	457,724	500,734	387,418	436,891	1,782,767
Time (ms)	2.29	2.50	1.94	2.18	8.91

[1] Sixteen CCs are needed in the MM and $f_{clk} = 138.9$ MHz.
[2] Twelve CCs are needed in the MM and $f_{clk} = 200.0$ MHz.

The round computations for the SIKEp751 over the two FPGA devices are shown in Table IV. The latency is computed through simulation, using the ending time minus the starting time and then dividing the interval time. Since the CCs used in the MM of the second device are less than those of the first one, the latencies in all rounds also have the same trend. The time is calculated by using the latency dividing the frequency. The superiority in frequency of the second device further reduces the required time in each round. Generally, more than 30% time is saved by using the second device.

The resource utilizations of the new SIKE implementation over the two FPGA devices are shown in Table V. We can see that the numbers of LUTs, DSPs, and BRAMs of the two devices are close or equal to each other. The FFs of the second device are reduced by nearly half compared to those of the first device, due to the reduction of the MM. For the Virtex-7, the number of logic cells is counted by slices in the implementation report and one slice is equal to 8 FFs or 4 LUTs. For the Kintex UltraScale+, that number is counted by configurable logic blocks (CLBs) and one CLB equals 16 FFs or 8 LUTs. That means one CLB is equivalent to two slices. Clearly, the used equivalent slices of the second device are slightly less than the slices used in the first one. On the other hand, we can find that the available resources

TABLE V
RESOURCE UTILIZATIONS OF MAJOR COMPONENTS FOR SIKEP751

Component	# FFs	# LUTs	# DSPs	# BRAMs	# Slices /CLBs
Virtex-7					
FALU	75,440	83,362	966	0	29,158
MM	72,318	77,009	960	0	27,070
MA	768	2,508	0	0	1,038
MS	769	2,805	0	0	984
IDCM	1,585	1,043	6	0	746
Control Unit	246	2,352	0	25	1,251
Keccak-1088	2,703	5,309	0	0	1,581
Register File	0	0	0	21	0
ROM	0	0	0	2	0
Total	81,518/ 866,400 9.41%	92,557/ 433,200 21.37%	966/ 3,600 26.83%	48/ 1,470 3.27%	31,407/ 108,300 29.00%
Kintex UltraScale+					
FALU	42,535	83,957	966	0	13,576
MM	39,413	77,610	960	0	12,522
MA	768	2,505	0	0	642
MS	769	2,805	0	0	628
IDCM	1,585	1,037	6	0	455
Control Unit	200	2,287	0	23	862
Keccak-1088	2,703	5,309	0	0	983
Register File	0	0	0	21	0
ROM	0	0	0	2	0
Total	48,566/ 682,560 7.12%	93,366/ 341,280 27.36%	966/ 3,528 27.38%	46/ 744 6.18%	14,763/ 42,660 34.61%

of the second device are less than the first device. It means that the improvement in efficiency is mainly brought by the technology.

We compare our SIKE results with other results in the literature [18], [24]–[29] in Table VI, including the area and timing comparisons. The term, ST product computed by slice$\times$time, is a combination of the area and timing to make a fair comparison. The normalized latency is also included. It can be seen that our design achieves the shortest latency and the fastest speed over either FPGA device at a cost of resource increase.

Especially, the latency, which is usually the bottleneck because of the inevitable data dependency, is drastically reduced compared to the prior arts. But its reduction ratio is not as much as the proposed MM's. We find that though the MM takes up the major portion of the whole computation, the time for memory access is not negligible and cannot be removed in such scheduling. This phenomenon can be directly explained by the above example shown in Fig. 9. Additionally, the parallelism is not fully used.

