Optimized Algorithms and Architectures for Montgomery Multiplication for Post-quantum Cryptography

Rami El Khatib[1](✉), Reza Azarderakhsh[1](✉), and Mehran Mozaffari-Kermani[2](✉)

[1] Florida Atlantic University, Boca Raton, FL, USA
{relkhatib2015,razarderakhsh}@fau.edu

[2] University of South Florida, Tampa, FL, USA
mehran2@usf.edu

Abstract. Finite field multiplication plays the main role determining the efficiency of public key cryptography systems based on RSA and elliptic curve cryptography (ECC). Most recently, quantum-safe cryptographic systems are proposed based on supersingular isogenies on elliptic curves which require large integer multiplications over extended prime fields. In this work, we present two Montgomery multiplication architectures for special primes used in a post-quantum cryptography system known as supersingular isogeny key encapsulation (SIKE). We optimize two existing Montgomery multiplication algorithms and develop area-efficient and time-efficient Montgomery multiplication architectures for hardware implementations of post-quantum cryptography. Our proposed time-efficient architecture is 32% to 42% faster than the leading one (depending on the prime size) available in the literature which has been used in original SIKE submission to the NIST standardization process. The area-efficient architecture is 42% to 50% smaller than the counterparts and is about 3% to 11% faster depending on the NIST security level.

Keywords: Hardware architectures · Isogeny-based cryptosystems · Montgomery multiplication · Post-quantum cryptography

1 Introduction

Post-quantum cryptography (PQC) refers to the research of cryptographic primitives (usually public-key cryptosystems) that are not efficiently breakable using quantum computers. Most notably, Shor's algorithm [1] can be efficiently implemented on a quantum computer to break standard Elliptic Curve Cryptography (ECC) and RSA cryptosystems. There exist some alternatives secure against quantum computing threats [2], such as lattice-based cryptosystems, hash-based signatures, code-based cryptosystems, multivariate public key cryptography, and isogeny-based cryptography [3].

Y. Mu et al. (Eds.): CANS 2019, LNCS 11829, pp. 83–98, 2019.
https://doi.org/10.1007/978-3-030-31578-8_5

Isogeny-based cryptography or more specifically supersingular isogeny Diffie-Hellman (SIDH) key exchange has been proposed by Jao and De Feo [4] as an alternative to Elliptic Curve Diffie-Hellman (ECDH) resistant to Shor's quantum attack. A more secure model of SIDH a.k.a. SIKE (supersingular isogeny key encapsulation) has been submitted to NIST standardization process [3]. SIKE computations constitute an algebraic map between supersingular elliptic curves, which appear to be resistant to quantum attacks. Existing results on the hardware implementations of SIKE have appeared in [5–8]. SIKE's lower level computations are mainly over $\mathbb{F}_{p^2}$ or extended prime fields. The prime size which decides the security of SIKE determines the size of arithmetic unit for lower level multiplication, addition, squaring, and inversion. Among these operations, multiplication plays the main role determining the performance of SIKE cryptosystem. Therefore, efficient and high-performance implementations of the multiplier is crucial. In comparison to the other post-quantum candidates, SIKE offers smallest key size for the same security level which is more attractive for bandwidth-constrained applications. However, SIKE is not the fastest quantum-safe candidate, and its performance still needs to be improved as stated in NIST submission [3].

In this paper, we focus on the optimization of arithmetic operations employed in SIKE and propose two new hardware architectures for modular multiplication algorithm targeting SIKE primes based on the well-known Montgomery modular multiplication algorithm [9]. Previous work on hardware implementation of Montgomery multiplication has been proposed in [10,11] for arbitrary primes and in [5,8] for SIKE primes. Since the primes employed in SIKE (SIKEp434, SIKEp503, SIKEp610, and SIKEp751 for NIST level-1, -2, -3, -5, respectively) have special forms, we developed a time-efficient implementation and an area-efficient implementation of Montgomery multiplication to be used in future work of SIKE. The time-efficient implementation reduces the latency and the area usage compared to previous work, maintaining high frequency while the area-efficient implementation significantly reduces the area.

Our Contributions

- We optimize two existing Montgomery multiplication algorithms for special primes used in post-quantum cryptography, SIKE.
- We provide efficient hardware architecture for the proposed Montgomery multiplication algorithms.
- We evaluate time and area performance of the proposed hardware architecture benchmarked on FPGA and compare with counterparts.

The organization of the paper is as follows. In Sect. 2, we discuss the Montgomery modular multiplication algorithm and two algorithms: Coarsely Integrated Operand Scanning (CIOS) and Finely Integrated Operand Scanning (FIOS) algorithms that perform Montgomery multiplication word-by-word. In Sect. 3, we provide optimization techniques for the CIOS and FIOS Montgomery multiplication algorithms. In Sect. 4, we propose efficient hardware architectures of the proposed Montgomery multiplier algorithms. In Sect. 5, we implement the

Algorithm 1. Montgomery Multiplication [9]

```
   Input  : $p < 2^K$, $R = 2^K$, $p' = -p^{-1} \bmod R$, $a, b < p$
   Output: $a \cdot b \cdot R^{-1} \bmod p$
1  $T \leftarrow a \cdot b$
2  $m \leftarrow T \cdot p' \mod R$
3  $T \leftarrow (T + m \cdot p)/R$
4  if $T > m$ then return $T - p$
5  return $T$
```

proposed hardware architectures on FPGA, provide area and timing results, and compare with counterparts available in the literature. Finally, in Sect. 6, we give our final thoughts and discuss future work.

