Université de Montréal

On Space-Time Trade-Off for Montgomery Multipliers over Finite Fields

par
Yiyang Chen

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Avril 2015

RÉSUMÉ

La multiplication dans les corps de Galois à 2^m éléments (i.e. $GF(2^m)$) est une opérations très importante pour les applications de la théorie des codes correcteurs et de la cryptographie. Dans ce mémoire, nous nous intéressons aux réalisations *parallèles* de multiplicateurs dans $GF(2^m)$ lorsque ce dernier est généré par des trinômes irréductibles. Notre point de départ est le *multiplicateur de Montgomery* qui calcule $A(x) \cdot B(x) \cdot x^{-u}$ efficacement, étant donné $A(x), B(x) \in GF(2^m)$ pour u choisi judicieusement. Nous étudions ensuite l'algorithme *diviser pour régner* PCHS qui permet de partitionner les multiplicandes d'un produit dans $GF(2^m)$ lorsque m est impair. Nous l'appliquons pour la partitionnement de $A(x)$ et de $B(x)$ dans *la multiplication de Montgomery* $A(x) \cdot B(x) \cdot x^{-u}$ pour $GF(2^m)$ même si m est pair. Basé sur cette nouvelle approche, nous construisons un multiplicateur dans $GF(2^m)$ généré par des trinômes irréductibles. Une nouvelle astuce de réutilisation des résultats intermédiaires nous permet d'éliminer plusieurs portes XOR redondantes. Les complexités de temps (i.e. le délai) et d'espace (i.e. le nombre de portes logiques) du nouveau multiplicateur sont ensuite analysées :

1. Le nouveau multiplicateur demande environ 25% moins de portes logiques que les multiplicateurs de Montgomery et de Mastrovito lorsque $GF(2^m)$ est généré par des trinômes irréductibles et m est suffisamment grand. Le nombre de portes du nouveau multiplicateur est presque identique à celui du multiplicateur de Karatsuba proposé par Elia.
2. Le délai de calcul du nouveau multiplicateur excède celui des meilleurs multiplicateurs d'au plus deux évaluations de portes XOR.
3. Nous determinons le délai et le nombre de portes logiques du nouveau multiplicateur sur les deux corps de Galois recommandés par le National Institute of Standards and Technology (NIST). Nous montrons que notre multiplicateurs contient 15% moins de portes logiques que les multiplicateurs de Montgomery et de Mastrovito au coût d'un délai d'au plus une porte XOR supplémentaire. De plus, notre multiplicateur a un délai d'une porte XOR moindre que celui du multiplicateur d'Elia au coût d'une augmentation de moins de 1% du nombre total de portes logiques.

Mots-clés : Corps de Galois, calcul parallèle, multiplication Montgomery, multiplication Karatsuba, trinôme irréductible.

ABSTRACT

The multiplication in a Galois field with 2^m elements (i.e. $GF(2^m)$) is an important arithmetic operation in coding theory and cryptography. In this thesis, we focus on the bit-parallel multipliers over the Galois fields generated by trinomials. We start by introducing the $GF(2^m)$ *Montgomery multiplication*, which calculates $A(x) \cdot B(x) \cdot x^{-u} \in GF(2^m)$ with two polynomials $A(x),\ B(x) \in GF(2^m)$ and a properly chosen u. Then, we investigate the rule for multiplicand partition used by a *divide-and-conquer* algorithm PCHS originally proposed for the multiplication over $GF(2^m)$ with odd m. By adopting similar rules for splitting $A(x)$ and $B(x)$ in $A(x) \cdot B(x) \cdot x^{-u}$, we develop new Montgomery multiplication formulae for $GF(2^m)$ with m either odd or even. Based on this new approach, we develop the corresponding *bit-parallel Montgomery multipliers* for the Galois fields generated by trinomials. A new bit-reusing trick is applied to eliminate redundant XOR gates from the new multiplier. The time complexity (i.e. the delay) and the space complexity (i.e. the logic gate number) of the new multiplier are explicitly analysed:

1. This new multiplier is about 25% more efficient in the number of logic gates than the previous trinomial-based Montgomery multipliers or trinomial-based Mastrovito multipliers on $GF(2^m)$ with m big enough. It has a number of logic gates very close to that of the Karatsuba multiplier proposed by Elia.
2. While having a significantly smaller number of logic gates, this new multiplier is at most two T_X larger in the total delay than the fastest bit-parallel multiplier on $GF(2^m)$, where T_X is the XOR gate delay.
3. We determine the space and time complexities of our multiplier on the two fields recommended by the National Institute of Standards and Technology (NIST). Having at most one more T_X in the total delay, our multiplier has a more-than-15% reduced logic gate number compared with the other Montgomery or Mastrovito multipliers. Moreover, our multiplier is one T_X smaller in delay than the Elia's multiplier at the cost of a less-than-1% increase in the logic gate number.

