# A Chinese Remainder Theorem Approach to Bit-Parallel $GF(2^n)$ Polynomial Basis Multipliers for Irreducible Trinomials

Haining Fan

**Abstract**

We show that the step "modulo the degree-$n$ field generating irreducible polynomial" in the classical definition of the $GF(2^n)$ multiplication operation can be avoided. This leads to an alternative representation of the finite field multiplication operation. Combining this representation and the Chinese Remainder Theorem, we design bit-parallel $GF(2^n)$ multipliers for irreducible trinomials $u^n + u^k + 1$ on $GF(2)$ where $1 < k \leq n/2$. For some values of $n$, our architectures have the same time complexity as the fastest bit-parallel multipliers – the quadratic multipliers, but their space complexities are reduced. Take the special irreducible trinomial $u^{2k} + u^k + 1$ for example, the space complexity of the proposed design is reduced by about $1/8$, while the time complexity matches the best result. Our experimental results show that among the 539 values of $n$ such that $4 < n < 1000$ and $x^n + x^k + 1$ is irreducible over $GF(2)$ for some $k$ in the range $1 < k \leq n/2$, the proposed multipliers beat the current fastest parallel multipliers for 290 values of $n$ when $(n-1)/3 \leq k \leq n/2$: they have the same time complexity, but the space complexities are reduced by $8.4\%$ on average.

**Index Terms**

Finite field, multiplication, polynomial basis, the Chinese Remainder Theorem.

## I. Introduction

Finite field $GF(2^n)$ multipliers can be classified according to different criteria [1], for example, bases (polynomial, normal and dual bases etc.), working mode (bit-serial, bit-parallel and

Haining Fan is with the School of Software, Tsinghua University, Beijing, China. E-mail: fhn@tsinghua.edu.cn

digital serial), design approach (polynomial-based and matrix-based) and algorithm complexity (quadratic, subquadratic and hybrid), etc. In VLSI, the space and time complexities are two major factors to measure the efficiency of a bit-parallel multiplier. The space complexity is usually represented in terms of the total number of 2-input XOR gates (the $GF(2)$ addition) and AND gates (the $GF(2)$ multiplication) used. The corresponding time complexity is given in terms of the maximum delay faced by a signal due to these XOR and AND gates. Symbols "$T_A$" and "$T_X$" are often used to represent the delays of one 2-input AND gate and one 2-input XOR gate, respectively.

The quadratic bit-parallel multipliers usually adopt the schoolbook methods to compute the product of two polynomials or a Toeplitz matrix-vector product (TMVP), e.g.,

$$(a_1x + a_0) \cdot (b_1x + b_0) = a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0 \tag{1}$$

and

$$T \cdot V = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} T_1V_0 + T_0V_1 \\ T_2V_0 + T_1V_1 \end{pmatrix}, \tag{2}$$

where $T$ is a $2^i \times 2^i$ Toeplitz matrix and $V$ a $2^i \times 1$ column vector, $T_0$, $T_1$ and $T_2$ are $2^{i-1} \times 2^{i-1}$ block submatrices of $T$, and $V_0$ and $V_1$ are $2^{i-1} \times 1$ subvectors of $V$.

On the other hand, the subquadratic bit-parallel multipliers usually adopt the Karatsuba formulae or their matrix versions, e.g.,

$$(a_1x + a_0)(b_1x + b_0) = (a_1b_1)x^2 + \{[(a_0 + a_1)(b_0 + b_1)] - [a_1b_1 + a_0b_0]\}x + a_0b_0 \tag{3}$$

and

$$T \cdot V = \begin{pmatrix} T_1 & T_0 \\ T_2 & T_1 \end{pmatrix} \begin{pmatrix} V_0 \\ V_1 \end{pmatrix} = \begin{pmatrix} P_0 + P_2 \\ P_1 + P_2 \end{pmatrix}, \tag{4}$$

where $P_0$, $P_1$ and $P_2$ are three TMVPs of size $2^{i-1}$:

$$\begin{cases} P_0 = (T_0 + T_1)V_1, \\ P_1 = (T_1 + T_2)V_0, \\ P_2 = T_1(V_0 + V_1). \end{cases}$$

The main advantage of the subquadratic multipliers is that their space complexities are often small. But their time complexities are often larger than their quadratic counterparts. In fact, the current fastest bit-parallel $GF(2^n)$ multiplier are all based on the quadratic approach. For

practical applications, the hybrid approach can provide a trade-off between the time and space complexities, see for example, [2], [3] and [4] etc. These multipliers first perform a few sub-quadratic iterations to reduce the whole space complexities, and then a quadratic algorithm on small input operands to achieve lower time complexity.

There also exist some other hybrid multipliers. In [5], the Karatsuba formula (3) is used only in the first step to compute the product of two degree-$(n-1)$ polynomials, and then the schoolbook formula (1) is used to compute the 3 products of 6 degree-$\frac{n-1}{2}$ polynomials. Compared to pure quadratic multipliers, the space complexity of this multiplier is reduced by about $1/4$, but its time complexity increases by $1T_X$.

Regarding the matrix approach, the block recombination method presented in [6] computes a $2^i \times 2^i$ TMVP $(T \cdot V)$ using the following two steps:

(i). Transfer the $2^i \times 2^i$ TMVP $(T \cdot V)$ into 4 TMVPs of size $2^{i-1}$ using (2);

(ii). Compute these 4 TMVPs using (4).

While keeping the time complexity unchanged, this method reduces the XOR gate complexity at the cost of an increase of the AND gates. Therefore, "it is more suitable for ASIC implementations as the area of an XOR gate is larger than that of an AND gate in CMOS libraries".

In this work, we also follow the "1-subquadratic-and-then-quadratic" computational mode, and present a "symmetrical" result of [6]. Instead of using the Karatsuba formula (3) in the first step, we use the Chinese Remainder Theorem (CRT). The key point of the proposed multipliers is an alternative representation of the finite field multiplication operation, which is introduced in Section II. Under this representation, the step "modulo the degree-$n$ field generating irreducible polynomial" in the classical definition of the $GF(2^n)$ multiplication operation can be avoided. In Section III, we present a new explicit formula of the product of two $GF(2^n)$ elements for irreducible trinomials $u^n + u^k + 1$ $(1 < k < n/2)$. In Section IV, we describe the structures of two types of parallel multipliers, and compare them with the fastest bit-parallel multipliers – the quadratic multipliers. For some values of $n$, our architectures have the same time complexity as the fastest quadratic multipliers, but their space complexities are reduced. Especially, we present two types of parallel multipliers for irreducible trinomials $u^{2k} + u^k + 1$ in Section V. The space complexity of these multipliers is only about $7/8$ that of the current fastest multiplier, while they have the same time complexity. Finally, concluding remarks are made in Section VI.

## II. BASIC IDEA

Let $f(u) = u^n + u^k + 1$ $(n > 2)$ be an irreducible trinomial of degree $n$ over $GF(2)$. All elements of the finite field $GF(2^n) := GF(2)[u]/(f(u))$ can be represented using a polynomial basis $\{x^i | 0 \leq i \leq n-1\}$, where $x$ is a root of $f$. Given two field elements $a(x) = \sum_{i=0}^{n-1} a_i x^i$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i$, where $a_i, b_i \in GF(2)$, the classical polynomial basis multiplication algorithm computes the $GF(2^n)$ product $c(x) = \sum_{i=0}^{n-1} c_i x^i$ of $a(x)$ and $b(x)$ using the following two steps. For the sake of simplicity, we omit "$(x)$" in polynomial "$a(x)$" and denote $a(x)$ by $a$.

(i) Conventional polynomial multiplication:

$$s = a \cdot b = \sum_{t=0}^{2n-2} s_t x^t,$$

where

$$s_t = \sum_{\substack{i+j=t \\ 0 \leq i,j < n}} a_i b_j = \begin{cases} \sum\limits_{i=0}^{t} a_i b_{t-i}, & 0 \leq t \leq n-1, \\ \sum\limits_{i=t+1-n}^{n-1} a_i b_{t-i}, & n \leq t \leq 2n-2. \end{cases} \tag{5}$$

(ii) Reduction $\bmod f = x^n + x^k + 1$:

$$c = \sum\nolimits_{i=0}^{n-1} c_i x^i = s \bmod f. \tag{6}$$

For the quadratic parallel multipliers, the schoolbook method is often used in the first step. On the other hand, the subquadratic parallel multipliers often use the Karatsuba formula, which is a special case of the following Chinese Remainder Theorem [7].

