High-Speed Modular Multipliers for Isogeny-Based Post-Quantum Cryptography

Jing Tian, *Student Member, IEEE*, Zhe Liu, *Senior Member, IEEE*, Jun Lin, *Senior Member, IEEE*, Zhongfeng Wang, *Fellow, IEEE*, and Binjing Li

Abstract—As one of the post-quantum protocol candidates, the supersingular isogeny key encapsulation (SIKE) protocol delivers promising public and secret key sizes over other candidates. Nevertheless, the considerable computations form the bottleneck and limit its practical applications. The modular multiplication operations occupy a large proportion of the overall computations required by the SIKE protocol. The VLSI implementation of the high-speed modular multiplier remains a big challenge. In this paper, we propose three improved modular multiplication algorithms based on an unconventional radix for this protocol, all of which cost about 20% fewer computations than the prior art. Besides, a multi-precision scheme is also introduced for the proposed algorithms to improve the scalability in hardware implementation, resulting in three new algorithms. We then present very efficient high-speed modular multiplier architectures for the six algorithms. It is shown that these new architectures can be highly optimized and extensively pipelined to obtain high throughput thanks to the adopted overlapping processing scheme. The FPGA implementation results show the proposed multipliers without the multi-precision scheme all achieve about 60 times higher throughput than the state-of-the-art design (the FFM2 multiplier), and those with the multi-precision scheme all acquire almost 10 times higher throughput than this work. Meanwhile, each of the multi-precision based designs has almost the same resource consumptions as the FFM2 does.

Index Terms—Modular multiplication, supersingular isogeny Diffie-Hellman (SIDH) key exchange, post-quantum cryptography (PQC), FPGA, VLSI.

1 Introduction

Recent improvements in quantum system control make it seem feasible to finally build a powerful quantum computer in the near future [1]. Many companies like IBM, Google, and Intel have enthusiastically joined this field. A company named IonQ reported in December 2018 that its machine could be built as large as 160 qubits [2]. These achievements have actually brought to the flurry of research in public-key cryptography since most of the popular public-key ciphers, such as the RSA [3] and ECC [4] schemes based on the difficulty of factoring integers or the discrete logarithm problem, can be solved by Shor's algorithm [5] with quantum computers. Meanwhile, they have also accelerated the development in post-quantum cryptography (PQC) protocols. For example, the call for proposals for PQC standards hosted by the National Institute of Standards and Technology (NIST) [6] is driven by this demand.

The supersingular isogeny key encapsulation (SIKE) protocol [7] has won the second round of fierce competitions and been one of the 26 candidates in April 2019 after submitted to the NIST in November 2017. The possible reason is that it is the only one which is similar to the classical ECC having very small public and secret keys and owning perfect forward secrecy. The core structure of SIKE is the supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol, packaged by the key encapsulation mechanism [8] to defend against various side-channel attacks [9]. The SIDH was first introduced by Jao and De Feo in 2011 to resist quantum attack based on the difficulty of finding isogenies between supersingular elliptic curves [10]. The zero-knowledge identification scheme was proposed based on this protocol in [11]. Jao and Soukharev presented the undeniable signatures based on the SIDH in [12]. In [13], Azarderakhsh *et al.* provided a key compression method which brings in about twice of reduction in sizes of public information without an impact on security. However, the computations of these algorithms are still huge and make them encounter difficulties in practical applications.

To alleviate this problem, many researchers have focused on speed-up for the SIDH key exchange on hardware, like on FPGA [14], [15], [16] or on ARM [17], [18], [19]. Through breaking down the computations, the modular multiplication is one of the fundamental operations, which is the main concerned issue in these designs. The first FPGA implementation for SIDH key exchange was proposed by Koziel *et al.* in [14] by parallelizing the modular multipliers based on the high-radix Montgomery multiplication algorithm [20]. They further sped up this protocol by adding more modular multipliers in [15]. In [16], Liu *et al.* presented two fast modular multipliers, the FFM1 and FFM2, for SIDH based on an unconventional radix inspired by the efficient finite field multiplication (EFFM) algorithm proposed in [21]. Additionally, the SIDH is implemented on ARM-embedded systems as well. In [17], Seo *et al.* proposed a unified ARM/NEON multi-precision modular multiplication architecture based on the specialized Montgomery reduction and integrated it to the SIDH library [22] to accelerate the original ARM design. Jalali *et al.* implemented

- *J. Tian, J. Lin, Z. Wang, and B. Li are with Nanjing University, China. Email: jingtian@smail.nju.edu.cn, {jlin, zfwang}@nju.edu.cn, banxiabaisu@163.com*
- *Z. Liu is with Nanjing University of Aeronautics and Astronautics, China. Email: zhe.liu@nuaa.edu.cn*

Date of the manuscript: August 9, 2019.

the optimized field arithmetic operations on ARM for SIKE in [18] and commutative SIDH (CSIDH) protocols in [19], respectively. Indeed, much progress has been made to speed up the SIKE protocol and make it more practical. But, these implementations for this PQC candidate still suffer more than one order of magnitude slower speed than those for most of the other candidates.

Notice that the smooth isogeny modulus for SIDH [1] has the form of $p = f \cdot a^x b^y \pm 1$, where a and b are small primes, x and y are positive integers, and f is a small cofactor to make p prime. To simplify the modular operations, especially for the modular multiplication, the parameter a is usually set to 2. The form of p then becomes $f \cdot 2^x b^y \pm 1$. The EFFM in [21] uses the p with the form of $2 \cdot 2^x b^y - 1 = 2R^2 - 1$, where x and y must be even, and $R = 2^{x/2} b^{y/2}$, the input numbers of which are transformed into quadratic polynomials based on the unconventional radix R and about half of the multiplications can be reduced for the reduction operation based on R at the cost of some additions. The FFM1 in [16] reduces the coefficients of EFFM from three to two by using an extra mapping function for the input and output, which could efficiently discard the precomputing constant without any increase in complexity. The FFM2 in [16] extends the searching scope of the prime with the form of $f \cdot 2^x b^y \pm 1$ at the expense of more computations. It should be pointed out that a good prime could more possibly be found with a larger searching scope, which could also help increase the efficiency of the algorithm. Therefore, it is important to develop efficient modular multiplication algorithms with loose constraints for the prime.

In this paper, we propose three new modular multiplication algorithms for different forms of prime based on an unconventional radix adopted in [16], [21], [23], and all of them have lower computational complexity than previous algorithms. We aim to extend the previous prime used in [21] into prime with form of $f \cdot 2^x b^y \pm 1$ where $f \in \{1, 2\}$, x and y are even. The prime can be split into three forms: $2 \cdot 2^x b^y - 1$, $2 \cdot 2^x b^y + 1$, and $2^x b^y + 1$. Accordingly, the corresponding new algorithms are named as IFFM$^-$, IFFMo$^+$, and IFFMe$^+$. We set x and y to even, and use $R = 2^{x/2} b^{y/2}$ as the unconventional radix for the proposed algorithms. For the IFFM$^-$ algorithm, the usage of the radix is almost the same as before, which has been preliminarily presented in our conference paper [24]. For the IFFMo$^+$ and IFFMe$^+$, we use the radix $R = 2^{x/2} b^{y/2}$ to reduce the complexity for the first time by expanding the range of the constant coefficient of a quadratic polynomial. A detailed discussion can be found in Section 3.2. The reduction and multiplication of the proposed algorithms are optimized to reduce the computational complexity by about 20%. Meanwhile, we also develop their multi-precision based versions with a clever interleaving scheme and present three other new algorithms to improve the scalability and reduce the resource consumption in hardware implementation.

Moreover, we have devised new architectures for the proposed algorithms with fully parallelizing and overlapping schedules which enable to utmostly reduce the required clock cycles and highly optimize each sub-module. We have also coded the proposed architectures with the Verilog language and implemented them on FPGA. The implementation results show that the designs without the multi-precision scheme have about 60 times faster throughput than the state-of-the-art design by introducing a relatively small portion of extra hardware resources. When applying the multi-precision scheme, these designs achieve significant reductions on total resource consumptions at the cost of slower throughput compared to their original versions while still being much better than prior arts.

1. The SIKE has the same form of modulus; we mainly refer to the SIDH in the following discussion.

2 Background

The multiplication for cryptography is based on finite fields, called modular multiplication, requiring the modular reduction after the multiplication operation. In the following we will first introduce the Montgomery reduction, the Barrett reduction, and the efficient Barrett reduction for the SIDH. Then, several efficient modular multiplications for SIDH will be presented.

