Bit-Serial and Bit-Parallel Montgomery Multiplication and Squaring over $GF(2^m)$

Arash Hariri, *Student Member*, *IEEE*, and Arash Reyhani-Masoleh, *Member*, *IEEE*

Abstract—Multiplication and squaring are main finite field operations in cryptographic computations and designing efficient multipliers and squarers affect the performance of cryptosystems. In this paper, we consider the Montgomery multiplication in the binary extension fields and study different structures of bit-serial and bit-parallel multipliers. For each of these structures, we study the role of the Montgomery factor, and then by using appropriate factors, propose new architectures. Specifically, we propose two bit-serial multipliers for general irreducible polynomials, and then derive bit-parallel Montgomery multipliers for two important classes of irreducible polynomials. In this regard, first we consider trinomials and provide a way for finding efficient Montgomery factors which results in a low time complexity. Then, we consider type-II irreducible pentanomials and design two bit-parallel multipliers which are comparable to the best finite field multipliers reported in the literature. Moreover, we consider squaring using this family of irreducible polynomials and show that this operation can be performed very fast with the time complexity of two XOR gates.

Index Terms—Montgomery multiplication, squaring, finite (or Galois) fields, bit-serial, bit-parallel, trinomials, pentanomials.

1 Introduction

Finite fields have an important role in cryptographic algorithms. Among the arithmetic operations which are performed over finite fields, multiplication and squaring are the most important ones. Other arithmetic operations such as inversion and exponentiation can be performed using multiplication and squaring. The multiplication in finite field has been extensively considered in the literature; see, for example, [1], [2], [3], [4], [5], and [6]. They cover a wide variety of cases regarding different basis representations (e.g., polynomial basis (PB), normal basis, etc.), irreducible polynomials (e.g., trinomials, pentanomials, etc.), and the architecture (e.g., bit-serial, digit-serial, bit-parallel, etc.).

Montgomery multiplication (MM) algorithm has been proposed in [7] for fast modular integer multiplication. In [5], Koç and Acar have introduced a class of algorithms for bit-serial, digit-serial, and bit-parallel Montgomery multiplication over binary extension fields. They have proposed that by choosing the Montgomery factor $r = x^m$, the multiplication can be efficiently implemented in hardware and software. The Montgomery multiplication is used to design an Elliptic Curve Cryptography (ECC) based cryptoprocessor in [8]. Also, it is implemented with a semisystolic array structure in [9]. In [10], another semisystolic array structure is designed for the Montgomery multiplication which uses $r = x^m$. A digit-serial Montgomery multiplication algorithm is proposed in [11] which is based on the algorithm proposed in [5] and the polynomial basis multiplication. Also in the literature, some scalable architectures are proposed for the Montgomery multiplication over finite fields, e.g., [12], [13], and [14]. In [15], the Montgomery multiplication is implemented using systolic arrays for all-one polynomials and trinomials. A new Montgomery factor has been considered by Wu in [16] for the Montgomery multiplication. His design is based on the method proposed in [5] and has shown that choosing the middle term of the irreducible trinomial $F(z) = z^m + z^k + 1$ as the Montgomery factor, i.e., $r = x^k$, results in more efficient bit-parallel multipliers and squarers. Although the Montgomery multiplication is suitable for designing scalable and versatile multipliers, according to [5] and [16], the important advantage of the Montgomery multiplication over $GF(2^m)$ is its low time complexity. In this paper, we provide more results to support this advantage.

Our objective in this work is to reduce the time complexity of Montgomery multipliers and squarers so as to accelerate scalar multiplication in ECC, which is included in the recent standards such as FIPS 186-2, ANSI X9.62, and IEEE 1363-2000. To achieve this, we use a different approach to formulate the Montgomery multiplication, and then, we study different Montgomery factors to find the most efficient ones. We begin by presenting two new bit-serial algorithms and their hardware architectures, and then, by unfolding one of the algorithms, we design a new general bit-parallel multiplication architecture which is different from the architecture proposed in [5] and [16]. Due to the popularity of irreducible trinomials and pentanomials in cryptography, we optimize our general architecture using efficient Montgomery factors for faster implementation. Finally, we design an efficient squarer for a family of irreducible pentanomials. Note that ECC is typically implemented with a fixed field size (e.g., [17], [18], and [19]) using the recommendations by NIST [20] for Elliptic Curve Digital Signature Algorithm (ECDSA). Therefore, to avoid any area, time, or power overheads, we design our multipliers assuming that the field size is fixed.

Bit-serial multipliers provide the lowest possible area complexity. In the literature, the bit-serial algorithms have been studied for the polynomials basis and two different

• *The authors are with the Department of Electrical and Computer Engineering, Faculty of Engineering, The University of Western Ontario, 1151 Richmond Street North, Thompson Engineering Building, London, ON N6A 5B9, Canada. E-mail: hariri@ieee.org, areyhani@eng.uwo.ca.*

Manuscript received 26 Apr. 2008; revised 3 Feb. 2009; accepted 23 Feb. 2009; published online 29 Apr. 2009.
Recommended for acceptance by E. Antelo.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-04-0179.
Digital Object Identifier no. 10.1109/TC.2009.70.

algorithms have been proposed, namely the Least Significant Bit (LSB) first and the Most Significant Bit (MSB) first bit-serial algorithms [21]. The bit-serial Montgomery multiplication algorithm proposed in [5] is an LSB-first bit-serial algorithm which uses $r = x^m$ as the Montgomery factor. In this paper, we propose two bit-serial Montgomery multipliers and a new Montgomery factor to reduce their time complexities.

The main objective of designing bit-parallel multipliers which makes them different from bit-serial and digit-serial multipliers (e.g., [2] and [22]) is to provide the lowest possible time complexity. In this paper, we study two classes of irreducible polynomials. The first class is the irreducible trinomials ($F(z) = z^m + z^k + 1$). For this class, we prove that two Montgomery factors result in an efficient hardware implementation, where their complexity results match the best results reported in the literature for different bit-parallel finite field multipliers including [6], [23], [3], and [16]. In this paper, we also consider the irreducible pentanomials and show that type-II irreducible pentanomials defined in [4] are very suitable for our general bit-parallel architecture. In this regard, we propose two Montgomery factors which result in very efficient implementations. Then, we propose two different bit-parallel Montgomery multipliers for this class of irreducible polynomials and compare their complexities with the ones of recent bit-parallel multipliers. We show that our results outperform the existing Montgomery multipliers in the literature. Finally, we consider squaring over $GF(2^m)$ and present a squarer for type-II irreducible pentanomials. The proposed squarer has the constant delay of two exclusive-or (XOR) gates which is the lowest reported delay for squaring using pentanomials.

The rest of the paper is organized as follows: In Section 2, we provide some background information. In Section 3, we introduce two new bit-serial algorithms as well as a new Montgomery factor. In Section 4, we consider a new general formulation of bit-parallel Montgomery multipliers and study it for two special cases of irreducible polynomials, namely irreducible trinomials and irreducible pentanomials in Sections 5 and 6, respectively. In Section 7, we consider squaring over binary extension fields. In Section 8, we present our comparison results, and finally, we conclude this paper in Section 9.

2 Preliminaries

In this section, we briefly introduce some basic concepts about Galois fields and the Montgomery multiplication over $GF(2^m)$.

2.1 Finite Fields

$GF(2^m)$ is a kind of finite field [24] that contains 2^m different elements. This finite field is an extension of $GF(2)$ which contains 0 and 1. The extended binary field, $GF(2^m)$, is associated with an irreducible polynomial of degree m over $GF(2)$, i.e.,

$$F(z) = f_m z^m + f_{m-1} z^{m-1} + \cdots + f_1 z + f_0,\ f_i \in GF(2). \quad (1)$$

Assuming x is a root of $F(z)$, i.e., $F(x) = 0$, each element of $GF(2^m)$ can be represented as a polynomial of degree $m-1$ over $GF(2)$, i.e., $A \in GF(2^m) \Leftrightarrow A = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$, where $a_i \in \{0, 1\}, i \in [0, m-1]$.

Algorithm 1 The MM over GF(2^m)

Inputs: $A, B \in GF(2^m), r, F(x), F'(x)$
Output: $C = A \cdot B \cdot r^{-1} \bmod F(x)$
Step 1: $t := A \cdot B$
Step 2: $u := t \cdot F'(x) \bmod r$
Step 3: $C := (t + u \cdot F(x))/r$

(a)

Algorithm 2 The bit-level MM over GF(2^m)

Inputs: $A, B \in GF(2^m), F(x)$
Output: $C = A \cdot B \cdot x^{-m} \bmod F(x)$
Step 1: $C := 0$
Step 2: For $i := 0$ to $m-1$
Step 3: $\quad C := C + b_i A$
Step 4: $\quad C := C + c_0 F(x)$
Step 5: $\quad C := C/x$

(b)

Fig. 1. (a) The Montgomery multiplication (MM) over $GF(2^m)$ [5]. (b) The bit-serial MM [5].

This representation is called the PB representation. In this case, the addition of any two elements is easily performed by the XOR operation. However, the multiplication and squaring operations are complicated as the intermediate product needs further reduction by $F(x)$.

2.2 Montgomery Multiplication over $GF(2^m)$

Let α and β be two elements of $GF(2^m)$ to be multiplied, and $\phi = \alpha \cdot \beta \bmod F(x)$ be their multiplication product. Also, let A and B be two Montgomery residues defined as

$$A = \alpha \cdot r \bmod F(x) = \sum_{i=0}^{m-1} a_i x^i, \quad (2)$$

and

$$B = \beta \cdot r \bmod F(x) = \sum_{i=0}^{m-1} b_i x^i, \quad (3)$$

where, r, a polynomial satisfying $gcd(r, F(x)) = 1$, is called the Montgomery factor and gcd means the greatest common divisor. Then, the MM algorithm over $GF(2^m)$ can be formulated as [5]

$$C = A \cdot B \cdot r^{-1} \bmod F(x), \quad (4)$$

where $r \cdot r^{-1} + F(x) \cdot F'(x) = 1$ and r^{-1} is the inverse of r modulo $F(x)$, i.e., $r \cdot r^{-1} = 1 \bmod F(x)$. Based on [5], the MM over $GF(2^m)$ can be carried out by using Algorithm 1 shown in Fig. 1a. The polynomial r plays an important role in the complexity of the algorithm as we need to do modulo r multiplication and a final division by r. In [5], r is chosen as x^m, and this is because the modular operation using $r = x^m$ only requires ignoring the terms whose powers of x are greater than or equal to m. Furthermore, dividing a polynomial by $r = x^m$ can be easily carried out by m right shifts. In [5], an LSB-first bit-serial MM algorithm is also introduced. This algorithm is shown in Fig. 1b.

Using the definition of the Montgomery residue as shown in (2) and (3), one can write (4) as

$$C = (\alpha \cdot r) \cdot (\beta \cdot r) \cdot r^{-1} \bmod F(x) = \phi \cdot r \bmod F(x).$$

In other words, C is the Montgomery residue of ϕ. This makes it possible to convert the operands to Montgomery residues once at the beginning, and then, do several consecutive multiplications/squarings, and convert the final result to the original representation. The final conversion is a multiplication by r^{-1} followed by a reduction by $F(x)$, i.e., $\phi = C \cdot r^{-1} \bmod F(x)$. The elliptic curve cryptography can be a good example. A straightforward implementation of the Montgomery scalar multiplication using projective coordinates requires up to $(m-1)(6M+3A+5S)+(10M+7A+4S+I)$ clock cycles, where M, A, S, and I represent the number of clock cycles for multiplication, addition, squaring, and inversion, respectively [17]. Furthermore, inversion using Itoh-Tsujii algorithm requires $\lfloor \log_2(m-1) \rfloor + H(m-1) - 1$ multiplications and $m-1$ squarings, where $H(m-1)$ denotes the Hamming weight of $(m-1)$ [17]. For instance, inversion over $GF(2^{163})$ requires nine multiplications and 162 squarings. Hence, the scalar multiplication requires $991M + 976S + 493A$ clock cycles for $m = 163$. If the designer changes the operands to the original form, it is only enough to do the conversion once before and once after the scalar multiplication. It is worthwhile to mention that, in the general case, where $r = x^u$, the conversion requires at most $3u$ XOR gates and has the delay of at most $2T_X$ for irreducible trinomials and pentanomials [25]. Note that multiplication using the shifted polynomial basis (SPB) requires the same conversions as well (see [25], Section 2). As a result, using efficient Montgomery multiplication/squaring with low delay, significantly reduces the overall time complexity of the scalar point multiplication, and hence, increases the speed of the elliptic curve processor.