*Theorem 1 (The polynomial CRT):* Let $F$ be a field, $t > 1$, $m_1, m_2, \cdots, m_t$ be pairwise coprime polynomials in $F[x]$, $M = \prod_{i=1}^{t} m_i$ and $M_i = \frac{M}{m_i}$. The unique solution $y \bmod M$ to the system of linear congruences $\langle y \rangle_{m_i} = y_i$ is

$$y = \left\langle \sum_{i=1}^{t} y_i \cdot M_i \cdot \left\langle M_i^{-1} \right\rangle_{m_i} \right\rangle_M,$$

where $\langle y \rangle_{m_i}$ denotes the remainder of $y \bmod m_i$, and $\left\langle M_i^{-1} \right\rangle_{m_i}$ is the multiplicative inverse of $M_i \bmod m_i$ and $1 \leq i \leq t$.

The product $c$ obtained in the second step is the remainder of $s$ divided by $f$, which is defined in the following polynomial division algorithm.

*Theorem 2 (The polynomial division algorithm):* Let $F$ be a field, $f, s \in F[x]$ with $deg(f) > 0$. Then there exist unique polynomials $q$ (the quotient) and $c$ (the remainder) in $F[x]$ with $0 \leq degree(c) < degree(f)$ such that

$$s = f \cdot q + c. \tag{7}$$

Because $f$ is irreducible over $GF(2)$, the CRT cannot be used directly in the second step, and it seems that the step "modulo the degree-$n$ irreducible polynomial $f$" in this classical definition of the $GF(2^n)$ multiplication operation can not be avoided. In fact, it can be removed.

In the following, we use the CRT to perform the $GF(2^n)$ multiplication operation. Instead of reducing the irreducible polynomial $f$ in (6), we compute the quotient and the remainder of $s$ modulo the $reducible$ polynomial $f + 1 = x^n + x^k$. Our design is based on the following identity:

$$s = f \cdot q + c = (f + 1)q + (c + q). \tag{8}$$

We note that addition and subtraction are the same in fields of characteristic 2.

Because $degree(s) \leq 2n-2$, $degree(f+1) = n$, $degree(c) \leq n-1$ and $degree(q) \leq n-2$, the degree of the polynomial $(c+q)$ satisfies the condition $degree(c+q) \leq n-1$. Therefore, equation (8) is essentially the division algorithm of $s$ divided by the reducible polynomial $f+1 = x^n+x^k$.

Based on (8), the product $c$ of two $GF(2^n)$ elements $a$ and $b$ can be constructed via the following two steps:

(i). Compute the quotient $q$ and the remainder $(c + q)$ of $s = a \cdot b$ divided by $f + 1$;

(ii). Output the summation of $q$ and $(c + q)$ obtained in step (i), i.e., $c = q + (c + q)$.

In the next section, we derive the explicit formula of $c = ab$ in $GF(2^n)$ for the case $1 < k < n/2$. For the case $k = 1$, i.e., $f + 1 = x^n + x = (x^{n-1} + 1)x$ , our multiplier is the same as the quadratic polynomial basis multipliers, and provides no improvement. In this case, equation (8) is

$$s = (f + 1)q + (c + q) = (x^n + x)q + (c + q) = x^n q + (c + q + xq).$$

Because $degree(xq) \leq n - 1$ and $degree(c) \leq n - 1$, the degree of the remainder $(c + q + xq)$ is also no more than $n - 1$. Therefore, the quotient $q$ is just the most significant part of the degree-$(2n - 2)$ polynomial $s = a \cdot b$, and the proposed two-step method to compute $c$ is the same as the direct calculation of $s \bmod x^n + x + 1$.

## III. The Expression of $c = ab \in GF(2^n)$ for $1 < k < n/2$

In this section, we consider the case $1 < k < n/2$. The special case $2k = n$ will be discussed later. We first derive the expressions of the quotient and the remainder of $s = a \cdot b$ divided by $(f + 1) = x^n + x^k$, and then present the expression of $c = ab$ in $GF(2^n)$.

### A. Compute the quotient $q$ of $s = a \cdot b$ divided by $(f + 1) = x^n + x^k$

Since $degree(q) \leq n - 2$ in (8), we can define $q$ as $q := \sum_{i=0}^{n-2} q_i x^i$. Replacing $f$ in (8) by $f = x^n + x^k + 1$, we have

$$s = (f + 1)q + (c + q) = q \cdot x^n + q \cdot x^k + (c + q).$$

Table I depicts bit positions of the terms $q \cdot x^n$, $q \cdot x^k$ and $(c + q)$ in this equation.

TABLE I

BIT POSITIONS OF THE TERMS IN $s = q \cdot x^n + q \cdot x^k + (c + q)$.

| $2n-2$ | | $2n-k$ | | $n+k-2$ | | $n$ | $n-1$ | | $k$ | $0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $q_{n-2}$ | ... | $q_{n-k}$ | ... | $q_{k-2}$ | ... | $q_0$ | Polynomial $c+q$ | | | |

| | | | | | |
|---|---|---|---|---|---|
| $q_{n-2}$ | ... | $q_{n-k}$ | $q_{n-k-1}$ | ... | $q_0$ |

It is clear that

$$q_i = s_{i+n}, \text{ for } \ k - 1 \leq i \leq n - 2. \tag{9}$$

Moreover, we have $s_{i+n} = q_i + q_{i+n-k}$ for $0 \leq i \leq k - 2$. The terms $q_{i+n-k}$ for $0 \leq i \leq k - 2$ are the same as the terms $q_j$ for $n - k \leq j \leq n - 2$, which also appear in (9). Thus we have

$$q_i = s_{i+n} + s_{i+2n-k}, \quad \text{where } \ 0 \leq i \leq k - 2. \tag{10}$$

Using (9) and (10), we get the following expression of $q$:

$$\begin{aligned} q &= \sum_{i=0}^{n-2} q_i x^i = \sum_{i=0}^{k-2} (s_{i+n} + s_{i+2n-k}) x^i + \sum_{i=k-1}^{n-2} s_{i+n} x^i \\ &= \sum_{i=0}^{n-2} s_{i+n} x^i + \sum_{i=0}^{k-2} s_{i+2n-k} x^i. \end{aligned} \tag{11}$$

*B. Compute the remainder* $(c+q) = \langle a \cdot b \rangle_{f+1}$

Because $k < n/2$, i.e., $2k < n$, we have the Bezout identity [8]

$$x^k \cdot x^{n-2k} + (x^{n-k}+1) \cdot 1 = 1.$$

So we know that $x^{n-2k}$ is the multiplicative inverse of $x^k \bmod (x^{n-k}+1)$ and 1 is the multiplicative inverse of $(x^{n-k}+1) \bmod x^k$. Therefore, the remainder $(c+q) = \langle a \cdot b \rangle_{x^n+x^k}$ can be computed using the CRT as follows:

$$\langle a \cdot b \rangle_{x^n+x^k} = \left\langle \langle a \cdot b \rangle_{x^k} \cdot (x^{n-k}+1) \cdot 1 + \langle a \cdot b \rangle_{x^{n-k}+1} \cdot x^k \cdot x^{n-2k} \right\rangle_{x^n+x^k}. \tag{12}$$

Thus, we need to compute the terms $\langle a \cdot b \rangle_{x^k}$ and $\langle a \cdot b \rangle_{x^{n-k}+1}$ in (12) first. The expression of $\langle a \cdot b \rangle_{x^k}$ can be derived from the expression of $a \cdot b$ given in (5). It is clear that

$$\langle a \cdot b \rangle_{x^k} = \sum_{i=0}^{k-1} s_i x^i. \tag{13}$$

In order to compute the term $\langle a \cdot b \rangle_{x^{n-k}+1} = \left\langle \langle a \rangle_{x^{n-k}+1} \cdot \langle b \rangle_{x^{n-k}+1} \right\rangle_{x^{n-k}+1}$ in (12), we first compute $\langle a \rangle_{x^{n-k}+1}$. Since it is the input of the operation $\left\langle \langle a \rangle_{x^{n-k}+1} \cdot \langle b \rangle_{x^{n-k}+1} \right\rangle_{x^{n-k}+1}$, we define it as $\sum_{i=0}^{n-k-1} g_i x^i$. Because $k < n/2$, i.e., $n-k > n/2$, we have

$$\begin{aligned}
&\sum_{i=0}^{n-k-1} g_i x^i := \langle a \rangle_{x^{n-k}+1} \\
= &\left\langle \sum_{i=0}^{n-k-1} a_i x^i + \sum_{i=n-k}^{n-1} a_i x^i \right\rangle_{x^{n-k}+1} \\
= &\sum_{i=0}^{n-k-1} a_i x^i + \left\langle \sum_{j=0}^{k-1} a_{j+n-k} x^{j+n-k} \right\rangle_{x^{n-k}+1} \\
= &\sum_{i=0}^{n-k-1} a_i x^i + \sum_{i=0}^{k-1} a_{i+n-k} x^i \\
= &\sum_{i=k}^{n-k-1} a_i x^i + \sum_{i=0}^{k-1} (a_i + a_{i+n-k}) x^i.
\end{aligned} \tag{14}$$

Similarly, we can obtain the expression of $\sum_{i=0}^{n-k-1} h_i x^i := \langle b \rangle_{x^{n-k}+1}$.