2.1 Modular Reduction Algorithms

2.1.1 Montgomery Reduction

The main idea of the Montgomery reduction [25] is to replace the ordinary modulus by a power of two so that the modular reduction operation is inexpensive to handle in hardware implementation. The detailed process is shown in Alg. 1. The modulus p is an arbitrary number, which is less than R (equal to 2^N). The term $(-p^{-1}) \bmod R$ is precomputed and saved. As integers modulo R is very easy, we will not take this kind of computations into consideration in the following counting. It can be found that the complexity is only related to the bit width of the modulus p. This algorithm totally requires two $N \times N$ multiplications, one $2N + 2N$ and one $N + N$ adders. Note that the output remainder is not $c \bmod p$ but $cR^{-1} \bmod p$. Luckily, if the operands are converted into Montgomery presentations by multiplying R, all of the arithmetic operations can be normally used. when an algorithm contains many modular multiplications and divisions, this conversion overhead becomes negligible. Therefore, the Montgomery reduction is usually used for the SIDH in conventional designs [14], [15].

Algorithm 1: The Montgomery reduction [25].

Input: $0 \le c < Rp$, where $R = 2^N$ and $2^{N-1} < p < 2^N$; precompute $p' = (-p^{-1}) \bmod R$.

1: $t = ((c \bmod R)p') \bmod R$
2: $r = (c + t \cdot p)/R$
3: **if** $r \ge p$ **then**
4: $\quad r = r - p$
5: **end if**

Output: $r = cR^{-1} \bmod p$.

2.1.2 Barrett Reduction

Another hardware-friendly modular reduction algorithm is the Barrett reduction, proposed by Paul Barrett in 1986 [26]. The key idea is also to transfer the complex division to an easier one by introducing an extra parameter.

Algorithm 2: The Barrett reduction [26].

Input: $0 \leq c < 2^{\alpha}$; $2^{N-1} < p < 2^{N}$;
precompute $\lambda = \lfloor 2^{\alpha}/p \rfloor$.
1: $q = \lfloor \frac{c \cdot \lambda}{2^{\alpha}} \rfloor$
2: $r = c - q \cdot p$
3: **if** $r \geq p$ **then**
4: $r = r - p$, $q = q + 1$
5: **end if**
Output: $q = \lfloor c/p \rfloor$, $r = c \bmod p$.

The flow is described in Alg. 2. It should be noticed that the complexity is changed with the input width α. When $\alpha = 2N$, this algorithm costs one $2N \times (N+1)$ and one $N \times (N+1)$ multiplications, and one $2N + 2N$ and one $N + N$ adders. The complexity of multiplication is almost 1.5 times of the Montgomery reduction's. The benefit is that this algorithm can directly compute the quotient and the remainder.

2.1.3 Efficient Barrett Reduction for SIDH

As introduced above, the form of modulus for SIDH is $f \cdot a^x b^y \pm 1$. In [21], the authors have constrained the values of f and a to 2, respectively. Meanwhile, x and y must be even. Therefore, the unconventional radix R has the form of $2^x b^y$ (x and y are arbitrary positive integers here). The Barrett reduction is used for such kind of moduli. Intuitively, the modulus R can be split into two parts — 2^x and b^y, and the reduction can be computed in two steps as introduced in [21]. This algorithm is summarized in Alg. 3. Obviously,

Algorithm 3: The Barrett reduction (BR) for modulus $R = 2^x b^y$ [21].

Input: $c \in N^+$, $0 \leq c < 2^{\alpha}$; $R' = R/2^x = b^y$;
precompute $\lambda = \lfloor \frac{2^{\alpha}}{R} \rfloor$.
1: $t = \lfloor c/2^{N_1} \rfloor$, $s = c \bmod 2^{N_1}$
2: $q = \lfloor \frac{t \cdot \lambda}{2^{\alpha - N_1}} \rfloor$
3: $r = t - q \cdot R'$
4: $r = r \ll N_1 + s$
5: **if** $r \geq R$ **then**
6: $r = r - R$, $q = q + 1$
7: **end if**
Output: q, r.

the main cost is for modulo b^y. Assume that $N_1 = x$, $N_2 = \lceil \log_2(b^y) \rceil$, and $N_1 \approx N_2 \approx N/2$. When $\alpha = 2N$, this algorithm costs one $3N/2 \times (N+1)$ and one $N/2 \times (N+1)$ multiplications, and one $3N/2 + 3N/2$ and one $N + N$ adders. Clearly, this reduction algorithm is more efficient than the other two algorithms.

2.2 Efficient Modular Multiplications for SIDH

2.2.1 EFFM

The modular multiplication named EFFM algorithm proposed in [21] is generalized in Alg. 4. It works in an interleaved way with an unconventional radix to compute the multiplication and reduction. It is a kind of simplifications for the normal modular multiplication by reducing the

Algorithm 4: The EFFM modular multiplication proposed in [21].

Input: $A = a_2R^2 + a_1R + a_0$, $B = b_2R^2 + b_1R + b_0$;
$p = 2 \cdot 2^{2x}b^{2y} - 1 = 2R^2 - 1$;
$2^{N-1} < p < 2^{N}$, $R < 2^{N/2}$.
1) The first tentative computing:
Common items:
1: $t_1 = a_2b_1 + a_1b_2$, $t_2 = a_2b_0 + a_1b_1 + a_0b_2$
Results:
2: $c_2 = t_2 \bmod 2$, $c_1 = \lfloor \frac{t_1}{2} \rfloor + a_1b_0 + a_0b_1$,
$c_0 = (2^{-2} \bmod p)a_2b_2 + a_0b_0 + (t_1 \bmod 2)\frac{R}{2} + \lfloor \frac{t_2}{2} \rfloor$
2) The second tentative computing:
Reduction:
3: $[q_0, r_0] = BR(c_0, R)$
4: $[q_1, r_1] = BR(c_1 + q_0, R)$
Common item:
5: $t = q_1 + c_2$
Results:
6: $c_2 = t \bmod 2$, $c_1 = r_1$, $c_0 = \lfloor \frac{t}{2} \rfloor + r_0$
3) Post processing:
Normalization:
7: **while** $c_0 \geq R$ **do**
8: $c_0 = c_0 - R$
9: $c_1 = c_1 + 1$
10: **if** $c_1 \geq R$ **then**
11: $c_0 = c_2 + c_0$, $c_1 = R - 1$, $c_2 = (\sim c_2)$
12: **end if**
13: **end while**
Output: $C = A \times B \bmod p = c_2R^2 + c_1R + c_0$.

modulus p to R, where $p = 2 \cdot 2^{2x}b^{2y} - 1$ and $R = 2^x b^y$. The input A, which is a field element in F_p, is expressed as in quadratic polynomial as:

$$A = a_2R^2 + a_1R + a_0 \tag{1}$$

in which $a_2 \in \{0, 1\}$ and $0 \leq a_1, a_0 < R$. We divide the process of this algorithm into three steps: 1) the first tentative computing; 2) the second tentative computing; and 3) post processing. In the first step, the higher order (larger than two orders) terms are reduced and merged with the lower order terms according to the rules deduced in [21]. The second step is to further reduce the coefficients by adopting two *BR* functions as presented in Alg. 3. The data width α of the *BR* function is about equal to N (half of that of the original input) and $N_1 \approx N_2 \approx N/4$. In the post processing step, the *while* loop could be executed at most once as introduced in [21]. Since adding or multiplying one number by a single-bit number is very easy, these kinds of operations are not taken into account in this paper. Thus, this algorithm takes four $N/2 \times N/2$, two $3N/4 \times (N/2+1)$, and two $N/4 \times (N/2+1)$ multiplications, and six $N/2+N/2$, two $3N/4+3N/4$, three $N/2+N$, and three $N+N$ additions.

2.2.2 FFM1

Recently, the authors in [16] have proposed the FFM1 algorithm to remove the coefficients a_2 and b_2 of the inputs

of Alg. 4. Taking input A for example, they have used the following formula:

$$a_i = \begin{cases} a_i, \ a_2 = 0 \\ R - a_i - 1, \ a_2 = 1 \end{cases}, i = \{1, 0\}, \tag{2}$$

based on the fact that

$$AB \equiv (p - A)(p - B) \equiv p - (p - A)B \bmod p \tag{3}$$

and

$$\begin{aligned} p - A &= 2R^2 - 1 - (a_2R^2 + a_1R + a_0) \\ &= (1 - a_2)R^2 + (R - a_1 - 1)R + (R - a_0 - 1). \end{aligned} \tag{4}$$

The transformation in Eq. (2) is also needed for the output and a_2 is replaced by $a_2 \oplus b_2$. This modification can remove the precomputing parameter $(2^{-2} \bmod p)$. The complexity of multiplications is the same as the EFFM, and it takes ten $N/2+N/2$, two $3N/4+3N/4$, one $N/2+N$, and two $N+N$ additions. As the transformation for the inputs and output requires more extra additions, the complexity reduction is limited.