3 New Bit-Serial Montgomery Multipliers

Using $r = x^u, 1 \le u \le m$, as the general Montgomery factor, the Montgomery multiplication over $GF(2^m)$ can be formulated as

$$C = A \cdot B \cdot x^{-u} \bmod F(x). \tag{5}$$

Using (3), one can rewrite (5) as

$$C = b_0 A x^{-u} + b_1 A x^{-u+1} + \cdots + b_{m-1} A x^{m-u-1} \bmod F(x). \tag{6}$$

We know that x is a root of the polynomial $F(z)$, $F(x) = 0$, and using (1), one can write

$$f_m x^m + f_{m-1} x^{m-1} + \cdots + f_1 x + f_0 = 0. \tag{7}$$

For any irreducible polynomial, we have $f_0 = 1$ and $f_m = 1$. Thus, using this fact, multiplying both sides of (7) by x^{-1}, and rearranging the terms, one can obtain

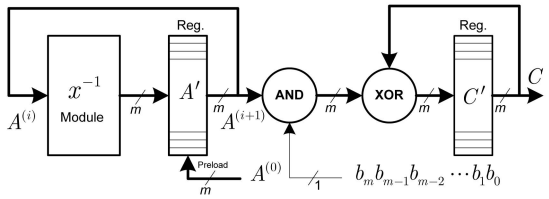
$$x^{-1} \bmod F(x) = x^{m-1} + \cdots + f_2 x + f_1. \tag{8}$$

In [10], (8) is used to design a semisystolic array structure for the MM using $r = x^m$. In the following sections, we use (8) to develop two different bit-serial multiplication algorithms based on (6) using $r = x^u, 1 \le u \le m$. Then, we show that the efficient Montgomery factor for such bit-serial structures is $r = x^{m-1}$.

Algorithm 3 The MSB-first bit-serial MM

Inputs: $A, B \in GF(2^m), F(x)$
Output: $C = A \cdot B \cdot x^{-u} \bmod F(x)$
Step 1: $A^{(0)} := Ax^{m-u-1} \bmod F(x)$, $C^{(0)} := 0$
Step 2: For $i := 0$ to $m-1$
Step 3: $C^{(i+1)} := b_{m-i-1} A^{(i)} + C^{(i)}$
Step 4: $A^{(i+1)} := A^{(i)} \cdot x^{-1} \bmod F(x)$
Step 5: $C := C^{(m)}$

(a)



(b)

Fig. 2. The proposed MSB-first bit-serial Montgomery multiplication (MM) using $r = x^u$: (a) algorithm, (b) architecture.

3.1 MSB-First Bit-Serial MM

In an MSB-first bit-serial MM algorithm, the operand B is processed from its MSB, i.e., b_{m-1}, and one bit at each cycle is considered. By rewriting (6) and changing the order of addition, one can obtain

$$C = b_{m-1} A x^{m-u-1} + \cdots + b_1 A x^{-u+1} + b_0 A x^{-u} \bmod F(x). \tag{9}$$

Now, we introduce Algorithm 3 based on (9) using the general Montgomery factor $r = x^u$ in Fig. 2a, where $A^{(i)}$ and $C^{(i)}$ denote the intermediate results at the ith iteration. It is clear from (9) that, first, we need to precompute $A^{(0)} = Ax^{m-u-1} \bmod F(x)$, as shown in Step 1 of this algorithm. As a result, the complexity of the MSB-first bit-serial MM depends on the complexity of Step 1 and the complexity of the main multiplication in Steps 2-5. First, we consider Step 4 in this algorithm as

$$\begin{aligned} A^{(i+1)} &= A^{(i)} \cdot x^{-1} \bmod F(x), \\ &= \left(a_{m-1}^{(i)} x^{m-2} + \cdots + a_1^{(i)} + a_0^{(i)} x^{-1}\right) \bmod F(x). \end{aligned} \tag{10}$$

Now, similar to [10], we substitute (8) in (10) and write the result as

$$\begin{aligned} A^{(i+1)} = a_0^{(i)} x^{m-1} &+ \left(a_{m-1}^{(i)} + a_0^{(i)} f_{m-1}\right) x^{m-2} + \cdots \\ &+ \left(a_2^{(i)} + a_0^{(i)} f_2\right) x^1 + \left(a_1^{(i)} + a_0^{(i)} f_1\right). \end{aligned} \tag{11}$$

Consequently, the architecture of Algorithm 3 is depicted in Fig. 2b. In this figure, A' and C' are two m-bit registers, which store values of $A^{(i)}$ and $C^{(i)}$, respectively. We assume that A' is loaded with $A^{(0)} = Ax^{m-u-1} \bmod F(x)$ at the beginning. There are two main loops in Fig. 2b. The right loop calculates the value of $C^{(i+1)}$ in Step 3 of Algorithm 3 and includes m two-input XOR gates. The left loop calculates the value of $A^{(i+1)}$ in Step 4 Algorithm 3 using the x^{-1}-module. This module multiplies $A^{(i)}$ by x^{-1} and reduces the results by

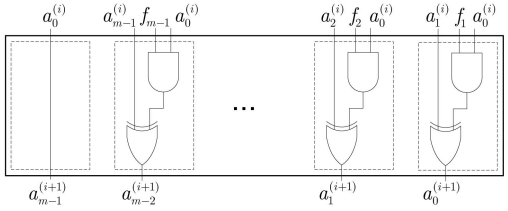


Fig. 3. The architecture of the x^{-1}-module for general irreducible polynomials.

$F(x)$. The architecture of the x^{-1}-module, which is obtained from (11), is depicted in Fig. 3. It is clear that if $F(x)$ is an ω-nomial, i.e., ω nonzero terms in (1), then we will need $\omega - 2$ two-input XOR gates to obtain $A^{(i+1)}$ in (11). So, for general irreducible polynomials of degree m in (1), it includes at most $(m-1)$ two-input AND gates, as well as $(m-1)$ two-input XOR gates to realize (11). Besides this module, we require m two-input AND gates to compute $b_{m-i-1}A^{(i)}$ in Step 3 of Algorithm 3. As a result, Steps 2-5 of the MSB-first bit-serial Montgomery multiplier require $(2m-1)$ two-input AND gates and $(2m-1)$ two-input XOR gates for general irreducible polynomials of degree m.

It is clear from Fig. 2b that two loops can be computed in parallel. Thus, a cycle of the multiplication algorithm requires the delay of $T_A + T_X$, where T_A and T_X represent the delays of a two-input AND gate and a two-input XOR gate, respectively. Also, the latency of the MSB-first bit-serial Montgomery multiplier equals m clock cycles.

Now, we consider the complexity of $Ax^{m-u-1} \bmod F(x)$. For $u < m-1$, this operation requires multiplications by positive powers of x followed by a reduction by $F(x)$. Implementing this operation with minimum hardware requires one multiplication by x followed by a reduction by $F(x)$ in a cycle, which has the time complexity of $T_A + T_X$ [21]. Consequently, $Ax^{m-u-1} \bmod F(x)$ is obtained with the linear time complexity of $(m-u-1)(T_A + T_X)$. Note that a multiplication by x results in including extra hardware. If $u = m$, we need to precompute $Ax^{-1} \bmod F(x)$ which requires the time complexity of $T_A + T_X$ using an x^{-1}-module as explained above.

It is clear that simplifying Step 1 of Algorithm 3 results in better time and area complexities. Here, the Montgomery factor plays an important role in simplifying this operation. The ideal case is $Ax^{m-u-1} = A$ or $x^{m-u-1} = 1$. This results in $u = m-1$, which suggests $r = x^{m-1}$ as a new efficient Montgomery factor. In this case, Step 1 is just a load operation of the coordinates of A into the register A'. The following summarizes the area and time complexities of the proposed multiplier using the Montgomery factor $r = x^{m-1}$:

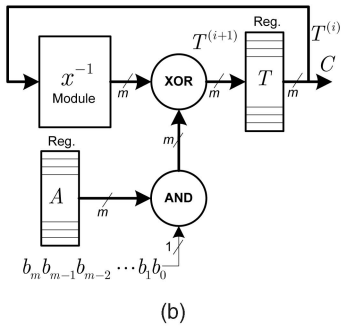
Proposition 1. *Using the new Montgomery factor $r = x^{m-1}$ for a general irreducible polynomial of degree m, the proposed MSB-first bit-serial MM over $GF(2^m)$ can be realized by using $(2m-1)$ two-input AND gates, $(2m-1)$ two-input XOR gates, and two m-bit registers. The critical path delay and the latency of this multiplier are $T_A + T_X$ and m clock cycles, respectively.*

Remark 1. The proposed MSB-first bit-serial multiplier is as efficient as the best bit-serial PB multiplier (LSB-first). One can use such a multiplier to improve the multiplication algorithm proposed in [11], which splits the multiplication into two concurrent multiplications: one PB and one MM. It is noted that the Montgomery multiplier of [11], which is used in [8] to design an ECC processor, is based on the algorithm proposed in [5] and has the critical path delay of $2(T_A + T_X)$. By replacing their Montgomery multiplier with our MSB-first multiplier and using the LSB-first bit-serial PB algorithm, the critical path delay can be reduced from $2(T_A + T_X)$ to $T_A + T_X$ with the same latency as $\lceil \frac{m}{2} \rceil$.

Algorithm 4 The LSB-first bit-serial MM

Inputs: $A, B \in GF(2^m), F(x)$
Output: $C = A \cdot B \cdot x^{-u} \bmod F(x)$
Step 1: $T^{(0)} := 0, A^{(0)} = Ax^{m-u-1} \bmod F(x)$
Step 2: For $i := 0$ to $m-1$
Step 3: $T^{(i+1)} := T^{(i)} \cdot x^{-1} \bmod F(x) + b_i \cdot A^{(0)}$
Step 4: $C := T^{(m)}$

(a)



(b)

Fig. 4. The proposed LSB-first bit-serial MM using $r = x^u$: (a) algorithm, (b) architecture.

Finally, note that the MSB-first bit-serial Montgomery multiplier using $r = x^m$ can be obtained by modifying Fig. 2b by adding a zero to the MSB of the operand B (i.e., $0b_{m-1} \cdots b_1 b_0$). Therefore, the latency of the MSB-first bit-serial MM using $r = x^m$ is increased to $m+1$ clock cycles. Note that A' is loaded with A.

3.2 LSB-First Bit-Serial MM

To design the LSB-first bit-serial MM, we rewrite (9) by using Horner's rule and $A^{(0)} = Ax^{m-u-1} \bmod F(x)$ to obtain

$$C = \big(\cdots \big(b_0 A^{(0)} x^{-1} \bmod F(x) + b_1 A^{(0)}\big) x^{-1} \bmod F(x) + \cdots + b_{m-2} A^{(0)}\big) x^{-1} \bmod F(x) + b_{m-1} A^{(0)}. \quad (12)$$

A similar formulation has previously been outlined in [10] using $u = m$ to design a semisystolic array structure for the MM. Based on (12), we can propose Algorithm 4 (Fig. 4a) for the MM algorithm over $GF(2^m)$ using the general Montgomery factor $r = x^u$. In this algorithm, we begin processing the operand B from its LSB, and again, we only process one bit at each cycle.

Similar to the discussion for the MSB-first bit-serial MM, in this case, again, we are interested in $A^{(0)} = Ax^{m-u-1} \bmod F(x) = A$ to simplify the multiplication process. This results in $u = m-1$ or $r = x^{m-1}$ as the new efficient Montgomery factor.

The hardware architecture of Algorithm 4 using $r = x^{m-1}$ can be obtained by a similar means as that of

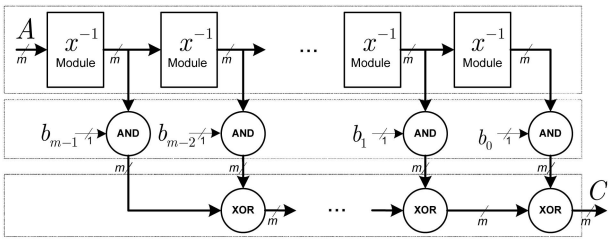


Fig. 5. The architecture of the bit-parallel Montgomery multiplier over $GF(2^m)$ with $r = x^m$.