Now the term $\langle a \cdot b \rangle_{x^{n-k}+1}$ in (12) can be calculated using the schoolbook polynomial multiplication algorithm in (5).

$$\begin{aligned}
& \langle a \cdot b \rangle_{x^{n-k}+1} = \left\langle \langle a \rangle_{x^{n-k}+1} \cdot \langle b \rangle_{x^{n-k}+1} \right\rangle_{x^{n-k}+1} \\
= & \left\langle \left( \sum_{i=0}^{n-k-1} g_i x^i \right) \left( \sum_{i=0}^{n-k-1} h_i x^i \right) \right\rangle_{x^{n-k}+1} \\
= & \left\langle \sum_{i=0}^{n-k-1} \left( \sum_{j=0}^{i} g_j h_{i-j} \right) x^i + \sum_{i=n-k}^{2n-2k-2} \left( \sum_{j=i-n+k+1}^{n-k-1} g_j h_{i-j} \right) x^i \right\rangle_{x^{n-k}+1} \\
= & \sum_{i=0}^{n-k-1} \left( \sum_{j=0}^{i} g_j h_{i-j} \right) x^i + \sum_{t=0}^{n-k-2} \left( \sum_{j=t+1}^{n-k-1} g_j h_{t+n-k-j} \right) x^t \\
= & \left( \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j} \right) x^{n-k-1} + \sum_{i=0}^{n-k-2} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^i. \qquad (15)
\end{aligned}$$

Based on the above equations (12), (13) and (15), we can obtain the following expression of the remainder $(c+q) = \langle a \cdot b \rangle_{f+1}$.

$$\begin{aligned}
& (c+q) = \langle a \cdot b \rangle_{x^n+x^k} \\
= & \left\langle \langle a \cdot b \rangle_{x^k} \cdot (x^{n-k}+1) \cdot 1 + \langle a \cdot b \rangle_{x^{n-k}+1} \cdot x^k \cdot x^{n-2k} \right\rangle_{x^n+x^k} \\
= & \left\langle \left( \sum_{i=0}^{k-1} s_i x^i \right) \cdot (x^{n-k}+1) + \right. \\
& \left. \left[ \left( \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j} \right) x^{n-k-1} + \sum_{i=0}^{n-k-2} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^i \right] x^{n-k} \right\rangle_{x^n+x^k} \\
= & \sum_{i=0}^{k-1} s_i x^{i+n-k} + \sum_{i=0}^{k-1} s_i x^i + \left( \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j} \right) x^{n-k-1} + \\
& \left\langle \sum_{i=0}^{n-k-2} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^{i+n-k} \right\rangle_{x^n+x^k}. \qquad (16)
\end{aligned}$$

The degree of the polynomial in the last angle bracket, i.e., $\sum_{i=0}^{n-k-2} (...) x^{i+n-k}$, is $2n-2k-2$. It is greater than or equal to $n$ when $2k \leq n-2$. In such cases, monomials in the angle bracket with degrees greater than or equal to $n$ should be reduced by $x^n + x^k$. For the cases $2k = n-1$ and $2k = n$, the degree of the polynomial in the angle brackets is less than $n$, so we have $\left\langle \sum_{i=0}^{n-k-2} (...) x^{i+n-k} \right\rangle_{x^n+x^k} = \sum_{i=0}^{n-k-2} (...) x^{i+n-k}$. Therefore, these two cases have different

expressions of $(c+q)$ from the case $2k \leq n-2$, and we discuss them later. For the case $2k \leq n-2$, equation (16) can be rewritten as

$$
\begin{aligned}
& (c+q) \\
= & \sum_{i=0}^{k-1} s_i x^i + \left( \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j} \right) x^{n-k-1} + \sum_{i=0}^{k-1} s_i x^{i+n-k} + \\
& \left\langle \sum_{i=0}^{k-1} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^{i+n-k} \right\rangle_{x^n+x^k} + \\
& \left\langle \sum_{i=k}^{n-k-2} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^{i+n-k} \right\rangle_{x^n+x^k} \\
= & \sum_{i=0}^{k-1} s_i x^i + \left( \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j} \right) x^{n-k-1} + \\
& \sum_{i=0}^{k-1} \left( s_i + \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^{i+n-k} + \\
& \sum_{i=k}^{n-k-2} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^i \\
= & \sum_{i=0}^{k-1} s_i x^i + \sum_{i=k}^{n-k-2} \left( \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j} \right) x^i + \left( \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j} \right) x^{n-k-1} + \\
& \sum_{t=n-k}^{n-1} \left( s_{t-n+k} + \sum_{j=0}^{t-n+k} g_j h_{t-n+k-j} + \sum_{j=t-n+k+1}^{n-k-1} g_j h_{t-j} \right) x^t. \qquad (17)
\end{aligned}
$$

### C. The expression of $c = ab$ in $GF(2^n)$

Based on the expressions of $q$, i.e., (11), and $c+q$, i.e., (17), we can obtain the expression of $c = q + (c+q)$ in $GF(2^n)$.

$$
\begin{aligned}
c &= \sum_{i=0}^{n-2} s_{i+n}x^i + \sum_{i=0}^{k-2} s_{i+2n-k}x^i + \sum_{i=0}^{k-1} s_i x^i + \sum_{i=k}^{n-k-2}\left(\sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j}\right)x^i + \\
&\quad \left(\sum_{j=0}^{n-k-1} g_j h_{n-k-1-j}\right)x^{n-k-1} + \sum_{i=n-k}^{n-1}\left(s_{i-n+k} + \sum_{j=0}^{i-n+k} g_j h_{i-n+k-j} + \sum_{j=i-n+k+1}^{n-k-1} g_j h_{i-j}\right)x^i \\
&= \sum_{i=0}^{n-k-1} s_{i+n}x^i + \sum_{i=n-k}^{n-2}\left[s_{i+n} + s_{i-n+k}\right]x^i + \sum_{i=0}^{k-2}\left[s_{i+2n-k} + s_i\right]x^i + s_{k-1}x^{k-1} + \\
&\quad \sum_{i=k}^{n-k-2}\left(\sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j}\right)x^i + \\
&\quad \left(\sum_{j=0}^{n-k-1} g_j h_{n-k-1-j}\right)x^{n-k-1} + \sum_{i=n-k}^{n-1}\left(\sum_{j=0}^{i-n+k} g_j h_{i-n+k-j} + \sum_{j=i-n+k+1}^{n-k-1} g_j h_{i-j}\right)x^i + s_{k-1}x^{n-1} \\
&= \sum_{i=0}^{k-2}\left\{s_{i+n} + \left[s_{i+2n-k} + s_i\right]\right\}x^i + \left\{s_{n+k-1} + (s_{k-1})\right\}x^{k-1} + \\
&\quad \sum_{i=k}^{n-k-2}\left\{s_{i+n} + \sum_{j=0}^{i} g_j h_{i-j} + \sum_{j=i+1}^{n-k-1} g_j h_{i+n-k-j}\right\}x^i + \\
&\quad \left\{s_{2n-k-1} + \sum_{j=0}^{n-k-1} g_j h_{n-k-1-j}\right\}x^{n-k-1} + \\
&\quad \sum_{i=n-k}^{n-2}\left\{\left[s_{i+n} + s_{i-n+k}\right] + \sum_{j=0}^{i-n+k} g_j h_{i-n+k-j} + \sum_{j=i-n+k+1}^{n-k-1} g_j h_{i-j}\right\}x^i + \\
&\quad \left\{(s_{k-1}) + \sum_{j=0}^{k-1} g_j h_{k-1-j} + \sum_{j=k}^{n-k-1} g_j h_{n-1-j}\right\}x^{n-1}. \qquad (18)
\end{aligned}
$$

It is clear that the term $s_{k-1}$ in the two round brackets are the same, and the two terms in the two square brackets are also the same. Therefore these expressions can be reused.