Clearly, the major bottleneck in our design is the clock frequency. When considering the ST product, the results on the Kintex UltraScale+ are generally better than those on the Virtex-7 device but slightly superior to the state-of-the-art. For the Virtex 7, our design is better than most of the previous works except the design in [29]. It should be pointed out that all the previous designs are based on the Montgomery multiplication algorithm. Many existing architectures can be referred to for them. Especially, the design in [29] is highly sped up based on their several rounds of optimizations. Note that our design is based on our recently proposed field arithmetic

TABLE VI
OVERALL COMPARISONS OF SIKEP751/SIDHP751 FOR TWO FPGA DEVICES

Work	# Mults	# FFs	# LUTs	# DSPs	# BRAMs	# Slices	Frequency (MHz)	Latency ($cc \times 10^6$) / Normalized	Total Time (ms)	ST Product (#Slice$\times s$)
Virtex-7										
Koziel *et al.* [24]	8	46,857	32,726	376	45.5	15,224	182.1	7.74 / 4.2	42.5	647
Koziel *et al.* [25]	8	48,688	34,742	384	58.5	14,447	203.7	6.86 / 3.7	33.7	487
Jao *et al.* [18]	8	51,914	44,822	376	56.5	16,756	198	6.60 / 3.6	33.4	560
Massolino *et al.* [26]	1[1]	13,657	21,210	162	38.0	7,408	142.2	8.60 / 4.7	60.8	450
Roy *et al.* [27]	3[2]	62,124	49,099	294	22.5	18,711	225.7	7.12 / 3.9	31.6	591
Koziel *et al.* [28]	8	50,079	39,953	512	43.5	15,834	163.1	4.55 / 2.5	27.8	440
Elkhatib *et al.* [29]	8	39,339	20,207	452	41.5	11,136	232.7	5.93 / 3.2	25.5	284
This Work	1	81,518	92,557	966	48	31,407	138.9	1.83 / 1.0	13.2	415
Kintex UltraScale+[3]										
Koziel *et al.* [28]	8	50,143	40,700	512	43.5	15,452	296.9	4.55 / 2.6	15.3	236
This Work	1	48,566	93,336	966	46	29,526	200.0	1.78 / 1.0	8.9	263

[1] A scalable multiplier called Carmela256.
[2] One multiplier includes three or four parallel modular multiplications.
[3] The number of slices are computed by doubling the CLBs.

algorithms, totally different from the previous algorithms. The architectures for them are devised in this paper for the first time. We can believe that it has great potential to be a good alternative with our substantial efforts.

VI. CONCLUSION

In this paper, we presented a high-speed FPGA implementation for the SIKE protocol based on the proposed ultra-low-latency field multiplier, adder, and subtracter. The corresponding field arithmetic algorithms are based on a new data representation, which facilitates these algorithms to obtain lower complexity and higher parallelism than the state-of-the-art. The experiment results show that the proposed field arithmetic modules achieve very low latency with acceptable resource consumption. Since those modules are totally feed-forward, a more compact control logic is devised. A new SIKE is obtained by integrating the proposed ALU and the updated instructions into the original SIKE design. The implementation results demonstrate that the latency and the total time of a SIKE both are drastically reduced compared to the prior arts. We expect that these achievements would greatly contribute to the SIKE's competitiveness over other PQC candidates. Our future work is to further reduce the latency and resource consumption and increase the clock frequency.

REFERENCES

[1] R. F. Mandelbaum, "This could be the best quantum computer yet," 2018. https://gizmodo.com/this-could-be-the-best-quantum-computer-yet-1831085617.

[2] F. Lardinois, "Ibm unveils its first commercial quantum computer," 2019. https://techcrunch.com/2019/01/08/ibm-unveils-its-first-commercial-quantum-computer/.

[3] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of The ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[4] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*, 1985.

[5] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, Nov 1994.

[6] L. Chen, L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *Report on post-quantum cryptography*, vol. 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[7] R. Azarderakhsh, M. Campagna, L. Costello, B. Feo, A. Hess, D. Jao, B. Koziel, and P. Longa, "Supersingular isogeny key encapsulation," *Submission to the NIST Post-Quantum Standardization project: https://sike.org/*, 2019.

[8] D. Jao and L. De Feo, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," in *International Workshop on Post-Quantum Cryptography*, pp. 19–34, Springer, 2011.

[9] B. T. Yan, "On the security of supersingular isogeny cryptosystems," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2016.

[10] A. Gélin and B. Wesolowski, "Loop-abort faults on supersingular isogeny cryptosystems," in *International Workshop on Post-Quantum Cryptography*, pp. 93–106, Springer, 2017.