2.2.3 FFM2

The FFM2 algorithm is another efficient modular algorithm proposed in [16], to extend the searching space of the modulus p with the form of $f \cdot 2^x b^y \pm 1$ (x and y are arbitrary positive integers here) at a cost of more multiplications. It costs one $N \times N$, one $3N/2 \times (N+1)$, and one $N/2 \times (N+1)$ multiplications, and two $N + N$ and one $3N/2 + 3N/2$ additions.

3 Proposed Modular Multiplication Algorithms

According to the analysis above, the complexity of modular multiplication algorithms in [21] and [16] is mainly dominated by the multiplications used in the first tentative computing and the two *BR* functions. We will detail our improvements from the two aspects in the following. Meanwhile, we will extend the searching space of the modulus with a form of $p = f \cdot 2^{2x} b^{2y} \pm 1$.

3.1 Improved Barrett Reduction

The improved Barrett reduction (IBR) is presented as shown in Alg. 5. We assume $\alpha = 2N$ and $N_1 \approx N_2 \approx N/2$, the same as in Section 2.1.3 for the BR algorithm. A simple improvement is to move the combination step to the end, which reduces the size of the subtraction in Step 6 of Alg. 3 from $N + N$ to $N_2 + N_2$. The other improvement is that the subtraction and multiplication operations in Step 3 of Alg. 3 are simplified (shown in Steps 3-4 of Alg. 5). Since the tentative remainder r is smaller than 2^{N_2+1}, the difference value of the $(N-1)$ MSBs of t and those of $q \cdot R$ is no more than one, which can be made out by their (N_2+1)-th bits. Accordingly, we can reduce the sizes of the subtraction and multiplication from $\frac{3}{2}N + \frac{3}{2}N$ and $\frac{3}{2}N \times (N+1)$ to $\frac{N}{2} + \frac{N}{2}$ and $(\frac{N}{2}+1) \times \frac{N}{2}$, respectively. If their (N_2+1)-th MSBs are not equal, the remainder r would be adjusted by adding the parameter 2^{N_2}. Therefore, the IBR algorithm only requires one $3N/2 \times (N+1)$ and one $N/2 \times (N/2+1)$ multiplications, and three $N/2 + N/2$ additions. When compared to the BR algorithm, the complexities of multiplication and addition are reduced by about 12.5% and about 40%, respectively.

Algorithm 5: The improved *BR* (IBR) for hardware efficiency.

Input: $c \in N^+$, $0 \le c < 2^\alpha$; $R' = R/2^x = b^y$;
precompute $\lambda = \lfloor 2^\alpha / R \rfloor$.
1: $t = \lfloor c/2^{N_1} \rfloor$, $s = c \bmod 2^{N_1}$
2: $q = \lfloor \frac{t \cdot \lambda}{2^{\alpha - N_1}} \rfloor$
3: $t_1 = ((q \bmod 2^{N_2+1}) \cdot R') \bmod 2^{N_2+1}$, $t = t \bmod 2^{N_2+1}$
4: $r = (t \bmod 2^{N_2}) - (t_1 \bmod 2^{N_2})$
5: **if** $\lfloor \frac{t}{2^{N_2}} \rfloor \neq \lfloor \frac{t_1}{2^{N_2}} \rfloor$ **then**
6: $\quad r = r + 2^{N_2}$
7: **end if**
8: **if** $r \ge R'$ **then**
9: $\quad r = r - R'$, $q = q + 1$
10: **end if**
11: $r = r \ll N_1 + s$
Output: q, r.

As the output r is expected to have the range of $[0, R)$, this function will not obtain the required results if the input integer c is a negative number. To deal with this problem, we first take the absolute value to the *IBR* function, and then correct the remainder with $R - r$ and the quotient with $-(q+1)$ when c is negative. This modified reduction algorithm is defined as *IBR*$^+$.

3.2 Modular Multiplication Algorithms with Modulus $p = 2^x b^y \pm 1$

Based on the introduction in [21], [23], and [16], the unconventional radix for modular multiplication of SIDH shows more efficiency than conventional methods. This concept is first proposed in [21] for the smooth isogeny prime modulus p with the form $p = 2 \cdot 2^x b^y - 1$ where x and y must be even. In this section, we will extend this prime with the form $p = 2^x b^y \pm 1$ for an even y. For convenience, we reformulate the prime as $p = f \cdot 2^{2x} b^{2y} \pm 1$ where $f = 1$ or 2 to make the transformation hold. Here, we propose two algorithms for $p = f \cdot 2^{2x} b^{2y} + 1$ with unconventional radix for the first time. There are two issues required to be solved for this kind of modulus p: 1) how to construct an unconventional radix; 2) how to reduce the coefficients with such a modulus.

For the first issue, we still keep the field elements in $\mathbb{F}_p$ with the form of quadratic polynomials based on the unconventional radix of $R = 2^x b^y$ as Eq. (1). This form is a one-to-one mapping for the modulus p with minus sign, where all elements in $\mathbb{F}_p$ are exactly expressed and the operation $p - A$ is still in this field. Back to the original motivation, the target is to replace the large modulus p with a small modulus R, not to construct an onto mapping. Thus, we try to build a mapping which may not be so exact but can involve all elements in $\mathbb{F}_p$. With this clue, we find that if we extend the range of the coefficient of the constant term to $R+1$, saying $0 \le a_0 \le R+1$, this goal will be achieved. We only need to add this constraint for c_0 in the post-processing step and most of the processing steps are almost the same as the EFFM or FFM1. Note that in some cases two polynomials

would equal the same value in $\mathbb{F}_p$, which however do not affect the calculations and the final results.

For the second issue, we review that the basic idea of EFFM [21] is firstly to resolve the quadratic or higher-order product terms modulo p and then to reduce the coefficients by modulo or subtracting R. We suppose that the quadratic and higher-order product terms are combined as tR^2. The formula tR^2 modulo $p = f \cdot R^2 \pm 1$ can be computed as

$$\begin{aligned} & tR^2 \bmod p \\ \equiv\ & ((tR^2 \bmod (fR^2)) \mp \lfloor \frac{tR^2}{fR^2} \rfloor) \bmod p \\ =\ & (((t \bmod f) \cdot R^2) \mp \lfloor \frac{t}{f} \rfloor) \bmod p. \end{aligned} \tag{5}$$

When $p = 2 \cdot R^2 - 1$, the plus sign is taken and this equation is equivalent to the deduced equation in [21]. For $p = f \cdot R^2 + 1$, this equation can also be used.

Since the modulus p is prime, we have three forms for p: 1) $p = 2 \cdot 2^{2x} b^{2y} - 1$; 2) $p = 2 \cdot 2^{2x} b^{2y} + 1$; and 3) $p = 2^{2x} b^{2y} + 1$. For the three different p forms, we apply the proposed *IBR* (*IBR*$^+$) function and obtain three different modular multiplication algorithms named as IFFM$^-$, IFFMo$^+$, and IFFMe$^+$, respectively. We will discuss more details in the following.

3.2.1 IFFM$^-$ for Modulus $p = 2 \cdot 2^{2x} b^{2y} - 1$

For $p = 2R^2 - 1$, the optimization methods have been fully discussed in [21] and [16]. Equation 5 for this modulus turns into

$$tR^2 \bmod p \equiv (((t \bmod 2) \cdot R^2) + \lfloor \frac{t}{2} \rfloor) \bmod p. \tag{6}$$

In the proposed IFFM$^-$, besides applying the *IBR* function, we also use the mapping function proposed in [16] as shown in Eq. (2). Meanwhile, the number of multiplications is further reduced by using the formula:

$$a_1 b_0 + a_0 b_1 = (a_1 + a_0)(b_1 + b_0) - a_0 b_0 - a_1 b_1. \tag{7}$$

The proposed algorithm is shown in Alg. 6. The *post_proc* function is given in Alg. 8, where the range of the input c_0^o is deduced shown in Section 7.1. Therefore, the proposed IFFM$^-$ only needs two $N/2 \times N/2$, one $(N/2+1) \times (N/2+1)$, two $3N/4 \times N/2$, and two $N/4 \times N/4$ multiplications, and six $N/4 + N/4$, ten $N/2 + N/2$, one $N/2 + N$, and three $N + N$ additions.