Algorithm 3. This is shown in Fig. 4b. In this case, we require two m-bit registers to hold the value of $T^{(i)}$ and A. Also, we require m two-input AND gates as well as m two-input XOR gates as labeled with AND and XOR in Fig. 4b. The x^{-1}-module in Fig. 4b is the same as introduced for Algorithm 3 in Fig. 3. We summarize the complexity results by the following proposition:

Proposition 2. *Let $r = x^{m-1}$ be the Montgomery factor. Then, the proposed LSB-first bit-serial Montgomery multiplier over $GF(2^m)$ requires $(2m-1)$ two-input AND gates, $(2m-1)$ two-input XOR gates, and two m-bit registers. The critical path delay of this multiplier equals $T_A + 2T_X$ and its latency is m clock cycles.*

Similar to the MSB-first bit-serial MM algorithm, we can present the following remark for the LSB-first bit-serial MM algorithm:

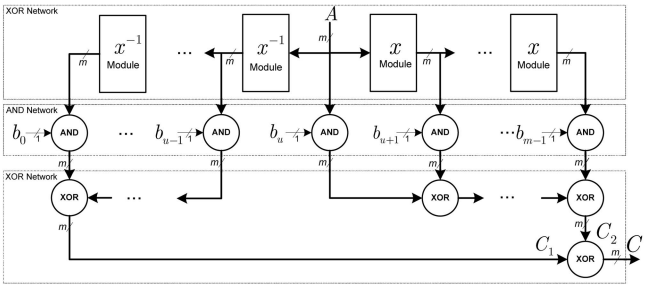
Remark 2. Assuming $r = x^m$ is the Montgomery factor, the LSB-first bit-serial MM over $GF(2^m)$ has the latency of $m+1$ clock cycles. In this case, Fig. 4b is modified by adding a zero to the MSB of the operand B (i.e., $0b_{m-1}\cdots b_1 b_0$).

Finally, we present the following remark:

Remark 3. It is interesting to note that using our proposed Montgomery factor $r = x^{m-1}$, one can simplify the semisystolic array structure proposed in [10]. As a result, its latency is reduced from $m+1$ to m clock cycles. Also, the number of the required cells is reduced from $m \times (m+1)$ to $m \times m$.

4 Bit-Parallel Montgomery Multiplication

Based on the formulation used in the previous sections, we present a new bit-parallel Montgomery multiplier over $GF(2^m)$ in this section. As shown in (4), the MM, in general, can be formulated as $C = A \cdot B \cdot r^{-1} \bmod F(x)$, where r can be chosen as $r = x^u,\ 0 < u \le m$. The algorithm proposed in [5] uses $u = m$ and generates (5) which can be rewritten as (6). Fig. 5 depicts a new architecture of the bit-parallel Montgomery multiplier for $u = m$. This architecture is also obtained by unfolding the loop in Algorithm 4. In this architecture, the AND modules multiply a field element by a bit, whereas the XOR modules add two field elements. The architecture shown in Fig. 5 is very similar to the architecture of the conventional bit-parallel polynomial basis multiplier. However, in the latter, instead of x^{-1}-modules, x-modules are used which perform a multiplication by x followed by a reduction modulo $F(x)$. Also, the order of processing the coordinates of B is reverse. Note that the x^{-1}-module in Fig. 5 is shown in Fig. 3 for general irreducible polynomials.



(a)

Algorithm 5 Bit-parallel MM

Inputs: $A, B \in GF(2^m), F(x)$
Output: $C = A \cdot B \cdot x^{-u} \bmod F(x)$
Step 1: Generate the matrix $\mathbf{M}$ for the given irreducible polynomial $F(x)$ using general u
Step 2: Find an efficient u to minimize the number of terms summed up in the entities of the matrix
Step 3: Re-generate the matrix $\mathbf{M}$ for the found u
Step 4: Implement $[c_0, c_1, \cdots, c_{m-1}]^T = \mathbf{M} \cdot [b_0, b_1, \cdots, b_{m-1}]^T$.

(b)

Fig. 6. The general architecture of the bit-parallel Montgomery multiplier over $GF(2^m)$ with $r = x^u, 1 \le u \le m-1$: (a) architecture, (b) algorithm.

By choosing u in the range of $[1, m-1]$, we can rewrite the Montgomery multiplication as

$$C = b_0 A x^{-u} + b_1 A x^{-u+1} + \cdots + b_{u-1} A x^{-1} + b_u A + b_{u+1} A x + \cdots + b_{m-1} A x^{m-u-1} \bmod F(x). \quad (13)$$

In this case, the main difference is that we multiply A by negative and positive powers of x to calculate the terms in (13). We can rewrite (13) as $C = C_1 + C_2$, where $C_1 = b_0 A x^{-u} + b_1 A x^{-u+1} + \cdots + b_{u-1} A x^{-1} \bmod F(x)$ and $C_2 = b_u A + b_{u+1} A x + \cdots + b_{m-1} A x^{m-u-1} \bmod F(x)$. Now, we can design the new architecture of the general case of the MM with $r = x^u$, as depicted in Fig. 6a. Note that for $1 \le u \le m-1$, the number of the x and x^{-1}-modules is $m-1$, as $b_u A$ is obtained directly from A.

Based on the architecture depicted in Fig. 6a, the first step of the multiplication is to compute the terms $Ax^i \bmod F(x)$, for $i \in [-u, m-u-1]$. In this paper, we use $A'_{(i)}$ to represent $Ax^i \bmod F(x)$. This can be done by using the matrix $\mathbf{M}$, whose columns show the PB representation of $A'_{(i)}$ for $i \in [-u, m-u-1]$. So, the matrix $\mathbf{M}$ has m rows and m columns. Then, the MM over $GF(2^m)$ can be formulated as

$$[c_0, c_1, \ldots, c_{m-1}]^T = \mathbf{M} \cdot [b_0, b_1, \ldots, b_{m-1}]^T. \quad (14)$$

Note that this formulation is similar to the Mastrovito multiplication [1]. We have shown the steps to construct the bit-parallel Montgomery multiplier in Fig. 6b.

Proposition 3. *Assume that an ω-nomial irreducible polynomial is used to construct $GF(2^m)$. In this case, the architecture shown in Fig. 6a requires m^2 AND gates and $(m-1)(m+\omega-2)$ XOR gates.*

Proof. In the architecture shown in Fig. 6a, the AND gates are only required in the AND network. So, the total number of AND gates is m^2. In this architecture, XOR gates are used in two XOR networks. The first XOR network obtains the matrix $\mathbf{M}$ and each column of this matrix requires $(\omega-2)$ XOR gates except the column corresponding to $A'_{(0)}$ which requires no XOR gate. Thus, $(m-1)(\omega-2)$ XOR gates are required in the first XOR network. In the second one, m XOR trees are used to obtain c_i for $i=0$ to $m-1$, where each XOR tree adds m terms. Thus, this network requires $m(m-1)$ XOR gates. As a result, the architecture presented in Fig. 6a requires $(m-1)(m+\omega-2)$ XOR gates. □

Remark 4. The proposed architecture for the bit-parallel MM is different from the structure proposed in [5] and [16]. In our structure, first the entries of matrix $\mathbf{M}$ are obtained by an XOR network which consists of XOR gates, then an AND network performs AND operations between the entries of the matrix $\mathbf{M}$ and the coordinates of B. Finally, another XOR network is used to obtain c_i for $i=0$ to $m-1$. In the algorithm proposed in [5], which is also used to design the original bit-parallel Montgomery multiplier in [16], an AND network followed by an XOR network are only used.

To consider the architecture of the bit-parallel Montgomery multiplier in details, we design bit-parallel Montgomery multipliers for two important classes of irreducible polynomials, namely irreducible trinomials and a special class of irreducible pentanomials.

5 Bit-Parallel Montgomery Multiplier for Irreducible Trinomials

By presenting the following lemma, we consider the properties of the matrix $\mathbf{M}$ to find the most efficient Montgomery factors (Step 2 in Algorithm 5):

Lemma 1. *Let $F(z)=z^m+z^k+1$ be an irreducible trinomial and x be the root of $F(z)$. Then, the Montgomery factor $r=x^u$ is obtained from the following in order to design a fast Montgomery multiplier (FMM)*

$$u=\begin{cases}1, & k=1,\\ k \text{ or } k-1, & k>1.\end{cases} \tag{15}$$

In this case, the entries of the matrix $\mathbf{M}$ will be the additions of at most two terms.

Proof. Using (13), we need to reduce the following polynomial by $F(x)$ for negative ($[-u,-1]$) and positive ($[1,m-u-1]$) values of i

$$Ax^i=a_0x^i+a_1x^{i+1}+\cdots+a_{m-1}x^{m+i-1}. \tag{16}$$

First, we consider (16) for $i\in[1,m-u-1]$ and it is easy to notice that the following requires no further reduction, $x^{m+i-1} \bmod F(x)=x^{k+i-1}+x^{i-1}$, if $i\le m-k$ (or $k+i-1\le m-1$). Note that x^{m+i-1} is the greatest power of x in (16). Thus, for $i\le m-k$, we can rewrite (16) by one step of reduction as

$$\begin{aligned}A'_{(i)}&=\sum_{j=0}^{m-1-i}a_jx^{j+i}+\sum_{j=m-i}^{m-1}a_jx^{j+i},\\ &=\sum_{j=0}^{m-1-i}a_jx^{j+i}+\sum_{j=0}^{i-1}a_{m-i+j}(x^{k+j}+x^j).\end{aligned} \tag{17}$$

It is clear from (17) that there are at most two terms at each position. Thus, we substitute i in (17) with the greatest positive power of x from (13), i.e., $i=m-u-1$, and we can conclude that $i\le m-k\Rightarrow m-u-1\le m-k$, which results in

$$u\ge k-1. \tag{18}$$

Now, we consider (16) for $i\in[-u,-1]$. From trinomial representation, one can find $1=x^m+x^k$, and by multiplying both sides by x^i, we have

$$x^i=x^{m+i}+x^{k+i} \bmod F(x). \tag{19}$$

Note that, in (19), $m+i$ is a positive number for $i\in[-u,-1]$. Therefore, (19) will be in the PB representation if $k+i\ge 0$. Thus, for $k+i\ge 0$, we can use (19) to simplify $A'_{(i)}$ by one step of reduction as

$$\begin{aligned}A'_{(i)}&=\sum_{j=|i|}^{m-1}a_jx^{j+i}+\sum_{j=0}^{|i|-1}a_jx^{j+i} \bmod F(x),\\ &=\sum_{j=|i|}^{m-1}a_jx^{j+i}+\sum_{j=0}^{|i|-1}a_j(x^{m+i+j}+x^{k+i+j}) \bmod F(x).\end{aligned} \tag{20}$$

Note that x^i is the least power of x in (16) which is in the PB representation for $k+i\ge 0$. As a result, (20) is in the PB representation for $k+i\ge 0$, and again, there are at most two terms in each position. By replacing i with the least value of i from (13), i.e., $-u$, we have

$$0\le k+i\Rightarrow u\le k. \tag{21}$$

It can be concluded from (18) and (21) that the elements of the matrix $\mathbf{M}$ are summations of at most two terms if $k-1\le u\le k$, and the proof is complete. □

Now, we can present the following proposition to determine the area complexity of the Montgomery multiplier based on the values of u obtained from Lemma 1:

Proposition 4. *Assume that the Montgomery factor is chosen based on (15). Then, the bit-parallel Montgomery multiplier using irreducible trinomials requires (m^2-1) two-input XOR gates if $k\neq\frac{m}{2}$. Otherwise, i.e., $k=\frac{m}{2}$, it requires $(m^2-\frac{m}{2})$ two-input XOR gates. In both cases, the multiplier also requires m^2 AND gates.*

Proof. Using Proposition 3, the proof is straightforward. □

Now, the entries in each row of the matrix include single or two-term elements (Step 3 in Algorithm 5). Those entries should finally be summed up by using an XOR tree after the AND operation with the corresponding coordinates of B (Step 4 in Algorithm 5). To reduce the delay of the MM, it is possible to use the method of [6]. This involves doing a part of the final addition operation in parallel with the computation of the elements of the matrix $\mathbf{M}$. In other

TABLE 1
The Number of the Elements in the Matrix $\mathbf{M}$ for Irreducible Trinomials

Position	$u=k-1$		$u=k$	
	Single-term	Two-term	Single-term	Two-term
x^0	m	0	$m-1$	1
x^1	$m-1$	1	$m-2$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x^{k-1}	$m-k+1$ $(m-u)$	$k-1$ (u)	$m-k$ $(m-u)$	k (u)
x^k	k $(u+1)$	$m-k$ $(m-u-1)$	$k+1$ $(u+1)$	$m-k-1$ $(m-u-1)$
x^{k+1}	$k+1$	$m-k-1$	$k+2$	$m-k-2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x^{m-1}	$m-1$	1	m	0

words, while we compute the two-term elements of the matrix $\mathbf{M}$, it is possible to add the single-term elements pairwise after the bitwise AND operation with the corresponding coordinates of B.