### D. An Example

From the expression of the quotient $q$ of $a \cdot b$ divided by $(f+1) = x^5 + x^2$, i.e., (11), we have

$$q = a_4b_4x^3 + (a_3b_4 + a_4b_3)x^2 + (a_2b_4 + a_4b_2 + a_3b_3)x + a_1b_4 + a_4b_1 + a_2b_3 + a_3b_2 + a_4b_4.$$

In order to obtain $(c+q) = \langle a \cdot b \rangle_{x^2(x^3+1)}$ using the CRT, we compute $\langle a \rangle_{x^3+1} = a_2x^2 + (a_1 + a_4)x + (a_0 + a_3)$ and $\langle b \rangle_{x^3+1} = b_2x^2 + (b_1 + b_4)x + (b_0 + b_3)$ first. Next, we compute the two intermediate values in the CRT formula, i.e., (13) and (15), using the schoolbook polynomial multiplication algorithm:

$$\langle a \cdot b \rangle_{x^k} = (a_0b_1 + a_1b_0)x + a_0b_0$$

and

$$\begin{aligned}
\langle a \cdot b \rangle_{x^{n-k}+1} &= \left\langle \langle a \rangle_{x^3+1} \cdot \langle b \rangle_{x^3+1} \right\rangle_{x^3+1} \\
&= [(a_0 + a_3)b_2 + a_2(b_0 + b_3) + (a_1 + a_4)(b_1 + b_4)]x^2 + \\
&\quad [(a_0 + a_3)(b_1 + b_4) + (a_1 + a_4)(b_0 + b_3) + a_2b_2]x + \\
&\quad (a_0 + a_3)(b_0 + b_3) + (a_1 + a_4)b_2 + a_2(b_1 + b_4).
\end{aligned}$$

Then, from the CRT expression of the remainder $(c + q) = \langle a \cdot b \rangle_{f+1}$, i.e., (12), we have

$$\begin{aligned}
c + q &= \left\langle \langle a \cdot b \rangle_{x^k} \cdot (x^{n-k} + 1) \cdot 1 + \langle a \cdot b \rangle_{x^{n-k}+1} \cdot x^k \cdot x^{n-2k} \right\rangle_{x^n+x^k} \\
&= [a_0b_1 + a_1b_0 + (a_0 + a_3)(b_1 + b_4) + (a_1 + a_4)(b_0 + b_3) + a_2b_2]x^4 + \\
&\quad [a_0b_0 + (a_0 + a_3)(b_0 + b_3) + (a_1 + a_4)b_2 + a_2(b_1 + b_4)]x^3 + \\
&\quad [(a_0 + a_3)b_2 + a_2(b_0 + b_3) + (a_1 + a_4)(b_1 + b_4)]x^2 + \\
&\quad (a_0b_1 + a_1b_0)x + a_0b_0.
\end{aligned}$$

Finally, by (18), the expression of $c = ab = q + (c + q)$ is

$$\begin{aligned}
&([a_0b_1 + a_1b_0] + (a_0 + a_3)(b_1 + b_4) + (a_1 + a_4)(b_0 + b_3) + a_2b_2)x^4 + \\
&([a_0b_1 + a_1b_0] + a_2b_4 + a_4b_2 + a_3b_3)x + \\
&([a_0b_0 + a_4b_4] + (a_0 + a_3)(b_0 + b_3) + (a_1 + a_4)b_2 + a_2(b_1 + b_4))x^3 + \\
&([a_0b_0 + a_4b_4] + a_1b_4 + a_4b_1 + a_2b_3 + a_3b_2) + \\
&(a_3b_4 + a_4b_3 + (a_0 + a_3)b_2 + a_2(b_0 + b_3) + (a_1 + a_4)(b_1 + b_4))x^2,
\end{aligned}$$

where the two reusable terms are in the square brackets. It is equal to the following one obtained

using the schoolbook method.

$$(a_0b_4 + a_4b_0 + a_1b_3 + a_3b_1 + a_2b_2 + a_3b_4 + a_4b_3)x^4 +$$
$$(a_0b_3 + a_3b_0 + a_1b_2 + a_2b_1 + a_2b_4 + a_4b_2 + a_3b_3 + a_4b_4)x^3 +$$
$$(a_0b_2 + a_2b_0 + a_1b_1 + a_1b_4 + a_4b_1 + a_2b_3 + a_3b_2 + a_3b_4 + a_4b_3 + a_4b_4)x^2 +$$
$$(a_0b_1 + a_1b_0 + a_2b_4 + a_4b_2 + a_3b_3)x +$$
$$(a_0b_0 + a_1b_4 + a_4b_1 + a_2b_3 + a_3b_2 + a_4b_4).$$

The space and time complexities of the proposed multipliers are as follows: 22 AND gates, 23 XOR gates and $1T_A + 3T_X$ gate delays. On the other hand, the complexities of the current fastest quadratic $GF(2^5)$ parallel multipliers, i.e., [9], [10] and [11] etc., are 25 AND gates, 24 XOR gates and $1T_A + 3T_X$ gate delays (see Table III in the end of the next section). We note that these multipliers adopt either the polynomial basis Montgomery representation ([10]) or the shifted polynomial basis ([9] and [11]). On the other hand, the minimal gate delay of the polynomial basis non-Montgomery multipliers is $1T_A + 4T_X$. Please refer to multipliers of [12], [13] and [14] in Table III.

## IV. Two types of the CRT-based multipliers

In this section, we present two different computational procedures to compute the reusable terms, and thus lead to two types of multipliers. The type-A multipliers can achieve the minimal number of the XOR gates. For some irreducible trinomials $u^n + u^k + 1$, the time complexity of this type of multiplier is not optimal. The other type of multipliers – Type-B multipliers – can overcome this disadvantage at the cost of some more XOR gates. Therefore, these two types of CRT-based multipliers can provide a space-time trade-off.

### *A. Complexities of the Type-A multipliers*

We need to determine the complexities of all coefficients $c_i$'s ($0 \leq i \leq n-1$) in equation (18). This equation includes only terms $s_i$, $h_i$ and $g_i$.

The expressions of $s_i$ ($0 \leq i \leq 2n-2$) are given in (5). For $0 \leq i \leq n-1$, the term $s_i = \sum_{j=0}^{i} a_j b_{i-j}$ is the summation of $i+1$ product terms $a_j b_{i-j}$. For $n \leq i \leq 2n-2$, the term $s_i = \sum_{j=i+1-n}^{n-1} a_j b_{i-j}$ is the summation of $2n-1-i$ product terms $a_j b_{i-j}$.

The expressions of $g_i$ $(0 \leq i \leq n-k-1)$ are given in (14). The expression of $h_i$ is similar to that of $g_i$ and they have the same complexity. Clearly, the complexities to compute all $g_i$ and $h_i$ for $0 \leq i \leq n-k-1$ are $2k$ XOR gates and 1 $T_X$ gate delay due to the parallelism.

TABLE II

THE NUMBER OF THE PRODUCT TERMS IN THE COEFFICIENT $c_i$ OF $x^i$.

| $x^i$ | The number of the product terms |
|---|---|
| $0 \leq i \leq k-2$ | $(2n-1-(n+i))+[\underline{(2n-1-(i+2n-k))+(i+1)}] = n+k-1-i$ |
| $i = k-1$ | $(2n-1-(n+k-1))+(\overline{k}) = n$ |
| $k \leq i \leq n-k-2$ | $(2n-1-(n+i))+(n-k) = 2n-k-1-i$ |
| $i = n-k-1$ | $(2n-1-(2n-k-1))+(n-k) = n$ |
| $n-k \leq i \leq n-2$ | $\underline{[(2n-1-(i+n))+(i-n+k+1)]}+(n-k) = n$ |
| $i = n-1$ | $(\overline{k})+(n-k) = n$ |

For the simplicity of description, we also call $g_i h_j$ in (18) a product term. Therefore, the number of the AND gates in the proposed multipliers is equal to the number of the product terms excluding the reusable terms. Table II lists the number of the product terms in each coefficient of $x^i$. The numbers of the two groups of the reusable product terms are overlined and underlined in the table. It is easy to see that $k + k(k-1) = k^2$ AND gates and $k * (k-1)$ XOR gates can be saved. Therefore, the total number of the AND gates used in the Type-A multiplier is

$$\begin{aligned} \Delta &= (k+2)n + \sum_{i=0}^{k-2}(n+k-1-i) + \sum_{i=k}^{n-k-2}(2n-k-1-i) - k^2 \\ &= (3n^2+3k^2-4kn-n+k)/2 \\ &= n^2 + \frac{(n-k)(n-1-3k)}{2}. \end{aligned} \tag{19}$$

The total number of the XOR gates is the summation of $2k$ (computing $g_i$ and $h_i$) and the number of "+" in equation (18) excluding the reusable terms:

$$2k + [(\Delta + k^2) - n] - k * (k-1) = \Delta + 3k - n. \tag{20}$$

The time complexity of a multiplier is the maximum AND and XOR gate delays of the coefficient $c_i$ for $0 \leq i \leq n-1$. Before deriving it using Table II, we prove the following lemma.