3.2.2 IFFMo$^+$ for Modulus $p = 2 \cdot 2^{2x} b^{2y} + 1$

For $p = 2R^2 + 1$, it means that f is set to 2 and the minus sign is adopted. Thus Eq. (5) becomes

$$tR^2 \bmod p \equiv (((t \bmod 2) \cdot R^2) - \lfloor \frac{t}{2} \rfloor) \bmod p. \tag{8}$$

The IFFMo$^+$ is very similar to the IFFM$^-$. We will mainly analyze the different operations of this algorithm. Firstly, the negation operation, $p - A$, will be

$$\begin{aligned} p - A &= 2R^2 + 1 - (a_2 R^2 + a_1 R + a_0) \\ &= (1 - a_2) R^2 + (R - a_1 - 1) R + (R - a_0 + 1), \end{aligned} \tag{9}$$

which is still a standard expression. Therefore, for a_1, the *map* function is $R - a_1 - 1$, and we define *map*$^+$ function as $R - a_0 + 1$ for a_0. And it is the same for b_1 and b_0. Secondly, when updating the constant term c_0, the subtraction should be taken, including Steps 5 and 9 in Alg. 6. Thirdly, the *IBR* function is replaced by the *IBR*$^+$ function. Finally, the *post_proc* function shown in Alg. 8 is updated differently according to the range of c_0^o deduced in Section 7.2. The complexity of this algorithm is almost the same as that of the IFFM$^-$, with two extra $N/2 + N/2$ additions.

Algorithm 6: The proposed IFFM$^-$ for $p = 2R^2 - 1$.

Input: $A = a_2 R^2 + a_1 R + a_0$, $B = b_2 R^2 + b_1 R + b_0$, $a_2, b_2 \in \{0, 1\}$, $a_1, a_0, b_1, b_0 \in [0, R-1]$; $2^{N-1} < p < 2^N$, $R < 2^{N/2}$.

1) The first tentative computing:

Mapping:

1: **for** $i = \{1, 0\}$ **do**
2: $a_i = map(a_i, a_2)$, $b_i = map(b_i, b_2)$
3: **end for**

Multiplication items:

4: $m_1 = a_1 b_1$, $m_2 = a_0 b_0$, $m_3 = (a_1 + a_0)(b_1 + b_0)$

Results:

5: $c_2 = m_1 \bmod 2$, $c_1 = m_3 - m_1 - m_2$, $c_0 = m_2 + \lfloor \frac{m_1}{2} \rfloor$

2) The second tentative computing:

Reduction:

6: $[q_0, r_0] = IBR(c_0, R)$
7: $[q_1, r_1] = IBR(c_1 + q_0, R)$

Common item:

8: $t = q_1 + c_2$

Results:

9: $c_2^o = t \bmod 2$, $c_1^o = r_1$, $c_0^o = \lfloor \frac{t}{2} \rfloor + r_0$

3) Post processing:

10: $[c_2, c_1, c_0] = post\_proc(c_2^o, c_1^o, c_0^o)$

Demapping:

11: $t = a_2 \oplus b_2$
12: $c_2 = c_2 \oplus t$, $c_1 = map(c_1, t)$, $c_0 = map(c_0, t)$

Output: $C = A \times B \bmod p = c_2 R^2 + c_1 R + c_0$.

3.2.3 IFFMe$^+$ for Modulus $p = 2^{2x} b^{2y} + 1$

For the IFFMe$^+$ algorithm, f is equal to 1, so Eq. (5) becomes

$$tR^2 \bmod p = -t \bmod p. \tag{10}$$

Meanwhile, for modulus $p = R^2 + 1$, the coefficient of the quadratic term is equal to zero. Therefore, multiplying two elements $A, B \in \mathbb{F}_p$ turns into

$$\begin{aligned} & A \times B \bmod p \\ \equiv\ & (a_1 R + a_0) \times (b_1 R + b_0) \\ \equiv\ & ((a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1) R + a_0 b_0 - a_1 b_1. \end{aligned} \tag{11}$$

It can be seen that there is no need to transform the inputs and output with Eq. (2), which can efficiently reduce the addition operations. The detailed process is shown in Alg. 7. c_0 in Step 8 has a range of $[-2R, R]$, the proof for which is attached in Section 7.3. Thus at most two additions are required to correct the final results. The number of multiplications is also the same as those of the other two algorithms. This algorithm totally costs six $N/4+N/4$, seven $N/2+N/2$, one $N/2 + N$, and three $N + N$ additions.

Algorithm 7: The proposed IFFMe$^+$ for $p = R^2 + 1$.

Input: $A = a_1R + a_0$, $B = b_1R + b_0$, $a_1, b_1 \in [0, R-1]$, $a_0, b_0 \in [0, R+1]$; $p = 2^{2x}b^{2y} + 1 = R^2 + 1$; $2^{N-1} < p < 2^N$, $R < 2^{N/2}$.

1) The first tentative computing:

Multiplication items:

1: $m_1 = a_1b_1$, $m_2 = a_0b_0$, $m_3 = (a_1 + a_0)(b_1 + b_0)$

Results:

2: $c_1 = m_3 - m_1 - m_2$, $c_0 = m_2 - m_1$

2) The second tentative computing:

Reduction:

3: $[q_0, r_0] = IBR^+(c_0, R)$

4: $[q_1, r_1] = IBR^+(c_1 + q_0, R)$

Results:

5: $c_1^o = r_1$, $c_0^o = r_0 - q_1$

3) Post processing:

6: $[c_1, c_0] = post\_proc(c_1^o, c_0^o)$

Output: $C = A \times B \bmod p = c_1R + c_0$.

Algorithm 8: The *post_proc* function for the proposed algorithms.

For IFFM$^-$:

Input : $c_2^o \in \{0, 1\}$, $0 \le c_1^o < R$, $0 \le c_0^o \le 2R - 2$.

if $c_0 \ge R$ **then**
  $c_0 = c_0^o - R$, $c_1 = c_1^o + 1$
  if $c_1 == R$ **then**
    $c_0 = c_2^o + c_0$, $c_1 = 0$, $c_2 = (\sim c_2^o)$

Output: $c_2 \in \{0, 1\}$, $0 \le c_1 < R$, $0 \le c_0 < R$.

For IFFMo$^+$:

Input : $c_2^o \in \{0, 1\}$, $0 \le c_1^o < R$, $-R \le c_0^o \le R$.

if $c_0 < 0$ **then**
  $c_0 = c_0^o + R$, $c_1 = c_1^o - 1$
  if $c_1 == -1$ **then**
    $c_2 = (\sim c_2^o)$, $c_1 = R - 1$, $c_0 = c_0 + c_2$

Output: $c_2 \in \{0, 1\}$, $0 \le c_1 < R$, $0 \le c_0 < R + 2$.

For IFFMe$^+$:

Input : $0 \le c_1^o < R$, $-2R \le c_0^o \le R$.

if $c_0 < 0$ **then**
  $c_0 = c_0^o + R$, $c_1 = c_1^o - 1$
  if $c_1 == -1$ **then**
    $c_1 = R - 1$, $c_0 = c_0 + 1$
  if $c_0 < 0$ **then**
    $c_0 = c_0 + R$, $c_1 = c_1 - 1$
    if $c_1 == -1$ **then**
      $c_1 = R - 1$, $c_0 = c_0 + 1$

Output: $0 \le c_1 < R$, $0 \le c_0 < R + 2$.

3.3 Complexity Analysis and Comparison

Assume that the data width of the input field elements A and B is N and the modulus p satisfies $2^{N-1} < p < 2^N$. We have normalized the numbers of additions and multiplications to those of $N+N$ additions and $N \times N$ multiplications for the previous and the proposed modular multiplication algorithms as listed in Table 1. The Montgomery and Barrett reduction algorithms associated with the multiplication part are abbreviated as MontM and BarM modular multiplication algorithms, respectively. Since N is usually as large as several hundred for public-key cryptosystems, the data width $N + 1$ is approximated to N. The $N + N/2$ addition, which can be split as one $N/2 + N/2$ and one $N/2 + 1$ additions, is approximately computed as one $N/2 + N/2$ addition. It can be seen that the proposed algorithms have the fewest number of multiplications. Note that the performance is mainly constrained by the computation of multiplications in these algorithms. Obviously, we have achieved nearly 20% reduction in computations compared to the state-of-the-art algorithms.