In this regard, Table 1 shows the number of single-term and two-term elements in each row (position) of the matrix $\mathbf{M}$ for two Montgomery factors mentioned in Lemma 1. We use Table 1 to obtain the time complexity of the MM using irreducible trinomials and we can present the following proposition:

Proposition 5. *Assuming that $F(z)=z^m+z^k+1$ is an irreducible trinomial, the delay of the bit-parallel Montgomery multiplier using $F(z)$ is as follows:*

$$\begin{cases} T_A+\lceil \log_2(2m-u-1)\rceil T_X, & u\le \frac{m-1}{2} \\ T_A+\lceil \log_2(m+u)\rceil T_X, & u>\frac{m-1}{2} \end{cases},$$

where u is defined in (15).

Proof. It is noted that the worst delay occurs in the position (row of the matrix $\mathbf{M}$) that includes the maximum number of two-term elements. The reason is that two-term elements will be ready after T_X and during this delay, we can add the single-term elements pairwise. Thus, if we have few two-term elements, more single-term elements can be added pairwise. Using Table 1, one can see that the worst case will be in the position x^{k-1} or x^k, which include u and $m-u-1$ two-term elements, as shown in the corresponding positions in Table 1. In other words, c_k or c_{k-1} have the longest critical path delay. In this regard and based on the Montgomery factor $r=x^u$, we study two possible cases:

Case I: If $u\le \frac{m-1}{2}$ (or $u\le m-u-1$), then in the position x^k, which now has the maximum number of two-term elements, there are $m-u-1$ two-term elements and $u+1$ single-term elements (see Table 1). The $u+1$ single-term elements can be added (after bitwise AND with the corresponding coordinates of B) by one level of XOR gates which results in $\lceil (u+1)/2\rceil$ terms. At the same time, the computation of the $m-u-1$ two-term elements is also complete and we can AND them with the corresponding coordinates of B. Thus, the total delay of the multiplication in the position x^k to generate c_k is $T_A+\left(1+\left\lceil \log_2(m-u-1+\left\lceil \frac{u+1}{2}\right\rceil)\right\rceil\right)T_X = T_A+\lceil \log_2(2m-u-1)\rceil T_X$.

Case II: If $u>\frac{m-1}{2}$ (or $u>m-u-1$), the maximum number of two-term elements occurs in the position x^{k-1}, where there are u two-term elements and $m-u$ single-term elements (see Table 1). Therefore, similar to Case I, the total delay of the whole operation in the position x^{k-1} to generate c_{k-1} is $T_A+(1+\lceil \log_2(\lceil (m+u)/2\rceil)\rceil)T_X = T_A+\lceil \log_2(m+u)\rceil T_X$. □

6 Bit-Parallel Montgomery Multiplier for Irreducible Pentanomials

Irreducible pentanomials form another family of irreducible polynomials which are used in finite field arithmetic, e.g., [6], [26], [25], [3], and [4], where there is no irreducible trinomial of the desired degree m. Generally, they can be formulated as

$$F(z)=z^m+z^{k_3}+z^{k_2}+z^{k_1}+1,\ 1\le k_1<k_2<k_3<m. \quad (22)$$

We assume that $r=x^u$ is the Montgomery factor. The matrix $\mathbf{M}$ plays an important role in designing efficient bit-parallel Montgomery multipliers. If each column of the matrix $\mathbf{M}$ is computed with one step of reduction, then the matrix $\mathbf{M}$ can be obtained faster. In this regard, we use a special type of irreducible pentanomials. This type of irreducible pentanomials, known as type-II irreducible pentanomials, is defined as $F(z)=z^m+z^{n+2}+z^{n+1}+z^n+1$, where $2\le n\le \lfloor \frac{m}{2}\rfloor -1$ [4]. Now, we can present the following remark:

Remark 5. Assume that $F(z)$ is an irreducible pentanomial (see (22)), $F(x)=0$ and $r=x^u$ is the Montgomery factor. The computation of matrix $\mathbf{M}$ defined in (14) is very fast for type-II irreducible pentanomials (Step 2 of Algorithm 5). In this case, for any u, there is at least one value for i, where the computation of $A'_{(i)}$ will require two steps of reduction. If $u=n$ or $u=n+1$, then the matrix $\mathbf{M}$ will be in the simplest form regarding the steps of reduction, where for $u=n$ (respectively, $u=n+1$) only the term $A'_{(m-u-1)}$ (respectively, $A'_{(-u)}$) will require two steps of reduction.

To verify the above remark, we start by considering the computation of $A'_{(i)}$, where $i\in[-u,-1]$. Similar to the proof of Lemma 1, it is easy to show that we should have $u\le k_1$. Similarly, in order to have one step of reduction for computation of $A'_{(i)}, i\in[1,m-u-1]$, the following condition should also be met, $m-u-1+k_3\le m\Rightarrow u\ge k_3-1$. Therefore, u should satisfy both $u\le k_1$ and $u\ge k_3-1$ which is impossible. In such cases, at least k_3-k_1-1 columns of the matrix $\mathbf{M}$ require more than one step of reduction. So, if one minimizes k_3-k_1-1, then less columns of the matrix $\mathbf{M}$ will require two steps of reduction which means it will be easier to obtain the matrix $\mathbf{M}$. Thus, k_1, k_2, and k_3 should be three consecutive numbers which means that the pentanomial should be a type-II irreducible pentanomial. In this case, only one column will require more than one step of reduction. Now, if we choose $u=k_1=n$, then only $A'_{(m-u-1)}$ will require two steps of reduction. Similarly, if we choose $u=k_3-1=n+1$, then only $A'_{(-u)}$ will require two steps of reduction.

In this paper, we obtain the time and area complexities of the bit-parallel Montgomery multiplier for $r=x^n$. We note that the same results can be obtained by using $r=x^{n+1}$ and

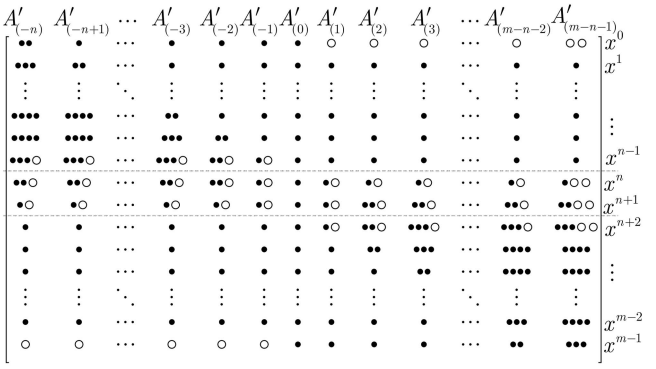


Fig. 7. The matrix $\mathbf{M}$ for type-II irreducible pentanomials using $u = n$.

this is not considered due to the page limit. The matrix $\mathbf{M}$ for type-II irreducible pentanomials is shown in Fig. 7. Here, we show how it is obtained (i.e., Step 3 of Algorithm 5). We assume that $A = A'_{(0)} = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$. The coefficients of $A'_{(0)}$ are shown with black nodes in Fig. 7 in the column $A'_{(0)}$. Then, $A'_{(1)}$ can be obtained by

$$\begin{aligned} A'_{(1)} &= A'_{(0)}x \bmod F(x), \\ &= \sum_{i=0}^{m-2} a_i x^{i+1} + a_{m-1}(x^{n+2} + x^{n+1} + x^n + 1). \end{aligned} \tag{23}$$

It is clear that the coefficients of (23) are obtained by shifting the coefficients of $A'_{(0)}$ to left and reducing the term $a_{m-1}x^m$ by $F(x)$. In this regard, the shifted terms, which are shifted vertically in the matrix $\mathbf{M}$, are depicted by black nodes in Fig. 7. Thus, in column $A'_{(1)}$ four new terms (a_{m-1}) are added to the positions x^0, x^n, x^{n+1}, and x^{n+2}. These new terms are depicted by white nodes in Fig. 7. Thus, (23) can be obtained with the delay of an XOR gate and using three two-input XOR gates.

Similarly, as depicted in Fig. 7, $A'_{(2)}$ can be obtained with the same delay and the same number of gates. Now, we can consider the general case of two consecutive columns. We assume that $2 \le j \le m-n-3$ and we obtain $A'_{(j+1)}$ by

$$\begin{aligned} A'_{(j)} = &\sum_{i=0}^{m-1-j} a_i x^{i+j} + \sum_{i=m-j}^{m-1} a_i \big(x^{n+2+i-(m-j)} \\ &+ x^{n+1+i-(m-j)} + x^{n+i-(m-j)} + x^{i-(m-j)}\big), \end{aligned} \tag{24}$$

and

$$\begin{aligned} A'_{(j+1)} = &\sum_{i=0}^{m-j-2} a_i x^{i+j+1} + a_{m-j-1}\big(x^{n+2} + x^{n+1} + x^n + 1\big) \\ &+ \sum_{i=m-j}^{m-1} a_i \big(x^{n+2+i-(m-j-1)} + x^{n+1+i-(m-j-1)} \\ &+ x^{n+i-(m-j-1)} + x^{i-(m-j-1)}\big). \end{aligned} \tag{25}$$

By comparing (24) and (25), it is clear that one can obtain (25) by shifting the coefficients of (24) to left, or equivalently, downshifting the entries of the column $A'_{(j)}$ in $\mathbf{M}$ down, and adding four new terms in the positions x^0, x^n, x^{n+1}, and x^{n+2}. Using (25), there are two terms in the position x^n, three terms in the position x^{n+1}, and four terms in the position x^{n+2}. So, we need one new XOR gate in the position x^n and one new XOR gate in the position x^{n+1}. But, for the position x^{n+2}, we can use two approaches. In the first approach, we can obtain the entry in the position x^{n+2} by reusing the shifted coefficient of $A'_{(j)}$ which is a three-term coefficient (three black nodes in Fig. 7) and adding it to the new term in the position x^{n+2} (one white node in Fig. 7). This results in having the delay of $3T_X$, however, we use only one extra XOR gate. In the second approach, we reuse one of the XOR gates of $A'_{(j)}$ in the position x^{n+1} and obtain the final result by using two more new XOR gates. This results in having the delay of $2T_X$; however, we use two additional XOR gates for this entry. Therefore, we can design two bit-parallel Montgomery multipliers. One is faster and we call it the Fast Montgomery Multiplier and the other one requires less area and we call it the Low-Complexity Montgomery Multiplier (LCMM). Now, we can conclude that in the FMM (respectively, LCMM), each new column in the matrix $\mathbf{M}$, i.e., $A'_{(j)}$, requires four (respectively, three) XOR gates for $3 \le j \le m-n-2$. For $j = m-n-2$ in (24), we have

$$\begin{aligned} A'_{(m-n-2)} = &\sum_{i=0}^{n+1} a_i x^{i+m-n-2} \\ &+ \sum_{i=n+2}^{m-1} a_i \big(x^i + x^{i-1} + x^{i-2} + x^{i-(n+2)}\big). \end{aligned} \tag{26}$$

Note that in (26), we have $a_{m-1} + a_{n+1}$ in the position x^{m-1}. Now, the rightmost column of the matrix $\mathbf{M}$, i.e., $A'_{(m-n-1)}$, is obtained by multiplying (26) by x and reducing with $F(x)$. Therefore, the term $a_{m-1} + a_{n+1}$ will be in the positions x^0, x^n, x^{n+1}, and x^{n+2}. This is shown with white nodes in the column $A'_{(m-n-1)}$ in Fig. 7. As $a_{m-1} + a_{n+1}$ is computed in $A'_{(1)}$, it does not require any new gate. Now, in the column $A'_{(m-n-1)}$, there is a five-term element in the position x^{n+2}. This element is a summation of a three-term and a two-term elements, which are reused. Here again, we have two possibilities. In the LCMM, these two can be summed up in the final XOR tree. So, we need two XOR gates to obtain $A'_{(m-n-1)}$ (one for the position x^n and one for the position x^{n+1}). For the LCMM, we can compute the five-term element directly with the delay of $3T_X$ and by using one new XOR gate. As a result, $A'_{(m-n-1)}$ requires three XOR gates. Note that the matrix $\mathbf{M}$ can be obtained for negative values of i, similarly. However, no column requires two steps of reduction.