2 Preliminaries: Montgomery Multiplication

Modular multiplication (i.e. $a \times b \mod p$) of large integers (especially the ones used in SIKE) can be efficiently implemented using Montgomery multiplication. Montgomery multiplication [9] avoids the division operation which is difficult to implement in an efficient way in hardware. Montgomery multiplication has been used in recent hardware implementations of isogeny-based cryptography including SIDH [5–7] and SIKE [12].

2.1 Montgomery Multiplication Algorithm

Montgomery multiplication performs modular multiplication by transforming the division by p into division by a power of 2, which is a simple shift. The overhead cost of Montgomery multiplication is the need to convert the inputs into the Montgomery domain, then perform all arithmetic operations in the Montgomery domain, and finally convert back to the ordinary domain. For simple applications with few modular multiplications, this conversion would be expensive. However, in SIKE, this is extremely useful because of its high dependence on a large number of modular multiplications.

Montgomery multiplication algorithm (MontMult) takes two inputs a and b with the remaining inputs constants and produces a single output $\text{MontMult}(a, b) = a \cdot b \cdot R^{-1} \mod p$ where R and p are co-prime. By taking R as a power of 2, the division becomes a simple shifting. Algorithm 1 shows the original Montgomery multiplication algorithm. The first part (line 1) performs the multiplication step while the second part (lines 2–4) performs the reduction step. Note that the same algorithm can be used to convert between the ordinary and Montgomery domain [9]. The conversion into the Montgomery domain can be done using $\text{MontMult}(a, R^2 \mod p)$ where a is the input in the ordinary domain and the conversion into the ordinary domain can be done using $\text{MontMult}(a, 1)$ where a is the input in the Montgomery domain.

The final subtraction step (line 4) can be removed from the algorithm and performed once at the very end after converting to the ordinary domain. However, in this case, the output from MontMult, which is going to be the input for subsequent MontMult, is $<2p-1$ instead of $<p$. Therefore, the algorithm needs to work for any input $<2p-1$. The condition $(T + m \cdot p)/R < 2p - 1$ needs to be satisfied so that the output from line 3 is $<2p-1$. If R is taken such that it is 2 bits larger than the size of p, then this condition is satisfied.

Hardware implementations of MontMult for public key cryptography have been studied in [5,10,11,13–21].

2.2 SIKE Primes

In SIKE submission for post-quantum cryptography [12], the primes employed have a special form where Montgomery multiplication can be optimized. These primes are given in Table 1. The main advantage of the primes used in SIKE is that the least significant bits of the prime are all 1s. This form can be utilized in Montgomery multiplication algorithm variants that perform word by word computation such as the Separated Operand Scanning (SOS), Coarsely Integrated Operand Scanning (CIOS) or Finely Integrated Operand Scanning (FIOS) algorithms [22]. In these word-by-word variants, $p'_0 = p' \mod 2^w = -p^{-1} \mod 2^w$, where w is the number of bits in a word, can be used instead of p' [23]. When more than 2 words are used for SIKE primes, $p'_0 = 1$ and $m = T \cdot p'_0 \mod R = T \mod R$ in line 2 becomes a simple copy register.

Table 1. SIKE primes for post-quantum cryptography based on NIST standardization process [3]

Prime form	Classical/quantum security	Public key size (bytes)
$p_{434} = 2^{216}3^{137} - 1$	NIST level 1	330
$p_{503} = 2^{250}3^{159} - 1$	NIST level 2	378
$p_{610} = 2^{305}3^{192} - 1$	NIST level 3	462
$p_{751} = 2^{372}3^{239} - 1$	NIST level 5	564

2.3 Coarsely Integrated Operand Scanning (CIOS) Montgomery Multiplication

The CIOS Montgomery multiplication algorithm [22] is a method that performs word-by-word Montgomery multiplication by alternating between the multiplication and reduction steps. The inputs a and b and prime p are split into s words of w-bit wide each. CIOS is shown in Algorithm 2. As seen, lines 4–9 perform the multiplication step while lines 11–18 perform the reduction step. Hardware implementations of CIOS have been carried out in [10] by Mrabet *et al.* and in [13] by McIvor *et al.*

Algorithm 2. CIOS Montgomery Multiplication Algorithm [22]

```
   Input  : $p < 2^K$, $R = 2^K$, $w$, $s$, $K = w \cdot s$, $p' = -p^{-1} \bmod 2^w$, $a, b < p$
   Output: MontMult$(a, b)$
 1 $T \leftarrow 0$
 2 for $i \leftarrow 0$ to $s-1$ do
 3     $C \leftarrow 0$
 4     for $j \leftarrow 0$ to $s-1$ do
 5         $(C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
 6         $T[j] \leftarrow S$
 7     $(C, S) \leftarrow T[s] + C$
 8     $T[s] \leftarrow S$
 9     $T[s+1] \leftarrow C$
10
11     $m \leftarrow T[0] \cdot p' \mod 2^w$
12     $(C, S) \leftarrow T[0] + m \cdot p[0]$
13     for $j \leftarrow 1$ to $s-1$ do
14         $(C, S) \leftarrow T[j] + m \cdot p[j] + C$
15         $T[j-1] \leftarrow S$
16     $(C, S) \leftarrow T[s] + C$
17     $T[s-1] \leftarrow S$
18     $T[s] \leftarrow T[s+1] + C$
19 return $T$
```