3.4 Multi-Precision Scheme for Proposed Algorithms

In order to improve the scalability in hardware design, we apply a multi-precision scheme to the proposed algorithms. Assume a number A with multi-precision format in radix of 2^k as

$$A = \sum_{j=0}^{n-1} A_j \cdot (2^k)^j, \tag{12}$$

where A_j is the j-th k-bit digit of A and n is the number of partition. To reduce the data width used in each iteration, the interleaving of multiplication and reduction is adopted as follows:

$$\begin{aligned} AB \bmod p = (...((0 \cdot 2^k + A_{n-1}B \bmod p) \cdot 2^k \\ + A_{n-2}B \bmod p) \cdot 2^k + ... + A_1B \bmod p) \cdot 2^k \\ + A_0B \bmod p. \end{aligned} \tag{13}$$

It can be observed that a recursive equation can be concluded as

$$C^{(j)} = C^{(j+1)} \cdot 2^k + A_jB \bmod p, \tag{14}$$

where $C^{(n)} = 0$, $C^{(j)}$ are the intermediate values for $0 < j < n$, and $C^{(0)}$ is the final result. In our proposed algorithms, we can transfer the modulus p to R. We represent the coefficients a_0, a_1 of the field element A with the form of Eq. (12). For the quadratic polynomial, a_0 and a_1 are the results after mapping. Take the IFFMe$^+$ as an example. The recursive process starts from Step 1 and finishes at Step 5 in Alg. 7. The post-processing step can be finally executed after the iterative process to reduce the computation consumption. If $n > 1$, the post-processing can be further simplified. The multi-precision based IFFMe$^+$ (Multi-IFFMe$^+$) algorithm is shown in Alg. 9 as an example. This scheme can be also applied to other proposed algorithms in the same way.

4 Hardware Architecture

4.1 Top-Level Architecture

The top-level architecture is shown in Fig. 1, where all the proposed algorithms are covered, and the variables and

TABLE 1
Estimations of the Normalized Numbers of $N + N$ Additions and $N \times N$ Multiplications for Different Algorithms

Algorithms	MontM [25]	BarM [26]	EFFM [21]	FFM1 [16]	FFM2 [16]	**IFFM$^-$**	**IFFMo$^+$**	**IFFMe$^+$**
moduli family	generic	generic	$2 \cdot 2^{2x}b^{2y} - 1$	$2 \cdot 2^{2x}b^{2y} - 1$	$f \cdot 2^x b^y \pm 1$	$2 \cdot 2^{2x}b^{2y} - 1$	$2 \cdot 2^{2x}b^{2y} + 1$	$2^{2x}b^{2y} + 1$
Norm. $(N + N)$	3	3	9	9	3.5	10	11	8.5
Norm. $(N \times N)$	3	4	2	2	3	1.625	1.625	1.625

Algorithm 9: The proposed multi-precision based IFFMe$^+$ modular multiplication algorithm.

Input: $A = a_1R + a_0 = \sum_{j=0}^{n-1} a_{1,j} \cdot (2^k)^j \cdot R + \sum_{j=0}^{n-1} a_{0,j} \cdot (2^k)^j$, $B = b_1R + b_0$.

1) The first tentative computing:

$c_1 = 0, c_0 = 0$.
for $j = n - 1$ to 0 **do**
  Multiplication items:
  $m_1 = a_{1,j}b_1, m_2 = a_{0,j}b_0,$
  $m_3 = (a_{1,j} + a_{0,j})(b_1 + b_0)$
  Results:
  $c_1 = (m_3 - m_1 - m_2) + c_1 \cdot 2^k$
  $c_0 = (m_2 - m_1) + c_0 \cdot 2^k$

2) The second tentative computing:

  Reduction:
  $[q_0, r_0] = IBR^+(c_0, R)$
  $[q_1, r_1] = IBR^+(c_1 + q_0, R)$
  Results:
  $c_1 = r_1, c_0 = r_0 - q_1$
end for

3) Post processing:

$[c_1, c_0] = post\_proc(c_1, c_0)$
Output: $C = A \times B \bmod p = c_1R + c_0$.

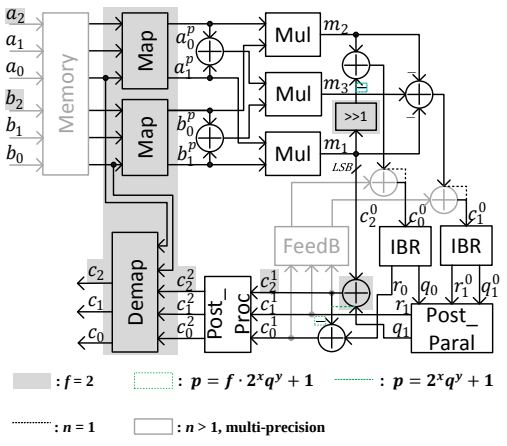


Fig. 1. The top-level architecture for the proposed algorithms including the IFFM$^-$, IFFMo$^+$, IFFMe$^+$, Multi-IFFM$^-$, Multi-IFFMo$^+$, and Multi-IFFMe$^+$.

data flow are labeled. The connecting lines and diagrams of common parts are shown in black. The parts in shaded area are used for the algorithms with $f = 2$, including the IFFM$^-$, IFFMo$^+$, and their multi-precision based algorithms. The subtracters in the green dotted box are adopted for the algorithms with $p = f \cdot 2^x b^y + 1$. When $f = 1$,

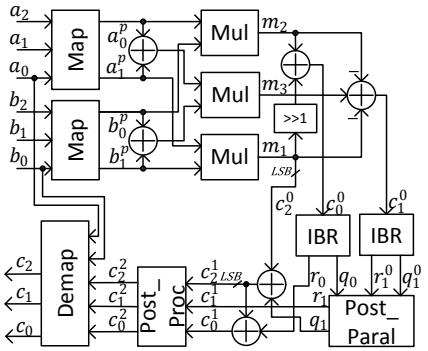


Fig. 2. The proposed top-level architecture for the IFFM$^-$.

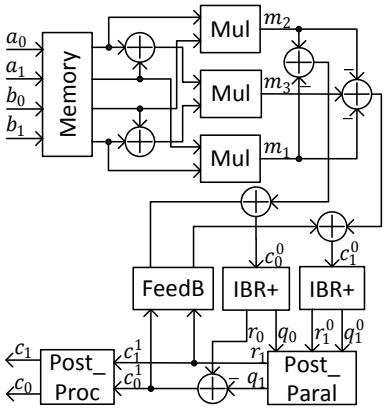


Fig. 3. The proposed top-level architecture for the Multi-IFFMe$^+$.

the green dotted line is connected. Modules in light gray, including the *Memory*, *FeedB*, and two adders, are used for multi-precision based designs. Totally, this figure covers six designs: the IFFM$^-$, IFFMo$^+$, IFFMe$^+$, Multi-IFFM$^-$, Multi-IFFMo$^+$, and Multi-IFFMe$^+$ modular multipliers.

To make Fig. 1 easier to read, we split it and give two of them as examples shown in Figs. 2 and 3. Fig. 2 shows the top-level architecture for the IFFM$^-$ algorithm, which is a completely feed-forward design. The red dashed lines illustrate the inserted pipeline stages to increase the clock speed, which would not increase the average required cycles due to the adopted overlapping processing schedule. Fig. 3 exhibits the top-level architecture for the Multi-IFFMe$^+$ algorithm, which is iteratively processed. In these designs with the multi-precision scheme, the data width of each module can be easily adjusted by the parameter n to make a tradeoff between the speed and the resource consumption. Since the input coefficients cannot be pushed into the computation modules at one time, the memory is necessary to save them, and it is also used to buffer the other groups of inputs to

accelerate the design. More details for scheduling will be introduced at the end of this section.

From the top-level architectures, we can see that besides the explicit adders, seven submodules are used: 1) *Map*; 2) *Mul*; 3) *IBR/IBR*$^+$; 4) *Post_Paral*; 5) *Post_Proc*; 6) *Demap*; and 7) *FeedB*. They will be detailed in the following. Since the modules except the *FeedB* module for the multi-precision based algorithms are almost the same as their originals, they will not separately be discussed below for brevity.

4.2 Proposed Submodules

4.2.1 Mapping & Demapping Modules

The mapping and demapping are used for the IFFM$^-$ and IFFMo$^+$ algorithms with $f = 2$ to transfer the three coefficients into two coefficients and vice versa, respectively. As the subtraction operation is implemented by using the addition operation with two's complement format of the minuend in hardware, we can reformulate Eq. (2) for $a_2 = 1$ part to reduce the complexity:

$$\begin{aligned} a_i^p &= R - a_i - 1 = R + (a_i)_{comp} - 1 \\ &= R + ((a_i)_{inv} + 1) - 1 \\ &= R + (a_i)_{inv},\ i = \{1, 0\},\ a_2 = 1, \end{aligned} \tag{15}$$

where the indexes *comp* and *inv* denote the two's and ones' complements of a_i, respectively. For *map*$^+$ function, the constant addend is replaced by $R+2$. By using this method, the critical path can be effectively reduced. The proposed architecture of the *map* module is presented in Fig. 4(a), where the constant R in blue is for *map* function and the $R + 2$ in green is for *map*$^+$ function. The *demap* module is

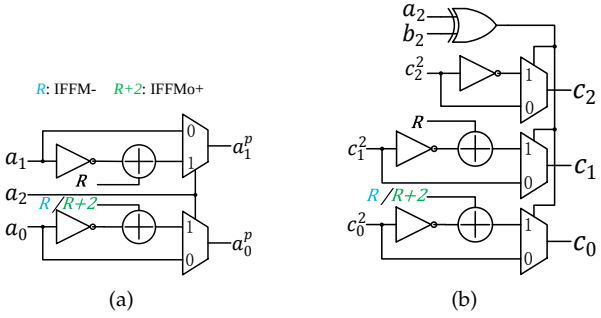


Fig. 4. The *map* and *demap* modules.

to make the final results back to normal. The architecture of this module shown in Fig. 4(b) is almost the same as that of the *map* module except the control signals and one more output for c_2.