Now, we consider Step 4 in Algorithm 5. Remark 5 and Fig. 7 show that it is not possible to compute all of the polynomials $A'_{(i)}$ by one step of reduction. It means that if we obtain the matrix $\mathbf{M}$ for type-II pentanomials, at least one of the elements of the matrix will be a summation of five terms. Having five terms in an entry of the matrix implies that direct computation of the matrix $\mathbf{M}$ requires the delay of $3T_X$ and $2T_X$ for the LCMM and the FMM, respectively. Then, the LCMM and the FMM require the total delay of $T_A + (3 + \lceil \log(m) \rceil)T_X$ and $T_A + (2 + \lceil \log(m+1) \rceil)T_X$,

respectively. In the case of the Montgomery multiplication with type-II irreducible pentanomials, similar procedure for trinomials can be used to reduce the delay. The main point is that, similar to the discussion for trinomials, while we are computing the elements of the matrix which are summations of three or four elements, we can build a part of the final XOR tree for the elements of the matrix $\mathbf{M}$ which are single terms or summation of two terms. This process is shown in Fig. 8 to obtain c_{n+1} for $n \leq \frac{m-1}{2}$ in the FMM. It is noted that this architecture is slightly different from the architecture depicted in Fig. 6. The main difference is the order of the AND and XOR operations. Now, we can present two propositions for the area and complexities of the bit-parallel Montgomery multipliers.

Proposition 6. *Let $r = x^n (u = n)$ be the Montgomery factor. The fast bit-parallel Montgomery Multiplier (FMM) using type-II irreducible pentanomials of degree m requires m^2 two-input AND gates and $m^2 + 3m - 9$ two-input XOR gates. Also, it has the time complexity of $T_A + (1 + \lceil \log_2(m+n) \rceil) T_X$, if $n \geq \frac{m-1}{2}$ and $T_A + (1 + \lceil \log_2(2m-n-2) \rceil) T_X$, if $n < \frac{m-1}{2}$.*

Proof. As stated above and shown in Fig. 7, the column $A'_{(0)}$ requires no XOR gate. Four columns of the matrix $\mathbf{M}$, i.e., ($A'_{(-2)}$, $A'_{(-1)}$, $A'_{(1)}$, and $A'_{(2)}$), require three XOR gates, one column ($A'_{(m-n-1)}$) requires two XOR gates and the rest of the columns require four XOR gates. As a result, the total number of the XOR gates to obtain the matrix $\mathbf{M}$ equals $(m-6) \times 4 + 4 \times 3 + 2 = 4m - 10$. Finally, the elements should be summed up by using m XOR trees (one for each position). There are $(m+1)$ elements in the position x^{n+2} as we break up the five-term element into two elements. The rest of the positions have m elements. Thus, the XOR trees require $(m-1) \times (m-1) + m = m^2 - m + 1$ XOR gates and consequently, the multiplier requires $m^2 + 3m - 9$ two-input XOR gates. Now, we can compute the number of AND gates. In the matrix $\mathbf{M}$, $(m-1)$ columns have m elements. The rightmost column, i.e., $A'_{(m-n-1)}$, has $(m+1)$ elements as we break up the five-term element into two parts. But the resulted two-term part is also used in the position x^0. Thus, in each column we need m AND gates, and consequently, the multiplier requires m^2 two-input AND gates.

Now, we consider the time complexity of the multiplier. The maximum delay occurs in a position (a row of the matrix $\mathbf{M}$) which contains the maximum number of elements with addition of more than two terms. Rows 0 to $n-1$ and $n+1$ to $m-1$ contain single-term elements for positive and negative values of i in $A'_{(i)}$, respectively (see Fig. 7). Thus, the maximum delay will be in one of the positions x^{n+1} or x^n which have $m-n-2$ and n three/four-term elements, respectively. This is shown by dashed lines in Fig. 7. We can consider the two possible cases.

Case I: If $n \leq \frac{m-1}{2}$ (or $n \leq m-n-2$), then the position x^{n+1} will have most of the three/four-term elements. As depicted in Fig. 7, it has $n+1$ two-term elements (the columns $A'_{(1)}$, and $A'_{(-1)}$ to $A'_{(-n)}$), $m-n-3$ three-term elements (the columns $A'_{(2)}$ to $A'_{(m-n-2)}$), one four-term element (the column $A'_{(m-n-1)}$), and one single-term element (the column $A'_{(0)}$). We compute the elements with three or four terms by the delay of $2T_X$. Meanwhile, it is possible to compute two-term elements, so after the delay of T_X, we will have $n+1$ single terms. We also have another single-term element in the column $A'_{(0)}$ which results in having $n+2$ single terms after the delay of T_X. Now, we can AND them with the corresponding coordinates of B. Therefore, after another delay of a two-input XOR gate, we will have $\lceil \frac{n+2}{2} \rceil$ single terms. At the same time, the computation of the three/four-term elements is complete and we can AND them with the corresponding coordinates of B. At this point of time, we have $(1 + m - n - 3 + \lceil \frac{n+2}{2} \rceil)$ single terms and consequently, the total delay equals $T_A + (2 + \lceil \log_2(1 + m - n - 3 + \lceil \frac{n+2}{2} \rceil) \rceil) T_X = T_A + (1 + \lceil \log_2(2m - n - 2) \rceil) T_X$. This is shown in Fig. 8.

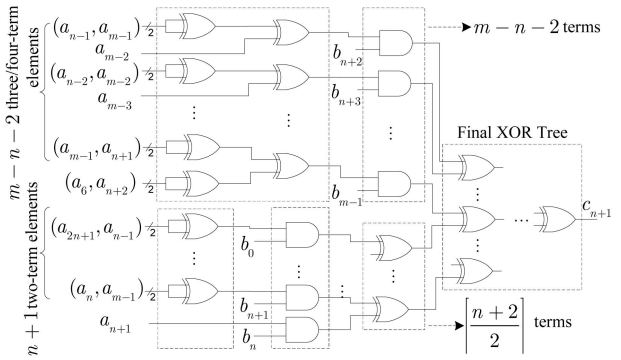


Fig. 8. The architecture of the FMM for type-II irreducible pentanomials to compute c_{n+1} for $u = n \leq \frac{m-1}{2}$.

Case II: If $n \geq \frac{m-1}{2}$ (or $n > m-n-2$), then the position x^n will have the most three/four-term elements, where there are $m-n-1$ two-term elements (the columns $A'_{(-1)}$ and $A'_{(1)}$ to $A'_{(m-n-2)}$), n elements with three terms (the columns $A'_{(m-n-1)}$ and $A'_{(-2)}$ to $A'_{(-n)}$), and one element with one term (the column $A'_{(0)}$). Thus, the delay of the bit-parallel Montgomery multiplier is $T_A + (2 + \lceil \log_2(n + \lceil \frac{m-n}{2} \rceil) \rceil) T_X = T_A + (1 + \lceil \log_2(m+n) \rceil) T_X$. □

Note that the same area/time complexity can be obtained by using $r = x^{n+1}$. Now, we present the following proposition for the LCMM:

Proposition 7. *Assuming $r = x^n$ $(u = n)$ is used as the Montgomery factor, the low-complexity bit-parallel Montgomery multiplier (LCMM) using the type-II irreducible pentanomial $F(z)$ requires m^2 two-input AND gates and $m^2 + 2m - 3$ two-input XOR gates. Also, it has the time complexity of $T_A + (1 + \lceil \log_2(\lceil \frac{m-u}{2} \rceil + 4u - 5) \rceil) T_X$, if $u > \frac{m-1}{2}$ and $T_A + (1 + \lceil \log_2(\lceil \frac{u+1}{2} \rceil + 4m - 4u - 9) \rceil) T_X$, if $u \leq \frac{m-1}{2}$.*

Proof. In this case, we need m two-input AND gates in each column, and totally, the multiplier requires m^2 two-input AND gates. Now, we can obtain the number of the XOR gates. The column $A'_{(0)}$ requires no XOR gate. The rest of the columns require three two-input XOR gates. So, the matrix $\mathbf{M}$ is obtained by using $3 \times (m-1) = 3m - 3$ two-input XOR gates. Each of the final XOR trees have m elements in each position, so they require

$m^2 - m$ two-input XOR gates. Therefore, this multiplier requires $m^2 + 2m - 3$ two-input XOR gates.

Now, we can obtain the time complexity of this multiplier. In this case, the number of four/five-term elements will determine the delay of the multiplier. Fig. 7 shows that the maximum number of four/five-term elements occurs in the positions x^{n+2} or x^{n-1} which have $m-u-3$ and $u-2$ four/five-term elements, respectively. So, we study the two possible cases as follows:

Case I: If $u-2 \leq m-u-3$ or $u \leq \frac{m-1}{2}$, then the position x^{n+2} will have the most four/five-term elements. In this column, we have $u+1$ single-term elements, one two-term element, one three-term element, and $m-u-3$ four/five-term elements. Similar to the proof of Proposition 6, we have the delay of $\mathrm{T_A} + (1 + \lceil \log_2(\lceil \frac{u+1}{2} \rceil + 4m - 4u - 9) \rceil) T_X$.

Case II: If $u-2 > m-u-3$ or $u > \frac{m-1}{2}$, then the position x^{n-1} will have the most four-term elements. In this column, we have $m-u$ single-term elements, one two-term element, one three-term element, and $u-2$ four/five-term elements. This results in the delay of $T_A + (1 + \lceil \log_2(\lceil \frac{m-u}{2} \rceil + 4u - 5) \rceil) T_X$. □

An example is presented in Appendix A to illustrate the proposed multipliers. Note that the same area/time complexity can be obtained by using $u = n+1$.

7 Montgomery Squaring over $GF(2^m)$

After multiplication, squaring is the most important operation in finite field arithmetic. This operation is considered in polynomial basis by Wu in [27] for the general case of irreducible polynomials and irreducible trinomials as a special case. In [5], some general squarers are proposed using the MM algorithm. An optimized squarer is proposed in [16] for irreducible trinomials using the MM algorithm. That squarer is designed using $r = x^k$ as the Montgomery factor for irreducible trinomials and it is shown that it has the delay of T_X, whereas the delay of squaring in PB is at most $2T_X$.