*Lemma 3:* Let $v$ and $i$ be positive integers. If $i \geq 2^v$ then $v + \left\lceil \log_2 \left\lceil \frac{i}{2^v} \right\rceil \right\rceil = \lceil \log_2 i \rceil$.

*Proof:* Suppose $i$ is a $t$-bit binary number. It is clear that $t - 1 \geq v$ for $i \geq 2^v$.

Let integers $Q$ and $R$ be the quotient and the remainder of $i$ divided by $2^v$. It is obvious that the lemma is true if $R = 0$.

Otherwise, $2^t > i > 2^{t-1}$ and we have $\lceil \log_2 i \rceil = \log_2 2^t = t$. On the other hand, because the $(t-v)$-bit quotient $Q$ is in the range $2^{t-v-1} \leq Q \leq 2^{t-v} - 1$, we have $2^{t-v-1} + 1 \leq \left\lceil \frac{i}{2^v} \right\rceil \leq 2^{t-v}$. Therefore, we have $\left\lceil \log_2 \left\lceil \frac{i}{2^v} \right\rceil \right\rceil = t - v$, and the lemma is also true for $2^v > R > 0$. ■

For $k \leq i \leq n-k-2$, the maximum gate delay is from the coefficient

$$c_k = s_{k+n} + \sum_{j=0}^{k} g_j h_{k-j} + \sum_{j=k+1}^{n-k-1} g_j h_{n-j},$$

which includes $(2n-1-(k+n)) = n-k-1$ product terms $a_i b_j$ (in $s_{k+n}$) and $n-k$ product terms $g_i h_j$. In order to compute $c_k$, the terms $g_i$ and $h_i$ should be generated first according to equation (14), and then they are ANDed in the schoolbook multiplication (15). The product terms $g_i h_j$ are just the intermediate result of this multiplication operation, and they can be generated at the cost of $(1T_A + 1T_X)$ gate delays. During this period, the $n-k-1$ product terms $a_i b_j$ in $s_{k+n}$ can be XORed pairwise and produce $\left\lceil \frac{n-k-1}{2} \right\rceil$ summations. After the above $(1T_A + 1T_X)$ gate delays, these summations and the $n-k$ product terms $g_i h_j$ are XORed using a binary XOR tree of the smallest height to obtain the coefficient $c_k$. Therefore, the total gate delay is

$$T_A + \left\lceil 1 + \log_2 \left( \left\lceil \frac{n-k-1}{2} \right\rceil + n - k \right) \right\rceil T_X = T_A + \lceil \log_2 (3n - 3k - 1) \rceil \, T_X \tag{21}$$

by lemma 3.

For $0 \leq i \leq k-2$, the maximum gate delay is from the coefficient $c_0 = s_n + [s_{2n-k} + s_0]$, which includes $(n-1) + [\underline{k}] = n + k - 1$ product terms $a_i b_j$. In order to obtain the explicit gate delay formula of $c_0$, we assume that $2^{v-1} < k \leq 2^v$ for some positive integer $v$. Because the term $[s_{2n-k} + s_0]$, which is the summation of $\underline{k}$ product terms $a_i b_j$, will be reused, these $k$ product terms can be XORed using a single binary XOR subtree of height $v$. Therefore, the total gate delay to compute the subtree of $[s_{2n-k} + s_0]$ is $T_A + vT_X$.

The $(n-1)$ product terms $a_i b_j$ in $s_n$ can then be processed as follows. Let $n - 1 = y \cdot 2^v + z$ by the division algorithm, where $0 \leq z < 2^v$. We split these $(n-1)$ product terms into $\left\lceil \frac{n-1}{2^v} \right\rceil$ groups, where the last group may have $z$ product terms if $z > 0$ and the other groups have

$2^v$ product terms each. During the period to compute $[s_{2n-k}+s_0]$, i.e., $T_A+vT_X$, the product terms in these $\lceil \frac{n-1}{2^v} \rceil$ groups can be XORed in parallel using $\lceil \frac{n-1}{2^v} \rceil$ binary XOR subtrees of the same height $v$. Finally, these $\lceil \frac{n-1}{2^v} \rceil$ summations and the result of $[s_{2n-k}+s_0]$ are XORed using a binary XOR tree at the cost of $\lceil \log_2(\lceil \frac{n-1}{2^v} \rceil + 1) \rceil$ XOR gate delays. Therefore, the total XOR gate delay to compute the coefficient $c_0$ is

$$v + \left\lceil \log_2(\left\lceil \frac{n-1}{2^v} \right\rceil + 1) \right\rceil = \lceil \log_2(n-1+2^v) \rceil \leq \lceil \log_2(n+2k-1) \rceil \tag{22}$$

by lemma 3 and $2^v < 2k \leq 2^{v+1}$.

We now count the gate delay of the coefficient $c_i$ for $n-k \leq i \leq n-2$, i.e.,

$$c_i = \sum_{i=n-k}^{n-2} [s_{i+n}+s_{i-n+k}] + \sum_{j=0}^{i-n+k} g_j h_{i-n+k-j} + \sum_{j=i-n+k+1}^{n-k-1} g_j h_{i-j}.$$

Each coefficient $c_i$ includes a reusable term $[s_{i+n}+s_{i-n+k}]$ and $n-k$ product terms $g_i h_j$. This reusable term, which is also in $c_i$ for $0 \leq i \leq k-2$, includes $(2n-1-(i+n))+(i-n+k+1)=k$ product terms $a_i b_j$. Therefore, these coefficients $c_i$'s have the same distribution pattern of the product terms $a_i b_j$ and $g_i h_j$, and their gate delays are equal. These coefficients $c_i$'s can be computed similar to the way we computed $c_0$. The only difference is that all product terms in $c_0$ are of the form $a_i b_j$, but the terms $g_i$ and $h_i$ here should be generated first using $1T_X$ delay. Therefore, the size of the groups should be halved, and the total XOR gate delay to compute these coefficients is

$$v + \left\lceil \log_2(\left\lceil \frac{n-k}{2^{v-1}} \right\rceil + 1) \right\rceil = \lceil \log_2(2n-2k+2^v) \rceil \leq \lceil \log_2(2n) \rceil \tag{23}$$

by lemma 3 and $2^v < 2k \leq 2^{v+1}$.

By (18), the gate delays of $c_i$ for the other two cases, i.e., $i=k-1$ and $i=n-k-1$, are not greater than those for $c_0$ and $c_k$.

Therefore, the XOR gate delay of the Type-A multiplier is the largest one among (21), (22) and (23). Because $2n-2k+2^v > n-1+2^v$ iff $2k < n+1$, the time complexity of the Type-A multiplier is equal to $T_A + \lceil \log_2(max(3n-3k-1, 2n-2k+2^v)) \rceil T_X$.

### *B. Complexities of the Type-B multipliers*

The method to compute the coefficient $c_0$ presented before (22) is not optimal for some values of $n$ and $k$. For example, $u^{68}+u^{33}+1$ is irreducible on $GF(2)$. The coefficient $c_0 = s_n + [s_{2n-k}+$

$s_0]$ includes $67+[\underline{33}] = 100$ product terms, The term $[s_{2n-k}+s_0]$ is the summation of 33 product terms and it will be reused in $c_{35}$. Following the above method, we should compute the summation of these 33 product terms using a single binary XOR tree of height 6, for $2^5 < 33 \leq 2^6$. The corresponding gate delay is $1T_A + 6T_X$. Then we should split the other 67 product terms of $s_n$ into 2 groups: one includes $2^6 = 64$ product terms and the other $67 - 64 = 3$. These two groups can also be computed at the cost of $1T_A + 6T_X$ gate delays. Finally, the two summations of these two groups and the value of $[s_{2n-k} + s_0]$, i.e., 3 values, are XORed at the cost of $2T_X$. Therefore, the time complexity of this method is $1T_A + 8T_X$. However, from Table III, the time and space complexities of the fastest quadratic bit-parallel $GF(2^{68})$ multipliers are: $68^2 = 4624$ AND gates, $68^2 - 1 = 4623$ XOR gates and $1T_A + \lceil \log_2(2 * 68 - 33) \rceil T_X = 1T_A + 7T_X$ gate delays.