[11] Y. B. Ti, "Fault attack on supersingular isogeny cryptosystems," in *Post-Quantum Cryptography* (T. Lange and T. Takagi, eds.), (Cham), pp. 107–122, Springer International Publishing, 2017.

[12] D. Jao, "Software for "towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies"," 2011. https://github.com/defeo/ss-isogeny-software.

[13] R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi, "Key compression for isogeny-based cryptosystems," in *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pp. 1–10, 2016.

[14] R. Azarderakhsh, D. Fishbein, and D. Jao, "Efficient implementations of a quantum-resistant key-exchange protocol on embedded systems," *Citeseer*, 2014.

[15] C. Costello, P. Longa, and M. Naehrig, "Efficient algorithms for supersingular isogeny diffie-hellman," in *Advances in Cryptology – CRYPTO 2016* (M. Robshaw and J. Katz, eds.), (Berlin, Heidelberg), pp. 572–601, Springer Berlin Heidelberg, 2016.

[16] A. Faz-Hernndez, J. Lpez, E. Ochoa-Jimnez, and F. Rodrguez-Henrquez, "A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1622–1636, 2018.

[17] G. H. M. Zanon, M. A. Simplicio, G. C. C. F. Pereira, J. Doliskani, and P. S. L. M. Barreto, "Faster key compression for isogeny-based cryptosystems," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 688–701, 2019.

[18] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev, and D. Urbanik, "PQCrypto-SIDH." Submission to the NIST Post-Quantum Standardization Project, 2020, [Online] https://github.com/Microsoft/PQCrypto-SIDH.

[19] H. Seo, Z. Liu, P. Longa, and Z. Hu, "SIDH on ARM: faster modular multiplications for faster post-quantum supersingular isogeny key exchange," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–20, 2018.

[20] A. Jalali, R. Azarderakhsh, and M. M. Kermani, "NEON SIKE: supersingular isogeny key encapsulation on ARMv7," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 37–51, Springer, 2018.

[21] A. Jalali, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Towards optimized and constant-time CSIDH on embedded devices," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 215–231, Springer, 2019.

[22] W. Liu, J. Ni, Z. Liu, C. Liu, and M. ONeill, "Optimized modular multiplication for supersingular isogeny Diffie-Hellman," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1249–1255, 2019.

[23] B. Koziel, R. Azarderakhsh, M. M. Kermani, and D. Jao, "Post-quantum cryptography on FPGA based on isogenies on elliptic curves," *IEEE Transactions on Circuits and Systems I-regular Papers*, vol. 64, no. 1, pp. 86–99, 2017.

[24] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani, "Fast hardware architectures for supersingular isogeny diffie-hellman key exchange on fpga," in *International Conference on Cryptology in India*, pp. 191–206, Springer, 2016.

[25] B. Koziel, R. Azarderakhsh, and M. M. Kermani, "A high-performance and scalable hardware architecture for isogeny-based cryptography," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1594–1609, 2018.

[26] P. M. C. Massolino, P. Longa, J. Renes, and L. Batina, "A compact and scalable hardware/software co-design of SIKE," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 245–271, 2020.

[27] D. B. Roy and D. Mukhopadhyay, "Post quantum ECC on FPGA platform," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 568, 2019.

[28] B. Koziel, A. Ackie, R. El Khatib, R. Azarderakhsh, and M. M. Kermani, "Sike'd up: Fast hardware architectures for supersingular isogeny key encapsulation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–13, 2020.

[29] R. Elkhatib, R. Azarderakhsh, and M. Mozaffari-Kermani, "Highly optimized montgomery multiplier for sike primes on fpga," in *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, pp. 64–71, 2020.

[30] T. Blum and C. Paar, "High-radix montgomery modular exponentiation on reconfigurable hardware," *IEEE Transactions on Computers*, vol. 50, no. 7, pp. 759–764, 2001.

[31] Montgomery and L. Peter, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–519, 1985.

[32] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Efficient finite field multiplication for isogeny based post quantum cryptography," *International Workshop on the Arithmetic of Finite Fields*, pp. 193–207, 2016.