Our proposed bit-serial and bit-parallel multipliers can be used to do the squaring for general case of irreducible polynomials. However, it is possible to design efficient squarers for some important cases of special irreducible polynomials. For irreducible trinomials, the Montgomery factor $r = x^{k-1}$ can be used to design squarers as well. However, the results will be similar to those of [16]. Therefore, we do not consider the squaring operation for irreducible trinomials and instead, we focus on bit-parallel squaring using type-II irreducible pentanomials. Squaring using the Montgomery multiplication can be formulated as

$$\begin{aligned} C &= A^2 \cdot x^{-u} \bmod F(x) \\ &= \left(\sum_{i=0}^{m-1} a_i x^{2i} \right) \cdot x^{-u} \bmod F(x). \end{aligned} \tag{27}$$

Now, we show that efficient squarers can be designed using the Montgomery factors x^n or x^{n+1}. Let x^n be the Montgomery factor. In this case, (27) can be rewritten as

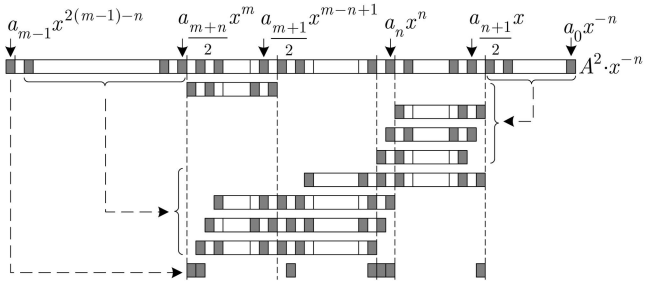


Fig. 9. Squaring for odd values of m and $n, n < \frac{m-3}{2}$.

$$C = \sum_{i=0}^{m-1} a_i x^{2i-n} \bmod F(x). \tag{28}$$

Here, we only consider odd values of m as they are more important than even values of m [20]. First, we assume that m and n are odd numbers. As a result, (28) can be written as

$$\begin{aligned} C = a_{m-1}x^{2m-n-2} &+ \sum_{i=\frac{m+n}{2}}^{m-2} a_i x^{2i-n} \\ &+ \sum_{i=\frac{n+1}{2}}^{\frac{m+n-2}{2}} a_i x^{2i-n} + \sum_{i=0}^{\frac{n-1}{2}} a_i x^{2i-n} \bmod F(x). \end{aligned} \tag{29}$$

Now, we present the following lemma to find the area and time complexities of Montgomery Squaring (MS) using type-II irreducible pentanomials:

Lemma 2. *Let m and n be odd positive integers and $n < \frac{m-3}{2}$, and $F(z) = z^m + z^{n+2} + z^{n+1} + z^n + 1$ be an irreducible polynomial. In this case, $C = A^2 \cdot x^{-n} \bmod F(x)$ can be obtained with the maximum delay of $2T_X$ using at most $(\frac{m-3}{2} + m + 4)$ two-input XOR gates.*

Proof. Let us represent (29) in the first row of Fig. 9, where the gray and white cells represent the coordinates of A and zeros, respectively. There are three sums in (29). The second sum in (29) does not require any reduction and is shown in the middle part of the first row in Fig. 9 indicated by indices from 1 to $m-2$. The last sum produces negative powers of x, and using the fact that $x^j = x^{m+j} + x^{n+2+j} + x^{n+1+j} + x^{n+j}$ for negative js, it can be reduced as

$$\begin{aligned} \sum_{i=0}^{\frac{n-1}{2}} a_i x^{2i-n} = &\sum_{i=0}^{\frac{n-1}{2}} a_i x^{m+2i-n} + \sum_{i=0}^{\frac{n-1}{2}} a_i x^{2i+2} \\ &+ \sum_{i=0}^{\frac{n-1}{2}} a_i x^{2i+1} + \sum_{i=0}^{\frac{n-1}{2}} a_i x^{2i}. \end{aligned} \tag{30}$$

Four sums on the right side of (30) are shown in rows 2-5 of Fig. 9. The first sum in (29) produces terms with degrees greater or equal to m and using $x^{m+j} = x^{n+2+j} + x^{n+1+j} + x^{n+j} + x^j$ for $j \geq 0$, it is reduced as

$$\begin{aligned} \sum_{i=\frac{m+n}{2}}^{m-2} a_i x^{2i-n} = &\sum_{i=\frac{m+n}{2}}^{m-2} a_i x^{2i-m+2} + \sum_{i=\frac{m+n}{2}}^{m-2} a_i x^{2i-m+1} \\ &+ \sum_{i=\frac{m+n}{2}}^{m-2} a_{\frac{n+i}{2}} x^{2i-m} + \sum_{i=\frac{m+n}{2}}^{m-2} a_{\frac{n+i}{2}} x^{2i-n-m}. \end{aligned} \tag{31}$$

TABLE 2
Comparison of Bit-Serial Multipliers over $GF(2^m)$

Algorithm	Type	#AND	#XOR	#Flip Flops	Latency	Critical path delay
$F(z) = f_m z^m + f_{m-1} z^{m-1} + \cdots + f_1 z + f_0$						
[21]: LSB-first	PB	$2m-1$	$2m-1$	$2m$	m	$T_A + T_X$
[21]: MSB-first		$2m-1$	$2m-1$	$2m$	m	$T_A + 2T_X$
Algorithm 2 [5]: LSB-first	MM	$2m-1$	$2m-1$	$2m$	$m\,(u=m)$	$2T_A + 2T_X$
Fig. 2: MSB-first		$2m-1$	$2m-1$	$2m$	$m\,(u=m-1)$ $m+1\,(u=m)$	$T_A + T_X$
Fig. 4: LSB-first		$2m-1$	$2m-1$	$2m$	$m\,(u=m-1)$ $m+1\,(u=m)$	$T_A + 2T_X$

The four sums in (31) are shown in rows 6-9 of Fig. 9. Finally, the term $a_{m-1}x^{2(m-1)-n}$ is reduced as

$$\begin{aligned} a_{m-1}x^{2m-n-2} &= a_{m-1}(x^m + x^{m-1} + x^{m-2} + x^{m-n-2}) \\ &= a_{m-1}(x^{m-1} + x^{m-2} + x^{m-n-2} + x^{n+2} \\ &\quad + x^{n+1} + x^n + 1) \bmod F(x). \end{aligned}$$

This is shown in the last row of Fig. 9. Considering the overlaps between the odd and even powers of x separately, at most four terms (gray cells) contribute to any position which results in the delay of $2T_X$.

To obtain the area complexity of squaring using Fig. 9, we start from position 0 and consider all the overlaps. For even i satisfying $0 \le i \le n-1$ ($\frac{n+1}{2}$ coordinates), c_i is a summation of three terms. For odd i satisfying $1 \le i \le n-2$ ($\frac{n-1}{2}$ coordinates), c_i is a summation of two terms. The coordinates c_n, c_{n+1}, and c_{n+2} are summations of four terms, however, one XOR gate is reused twice (first cells of rows 7-9 overlapping with row 10). For even i satisfying $n+3 \le i \le m-1$ ($\frac{m-n-2}{2}$ coordinates), c_i is a summation of two terms. For odd i satisfying $n+4 \le i \le m-2$ ($\frac{m-n-4}{2}$ coordinates), c_i is a summation of three terms. Thus, there are $\frac{m-3}{2}$ two-term, $\frac{m-3}{2}$ three-term, and three four-term coordinates (reusing one XOR gate twice), which result in using at most $(\frac{m-3}{2} + m + 4)$ two-input XOR gates to obtain the coordinates of C. □

We present the explicit formulation to obtain c_i below, where m and n are odd numbers and $n < \frac{m-3}{2}$. Note that in all of the cases, i is increased by 2 (e.g., $i = 1, \ldots n-2$ means $i = 1, 3, \ldots, n-4, n-2$)

$$\begin{cases} a_0 + a_{\frac{m+n}{2}} + a_{m-1}, & i = 0, \\ a_{\frac{i-1}{2}} + a_{\frac{n+1}{2}+\frac{i-1}{2}}, & i = 1, \cdots, n-2, \\ a_{\frac{i}{2}-1} + a_{\frac{i}{2}} + a_{\frac{m+n+i}{2}}, & i = 2, \cdots, n-1, \\ a_{\frac{n-1}{2}} + a_n + a_{\frac{m+n}{2}} + a_{m-1}, & i = n, \\ a_{\frac{n-1}{2}} + a_{\frac{m+n}{2}} + & i = n+1, \\ a_{\frac{m+2n+1}{2}} + a_{m-1}, & \\ a_{n+1} + a_{\frac{m+n}{2}} + & i = n+2, \\ a_{\frac{m+n}{2}+1} + a_{m-1}, & \\ a_{\frac{m+i-1}{2}} + a_{\frac{m+n}{2}+\frac{i}{2}}, & i = n+3, \cdots, m-n-2, \\ a_{\frac{n+1}{2}+\frac{i-1}{2}} + & i = n+4, \cdots, m-2, \\ a_{\frac{m+i}{2}-1} + a_{\frac{m+i}{2}}, & \\ a_{\frac{i-(m-n)}{2}} + a_{\frac{m+i-1}{2}}, & i = m-n, \cdots, m-1. \end{cases}$$

For other values of m and n, C can be obtained similarly. The results are presented in Appendix B and verified by Visual C++ simulations. Similar results can be obtained by using $r = x^{n+1}$ as the Montgomery factor.

8 Comparison

In this section, we compare our results to their best counterparts from the same category available in the literature. Table 2 compares our proposed bit-serial Montgomery multipliers to those of [21] and [5]. Note that these multipliers can be derived from the digit-serial multipliers of [2] and [5], respectively, if the digit size is equal to one. Although our bit-serial multipliers can be used for the general Montgomery factor $r = x^u, 1 \le u \le m$, they are only compared for two values of $u = m-1$ and $u = m$. This is because no precomputation is required in the initialization step of the multiplication algorithms. The critical path delay of the multiplier proposed in [5] is $2T_A + 2T_X$ and it has the latency of m clock cycles. Our proposed LSB-first bit-serial multiplier of Fig. 4 has the critical path delay of $T_A + 2T_X$ and the latency of $(m+1)$ and m clock cycles for $r = x^m$ and $r = x^{m-1}$, respectively. We have also proposed an MSB-first bit-serial multiplier (Fig. 2) which has the critical path delay of $T_A + T_X$ and the latency of $(m+1)$ and m clock cycles for $r = x^m$ and $r = x^{m-1}$, respectively. Thus, both of our bit-serial multipliers are faster than the bit-serial multiplier of [5]. Note that our MSB-first bit-serial Montgomery multiplier is the fastest bit-serial Montgomery multiplier. All of these three bit-serial Montgomery multipliers require $(2m-1)$ XOR gates and $(2m-1)$ AND gates and two m-bit registers. As seen from Table 2, the LSB-first bit-serial PB multiplier and our MSB-first Montgomery multiplier have the same and least time complexity. Thus, our MSB-first MM (with the LSB-first PB multiplier) can be used in the digit-serial MM algorithms, such as the one presented in [11], to reduce the overall time complexity (see Remark 1). Also, the LSB-first bit-serial PB multiplier and our MSB-first Montgomery multiplier have the same area/time complexity.

We compare our proposed bit-parallel Montgomery multiplier using irreducible trinomials with the Montgomery multiplier [16], two PB multipliers [23], [3], and an SPB multiplier as shown in Table 3. Note that although bit-parallel multipliers can be derived from digit-serial multipliers using the digit size m, they are not optimized. Based on Table 3 all of the multipliers have the same area complexity. The time complexity of our multiplier is lower than those of [23], [3], and [16] and equal to the ones proposed in [6].