4.2.2 IBR/IBR$^+$ & Post Processing Modules

The *IBR* module is to implement the reduction using the method presented as Alg. 5. The architecture is shown in Fig. 5, where the data widths of the data flow are marked and the parameter α is assumed as $2N$. After the positive integer c is input, its N_1 LSBs are saved in c_L and the other $2N - N_1$ bits are saved in c_H and sent into the following steps (Step 1 of Alg. 5). Firstly, c_H is multiplied by the constant parameter λ using a constant multiplier *cMul 0* and the tentative quotient q_0 is obtained by taking the $N + 1$

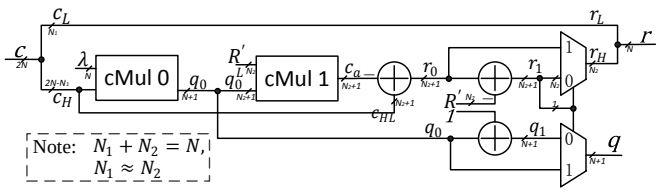


Fig. 5. The *IBR* module.

MSBs of the product (Step 2 of Alg. 5). Then, the $N_2 + 1$ LSBs of q_0 is multiplied by the parameter R' in the second constant multiplier *cMul 1* and the N_2+1 LSBs of the output is denoted as c_a (Step 3 of Alg. 5). Thirdly, the $N_2 + 1$ LSBs of c_H is subtracted by c_a to get the tentative remainder r_0. It should be pointed out that this subtraction has covered the computations from Step 4 to Step 7 of Alg. 5, as these steps are exactly the definition of two's complement. Fourthly, the quotient q and the partial remainder r_H are selected from the corresponding tentative values and their corrections based on whether r_0 is larger than R' (Steps 8 to 10 of Alg. 5). Note that the comparator is omitted and the sign bit of r_0 is used as the control signal. At last, the final remainder r is obtained by directly assembling r_L and r_H together. For the *IBR*$^+$ function, two multiplexers controlled by the sign bit of c and two adders are needed for the output of the *IBR* module.

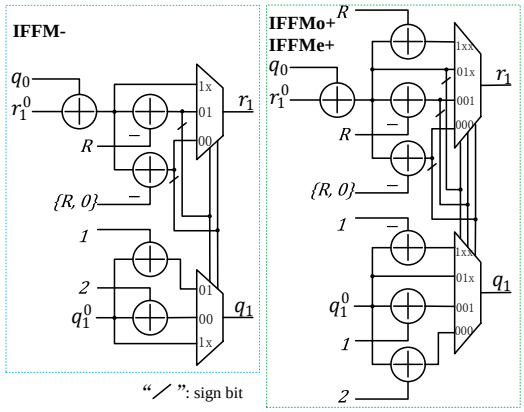


Fig. 6. The *Post_Paral* module.

As introduced in [21], the two *IBR* functions can be processed in parallel with some extra computations to reduce the whole latency. We follow this idea and further develop it for our proposed algorithms. For the IFFM$^-$, as $q_0 + r_1^0$ ranges in $[0, \frac{5}{2}R - 4]$, subtracting R is needed at most twice for r_1. For the IFFMo$^+$ and IFFMe$^+$, the ranges of $q_0 + r_1^0$ are $[-\frac{R}{2}, 2R + 1]$ and $[-R + 1, 2R + 1]$, respectively. Note that though they are different, the required operations, one addition and two subtractions, are the same. The architectures for the three algorithms are presented in Fig. 6. To reduce the data path, we arrange the adders in parallel as much as possible. It can be seen that the comparators are thoroughly removed by using the sign bits to select the right answer. The critical path is only two adders and two multiplexers for IFFM$^-$, and two adders and three multiplexers for IFFMo$^+$ and IFFMe$^+$.

4.2.3 Post Processing Module

This module is to satisfy the constraints of the output for the proposed algorithms. According to Alg. 8 for *post_proc* functions, the corresponding architectures can be devised as shown in Fig. 7. The data paths are optimized by parallelizing the adders and precomputing the constant parameters. The critical paths are two adders and one multiplexer for the IFFM$^-$, two adders and two multiplexers for the IFFMo$^+$, and one adder and four multiplexers for the IFFMe$^+$.

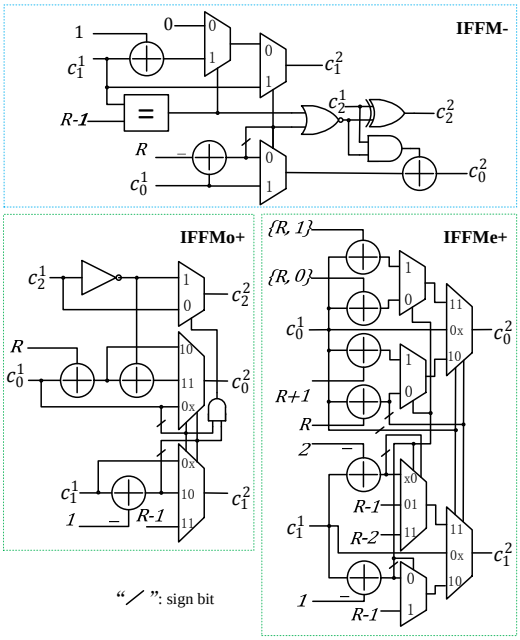


Fig. 7. The *Post_Proc* module.

4.2.4 Multipliers

Multipliers occupy the most hardware resources and they are usually located in the critical path. Since a constant multiplier consumes much fewer resources and has a shorter critical path than a normal multiplier under the same conditions, we design them separately in our work, but the basic optimization ideas are identical. If we multiply two integers directly, the critical path will be too long to be accepted because of the large input data width. Two strategies are adopted for multipliers in our designs: multi-precision and Karatsuba decomposition. Both of them can efficiently reduce the data width of a multiplier. Theoretically, the Karatsuba decomposition can unlimitedly reduce the complexity of multipliers. However, the other overheads usually quickly increase along with the growing orders. The number of orders of such decomposition is specifically designed for each multiplier, usually three or four orders used in our following designs. We have carefully devised them and made a good tradeoff between speed and resource consumption.

4.2.5 FeedB Module for Multi-Precision Based Algorithms

According to Steps 4 and 5 of Alg. 9, two k-bit left shifters are required for the outputs of last iteration. Therefore, for the Multi-IFFMe$^+$, the *FeedB* module is composed of two k-bit left shifters. For the Multi-IFFM$^-$ and Multi-IFFMo$^+$, the *FeedB* modules are a little different from that of the Multi-IFFMe$^+$ as three instead of two inputs are put into this module. We can deduce the formulas as:

$$\begin{aligned} c_2 &= ((m_1 \bmod 2) + c_2 \cdot 2^k) \bmod 2 = m_1 \bmod 2, \\ c_1 &= (m_3 - m_1 - m_2) + \mathbf{c_1} \cdot \mathbf{2^k}, \\ c_0 &= (m_2 \pm \lfloor \frac{m_1}{2} \rfloor) + c_0 \cdot 2^k \pm \frac{c_2}{2} \cdot 2^k \\ &= (m_2 \pm \lfloor \frac{m_1}{2} \rfloor) + (\mathbf{2c_0} \pm \mathbf{c_2}) \cdot \mathbf{2^{k-1}}, \end{aligned}$$

where the plus sign is for Multi-IFFM$^-$ and minus sign for Multi-IFFMo$^+$. It can be seen from the bold term that the *FeedB* module for them is made up of one 1-bit left shifter, one adder, and one $(k-1)$-bit left shifter, and one k-bit left shifter.

4.3 Processing Schedule

For high-speed applications, we should increase the clock frequency and reduce the required clock cycles (CCs). The parallel processing and pipelining are both adopted in our designs to increase the speed. Many efforts have been made to optimize the data path of each module as shown above. Several pipeline stages are inserted such that the proposed designs can achieve a reasonable clock frequency. To further enhance the throughput, the overlapping scheme is also used to make the hardware resources fully utilized. Assuming the number of pipeline stages is m, m pairs of inputs can simultaneously be computed in mn CCs. Figure 8 illustrates the proposed processing schedule, in which A_i^j denotes the j-th input A's i-th digit for $1 \le i \le n$ and $1 \le j \le m$. Though the latency for one pair of inputs is mn CCs, the average cost cycles for it is only n CCs. The throughput can be computed as:

$$throughput = \frac{m \cdot N \cdot f_{clk}}{mn} = \frac{N \cdot f_{clk}}{n}, \qquad (16)$$

where f_{clk} is the clock frequency and N is the output data width.