2.4 Finely Integrated Operand Scanning (FIOS) Montgomery Multiplication

The FIOS Montgomery multiplication algorithm [22] is a method that performs word-by-word Montgomery multiplication by performing the multiplication and reduction steps in the same loop. Similar to CIOS, the inputs a,b and prime p are split into s words of w-bit each. FIOS is shown in Algorithm 3. In FIOS, the first multiplication and m must be computed (3–6) before performing the remaining multiplication and reduction steps (lines 7–15). The main difference between FIOS and CIOS is that in FIOS, the multiplication and reduction can be parallelized while in CIOS, the reduction has to wait for the multiplication step.

Hardware implementation of FIOS has been conducted by McIvor *et al.* in [13]. It has been shown that FIOS performed slower than CIOS. The main reason for this is the need for carry propagation units (lines 4 and 9). To address, we will show in this paper that the carry propagation can be eliminated for 1-bit larger registers which adds minimal cost in hardware implementations and improves the critical path delay.

Algorithm 3. FIOS Montgomery Multiplication Algorithm [22]

```
Input  : $p < 2^K$, $R = 2^K$, $w$, $s$, $K = w \cdot s$, $p' = -p^{-1} \bmod 2^w$, $a, b < p$
Output: $\mathrm{MontMult}(a, b)$
1  $T \leftarrow 0$
2  for $i \leftarrow 0$ to $s - 1$ do
3      $(C, S) \leftarrow T[0] + a[i] \cdot b[0]$
4      $ADD(t(1), C)$
5      $m \leftarrow S \cdot p' \mod 2^w$
6      $(C, S) \leftarrow S + m \cdot p[0]$
7      for $j \leftarrow 1$ to $s - 1$ do
8          $(C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
9          $ADD(T[j + 1], C)$
10         $(C, S) \leftarrow S + m \cdot p[j]$
11         $T[j - 1] \leftarrow S$
12     $(C, S) \leftarrow T[s] + C$
13     $T[s - 1] \leftarrow S$
14     $T[s] \leftarrow T[s + 1] + C$
15     $T[s + 1] \leftarrow 0$
16 return $T$
```

3 Proposed Finite Field Multiplier Algorithms

In this section, based on the information provided in the previous section, we propose an optimized CIOS and FIOS Montgomery multiplication algorithms for primes employed in SIKE.

3.1 Optimized CIOS (O-CIOS) Montgomery Multiplication Algorithm

We propose a new optimized coarsely integrated operand scanning multiplier (O-CIOS) for SIKE which requires less hardware units as shown in Algorithm 4. We show that by taking R 3 bits larger than the size of the prime p, we only require $s+1$ registers. First, lines 11–12 in the original algorithm (Algorithm 2) are replaced by lines 9–10 for $s > 2$ since $p' = 1$ (as shown in Subsect. 2.2) and $p[0] = 2^w - 1$. Proposition 1 shows how lines 7–9 and 16–18 can be replaced by lines 7 and 14, respectively, for $w > 2$.

Proposition 1. *In Algorithm 2, lines 7–9 and 16–18 can be can be replaced by lines 7 and 14 in Algorithm 4, respectively, for* $w > 2$.

Proof. For $i = 0$, iteration $j = s - 1$ in line 5 has output $\leq (2^w - 1)(2^{w-2} - 1) + (2^w - 1) = 2^{w-2}(2^w - 1) \leq 2^{2w-2} - 1$ which implies $C \leq 2^{w-2} - 1 \leq 2^{w-1} - 1$. Therefore, $T[s] \leq 2^{w-1} - 1$ and $T[s+1] = 0$ in lines 8 and 9. Iteration $j = s - 1$ in line 5 has output $\leq (2^w - 1) + (2^w - 1)(2^{w-3} - 1) + (2^w - 1) = (2^{w-3} + 1)(2^w - 1) \leq 2^{w-2}(2^w - 1) \leq 2^{2w-1} - 1$ for $w > 2$ which implies

Algorithm 4. Optimized CIOS Montgomery Multiplication Algorithm for SIKE primes

```
   Input  : $p < 2^{K-3}$, $R = 2^K$, $w > 2$, $s > 2$, $K = w \cdot s$, $a, b < 2p - 1$
   Output: MontMult$(a, b)$
 1 $T \leftarrow 0$
 2 for $i \leftarrow 0$ to $s - 1$ do
 3     $C \leftarrow 0$
 4     for $j \leftarrow 0$ to $s - 1$ do
 5         $(C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$
 6         $T[j] \leftarrow S$
 7     $T[s] \leftarrow C$
 8
 9     $m \leftarrow T[0]$
10     $C \leftarrow T[0]$
11     for $j \leftarrow 1$ to $s - 1$ do
12         $(C, S) \leftarrow T[j] + m \cdot p[j] + C$
13         $T[j - 1] \leftarrow S$
14     $T[s - 1] \leftarrow T[s] + C$
15 return $T$
```