The Type-B multiplier is designed to overcome this disadvantage at the cost of some more XOR gates. Suppose that there are $l$ product terms $a_i b_j$ and $m$ product terms $g_i h_j$ in the coefficient $c_t$, we compute $c_t$ using a binary XOR tree of the smallest height with at least $(l + 2m)$ leaf nodes. The product terms $a_i b_j$ are placed at the first level – the leaf nodes, and $g_i h_j$ the second level. Finally, the coefficient $c_t$ is computed similar to the way we computed $c_k$ in the Type-A multiplier before (21).

Return to the above example, the Type-B multiplier computes $c_0$ using a binary XOR tree of height 7 ($100 < 2^7$): the 100 product terms of $s_{2n-k}$, $s_0$ and $s_{2n-k}$ are placed at the leaf nodes starting from the leftmost position successively. Therefore, the 32 product terms in $s_{2n-k}$ form a subtree of height 5, and the only product term $a_0 b_0$ in $s_0$ a subtree of height 0. The results of these two subtrees, i.e., the values of $s_{2n-k}$ and $s_0$, can then be reused in the coefficient $c_{35}$.

The coefficient $c_{35}$ is the summation of the reusable term $[s_{2n-k}+s_0]$ and 35 product terms $g_i h_j$. It can be computed using a binary XOR tree of the smallest height with at least $32+1+2*35 = 103$ leaf nodes so that the above two height-5 and height-0 subtrees can be embedded into it. Because $2^6 < 103 < 2^7$, the height of this tree is 7. The structure of this height-7 tree is similar to that of $c_k$ in the Type-A multiplier before (21), i.e., the 33 product terms of $s_{2n-k}$ and $s_0$ are placed at the leaf nodes starting from the leftmost position successively, and then the 35 product terms $g_i h_j$ at the second level. Clearly, the two values of the height-5 and height-0 subtrees constructed from the product terms in $s_{2n-k}$ and $s_0$ have been obtained in the binary XOR tree of $c_0$. Compared to the Type-A multiplier, it means that the Type-B multiplier first deletes the

whole height-6 subtree of the reusable term $[s_{2n-k}+s_0]$, and then XORs the height-5 and height-0 subtrees of $s_{2n-k}$ and $s_0$ into the height-7 binary XOR tree of $c_{35}$ using 2 XOR gates. From (18) and Table II, we know that there are $k$ reusable summations. Therefore, by (19) and (20), the AND and XOR complexities of the Type-B multiplier are $\Delta = (3n^2 + 3k^2 - 4kn - n + k)/2 = 4064$ AND gates and $\Delta + 3k - n + (2-1) \cdot k = 4128$ XOR gates respectively. Moreover, from Table II, the time complexity of the multiplier is the gate delay to compute $c_k = c_{33}$, and it is $1T_A + \lceil \log_2(3 * 68 - 3 * 33 - 1) \rceil T_X = 1T_A + 7T_X$. Clearly, this scheme has the same time complexity as the fastest parallel multiplier, but the AND and XOR gate complexities are reduced by about $12.1\%$ and $10.7\%$ respectively.

We note that it is a coincidence that $s_{2n-k}$ of the reusable term $[s_{2n-k} + s_0]$ in this example includes 32 product terms, and they form the height-5 subtree. Generally, for a reusable term $[s_i + s_j]$ in the Type-B multiplier, the leaf nodes of its subtrees may come from either $s_i$, $s_j$, or both $s_i$ and $s_j$.

The other examples of both types of multipliers in $GF(2^6)$ are given in the end of the next section.

In order to generalize this idea, we define $w(k)$ as the Hamming weight of the integer $k$. The $k$ reusable summations are in $c_i$ and $c_{i+n-k}$ for $0 \leq i \leq k-1$. They each include $k$ product terms $a_i b_j$, and we arrange them in the way discussed above. Thus we have $w(k)$ subtrees: the $j$-th nonzero bit in the binary expansion of $k$ corresponds to the subtree of height $j$. After constructing the binary XOR tree of $c_i$, we obtain the summation of leaf nodes in each subtree. These $w(k)$ subtree summations can then be XORed into the binary XOR tree of $c_{i+n-k}$ at the cost of $w(k)$ XOR gates.

Therefore, the number of the AND gates used in this type of multiplier is equal to that of the Type-A multiplier, but the number of the XOR gates increases by $[w(k)-1] \cdot k$. By (20), this number is $\Delta + 3k - n + k \cdot [w(k)-1] = \Delta + 2k - n + k \cdot w(k)$.

The time complexity of the Type-B multiplier can be obtained similar to the way we processed $c_k$ in the Type-A multiplier before (21). For $k \leq i \leq n-k-2$, the maximum XOR gate delay is also from the coefficient $c_k$, and it was given in (21) as $T_A + \lceil \log_2(3n - 3k - 1) \rceil T_X$. For $0 \leq i \leq k-2$, the maximum gate delay is from the coefficient $c_0$, which is $T_A + \lceil \log_2(n + k - 1) \rceil T_X$.

The gate delays of the other coefficients are not greater than those of $c_k$ and $c_0$. Because

$$3n - 3k - 1 > n + k - 1 \quad \text{for} \quad k < n/2,$$

the time complexity of the Type-B multiplier is equal to $T_A + \lceil \log_2(3n - 3k - 1) \rceil T_X$.

### C. The case $n = 2k + 1$

For the case $x^n + x^k + 1$ where $n = 2k + 1$, the expression of $c = ab$ is slightly different from equation (18), i.e., the summation $\sum_{i=k}^{n-k-2} \{\cdots\}$ in (18) should be deleted. But the space and time complexities of the two types of multipliers coincide with the values in Table III.

### D. Comparisons

We list the complexities of the proposed multipliers and other trinomial-based quadratic parallel multipliers in Table III, where WDB, PB and SPB denote weekly dual bases, polynomial bases and shifted polynomial bases respectively. Because their AND gate delays are all $1T_A$, we ignore them and list only the XOR gate delays in the last column.

TABLE III

COMPLEXITIES OF $f(u) = u^n + u^k + 1$-BASED PARALLEL MULTIPLIERS FOR $2 \leq k < n/2$

| Multipliers | # AND Gate | # XOR Gate | XOR Gate Delay |
|---|---|---|---|
| PB reduction matrix [12] 2004 | $n^2$ | $n^2 - 1$ | $\lceil \log_2(4n - 4) \rceil$ |
| WDB [15] 1998 | $n^2$ | $n^2 - 1$ | $\lceil \log_2(2n + 2k - 2) \rceil$ |
| PB Mastrovito [13] 2007 | $n^2$ | $n^2 - 1$ | $\lceil \log_2(2n + 2k - 3) \rceil$ |
| PB Mastrovito [14] 2003 | $n^2$ | $n^2 + (k^2 - 3k)/2$ | $\lceil \log_2(2n + k - 2) \rceil$ |
| SPB Mastrovito [16] 2005 | $n^2$ | $n^2 - 1$ | $\lceil \log_2 2n \rceil$ |
| SPB matrix [9] 2007 | $n^2$ | $n^2 + [(n-k)^2 + k^2 - 3n]/2$ | $\lceil \log_2(2n - k) \rceil$ |
| PB Montgomery [10] 2009 | $n^2$ | $n^2 - 1$ | $\lceil \log_2(2n - k) \rceil$ |
| SPB XOR tree [11] 2006 | $n^2$ | $n^2 - 1$ | $\lceil \log_2(2n - k) \rceil$ |
| Karatsuba hybrid $n$ even [5] 1999 | $\frac{3n^2}{4}$ | $\frac{3n^2}{4} + 2.5n + k - 4$ | $\lceil \log_2(8n - 8) \rceil$ |
| Karatsuba hybrid $n$ odd [5] 1999 | $\frac{3n^2+2n-1}{4}$ | $\frac{3n^2}{4} + 4n + k - 5.75$ | $\lceil \log_2(8n - 8) \rceil$ |
| PB Type-A | $\Delta$ | $\Delta + 3k - n$ | $\lceil \log_2(max(3n - 3k - 1, 2n - 2k + 2^v)) \rceil$ |
| PB Type-B | $\Delta$ | $\Delta + 2k - n + k \cdot w(k)$ | $\lceil \log_2(3n - 3k - 1) \rceil$ |
| where $\Delta = n^2 + \frac{(n-k)(n-1-3k)}{2} \leq n^2$ when $\frac{n-1}{3} \leq k < \frac{n}{2}$, $2^{v-1} < k \leq 2^v$. | | | |

The space complexity of the fastest quadratic parallel multipliers for irreducible trinomials are $n^2$ AND gates and $n^2-1$ XOR gates. For the purpose of comparison, we derive the condition that the proposed Type-A multipliers outperforms these results. The inequalities are as follows:

$$\begin{cases} \text{\# AND gates:} & \Delta = n^2 + \frac{(n-k)(n-1-3k)}{2} \leq n^2, \\ \text{\# XOR gates:} & \Delta + 3k - n \leq n^2 - 1, \end{cases}$$

and their solutions are

$$\begin{cases} \text{\# AND gates:} & (n-1)/3 \leq k < n/2, \\ \text{\# XOR gates:} & (n-1)/3 \leq k < n/2. \end{cases}$$

As for the time complexity, thanks to the property of the ceiling function "$\lceil \cdot \rceil$", these multipliers may have the $same$ XOR gate delay, depending on the values of $n$ and $k$. But the distribution of the irreducible trinomials over $GF(2)$ is not regular. Thus we can not give an explicit comparison result. Moreover, in Table III, the expression of the XOR gate delay of the Type-A multiplier involves the value of $v$, and the expression of the XOR gate number of the Type-B multiplier the Hamming weight $w(k)$. Therefore, it is hard to carry out a simple theoretical evaluation of these multipliers. So we resort to experiments.