5 Implementation Results

To compare with conventional implementations, the proposed designs are coded with RTL and implemented on FPGA. Since the complexity of IFFMo$^+$ is almost the same as that of IFFM$^-$, the implementations are mainly focused on the IFFM$^-$ and IFFMe$^+$, and their multi-precision versions. The adopted SIDH-friendly prime moduli are $p_{771} = 2 \cdot 2^{386}3^{242} - 1$ provided in [11] for IFFM$^-$ and $p_{752} = 2^{394}5^{154} + 1$ in [23] for IFFMe$^+$, targeting the 128-bit post-quantum security level. We will show more implementation details for them in the following.

5.1 FPGA Implementation

The Xilinx Vivado 2016.4 EDA platform is adopted. Since the FFM1 and FFM2 in [16] are tested on the Kintex-7 xc7k325tffg900-2 board, we implement the multi-precision based algorithms on this board for a fair comparison. Constrained by the resource limitation, the algorithms without the multi-precision scheme are not implemented on this device. We select the Virtex-7 xc7vx690tffg1157-3 board, which

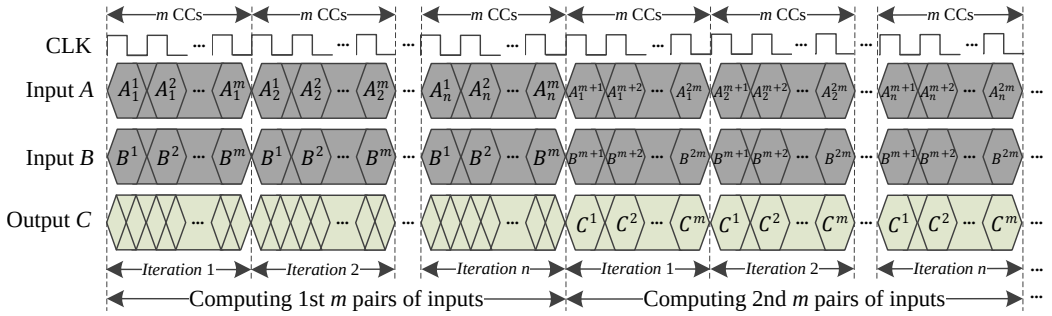


Fig. 8. The proposed processing schedule.

TABLE 2
Comparisons of Modular Multipliers for 128-bit Post-Quantum Security Level Implementing on FPGA

Algorithms	EFFM [21]	FFM1 [16]	FFM2 [16]	**Multi-IFFM$^-$**	**IFFM$^-$**	**Multi-IFFMe$^+$**	**IFFMe$^+$**
Prime[1]	p_{771}	p_{771}	p_{771}	p_{771}	p_{771}	p_{752}	p_{752}
Platform[2]	Virtex-6	Kintex-7	Kintex-7	Kintex-7/Virtex-7	Virtex-7	Kintex-7/Virtex-7	Virtex-7
FFs	11,924	9,675	11,635	12,902	38,976	9,882	25,055
LUTs	12,790	16,627	33,051	25,743	63,173	27,131/27,132	67,293
DSPs	-	122	529	210	729	216	549
BRAMs	-	-	-	302	574	290	561
f_{clk} (MHz)	31	55	25	57/56	60	61/62	52
CCs	236	64	28	7	1	8	1
Time (ns)	7,613	1,164	1,120	122/124	17	131/128	19
Throughput (Mb/s)	101	663	688	6,278/6,168	46,260	5,734/5,828	39,104

[1] $p_{771} = 2 \cdot 2^{386}3^{242} - 1$ and $p_{752} = 2^{394}5^{154} + 1$.
[2] The device type of Virtex-6 is xc6vcx240t-2ff784 and it contains 301,440 FFs, 150,720 LUTs, 416 BRAMs, and 768 DSPs. The device type of Kintex-7 is xc7k325tffg900-2 and it contains 407,600 FFs, 203,800 LUTs, 445 BRAMs, and 840 DSPs. The device type of Virtex-7 is xc7vx690tffg1157-3 and it contains 866,400 FFs, 433,200 LUTs, 1,470 BRAMs, and 3,600 DSPs.

TABLE 3
Calculation of Inserted Pipeline Stages m, Iterations n, and Latency for the Proposed Algorithms on FPGA

Algorithms		**Multi-IFFM$^-$**	**IFFM$^-$**	**Multi-IFFMe$^+$**	**IFFMe$^+$**
FPGA	m	10	18	11	17
	n	7	1	8	1
	Latency	70	18	88	17

is used by Koziel *et al.* for SIDH protocol implementation in [14] and [15], and implement all the proposed algorithms on it for comparisons. Table 3 shows the numbers of inserted pipelines, iterations, and the total latency of the proposed designs on FPGA. The comparisons with prior arts are provided in Table 2. Note that the Multi-IFFM$^-$ and Multi-IFFMe$^+$ implemented on the Kintex-7 consume almost as many resources as on the Virtex-7 FPGA boards. Besides, the clock frequencies on the two boards are also with a very small difference though the resource proportions by using the Kintex-7 are larger than those by using the Virtex-7. Since our multipliers require much fewer CCs and have higher clock frequency than the conventional ones, it is clear that each of the proposed designs consumes much less time than the prior art. We also provide the throughput based on Eq. (16) in this table. We can see that our designs without and with the multi-precision scheme achieve more than 60 times and about 10 times faster throughput than the state-of-the-art FFM2 design, respectively. However, the introduced extra hardware resources are relatively small. For example, compared to the best design FFM2, the multi-precision based designs consume some extra BRAMs but fewer FFs, LUTs, and DSPs, and meanwhile for the designs without the multi-precision scheme, except the extra BRAMs, the increasing ratio in other resources is only about 2-3 times, far smaller than the increase in speed.

6 Conclusion

In this paper, we have proposed three low-complexity modular multiplication algorithms called IFFM$^-$, IFFMo$^+$, and IFFMe$^+$, and their corresponding multi-precision versions for the SIKE protocol. Six new architectures were presented based on these algorithms. By incorporating the smart formula transformation, novel architectural optimization, and maximum overlapping processing, the proposed designs demonstrate significant advantages over conventional ones. We believe these achievements will greatly contribute to the practicability of this protocol.

7 Appendix
Deduction for the Range of c_0^o for Proposed Algorithms

7.1 For the IFFM$^-$

The deduction is based on the assumption that the input numbers A and B are the field elements in $\mathbb{F}_p$, where $p = 2R^2 - 1$. According to Alg. 6, after mapping, the coefficients of A and B are still in the normal range. After the first tentative computing in Step 5, we can compute the

ranges of the coefficients of C with the upper and lower limits as:

$$c_2 \in \{0,1\},\ 0 \le c_1 \le 2(R-1)^2,\ 0 \le c_0 \le \frac{3}{2}(R-1)^2.$$

With the first *IBR* function, c_0 is decomposed as q_0 and r_0 in Step 6, whose ranges are $[0, \frac{3}{2}R-3]$ and $[0, R-1]$, respectively, and thus we have

$$0 \le c_1 + q_0 \le 2R^2 - \frac{5}{2}R - 1 = 2R^2 - 3R + (\frac{1}{2}R - 1).$$

After the second *IBR* function in Step 7, we have

$$0 \le q_1 \le 2R-3,\ 0 \le r_1 \le R-1.$$

Since t ranges in $[0, 2R-2]$, the term $\lfloor \frac{t}{2} \rfloor$ has the range of $[0, R-1]$. Therefore, we can deduce the range of c_0^o for the IFFM$^-$ as $[0, 2R-2]$.