$C \leq 2^{w-1} - 1$. Therefore, $T[s-1] \leq 2^{w-1} - 1$ and $T[s] = 0$ in lines 17 and 18. Now, proving for iteration $i = k$ where $k > 0$, iteration $j = s - 1$ in line 5 has output $\leq (2^w - 1) + (2^w - 1)(2^{w-2} - 1) + (2^w - 1) = (2^{w-2} + 1)(2^w - 1) \leq 2^{w-1}(2^w - 1) \leq 2^{2w-1} - 1$ for $w > 1$ which implies $C \leq 2^{w-1} - 1$. Therefore, $T[s] \leq 2^{w-1} - 1$ and $T[s+1] = 0$ in lines 8 and 9. Iteration $j = s - 1$ in line 5 has output $\leq (2^w - 1) + (2^w - 1)(2^{w-3} - 1) + (2^w - 1) = (2^{w-3} + 1)(2^w - 1) \leq 2^{w-2}(2^w - 1) \leq 2^{2w-1} - 1$ for $w > 2$ which implies $C \leq 2^{w-1} - 1$. Therefore, $T[s-1] \leq 2^{w-1} - 1$ and $T[s] = 0$ in lines 17 and 18. This complete the proof.

3.2 Optimized FIOS (O-FIOS) Montgomery Multiplication Algorithm

We also propose an optimized FIOS algorithm for Montgomery multiplication as shown in Algorithm 5. Similar to CIOS, lines 5 and 6 can be modified since $p' = 1$ and $p[0] = 2^w - 1$ for $s > 2$. As for the carry propagation in lines 4 and 9, they can be directly integrated inside the other carry C in lines 6 and 10. However, the carry C must use a register of size $w + 1$ bits to accommodate the extra accumulated bits. The changes are shown in lines 4–5 and line 7 in the optimized algorithm. We notice that the two multiplications in line 7 can be performed in the same cycle in parallel and without the need to propagate the result of the first multiplication which will decrease architecture complexity and routing delays. Proposition 2 shows how lines 12–15 can be replaced by 9 for $w > 2$.

Algorithm 5. Optimized FIOS Montgomery Multiplication Algorithm for SIKE primes

```
   Input  : $p < 2^{K-2}$, $R = 2^K$, $w > 2$, $s > 2$, $K = w \cdot s$, $a, b < 2p - 1$
   Output: $\mathrm{MontMult}(a, b)$
1  $T \leftarrow 0$
2  for $i \leftarrow 0$ to $s - 1$ do
3      $(C, S) \leftarrow T[0] + a[i] \cdot b[0]$
4      $m \leftarrow S$
5      $C \leftarrow C + S$
6      for $j \leftarrow 1$ to $s - 1$ do
7          $(C, S) \leftarrow T[j] + a[i] \cdot b[j] + m \cdot p[j] + C$
8          $T[j - 1] \leftarrow S$
9      $T[s - 1] \leftarrow \mathrm{LSW}(C)$
10 return $T$
```

Proposition 2. *In Algorithm 3, lines 12–15 can be replaced by line 9 in Algorithm 5 for $w > 2$.*

Proof. For $i = 0$, iteration $j = s - 1$ in line 7 in new algorithm has output $(\leq (2^w-1)(2^{w-1}-1)+(2^w-1)(2^{w-2}-1)+(2^w-1) = (2^{w-1}+2^{w-2}+1)(2^w-1) \leq 2^w(2^w-1) = 2^{2w}-1$ for $w > 1$ which implies $C \leq 2^w - 1$. Therefore, $T[s-1] \leq 2^w - 1$ and $T[s] = 0$. Now, proving for iteration $i = k$ where $k > 0$, iteration $j = s-1$ in line 7 in new algorithm has output $(\leq (2^w-1)+(2^w-1)(2^{w-1}-1)+(2^w-1)(2^{w-2}-1)+(2^w-1) = (2^{w-1}+2^{w-2}+2)(2^w-1) \leq 2^w(2^w-1) = 2^{2w}-1$ for $w > 2$ which implies $C \leq 2^w - 1$. Therefore, $T[s-1] \leq 2^w - 1$ and $T[s] = 0$. This complete the proof.

4 Proposed Efficient Architecture for O-CIOS and O-FIOS Montgomery Multiplication Algorithms

In this section, we propose a hardware architecture design for each of the new optimized algorithms O-CIOS and O-FIOS discussed in the previous section. The O-CIOS design focuses on minimizing area usage while the design for O-FIOS focuses on maximizing the frequency and minimizing the total multiplication time.

4.1 Proposed O-CIOS Architecture

The proposed O-CIOS architecture is illustrated in Fig. 1 which mainly improves the area usage in comparison to the ones adopted before for hardware implementations. The architecture is composed of several processing elements (PEs) cascaded to perform the multiplication and reduction steps as can be seen in