TABLE 3
Comparison of Bit-Parallel Multipliers Using Irreducible Trinomials over $GF(2^m)$

Multiplier	Type	#AND	#XOR	Delay
$F(z) = z^m + z^k + 1,\ k \neq \frac{m}{2}$				
[6]	SPB $(v = k, k-1)$	m^2	$m^2 - 1$	$\begin{cases} T_A + \lceil \log_2 (m+v) \rceil T_X, & v > \frac{m-1}{2} \\ T_A + \lceil \log_2 (2m-v-1) \rceil T_X, & v \leq \frac{m-1}{2} \end{cases}$
[3]	PB	m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 (m-1) \rceil) T_X,\ 1 \leq k < \frac{m}{2}$
[23]		m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 (m-1) \rceil) T_X$
[16] $(u = k)$	MM	m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 (m-2) \rceil) T_X$
Proposed $(u = k, k-1)$		m^2	$m^2 - 1$	$\begin{cases} T_A + \lceil \log_2 (m+u) \rceil T_X, & u > \frac{m-1}{2} \\ T_A + \lceil \log_2 (2m-u-1) \rceil T_X, & u \leq \frac{m-1}{2} \end{cases}$
$F(z) = z^m + z^{\frac{m}{2}} + 1$				
[6]	SPB $(v = k, k-1)$	m^2	$m^2 - \frac{m}{2}$	$T_A + (\lceil \log_2 (m+v) \rceil) T_X$
[3]	PB	m^2	$m^2 - \frac{m}{2}$	$T_A + (1 + \lceil \log_2 (m) \rceil) T_X$
[23]		m^2	$m^2 - \frac{m}{2}$	$T_A + (1 + \lceil \log_2 (m) \rceil) T_X$
[16] $(u = k)$	MM	m^2	$m^2 - \frac{m}{2}$	$T_A + (1 + \lceil \log_2 (m-1) \rceil) T_X$
Proposed $(u = k, k-1)$		m^2	$m^2 - \frac{m}{2}$	$T_A + (\lceil \log_2 (m+u) \rceil) T_X$

Our second bit-parallel Montgomery multiplier is designed for Type-II irreducible pentanomials. In this case, we have proven that two Montgomery factors can result in efficient hardware implementation. Then, we have designed two bit-parallel multipliers. Here, we compare our multipliers to the multipliers of [6] and [4] which are based on type-II irreducible pentanomials. The results are shown in Table 4. The multiplier of [6] uses $v = n + 1$ for the SPB and it has the same time complexity as our FMM. However, our multiplier uses two Montgomery factors, i.e., $u = n, n + 1$, and requires a few gates less than the one presented in [6]. The multiplier of [4] has higher delay than our fast multiplier, but it requires less hardware. The comparison of the multiplier of [4] and our LCMM depends on the value of n. For some value of n, our LCMM is faster and for some values of n, they have the same delay. Note that in [4], n should satisfy $2 \leq n \leq \lfloor \frac{m}{2} \rfloor - 1$, whereas in our design it should satisfy $2 \leq n \leq m - 3$. The area complexity of [4] also depends on n. For some values of n, it has less XOR gates than our LCMM, whereas for some values of n our LCMM requires less XOR gates. To show the differences among those multipliers, we use $m = 163$ which is recommended by NIST for elliptic curve digital signatures algorithm [20]. There are three irreducible pentanomials of degree 163 and $2 \leq n \leq \lfloor \frac{m}{2} \rfloor - 1$. We present the complexity of the multipliers using these three pentanomials in Table 5.

To the best knowledge of the authors, squaring using type-II irreducible pentanomials has not been considered before. However, in [27], it is shown that at most $4(m-1)$ additions are required for squaring using general irreducible pentanomials. In [28], the complexity of squaring is presented for some pentanomials after optimization. We compare the results reported in [27] and [28] to ours in Table 6. It is clear that in our presented squarers, the delay is reduced to $2T_X$ for type-II irreducible pentanomials with slightly less number of XOR gates. This delay is equal to the delay of squaring in the PB using trinomials $(x^m + x^k + 1)$, where $m + k$ is an odd number.

Therefore, our squarer together with our proposed FMM can be used to accelerate scalar multiplication in ECC.

9 Conclusions

In this paper, we have studied the Montgomery multiplication and squaring over $GF(2^m)$. Using new Montgomery factors, we have proposed two bit-serial Montgomery multipliers which are faster than the previously published Montgomery multipliers. Also, we have proposed new bit-parallel Montgomery multipliers for the general and two special classes of irreducible polynomials. The time and area complexities of these multipliers match the best results reported in the literature. We have shown that among the general irreducible pentanomials, type-II irreducible pentanomials are very suitable for the proposed multiplier. Then, we have designed two bit-parallel Montgomery multipliers. Our LCMM requires less hardware than the

TABLE 4
Comparison of Bit-Parallel Multipliers Using Irreducible Pentanomials over $GF(2^m)$

Multiplier	Type	#AND	#XOR	Delay
$F(z) = z^m + z^{n+2} + z^{n+1} + z^n + 1$				
[6]	SPB $(v = n+1)$	m^2	$m^2 + 3m - 7$	$\begin{cases} T_A + (1 + \lceil \log_2 (m+n) \rceil) T_X, & n > \frac{m-1}{2} \\ T_A + (1 + \lceil \log_2 (2m-n-2) \rceil) T_X, & n \leq \frac{m-1}{2} \end{cases}$
[4]	DB	m^2	$m^2 + 2m - \lceil \frac{m-2}{2} \rceil + 3n - 4$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X,\ n \leq \frac{m-1}{2}$
LCMM	MM $(u = n, n+1)$	m^2	$m^2 + 2m - 3$	$\begin{cases} T_A + \left(1 + \lceil \log_2 \left(\lceil \frac{m-u}{2} \rceil + 4u - 5 \right) \rceil \right) T_X, & u > \frac{m-1}{2} \\ T_A + \left(1 + \lceil \log_2 \left(\lceil \frac{u+1}{2} \rceil + 4m - 4u - 9 \right) \rceil \right) T_X, & u \leq \frac{m-1}{2} \end{cases}$
FMM		m^2	$m^2 + 3m - 9$	$\begin{cases} T_A + (1 + \lceil \log_2 (m+n) \rceil) T_X, & n \geq \frac{m-1}{2} \\ T_A + (1 + \lceil \log_2 (2m-n-2) \rceil) T_X, & n < \frac{m-1}{2} \end{cases}$

TABLE 5
Comparison of Multipliers for Irreducible Pentanomials

Multiplier	Type	#AND	#XOR	Delay
$F(z) = z^{163} + z^{68} + z^{67} + z^{66} + 1$				
[6]	SPB	26569	27051	$T_A + 10T_X$
[4]	DB	26569	27008	$T_A + 11T_X$
Proposed FMM	MM	26569	27049	$T_A + 10T_X$
Proposed LCMM		26569	26892	$T_A + 10T_X$
$F(z) = z^{163} + z^{70} + z^{69} + z^{68} + 1$				
[6]	SPB	26569	27051	$T_A + 10T_X$
[4]	DB	26569	27014	$T_A + 11T_X$
Proposed FMM	MM	26569	27049	$T_A + 10T_X$
Proposed LCMM		26569	26892	$T_A + 10T_X$
$F(z) = z^{163} + z^{72} + z^{71} + z^{70} + 1$				
[6]	SPB	26569	27051	$T_A + 9T_X$
[4]	DB	26569	27020	$T_A + 11T_X$
Proposed FMM	MM	26569	27049	$T_A + 9T_X$
Proposed LCMM		26569	26892	$T_A + 10T_X$

TABLE 6
Comparison of Squarers for Irreducible Pentanomials

Irreducible Polynomial	#XOR	Delay
Polynomial Basis		
$F(z) = z^m + z^{k_3} + z^{k_2} + z^{k_1} + 1$ [27]	$\leq 4(m-1)$	-
$F(z) = z^{163} + z^7 + z^6 + z^3 + 1$ [28]	246	$3T_X$
Proposed Montgomery squaring		
$F(z) = z^m + z^{n+2} + z^{n+1} + z^n + 1$	$\leq \frac{m-3}{2} + m + 4$	$2T_X$
e.g., $F(z) = z^{163} + z^{72} + z^{71} + z^{70} + 1$	245	$2T_X$

shifted polynomials basis multiplier, however, for a few irreducible pentanomials, it has a higher delay. Our FMM multiplier is faster than dual basis multiplier, but requires more hardware. Also, FMM has the same time complexity in comparison to the SPB multiplier, but it can be implemented with two Montgomery factors. Moreover, it can be used with our proposed squarer for type-II irreducible pentanomials which has the delay of two XOR gates. This is the lowest reported delay for squaring using pentanomials. As a result, scalar multiplication in ECC can be accelerated by using our multipliers and squarer.

APPENDIX A

Example 1. Assuming $F(z) = z^7 + z^4 + z^3 + z^2 + 1$ and using $u = 2$, we can obtain the matrix M for the bit-parallel Montgomery multiplication as follows:

$$\begin{bmatrix} A'_{(-2)} & A'_{(-1)} & A'_{(0)} & A'_{(1)} & A'_{(2)} & A'_{(3)} & A'_{(4)} & \\ a_2+a_0 & a_1 & a_0 & a_6 & a_5 & a_4 & a_3+a_6 & x^0 \\ a_3+a_0+a_1 & a_2+a_0 & a_1 & a_0 & a_6 & a_5 & a_4 & x^1 \\ a_4+a_0+a_1 & a_3+a_0 & a_2 & a_1+a_6 & a_0+a_5 & a_6+a_4 & a_5+a_3+a_6 & x^2 \\ a_5+a_1 & a_4+a_0 & a_3 & a_2+a_6 & a_1+a_6+a_5 & a_0+a_5+a_4 & a_6+a_4+a_3+a_6 & x^3 \\ a_6 & a_5 & a_4 & a_3+a_6 & a_2+a_6+a_5 & a_1+a_6+a_5+a_4 & a_0+a_5+a_4+a_3+a_6 & x^4 \\ a_0 & a_6 & a_5 & a_4 & a_3+a_6 & a_2+a_6+a_5 & a_1+a_6+a_5+a_4 & x^5 \\ a_1 & a_0 & a_6 & a_5 & a_4 & a_3+a_6 & a_2+a_6+a_5 & x^6 \end{bmatrix}$$

The column $A'_{(0)}$ requires no XOR gate. The Column $A'_{(1)}$ requires three XOR gates to compute $(a_1 + a_6)$, $(a_2 + a_6)$, and $(a_3 + a_6)$. The Column $A'_{(2)}$ also requires three XOR gates, one to compute $(a_0 + a_5)$, one for $((a_1 + a_6) + a_5)$ as $(a_1 + a_6)$ is reused, and one for $((a_2 + a_6) + a_5)$ as $(a_2 + a_6)$ is reused. In Column $A'_{(3)}$, one XOR is required for $(a_6 + a_4)$ and one for $((a_0 + a_5) + a_4)$ as $(a_0 + a_5)$ is reused. In the FMM, $(a_1 + a_6 + a_5 + a_4)$ is obtained by $((a_1 + a_6) + (a_5 + a_4))$ reusing $(a_1 + a_6)$ which results in using two XOR gates with the delay of $2T_X$. In the LCMM, $(a_1 + a_6 + a_5 + a_4)$ is obtained by $((a_1 + a_6 + a_5) + a_4)$ reusing $(a_1 + a_6 + a_5)$. In this case, one XOR gate is required but the delay is $3T_X$. In the column $A'_{(4)}$, one XOR gate is required for obtaining $(a_5 + (a_3 + a_6))$ as $(a_3 + a_6)$ is reused. Obtaining $((a_6 + a_4) + (a_3 + a_6))$ requires one XOR gate as both $(a_6 + a_4)$ and $(a_3 + a_6)$ are reused. There are two possibilities to compute $(a_0 + a_5 + a_4 + a_3 + a_6)$. In the FMM, it is considered as $((a_0 + a_5 + a_4) + (a_3 + a_6))$, where both $(a_0 + a_5 + a_4)$ and $(a_3 + a_6)$ are reused. But the addition is postponed to the final XOR tree, thus no XOR gate is required. In the LCMM, $((a_0 + a_5 + a_4) + (a_3 + a_6))$ is obtained by an XOR gate and the delay of $3T_X$. Note that $A'_{(-1)}$ and $A'_{(-2)}$ are obtained similarly.