In order to compare the proposed two types of multipliers with the current fastest parallel multipliers – the quadratic multipliers, we compute their space and time complexities for the 539 values of $n$ such that $4 < n < 1000$ and $x^n + x^k + 1$ is irreducible over $GF(2)$ for some $k \in [2, n/2]$. Our experimental results show that the proposed multipliers beat the current fastest parallel multipliers for 290 values of $n$: they have the same time complexity, but the space complexities are reduced by $8.4\%$ on average.

## V. Type-A and Type-B multipliers for $n = 2k > 2$

The trinomial $u^{2k}+u^k+1$ is irreducible over $GF(2)$ if and only if $k = 3^i$ for some nonnegative integer $i$ [17]. In this case, we have $(f+1) = x^{2k}+x^k = x^k(x^k+1)$, and thus the corresponding Bezout identity is

$$x^k \cdot 1 + (x^k + 1) \cdot 1 = 1.$$

Similarly, we can obtain the following expression of $c = ab$:

$$c = \sum_{i=0}^{k-2} \left(s_{i+n} + [s_{i+3k} + s_i]\right) x^i + \left((s_{k-1}) + \sum_{j=0}^{k-1} g_j h_{k-1-j}\right) x^{n-1} + \sum_{i=k}^{n-2} \left([s_{i+n} + s_{i-k}] + \sum_{j=0}^{i-k} g_j h_{i-k-j} + \sum_{j=i-k+1}^{k-1} g_j h_{i-j}\right) x^i + \left(s_{3k-1} + (s_{k-1})\right) x^{k-1}. \quad (24)$$

### A. Type-A and Type-B multipliers

The two types of multipliers are similar to those presented in the previous section.

Table IV lists the number of the product terms in each coefficient $c_i$ of $x^i$. The numbers of the AND gates in the Type-A and Type-B multipliers are all equal to the number of the product terms excluding the reusable terms. Similarly, $k^2$ AND gates and $k * (k-1)$ XOR gates can be saved. Therefore, the total number of the AND gates is

$$(n-k+1)n + \sum_{i=0}^{k-2}(3k-1-i) - k^2 = (7n^2 - 2n)/8,$$

which coincides with the value of $\Delta$ in (19).

The total number of the XOR gates in the Type-A multiplier is the summation of $2k$ (computing $g_i$ and $h_i$) and the number of "+" in equation (24) excluding the reusable terms:

$$2k + [((7n^2 - 2n)/8 + k^2) - n] - k * (k-1) = (7n^2 + 2n)/8.$$

TABLE IV

THE NUMBER OF THE PRODUCT TERMS IN THE COEFFICIENT $c_i$ OF $x^i$.

| $x^i$ | The number of the product terms |
|---|---|
| $0 \le i \le k-2$ | $(2n-1-(n+i)) + [\underline{(2n-1-(i+3k)) + (i+1)}] = 3k-1-i$ |
| $i = k-1$ | $(2n-1-(3k-1)) + (\overline{k}) = n$ |
| $k \le i \le n-2$ | $[\underline{(2n-1-(n+i)) + (i-k+1)}] + k = n$ |
| $i = n-1$ | $(\overline{k}) + k = n$ |

For $0 \le i \le k-1$, the maximum gate delay is from the coefficient $c_0 = s_n + [s_{3k} + s_0]$, which includes $3k-1$ product terms $a_i b_j$. Therefore, the gate delay of $c_0$ in the Type-B multiplier is $T_A + \lceil \log_2(3k-1) \rceil T_X$.

As for the Type-A multiplier, the reusable term $[s_{3k} + s_0]$, which includes $k$ product terms $a_i b_j$, should be computed using a subtree at the cost of $\lceil \log_2 k \rceil T_X$ gate delays. The rest of terms in

$c_0$, i.e., $s_n$, includes $2k-1$ product terms $a_ib_j$, and they should be divided into $\lceil (2k-1)/k \rceil = 2$ groups: one includes $k$ product terms and the other $k-1$. Finally, the two subtrees constructed from these two groups and the reusable term $[s_{3k}+s_0]$ are XORed at the cost of $2T_X$ gate delays. Therefore, in the Type-A multiplier, the total gate delay to compute $c_0$ is $T_A + \lceil 2 + \log_2 k \rceil T_X = T_A + \lceil \log_2 4k \rceil T_X$.

For $k \le i \le n-1$, the maximum gate delay is from the coefficient

$$c_k = [s_{3k} + s_0] + g_0h_0 + \sum_{j=1}^{k-1} g_jh_{k-j},$$

which includes $(2n-1-3k)+1 = k$ product terms $a_ib_j$ (in $[s_{3k}+s_0]$) and $k$ product terms $g_ih_j$. Therefore, the gate delay of $c_k$ in the Type-B multiplier is $T_A + \left\lceil 1 + \log_2(\lceil \frac{k}{2} \rceil + k) \right\rceil T_X = T_A + \lceil \log_2(3k) \rceil T_X$.

As for the Type-A multiplier, the gate delay of $c_k$ can be counted similar to the way we derived (23), and it is

$$v + \left\lceil \log_2\left( \left\lceil \frac{k}{2^{v-1}} \right\rceil + 1 \right) \right\rceil = \lceil \log_2(2k + 2^v) \rceil \le \lceil \log_2(4k) \rceil$$

for $2^v < 2k \le 2^{v+1}$.

Similarly, $k \cdot (w(k) - 1)$ more XOR gates are required in the Type-B multiplier.

Table V compares space and time complexities of different $GF(2^n)$ parallel multipliers for irreducible trinomial $u^n + u^k + 1$ ($n = 2k > 2$). Because $w(k) \le \lceil \log_2 k \rceil$, we have $k \cdot w(k) \le n \lceil \log_2 n - 1 \rceil / 2$. Therefore, the space complexity of the Type-B multiplier is only about $7/8$ that of the current fastest multipliers, but they have the same time complexity.

### *B. An example*

By (24), The expression of $c = ab$ for the irreducible trinomial $u^6 + u^3 + 1$ is

$$\begin{aligned}
&([a_1b_1 + a_0b_2 + a_2b_0] + (a_0 + a_3)(b_2 + b_5) + (a_1 + a_4)(b_1 + b_4) + (a_2 + a_5)(b_0 + b_3))x^5 + \\
&([a_1b_1 + a_0b_2 + a_2b_0] + a_4b_4 + a_3b_5 + a_5b_3)x^2 + \\
&([a_0b_1 + a_1b_0 + a_5b_5] + (a_0 + a_3)(b_1 + b_4) + (a_1 + a_4)(b_0 + b_3) + (a_2 + a_5)(b_2 + b_5))x^4 + \\
&([a_0b_1 + a_1b_0 + a_5b_5] + a_2b_5 + a_5b_2 + a_3b_4 + a_4b_3)x + \\
&([a_4b_5 + a_5b_4 + a_0b_0] + (a_1 + a_4)(b_2 + b_5) + (a_2 + a_5)(b_1 + b_4) + (a_0 + a_3)(b_0 + b_3))x^3 + \\
&[a_4b_5 + a_5b_4 + a_0b_0] + a_1b_5 + a_5b_1 + a_2b_4 + a_4b_2 + a_3b_3.
\end{aligned}$$