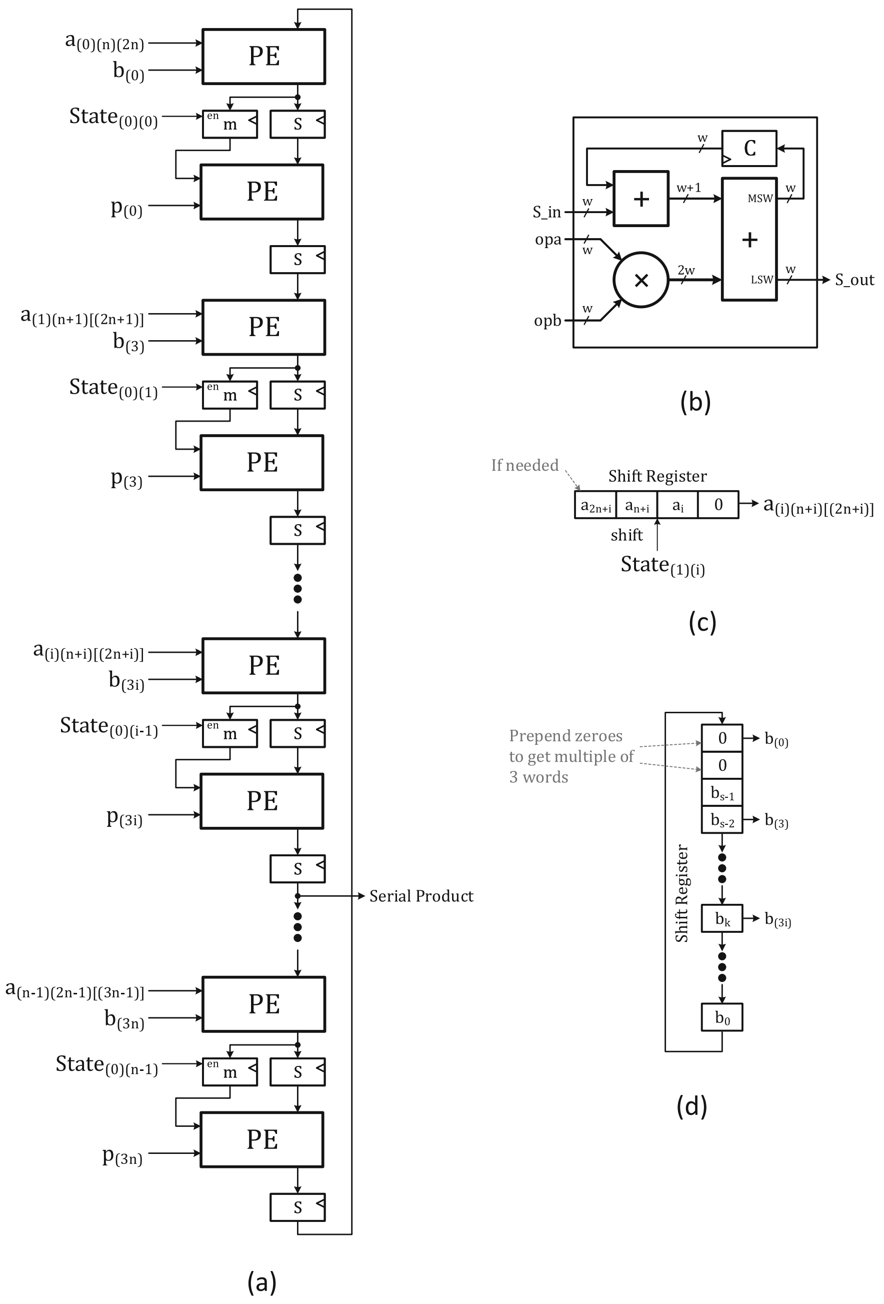

Fig. 1. (a) Proposed O-CIOS architecture. (b) Processing element. (c) Design for input b and p. (d) Design for input a.

Fig. 1(a). Each PE performs a multiplication and an addition in parallel followed by an addition corresponding to lines 5 and 12 in Algorithm 4 as shown in Fig. 1(b). Notice that unlike Mrabet's design [10], no final PEs for multiplication and reduction are required since lines 7 and 14 are respectively similar to lines 5 and 12 with some registers set to 0. In addition, line 10 can be integrated in the reduction PE (line 12) since $T[0] + m \cdot p[0] = T[0] \cdot 2^w$ which implies $C = T[0]$ before performing the first iteration.

Each *odd* PE performs the entire first inner loop (lines 4–6) and line 7 (multiplication step) of one outer iteration while each *even* PE performs entire second inner loop (lines 11–13) and line 14 (reduction step) of one outer iteration. There is one fan-out needed in the output of the *multiplication* PE to store m for the *reduction* PE.

Once the PE is finished with processing one iteration, it processes another iteration, which can be seen by the loop-back after the last PE. There is a delay of 3 cycles between two consecutive multiplication or reduction PEs corresponding to one multiplication, one reduction, and computing m cycles. Therefore, to minimize the number of cycles, the number of PEs used is $\lfloor (s + 1)/3 \rfloor$. This means that each PE performs 2 or 3 iterations. Thus, each PE uses 2 or 3 different words of input a as can be seen in Fig. 1(c). Words from inputs b and p rotate across each PE as shown in Fig. 1(d). The number of cycles required for this architecture to compute Montgomery multiplication is $4s$ cycles. For instance, for p434, O-CIOS requires 112 clock cycles.

4.2 O-FIOS Architecture

The proposed O-FIOS architecture is illustrated in Figs. 2 and 3. As one can see, Fig. 2(a) illustrates the proposed systolic architecture based on PEs. This architecture focuses on improving timing results by parallelizing the multiplication and reduction steps. The initial PE (Fig. 2(b)) computes m and the first carry for each iteration (lines 3–5 in Algorithm 5). The remaining PEs (Fig. 3(c)) perform two consecutive iterations of the inner loop (lines 7–8). Therefore, each PE processes two words of input b and prime p following Fig. 3(d) and outputs the two words of the output following Fig. 3(e). Words for input a are pushed serially using a shift register (Fig. 3(f)) into the initial PE and propagated to the next PE after two cycles corresponding to the two consecutive iterations mentioned earlier. Similarly, m is propagated through each PE after being evaluated in the initial PE.