[33] J. Bos and S. Friedberger, "Arithmetic considerations for isogeny based cryptography," *IEEE Transactions on Computers*, pp. 1–1, 2018.

[34] J. Tian, J. Lin, and Z. Wang, "Ultra-fast modular multiplication implementation for isogeny-based post-quantum cryptography," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 97–102, IEEE, 2019.

[35] B. Wu, J. Tian, X. Hu, and Z. Wang, "A novel modular multiplier for isogeny-based post-quantum cryptography," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 334–339, 2020.

[36] J. Tian, Z. Liu, J. Lin, Z. Wang, and B. Li, "High-speed modular multipliers for isogeny-based post-quantum cryptography." Cryptology ePrint Archive, Report 2019/1206, 2019. https://eprint.iacr.org/2019/1206.

[37] J. Tian, P. Wang, Z. Liu, J. Lin, Z. Wang, and J. Groschdl, "Faster software implementation of the SIKE protocol based on a new data representation." Cryptology ePrint Archive, Report 2020/660, 2020. https://eprint.iacr.org/2020/660.

[38] C. Costello, P. Longa, M. Naehrig, J. Renes, and F. Virdia, "Improved classical cryptanalysis of the computational supersingular isogeny problem," *IACR Cryptology ePrint Archive*, vol. 2019, p. 298, 2019.

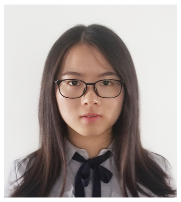
[39] J. Vélu, "Isogénies entre courbes elliptiques," *CR Acad. Sci. Paris, Séries A*, vol. 273, pp. 305–347, 1971.

[40] A. Karatsuba, "Multiplication of multidigit numbers on automata," *Soviet physics. Doklady*, vol. 7, pp. 595–596, 1963.

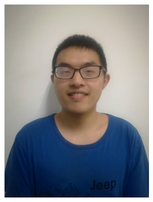
[41] Z. Wang, "High-speed recursion architectures for map-based turbo decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 470–474, 2007.

[42] M. P. G. V. A. G. Bertoni, J. Daemen and R. V. Keer, "Keccak implementation overview." https://keccak.team/files/Keccak-implementation-3.2.pdf, May 2012.

[43] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani, "Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA," in *International Conference in Cryptology in India*, pp. 191–206, Springer, 2016.



Jing Tian received her B.S. degree in microelectronics and Ph.D. degree in information and communication engineering from Nanjing University, Nanjing, China, in 2015 and 2020, respectively. She is now an associate researcher in Nanjing University. Her research interests include VLSI design for digital signal processing and cryptographic engineering.



Bo Wu received his B.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2020. He is currently working toward the M.S. integrated circuit engineering from Nanjing University. His research interest includes VLSI design for post-quantum cryptography.

Zhongfeng Wang received both the B.E. and M.S. degrees in the Dept. of Automation at Tsinghua University, Beijing, China, in 1988 and 1990, respectively. He obtained the Ph.D. degree from the University of Minnesota, Minneapolis, in 2000. He has been working for Nanjing University, China, as a Distinguished Professor since 2016. Previously he worked for Broadcom Corporation, California, from 2007 to 2016 as a leading VLSI architect. Before that, he worked for Oregon State University and National Semiconductor Corporation.

Dr. Wang is a world-recognized expert on Low-Power High-Speed VLSI Design for Signal Processing Systems. He has published over 200 technical papers with multiple best paper awards received from the IEEE technical societies, among which is the VLSI Transactions Best Paper Award of 2007. He has edited one book VLSI and held more than 20 U.S. and China patents. In the current record, he has had many papers ranking among top 25 most (annually) downloaded manuscripts in IEEE Trans. on VLSI Systems. In the past, he has served as Associate Editor for IEEE Trans. on TCAS-I, T-CAS-II, and T-VLSI for many terms. He has also served as TPC member and various chairs for tens of international conferences. Moreover, he has contributed significantly to the industrial standards. So far, his technical proposals have been adopted by more than fifteen international networking standards. In 2015, he was elevated to the Fellow of IEEE for contributions to VLSI design and implementation of FEC coding. His current research interests are in the area of Optimized VLSI Design for Digital Communications and Deep Learning.