TABLE V

COMPLEXITIES OF $f(u) = u^n + u^k + 1$-BASED PARALLEL MULTIPLIERS FOR $2k = n > 2$

| Multipliers | # AND Gate | # XOR Gate | XOR Gate Delay |
|---|---|---|---|
| PB Mastrovito [17] 1988 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(8k) \rceil$ |
| PB Mastrovito [18] 1999 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(4k) \rceil$ |
| PB mod reduction [19] 2002 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(4k - 2) \rceil$ |
| PB reduction matrix [12] 2004 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(4k) \rceil$ |
| WDB [15] 1998 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(3k) \rceil$ |
| SPB XOR tree [11] 2006 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(3k) \rceil$ |
| PB Mastrovito [13] 2007 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(3k) \rceil$ |
| PB Montgomery [10] 2009 | $n^2$ | $(2n^2 - n)/2$ | $\lceil \log_2(3k) \rceil$ |
| Karatsuba hybrid [5] 1999 | $3n^2/4$ | $3n^2/4 + 3n - 4$ | $\lceil \log_2(16k - 8) \rceil$ |
| PB Type-A | $(7n^2 - 2n)/8$ | $(7n^2 + 2n)/8$ | $\lceil \log_2(4k) \rceil$ |
| PB Type-B | $(7n^2 - 2n)/8$ | $(7n^2 - 2n)/8 + k \cdot w(k)$ | $\lceil \log_2(3k) \rceil$ |

It is equal to the following one obtained using the schoolbook method.

$$(a_0b_5 + a_5b_0 + a_1b_4 + a_4b_1 + a_2b_3 + a_3b_2 + [a_3b_5 + a_5b_3 + a_4b_4])x^5 +$$

$$(a_0b_4 + a_4b_0 + a_1b_3 + a_3b_1 + a_2b_2 + [a_2b_5 + a_5b_2 + a_3b_4 + a_4b_3])x^4 +$$

$$(a_0b_3 + a_3b_0 + a_1b_2 + a_2b_1 + [a_1b_5 + a_5b_1 + a_2b_4 + a_4b_2 + a_3b_3])x^3 +$$

$$(a_0b_2 + a_2b_0 + a_1b_1 + a_3b_5 + a_5b_3 + a_4b_4)x^2 +$$

$$(a_0b_1 + a_1b_0 + a_2b_5 + a_5b_2 + a_3b_4 + a_4b_3 + a_5b_5)x +$$

$$a_0b_0 + a_1b_5 + a_5b_1 + a_2b_4 + a_4b_2 + a_3b_3 + a_4b_5 + a_5b_4.$$

The Type-A multiplier computes $c_0$ and $c_3$ using the following formulae:

$reusable = [a_4b_5 + a_5b_4 + a_0b_0]$;

$c_0 = [reusable] + [a_1b_5 + a_5b_1 + a_2b_4 + a_4b_2] + [a_3b_3]$;

$c_3 = [reusable] + [(a_1 + a_4)(b_2 + b_5) + (a_2 + a_5)(b_1 + b_4)] + [(a_0 + a_3)(b_0 + b_3)]$.

The Type-B multiplier computes $c_0$ and $c_3$ using the following formulae:

$subtree_1 = a_5b_4 + a_4b_5$;

$subtree_2 = a_0b_0$;

$c_0 = [subtree_1] + [subtree_2] + [a_1b_5 + a_5b_1] + [a_2b_4 + a_4b_2] + [a_3b_3]$;

$c_3 = [subtree_1] + [subtree_2] + [(a_1 + a_4)(b_2 + b_5)] + [(a_2 + a_5)(b_1 + b_4)] + [(a_0 + a_3)(b_0 + b_3)]$.

The complexities of the Type-B multiplier are as follows: $30$ AND gates, $36$ XOR gates and $1T_A + 4T_X$ gate delays. It is less efficient than the Type-A multiplier, whose complexities are as follows: $30$ AND gates, $33$ XOR gates and $1T_A + 4T_X$ gate delays. On the other hand, the complexities of the current fastest quadratic $GF(2^6)$ parallel multipliers are $36$ AND gates, $33$ XOR gates and $1T_A + 4T_X$ gate delays, and it is also less efficient than the Type-A multiplier. We note that $6$ is a solution to $(7n^2 + 2n)/8 = (2n^2 - n)/2$. Therefore, the Type-A multiplier and the current fastest quadratic $GF(2^6)$ parallel multiplier have the same XOR gate complexity for only this special case.

## VI. Conclusions

We have presented an alternative representation of the finite field multiplication operation, and designed two types of bit-parallel $GF(2^n)$ multipliers for irreducible trinomials. For the 539 values of $n$ such that $4 < n < 1000$ and $u^n + u^k + 1$ is irreducible over $GF(2)$ for some $k \in [2, n/2]$, our experimental results show that the proposed multipliers beat the current fastest parallel multipliers for 290 values of $n$ when $k \in [(n-1)/3, n/2]$: they have the same time complexity, but the space complexities are reduced by $8.4\%$ on average. Especially, for the special irreducible trinomial $u^{2k} + u^k + 1$, the proposed multipliers set a new record for the space complexity.

Finally, we note that any polynomial $e$ of degree less than 2 can be used in (8), i.e.,

$$a \cdot b = f \cdot q + c = (f + e)q + (c + e \cdot q).$$

## References


[1] H. Fan and M. A. Hasan, "A survey of some recent bit-parallel $GF(2^n)$ multipliers," *Finite Fields and Their Applications*, vol. 32, pp. 5–43, March 2015.

[2] C. Grabbe, M. Bednara, J. Shokrollahi, J. Teich, and J. von zur Gathen, "FPGA designs of parallel high performance $GF(2^{233})$ multipliers," in *Proc. Int. Symposium on Circuits and Systems (ISCAS 2003), vol. II*, 2003, pp. 268–271.

[3] F. Rodríguez-Henríquez and Ç. K. Koç, "On fully parallel Karatsuba multipliers for $GF(2^m)$," in *Proc. Int. Conf. Computer Science and Technology (CST 2003)*. ACTA Press, 2003, pp. 405–410.

[4] J. von zur Gathen and J. Shokrollahi, "Efficient FPGA-based Karatsuba multipliers for polynomials over $F_2$," in *Proc. 12th Workshop on Selected Areas in Cryptography (SAC 2005)*. Springer, 2006, pp. 359–369.

[5] M. Elia, M. Leone, and C. Visentin, "Low complexity bit-parallel multipliers for $GF(2^m)$ with generator polynomial $x^m + x^k + 1$," *IEE Electronics Letters*, vol. 35, no. 7, pp. 551–552, 1999.

[6] M. A. Hasan, N. Méloni, A. H. Namin, and C. Negre, "Block recombination approach for subquadratic space complexity binary field multiplication based on Toeplitz matrix-vector product," *IEEE Transactions on Computers*, vol. 61, no. 2, pp. 151–163, 2012.

[7] D. J. Bernstein, "Multidigit multiplication for mathematicians," Tech. Rep., 2001.

[8] F. Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and Ç. K. Koç, *Cryptographic algorithms on reconfigurable hardware*. Springer, 2006.

[9] C. Negre, "Efficient parallel multiplier in shifted polynomial basis," *Journal of Systems Architecture*, vol. 53, pp. 109–116, 2007.

[10] A. Hariri and A. Reyhani-Masoleh, "Bit-serial and bit-parallel Montgomery multiplication and squaring over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 58, no. 10, pp. 1332–1345, 2009.

[11] H. Fan and M. A. Hasan, "Fast bit parallel-shifted polynomial basis multipliers in $GF(2^n)$," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 12, pp. 2606–2615, 2006.

[12] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 945–959, Aug. 2004.

[13] N. Petra, D. D. Caro, and A. G. Strollo, "A novel architecture for Galois fields $GF(2^m)$ multipliers based on Mastrovito scheme," *IEEE Transactions on Computers*, vol. 56, no. 11, pp. 1470–1483, 2007.

[14] S. O. Lee, S. W. Jung, C. H. Kim, J. Yoon, J. Koh, and D. Kim, "Design of bit parallel multiplier with lower time complexity," in *Proc. ICICS' 2003, Lect. Notes Comput. Sci., vol. 2971*, 2004, pp. 127–139.

[15] H. Wu, M. A. Hasan, and I. F. Blake, "New low-complexity bit-parallel finite field multipliers using weakly dual bases," *IEEE Transactions on Computers*, vol. 47, no. 11, pp. 1223–1234, Nov. 1998.

[16] H. Fan and Y. Dai, "Fast bit-parallel $GF(2^n)$ multiplier for all trinomials," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 485–490, 2005.

[17] E. Mastrovito, "VLSI designs for multiplication over finite fields $GF(2^m)$," *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pp. 297–309, 1988.

[18] B. Sunar and Ç. K. Koç, "Mastrovito multiplier for all trinomials," *IEEE Transactions on Computers*, pp. 522–527, 1999.

[19] H. Wu, "Bit-parallel finite field multiplier and squarer using polynomial basis," *IEEE Transactions on Computers*, vol. 51, no. 7, pp. 750–758, 2002.