For the last line 9 of the algorithm, we tried feeding back the carry output of the last PE into its sum input. However, this has caused a routing delay in the FPGA we are using outweighing the cycle saved in the process. Therefore, we have decided that for *even* s, the last PE can process the last line by grounding the second b_{in}, second p_{in}, and S_{in} while in *odd* s, a simple register is used to store the result before fed-back into S_{in} of the last PE. The number of cycles required for this architecture to compute Montgomery multiplication is $3s$ cycles. For example, in p434, O-FIOS requires 84 cycles.

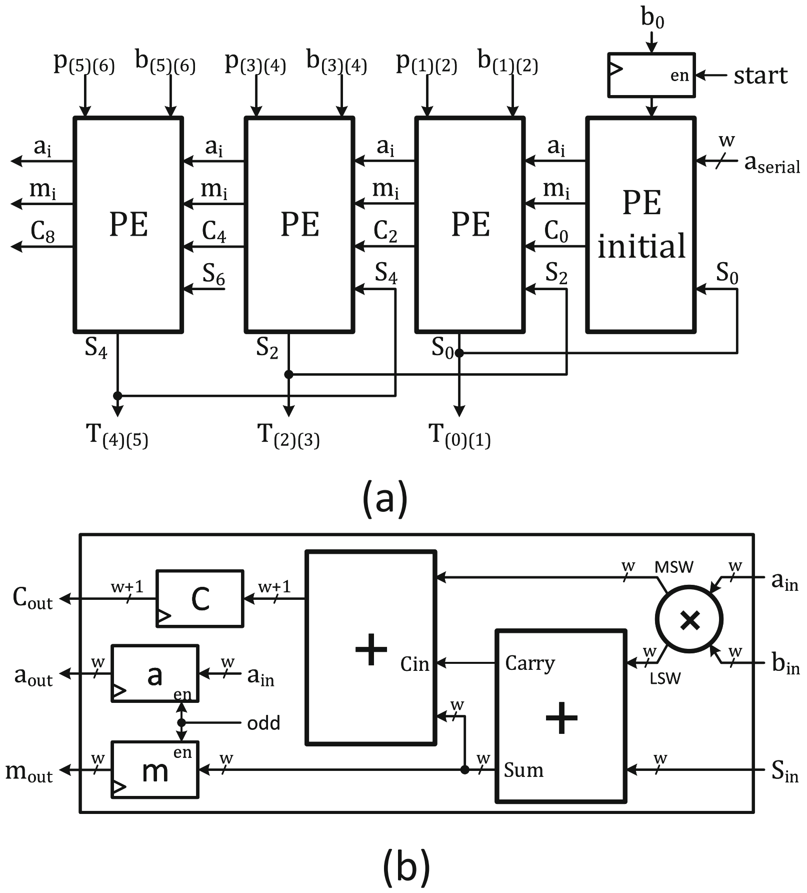

Fig. 2. (a) Proposed O-FIOS architecture. (b) Initial processing element.

4.3 Time Complexity Analysis

Table 2 provides a time complexity comparison between our O-CIOS and O-FIOS implementations and different Montgomery multiplication implementations. For a fair comparison, we have optimized Mrabet *et al.*'s implementation [13] for SIKE primes by changing the β-cell into a simple register since $p^{-1} = 1$. As for Koziel *et al.*'s implementation [5,11], we used a non-interleaved version of the multiplier. Our proposed O-CIOS uses less number of clock cycles while maintaining the same critical path delay of Mrabet's CIOS. Our proposed O-FIOS uses the least number of clock cycles of any design while maintaining the same critical path delay of Koziel's implementation. However, in the next section, our results show that O-FIOS perform at the same frequency of Mrabet's CIOS mainly because our design has minimal routing delays. Furthermore, our designs require less area which we will show in the next section after implementing in hardware.