The maximum delay in FMM occurs in the position x^3 (in computing c_3), where $c_3 = ((a_5 + a_1)b_0 + (a_4 + a_0)b_1) + (a_3b_2 + (a_2 + a_6)b_3) + ((a_1 + a_6) + a_5)b_4 + ((a_0 + a_5) + a_4)b_5 + ((a_3 + a_6) + (a_6 + a_4))b_6$. By implementing the terms with the order represented by the brackets, c_3 is obtained with the delay of $T_A + (2 + \lceil \log_2(5) \rceil)T_X = T_A + 5T_X$. Note that the delay of the LCMM can be obtained similarly. □

APPENDIX B

For odd m and $n, n \geq \frac{m+1}{2}$, we have $c_i =$

$$\begin{cases} a_0 + a_{\frac{m+n}{2}} + a_{m-1}, & i = 0, \\ a_{\lfloor \frac{i}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1 + \lfloor \frac{i}{2} \rfloor}, & i = 1, \ldots, n-2, \\ a_{\frac{i}{2}-1} + a_{\frac{i}{2}} + a_{\frac{m+n}{2}+\frac{i}{2}}, & i = 2, \ldots, m-n-2, \\ a_{\frac{i-(m-n)}{2}} + a_{\frac{i}{2}-1} + a_{\frac{i}{2}}, & i = m-n, \ldots, n-1, \\ a_{\lfloor \frac{n}{2} \rfloor} + a_n + a_{\frac{m+n}{2}} + a_{m-1}, & i = n, \\ a_{\frac{2n-m+1}{2}} + a_{\lfloor \frac{n}{2} \rfloor} + a_{\frac{m+n}{2}} + a_{m-1}, & i = n+1, \\ a_{n+1} + a_{\frac{m+n}{2}} + a_{\frac{m+n}{2}+1} + a_{m-1}, & i = n+2, \\ a_{\frac{i-(m-n)}{2}} + a_{\frac{m+n}{2}+\frac{i-n}{2}}, & i = n+3, \ldots, m-1, \\ a_{\lfloor \frac{n}{2} \rfloor + 1 + \lfloor \frac{i}{2} \rfloor} + a_{\frac{m+i}{2}-1} + a_{\frac{m+i}{2}}, & i = n+4, \ldots, m-2, \end{cases}$$

which requires at most $(\frac{m-3}{2} + m + 4)$ XOR gates. For odd m, even n, and $n < \frac{m-3}{2}$, we have $c_i =$

$$\begin{cases} a_0 + a_{\frac{n}{2}} + a_{m-1}, & i = 0, \\ a_{\lfloor \frac{i}{2} \rfloor} + a_{\lfloor \frac{m+n+2}{2} \rfloor + \lfloor \frac{i}{2} \rfloor}, & i = 1, \ldots, n-1, \\ a_{\frac{i}{2}-1} + a_{\frac{i}{2}} + a_{\frac{n}{2}+\frac{i}{2}}, & i = 2, \ldots, n-2, \\ a_{\frac{n}{2}-1} + a_n + a_{m-1}, & i = n, \\ a_{\lfloor \frac{m+n+2}{2} \rfloor} + a_{\lfloor \frac{m+n+2}{2} \rfloor + \lfloor \frac{n+1}{2} \rfloor} + a_{m-1}, & i = n+1, \\ a_{\frac{2n+2}{2}} + a_{\lfloor \frac{m+n+2}{2} \rfloor} + a_{m-1}, & i = n+2, \\ a_{\lfloor \frac{m+n}{2} \rfloor + \frac{i-n-1}{2}} + a_{\lfloor \frac{m+n}{2} \rfloor + \frac{i-n+1}{2}} + a_{\lfloor \frac{m+n+2}{2} \rfloor + \lfloor \frac{i}{2} \rfloor}, & i = n+3, \ldots, m-n-2, \\ a_{\frac{n}{2}+\frac{i}{2}} + a_{\lfloor \frac{m+n}{2} \rfloor + \lfloor \frac{i-n+1}{2} \rfloor}, & i = n+4, \ldots, m-1, \\ a_{\frac{i-(m-n)}{2}} + a_{\lfloor \frac{m+n}{2} \rfloor + \frac{i-n-1}{2}} + a_{\lfloor \frac{m+n}{2} \rfloor + \frac{i-n+1}{2}}, & i = m-n, \ldots, m-2, \end{cases}$$

which requires at most $(\frac{m-3}{2} + m + 2)$ two-input XOR gates. For odd m, even n, and $n \geq \frac{m+1}{2}$, we have $c_i =$

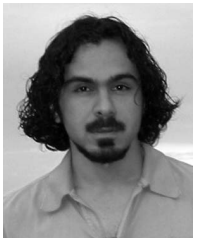
$$\begin{cases} a_0 + a_{\frac{n}{2}} + a_{m-1}, & i = 0, \\ a_{\lfloor \frac{i}{2} \rfloor} + a_{\lfloor \frac{m+n}{2} \rfloor + 1 + \lfloor \frac{i}{2} \rfloor}, & i = 1, \ldots, m-n-2, \\ a_{\frac{i}{2}-1} + a_{\frac{i}{2}} + a_{\frac{n}{2}+\frac{i}{2}}, & i = 2, \ldots, n-2, \\ a_{\frac{i-(m-n)}{2}} + a_{\lfloor \frac{i}{2} \rfloor}, & i = m-n, \ldots, n-1, \\ a_{\frac{n}{2}-1} + a_n + a_{m-1}, & i = n, \\ a_{\frac{2n-m+1}{2}} + a_{\lfloor \frac{m+n+2}{2} \rfloor} + a_{m-1}, & i = n+1, \\ a_{\frac{n}{2}+\frac{n+2}{2}} + a_{\lfloor \frac{m+n+2}{2} \rfloor} + a_{m-1}, & i = n+2, \\ a_{\frac{i-(m-n)}{2}} + a_{\lfloor \frac{m+n}{2} \rfloor + \frac{i-n-1}{2}} + & i = n+3, \ldots, m-2, \\ a_{\lfloor \frac{m+n}{2} \rfloor + \frac{i-n+1}{2}}, & \\ a_{\frac{n}{2}+\frac{i}{2}} + a_{\lfloor \frac{m+n}{2} \rfloor + \lfloor \frac{i-n+1}{2} \rfloor}, & i = n+4, \ldots, m-1. \end{cases}$$

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments. This work has been supported in part by an NSERC Discovery grant awarded to Arash Reyhani-Masoleh.

REFERENCES

[1] E.D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," PhD dissertation, Linkoping Univ., 1991.
[2] L. Song and K. Parhi, "Low-Energy Digit-Serial/Parallel Finite Field Multipliers," *J. Very Large Scale Integration (VLSI) Signal Processing,* vol. 19, no. 2, pp. 149-166, 1998.
[3] A. Reyhani-Masoleh and M. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$," *IEEE Trans. Computers,* vol. 53, no. 8, pp. 945-959, Aug. 2004.
[4] F. Rodriguez-Henriguez and C. Koc, "Parallel Multipliers Based on Special Irreducible Pentanomials," *IEEE Trans. Computers,* vol. 52, no. 12, pp. 1535-1542, Dec. 2003.
[5] C. Koc and T. Acar, "Montgomery Multiplication in $GF(2^k)$," *Designs, Codes and Cryptography,* vol. 14, no. 1, pp. 57-69, 1998.
[6] H. Fan and M. Hasan, "Fast Bit Parallel Shifted Polynomial Basis Multipliers in $GF(2^n)$," *IEEE Trans. Circuits and Systems I, Fundamental Theory and Applications,* vol. 53, no. 12, pp. 2606-2615, Dec. 2006.
[7] P. Montgomery, "Modular Multiplication without Trial Division," *Math. Computation,* vol. 44, no. 170, pp. 519-521, 1985.
[8] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "High-Performance Public-Key Cryptoprocessor for Wireless Mobile Applications," *Mobile Networks and Applications,* vol. 12, no. 4, pp. 245-258, 2007.
[9] N. Mentens, S.B. Ors, B. Preneel, and J. Vandewalle, "An FPGA Implementation of a Montgomery Multiplier over $GF(2^m)$," *Proc. Seventh IEEE Workshop Design and Diagnostics of Electronic Circuits and Systems (DDECS),* pp. 121-128, 2004.
[10] C. Chiou, C. Lee, A. Deng, and J. Lin, "Concurrent Error Detection in Montgomery Multiplication over $GF(2^m)$," *IEICE Trans. Fundamentals of Electronics, Comm. and Computer Sciences,* vol. 89, no. 2, pp. 566-574, 2006.
[11] L. Batina, N. Mentens, B. Preneel, and I. Verbauwhede, "Balanced Point Operations for Side Channel Protection of Elliptic Curve Cryptography," *Proc. IEE: Information Security,* vol. 152, no. 1, pp. 57-65, 2005.
[12] E. Savas, A. Tenca, M. Ciftcibasi, and Ç. Koc, "Novel Multiplier Architectures for $GF(p)$ and $GF(2^n)$," *Proc. IEE: Computers and Digital Techniques,* vol. 151, no. 2, pp. 147-160, 2004.
[13] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An Improved Unified Scalable Radix-2 Montgomery Multiplier," *Proc. 17th IEEE Symp. Computer Arithmetic,* pp. 172-178, 2005.
[14] A. Fournaris and O. Koufopavlou, "Versatile Multiplier Architectures in $GF(2^k)$ Fields Using the Montgomery Multiplication Algorithm," *Integration, the Very Large Scale Integration (VLSI) J.,* vol. 41, no. 3, pp. 371-384, 2008.
[15] J.S. Horng and E.H. Lu, "Low-Complexity Bit-Parallel Systolic Montgomery Multipliers for Special Classes of GF(2^m)," *IEEE Trans. Computers,* vol. 54, no. 9, pp. 1061-1070, Sept. 2005.
[16] H. Wu, "Montgomery Multiplier and Squarer for a Class of Finite Fields," *IEEE Trans. Computers,* vol. 51, no. 5, pp. 521-529, May 2002.
[17] B. Ansari and M. Hasan, "High Performance Architecture of Elliptic Curve Scalar Multiplication," *IEEE Trans. Computers,* vol. 57, no. 11, pp. 1443-1453, Nov. 2008.
[18] Y. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID," *IEEE Trans. Computers,* vol. 57, no. 11, pp. 1514-1527, Nov. 2008.
[19] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore Curve-Based Cryptoprocessor with Reconfigurable Modular Arithmetic Logic Units over $GF(2^n)$," *IEEE Trans. Computers,* vol. 56, no. 9, pp. 1269-1282, Sept. 2007.
[20] Recommended Elliptic Curves for Federal Government Use, csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, 2009.
[21] T. Beth and D. Gollman, "Algorithm Engineering for Public Key Algorithms," *IEEE J. Selected Areas in Comm.,* vol. 7, no. 4, pp. 458-466, May 1989.
[22] S. Kumar, T. Wollinger, and C. Paar, "Optimum Digit Serial $GF(2^m)$ Multipliers for Curve-Based Cryptography," *IEEE Trans. Computers,* vol. 55, no. 10, pp. 1306-1311, Oct. 2006.
[23] J. Imana and J. Sanchez, "Bit-Parallel Finite Field Multipliers for Irreducible Trinomials," *IEEE Trans. Computers,* vol. 55, no. 5, pp. 520-533, May 2006.
[24] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications.* Cambridge Univ. Press, 1986.
[25] S. Park, K. Chang, and D. Hong, "Efficient Bit-Parallel Multiplier for Irreducible Pentanomials Using a Shifted Polynomial Basis," *IEEE Trans. Computers,* vol. 55, no. 9, pp. 1211-1215, Sept. 2006.
[26] J. Imana, R. Hermida, and F. Tirado, "Low Complexity Bit-Parallel Multipliers Based on a Class of Irreducible Pentanomials," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 14, no. 12, pp. 1388-1393, Dec. 2006.
[27] H. Wu, "Low Complexity Bit-Parallel Finite Field Arithmetic Using Polynomial Basis," *Proc. First Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES),* pp. 280-291, 1999.
[28] J. Guajardo, T. Güneysu, S. Kumar, C. Paar, and J. Pelzl, "Efficient Hardware Implementation of Finite Fields with Applications to Cryptography," *Acta Applicandae Math.: Int'l Survey J. Applying Math. and Math. Applications,* vol. 93, no. 1, pp. 75-118, 2006.



Arash Hariri received the BSc degree from Amirkabir University of Technology, Iran, in 2003, and the MSc degree from Shahid Beheshti University, Iran, in 2006, both in computer engineering. He is currently working toward the PhD degree in electrical and computer engineering at The University of Western Ontario, Canada. His current research interests include computer arithmetic, cryptography, and fault tolerance. He is a student member of the IEEE.

Arash Reyhani-Masoleh received the BSc degree from the Iran University of Science and Technology in 1989, and the MSc degree from the University of Tehran in 1991, both with the first rank in electrical and electronic engineering, and the PhD degree in electrical and computer engineering from the University of Waterloo in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Centre for Applied Cryptographic Research, University of Waterloo. In October 2004, he joined the Department of Electrical and Computer Engineering, University of Western Ontario, London, Canada, as an assistant professor. His current research interests include algorithms and VLSI architectures for computations in finite fields, fault tolerant computing, and error-control coding. He was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) postdoctoral fellowship in 2002. Currently, he is an associate editor of *Integration, the VLSI Journal* (Elsevier). He is a member of the IEEE and the IEEE Computer Society.