7.2 For the IFFMo$^+$

Similar to the IFFM$^-$, we can begin our deduction after the mapping. The first tentative results have the ranges as:

$$c_2 \in \{0,1\},\ 0 \le c_1 \le 2(R^2-1),\ -\frac{(R-1)^2}{2} \le c_0 \le (R+1)^2.$$

By using the first *IBR*$^+$ function, r_0 is in $[0, R-1]$ and q_0 is in $[-\frac{R}{2}, R+2]$. Note that these arithmetic operations aforementioned are monotonous so we can directly operate the maximum and minimum values to obtain the upper and lower limits. But for the formula $c_1 + q_0$, the monotonicity is broken by the subtraction and the assumptions for the coefficients of A and B are not the same when computing the two variables' upper and lower limits. When calculating the lower limit of this formula, we should take the minimums of c_1 and q_0 into consideration. The lower limit of q_0 is achieved by assuming a_0 or b_0 equal to zero and $a_1 = b_1 = R-1$. When $a_0 = b_0 = 0$, the lower limit of c_1 can also be obtained. So we can get the lower limit for $c_1 + q_0$ as $-\frac{R}{2}$ when $a_0 = b_0 = 0$ and $a_1 = b_1 = R-1$. However, the upper limit of q_0 is achieved by setting a_0 and b_0 to $R+1$ and a_1 or b_1 to zero, while that of c_1 is obtained when $a_0 = b_0 = R+1$ and $a_1 = b_1 = R-1$. Since the q_0 is the quotient of c_0 divided by R, obviously, satisfying the assumptions for c_1 can obtain the largest value of $c_1 + q_0$. Hence, the upper limit of $c_1 + q_0$ is $2R^2 + \frac{R}{2} + 1$. After computing by the second *IBR*$^+$ function, we have

$$-1 \le q_1 \le 2R,\ 0 \le r_1 \le R-1.$$

Then the formula $\lfloor \frac{q_1 + c_0}{2} \rfloor$ ranges in $[-1, R]$. Therefore, we can obtain the range of c_0^o for the IFFMo$^+$ as $[-R, R]$.

7.3 For the IFFMe$^+$

The IFFMe$^+$ does not need the mapping and demapping process. We can refer to the deduction for the IFFMo$^+$. In Step 2 of Alg. 7, we have the constraints as:

$$c_2 \in \{0,1\},\ 0 \le c_1 \le 2(R^2-1),\ -(R-1)^2 \le c_0 \le (R+1)^2.$$

In Step 3, the ranges for q_0 and r_0 are

$$-R+1 \le q_0 \le R+2,\ 0 \le r_0 \le R-1.$$

According to the analysis above, the range of $c_1 + q_0$ becomes $[-R+1, 2R^2+2]$. After the second *IBR*$^+$ function in Step 4, we obtain

$$-1 \le q_1 \le 2R,\ 0 \le r_1 \le R-1.$$

Therefore, we can have the range of c_0^o for the IFFMe$^+$ as $[-2R, R]$.

REFERENCES

[1] B. Bauer, D. Wecker, A. J. Millis, M. B. Hastings, and M. Troyer, "Hybrid quantum-classical approach to correlated materials," *Physical Review X*, vol. 6, no. 3, p. 031045, 2016.

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

[3] R. L. Rivest, A. Shamir, and L. 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 *Conference on the theory and application of cryptographic techniques*. Springer, 1985, pp. 417–426.

[5] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[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*. US Department of Commerce, National Institute of Standards and Technology, 2016.

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

[8] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the fujisaki-okamoto transformation," in *Theory of Cryptography Conference*. Springer, 2017, pp. 341–371.

[9] S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti, "On the security of supersingular isogeny cryptosystems," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2016, pp. 63–91.

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

[11] L. De Feo, D. Jao, and J. Plût, "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies," *Journal of Mathematical Cryptology*, vol. 8, no. 3, pp. 209–247, 2014.

[12] D. Jao and V. Soukharev, "Isogeny-based quantum-resistant undeniable signatures," in *International Workshop on Post-Quantum Cryptography*. Springer, 2014, pp. 160–179.

[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*. ACM, 2016, pp. 1–10.

[14] 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.

[15] 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.

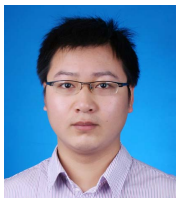
[16] W. Liu, J. Ni, Z. Liu, C. Liu, and M. O'Neill, "Optimized modular multiplication for supersingular isogeny Diffie-Hellman," *IEEE Transactions on Computers*, pp. 1–1, 2019.

[17] 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.

[18] 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*. Springer, 2018, pp. 37–51.

[19] 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*. Springer, 2019, pp. 215–231.

[20] 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.
[21] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Efficient finite field multiplication for isogeny based post quantum cryptography," in *International Workshop on the Arithmetic of Finite Fields*. Springer, 2016, pp. 193–207.
[22] C. Costello, P. Longa, and M. Naehrig, "SIDH/SIKE library," *https://github.com/Microsoft/PQCrypto-SIDH*, 2016-2019.
[23] J. Bos and S. Friedberger, "Arithmetic considerations for isogeny based cryptography," *IEEE Transactions on Computers*, 2018.
[24] J. Tian, J. Lin, and Z. Wang, "Ultra-fast modular multiplication implementation for isogeny-based post-quantum cryptography," in *2019 IEEE Workshop on Signal Processing Systems (SiPS)*, accepted to be published.
[25] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.
[26] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.



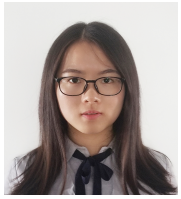
Jun Lin received the B.S. degree in physics and the M.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2007 and 2010, respectively, and the Ph.D. degree in electrical engineering from the Lehigh University, Bethlehem, in 2015. From 2010 to 2011, he was an ASIC design engineer with AMD. During summer 2013, he was an intern with Qualcomm Research, Bridgewater, NJ. In June 2015, he joined the school of electronic science and engineering of Nanjing University, where he is an associate professor. He is a member of the Design and Implementation of Signal Processing Systems (DISPS) Technical Committee of the IEEE Signal Processing Society.

His current research interests include low-power high-speed VLSI design, specifically VLSI design for digital signal processing and cryptography. He was a co-recipient of the Merit Student Paper Award at the IEEE Asia Pacific Conference on Circuits and Systems in 2008. He was a recipient of the 2014 IEEE Circuits & Systems Society (CAS) student travel award.

Zhongfeng Wang received both B.E. and M.S. degrees from Tsinghua University, Beijing, China. He obtained the Ph.D. degree from the Department of Electrical and Computer Engineering at the University of Minnesota, Minneapolis in Aug. 2000. He recently joined Nanjing University as a distinguished professor after serving Broadcom Corp. as a leading VLSI architect for nearly nine years. From 2003 to 2007, he was an assistant professor in the School of EECS at Oregon State University, Corvallis. Even earlier, he was working for National Semiconductor Corporation.

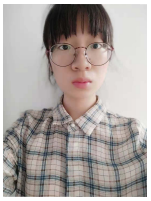
Dr. Wang is a world-recognized expert on VLSI for Forward Error Correction Codes. He has published over one hundred technical papers, edited one book ("VLSI") and filed tens of U.S. patent applications and disclosures. He was the recipient of the IEEE Circuits and Systems (CAS) Society VLSI Transactions Best Paper Award in 2007. In the current record (2007-present), he has had five papers ranked among top twenty most downloaded manuscripts in IEEE Trans. on VLSI Systems. During his tenure at Broadcom, he has contributed significantly on 10Gbps and beyond high-speed networking products. Additionally, he has made critical contributions in defining FEC coding schemes for 100Gbps and 400Gbps Ethernet standards. So far, his technical proposals have been adopted by many networking standard specs.

Since 2004, Dr. Wang has served as Associate Editor for the IEEE Trans. on Circuits and Systems I (TCAS-I), TCAS-II, and IEEE Trans. on VLSI Systems for many terms. He has also served as technical program committee member, session chair, track chair, and review committee member for many IEEE and ACM conferences. In 2013, he served in the Best Paper Award selection committee for the IEEE Circuits and System Society. His current research interests are in the area of VLSI for High-Speed Signal Processing Systems. He is a Fellow of IEEE.



Jing Tian received her B.S. degree in microelectronics from Nanjing University, Nanjing, China, in 2015. She is currently working toward the M.S. and Ph.D. integrated degree with information and communication engineering from Nanjing University. Her research interests include VLSI design for digital signal processing and cryptographic engineering.

Zhe Liu received the Ph.D. degree from the Laboratory of Algorithmics, Cryptology and Security (LACS), University of Luxembourg, Luxembourg. He is a professor with Nanjing University of Aeronautics and Astronautics, China. His Ph.D. thesis has received the prestigious FNR Awards 2016 Outstanding Ph.D. Thesis Award for his contributions in cryptographic engineering on IoT devices. He is the recipient of ACM China SIGSAC Rising Star Award and Honorable Mentions for ACM China Rising Star Award in 2017. His research interests include computer arithmetic and cryptographic engineering. He has co-authored more than 70 research peer-reviewed journal and conference papers. He is a senior member of the IEEE.



Binjing Li received the B.S. degree in electronic information science and technology from Nanjing University, Nanjing, China, in 2019, where she is currently pursuing the M.S. degree in integrated circuit engineering. Her research interests include VLSI design and efficient hardware architectures.