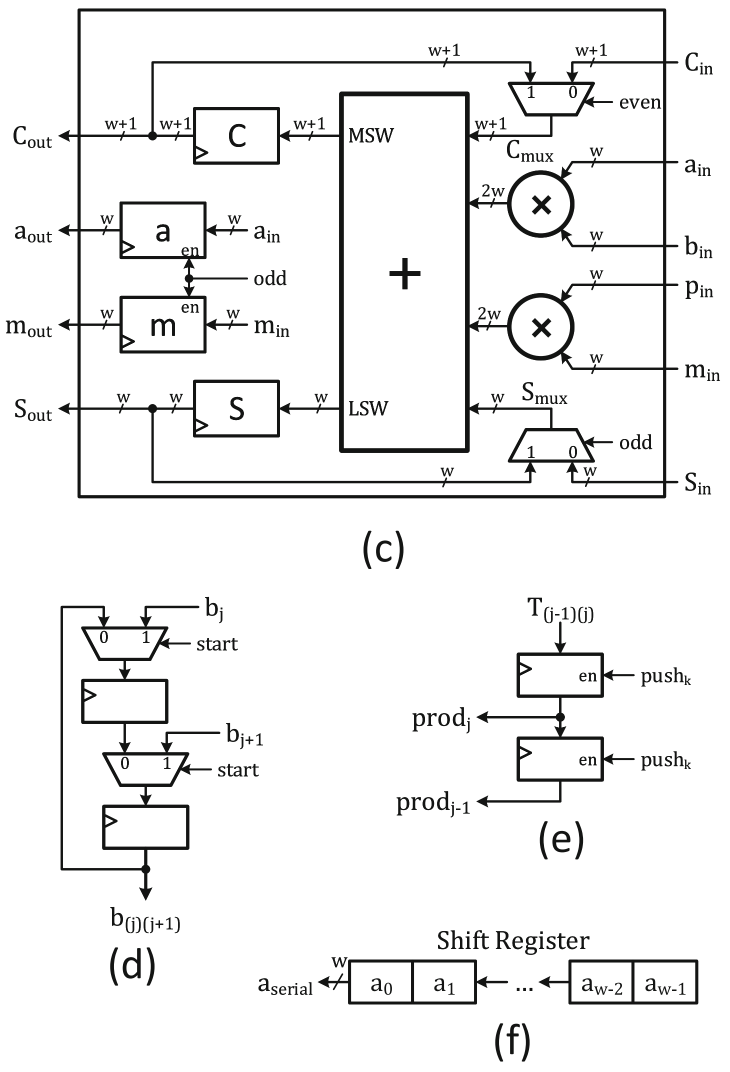

Fig. 3. Remaining components of O-FIOS architecture. (c) Single processing element. (d) Design for input b and p. (e) Design for input a. (f) Design for output.

5 Implementation Results

In this section, we are going to provide implementation results for the proposed Montgomery multiplication architectures, O-CIOS and O-FIOS, discussed in the previous sections. The implementations are performed in Xilinx Vivado 2018.2 for Xilinx Virtex-7 FPGA xc7vx690tffg1157-3. Table 2 reports area and timing results for O-CIOS and O-FIOS. As one can see, for NIST level-1, our proposed O-CIOS architecture operates in 233.5 MHz and occupies 770 Flip-flops, 1869

Table 2. Time complexity comparison of 448-bit Montgomery multipliers for SIKEp434. $T_{16\times}$ indicates the critical path of a 16-bit×16-bit multiplication. T_{32+} indicates the critical path of a 32-bit addition.

Work	Critical path delay	Latency (cc)
Mrabet *et al.* [10] ($w = 16$)	$T_{16\times} + T_{32+}$	114
McIvor/Koziel *et al.* [5,11] ($w = 16$)	$T_{16\times} + 2T_{32+}$	87
This work O-CIOS ($w = 16$)	$T_{16\times} + T_{32+}$	112
This work O-FIOS ($w = 16$)	$T_{16\times} + 2T_{32+}$	84

Table 3. Implementation results and comparison of proposed O-FIOS and O-CIOS Montgomery multiplication architectures on a Xilinx Virtex-7 FPGA device, xc7vx690tffg1157-3

Prime	NIST level	Area				Time			Area × Time
		# FFs	# LUTs	# DSPs	# Slices	Freq. (MHz)	Latency (cc)	Total time (ns)	(×1000)
This work O-CIOS ($w = 16$)									
p_{434}	1	770	1869	**40**	**447**	233.481	112	479.696	214.424
p_{503}	2	851	2094	**44**	**519**	216.685	128	590.720	306.584
p_{610}	3	1075	2609	56	**646**	217.297	156	717.912	463.771
p_{751}	5	1309	3188	68	**761**	219.635	192	874.176	665.248
This work O-FIOS ($w = 16$)									
p_{434}	1	1119	1905	43	607	271.370	**84**	**309.540**	**187.891**
p_{503}	2	1290	2219	49	554	**267.380**	**96**	**359.040**	**198.908**
p_{610}	3	1568	2704	61	835	**267.380**	**117**	**437.580**	**365.379**
p_{751}	5	1794	3308	73	967	232.829	**144**	**618.480**	**598.070**
Mrabet *et al.* [10]* ($w = 16$)									
p_{434}	1	3492	3737	40	2959	**273.973**	114	416.100	1,231.240
p_{503}	2	3884	4240	44	3334	251.130	129	513.678	1,486.911
p_{610}	3	4741	5148	**54**	4041	247.158	159	643.314	1,831.187
p_{751}	5	5814	6288	**66**	4970	**245.459**	195	794.430	2,267.751
McIvor/Koziel *et al.* [5,11]* ($w = 16$)									
p_{434}	1	687	3177	84	895	160.694	87	541.401	484.554
p_{503}	2	784	3641	96	1044	162.470	99	609.345	636.156
p_{610}	3	952	4040	117	1315	162.101	120	740.280	973.468
p_{751}	5	1168	4193	144	1310	159.974	147	918.897	1,203.755

∗Note that the original paper does not have the results for these primes. The numbers are based on our implementations using this work.

LUTs, 40 DSPs and 447 slices. The total time to perform one Montgomery multiplication is 480 ns in O-CIOS. On the other hand, O-FIOS operates at 271.4 MHz and occupies 1119 FFs, 1905 LUTs, 43 DSPs and 607 slices for NIST level-1. The total time to perform one Montgomery multiplication is 310 ns in O-FIOS (Table 3).

5.1 Comparison and Discussion

Table 2 compares our results with two different implementations; Mrabet *et al.* [10] and non-interleaved Koziel *et al.* [5,11]. Our O-FIOS is 22% to 31% faster than Mrabet's implementation and 32% to 42% faster than Koziel's implementation (a non-interleaved version of the one used in SIKE). In addition, the O-FIOS architecture uses less area compared to other implementations (other than O-CIOS). Our O-CIOS focuses on minimizing the area usage while maintaining high frequency and low total time. For example, in p434, the number of slices used in O-CIOS is 447 which is $2\times$ smaller than Koziel's implementation at 895 slices and $6.5\times$ smaller than Mrabet's implementation at 2959 slices. This implementation of O-CIOS is slightly slower (9% to 13%) than Mrabet's CIOS and slightly faster (3% to 11%) than Koziel's implementation.

6 Conclusion

In this paper, we discussed two optimized Montgomery multiplication algorithms O-CIOS and O-FIOS for SIKE primes. We then developed an architecture for each algorithm. The O-CIOS architecture focuses on minimizing the area usage while the O-FIOS architecture focuses on minimizing the total time. For performance evaluation and comparison, we implemented our proposed architectures in FPGA and showed area and timing results.

The Montgomery multiplication architectures developed in this paper show great potential in increasing the performance of SIKE. Our future work is to develop an optimized version of SIKE and employ the two Montgomery multiplier architectures developed in this paper.

Acknowledgement. The authors would like to thank the reviewers for their comments. This work is supported in parts by NSF CNS-1801341, NIST-60NANB16D246, and ARO W911NF-17-1-0311.

References

1. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, FOCS 1994, pp. 124–134 (1994)
2. Chen, L., et al.: Report on Post-Quantum Cryptography (2016). NIST IR 8105
3. Jao, D., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization Project (2019)
4. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
5. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M., Jao, D.: Post-quantum cryptography on FPGA based on isogenies on elliptic curves. IEEE Trans. Circuit. Syst. I: Regul. Pap. **64**(1), 86–99 (2017)

6. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M.: Fast hardware architectures for supersingular isogeny Diffie-Hellman key exchange on FPGA. In: Dunkelman, O., Sanadhya, S.K. (eds.) INDOCRYPT 2016. LNCS, vol. 10095, pp. 191–206. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49890-4_11
7. Koziel, B., Azarderakhsh, R., Mozaffari-Kermani, M.: A high-performance and scalable hardware architecture for isogeny-based cryptography. IEEE Trans. Comput. **67**(11), 1594–1609 (2018)
8. Roy, D.B., Mukhopadhyay, D.: Post quantum ecc on fpga platform. Cryptology ePrint Archive, Report 2019/568 (2019). https://eprint.iacr.org/2019/568
9. Montgomery, P.L.: Modular multiplication without trial division. Math. Comput. **44**(170), 519–521 (1985)
10. Mrabet, A., et al.: High-performance elliptic curve cryptography by using the CIOS method for modular multiplication. In: Cuppens, Frédéric, Cuppens, Nora, Lanet, Jean-Louis, Legay, Axel (eds.) CRiSIS 2016. LNCS, vol. 10158, pp. 185–198. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54876-0_15
11. McIvor, C., McLoone, M., McCanny, J.V.: High-radix systolic modular multiplication on reconfigurable hardware. In: IEEE International Conference on Field-Programmable Technology, pp. 13–18 (2005)
12. Jao, D., et al.: Supersingular isogeny key encapsulation. Submission to the NIST Post-Quantum Standardization Project (2017)
13. McIvor, C., McLoone, M., McCanny, J.V.: FPGA Montgomery multiplier architectures - a comparison. In: 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 279–282, April 2004
14. Alrimeih, H., Rakhmatov, D.: Fast and flexible hardware support for ECC over multiple standard prime fields. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **22**(12), 2661–2674 (2014)
15. Blum, T., Paar, C.: High-radix montgomery modular exponentiation on reconfigurable hardware. IEEE Trans. Comput. **50**(7), 759–764 (2001)
16. Chen, G., Bai, G., Chen, H.: A high-performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit. IEEE Trans. Circ. Syst. II: Express Briefs **54**(5), 412–416 (2007)
17. Eberle, H., Gura, N., Shantz, S.C., Gupta, V., Rarick, L., Sundaram, S.: A public-key cryptographic processor for RSA and ECC. In: Proceedings of 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 98–110, September 2004
18. Ghosh, S., Alam, M., Roy Chowdhury, D., Sen Gupta, I.: Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks. Comput. Electr. Eng. **35**, 329–338 (2009)
19. Sakiyama, K., Mentens, N., Batina, L., Preneel, B., Verbauwhede, I.: Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems. In: Bertels, K., Cardoso, J.M.P., Vassiliadis, S. (eds.) ARC 2006. LNCS, vol. 3985, pp. 347–357. Springer, Heidelberg (2006). https://doi.org/10.1007/11802839_43
20. Tenca, A.F., Koç, Ç.K.: A scalable architecture for montgomery nultiplication. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 94–108. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_10
21. Vliegen, J., et al.: A compact FPGA-based architecture for elliptic curve cryptography over prime fields. In: ASAP 2010–21st IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 313–316, July 2010

22. Kaya Koc, C., Acar, T., Kaliski, B.S.: Analyzing and comparing montgomery multiplication algorithms. IEEE Micro **16**(3), 26–33 (1996)
23. Dussé, S.R., Kaliski, B.S.: A cryptographic library for the motorola DSP56000. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 230–244. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-46877-3_21