Keywords: Galois field, parallel computation, Montgomery multiplication, Karatsuba multiplication, irreducible trinomial.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

VLSI	very large scale integration
XOR	exclusive OR
PB	polynomial basis
SPB	shifted polynomial basis
NB	normal basis
AOP	all-one polynomial
ESP	equally-spaced polynomial
ECC	elliptic curve cryptography
DB	dual basis
WDB	weak dual basis
DC	divide-and-conquer
PCHS	a divide-and-conquer algorithm by S. Park, K, Chang, D. Hong, and C. Seo[21]
NIST	National Institute of Standards and Technology
ECDSA	elliptic curve digital signature algorithm

NOTATION

$(n)_2$	the binary form of n
$\lfloor x \rfloor$	the floor function
$\lceil x \rceil$	the ceiling function
$\mathbb{Z}$	the integer ring
$\mathbb{Z}^+$	$\{n \in \mathbb{Z} \mid n > 0\}$
$\mathbb{F}$	a field
$GF(2)$ or $\mathbb{F}_2$	0-1 field
$GF(2)[x]$ or $\mathbb{F}_2[x]$	the polynomial ring over the 0-1 field
$GF(2^m)$ or $\mathbb{F}_2^m$	A Galois field having 2^m elements
$S^{\otimes}$	the number of AND gates in a sub-circuit
$S^{\oplus}$	the number of XOR gates in a sub-circuit
T_A	the AND gate delay
T_X	the XOR gate delay
$T^{\otimes}$	the delay caused by the AND operations in a sub-circuit
$T^{\oplus}$	the delay caused by the XOR operations in a sub-circuit
Delay	the total delay of a sub-circuit
O	the big O notation
$W(\cdot)$	the Hamming weight
#	the cardinality of a set
$\binom{n}{k}$	the binomial coefficient
$\bigcap$	intersection
$\bigcup$	union
$\emptyset$	the empty set

ACKNOWLEDGMENTS

This thesis is based upon my research conducted over the past three years during my study for my Master's Degree in Department of Informatics and Operational Research at Université de Montréal.

First of all, I would like to express my thanks to Professor Louis Savail for his support to my self-motivated research. His lectures on cryptography were very helpful to me in undertaking this research. I am grateful for his time and suggestions in reviewing my thesis. These suggestions have contributed a lot for improving my thesis.

I would like to thank Dr. Yin Li, my former colleague at Shanghai Jiaotong University (SJTU), for his suggestions and time in reviewing this work across the past two years.

I would like to thank Professor Gilles Brassard for his classes in Quantum Informatics. His teachings are precise and humorous, his assignments are entertaining. I am still interested by the bit communication problems that have puzzled me in the final exam. It is my hope that I can use my spare time in the future to find the solutions by myself.

I am also very grateful to several faculty and staff members for their daily help and support to my study in the department. I express my gratitude to Professor Jean-Yves Potvin and Professor Sébastien Roy for their time and effort for extending my study permit in Quebec and Canada. I want to express my heartfelt thanks to Professor Jianyun Nie for his teachings and guidance. Finally, I want to extend my sincere thanks to Ms. Mariette Paradis and Ms. Céline Bégin for their excellent service in student administration.

CHAPTER 1

INTRODUCTION

1.1 Background

Efficient hardware implementation of multiplications over $GF(2^m)$ are very important in many areas such as coding theory, computational algebra and public-key cryptosystems [1][2]. In general, there are three different architectures for implementing a $GF(2^m)$ multiplier: bit-serial, digit-serial and bit-parallel. Bit-serial multipliers use a single wire to process one bit at a time, while the bit-parallel multipliers use multiple wires to process multiple bits simultaneously. The bit-serial multipliers are the slowest in speed and the smallest in circuit size. In contrast, the bit-parallel multipliers are the fastest in speed but the greatest in circuit size. The digit-serial multipliers process a fixed number of bits (i.e. a digit) per cycle. They are a compromise between the bit parallel multipliers and the bit serial multipliers. Nowadays, with the development of very-large-scale integration (VLSI), more and more logic gates can be located on a single chip. This progress makes bit-parallel multipliers on chip possible and reasonable. A number of bit-parallel schemes for the multiplication over $GF(2^m)$ have been proposed during recent years. Characterized by a high computation speed or a small circuit size, these works cover extensive cases with respect to different base representations and different generating polynomials.

The choice of base representation, i.e. the way to represent multiplicands, has a direct influence on the performance of $GF(2^m)$ multipliers. The polynomial basis (PB) [9], with a form as $\{x^u \mid u = 0,1,2,\cdots,m-1\}$, is the simplest basis for constructing multipliers over $GF(2^m)$. Besides PB, some other bases have also been investigated for facilitating multiplications over special Galois fields. Fan proposed the shifted polynomial basis (SPB) and discussed two types of multipliers using SPB [7]. As a variation of PB, the SPB simplifies the modular operations. There are other unconventional bases such as the normal basis (NB) [28], the dual basis (DB) [29] and the general polynomial

basis (GPB) [10]. Compared with the polynomial basis, these unconventional bases are only better in some special cases.

The choice of irreducible polynomials for generating $GF(2^m)$ is crucial to the efficiency of the corresponding multiplier. Commonly used irreducible polynomials include all-one polynomial (AOP) [31], equally-spaced polynomial (ESP) [34], pentanomial [32] and trinomial [22][24]. Irreducible AOPs and ESPs have regular structures that can be exploited in the multiplication process, but such polynomials are quite rare. For example, there are no more than 100 irreducible AOPs in $\mathbb{F}_2[x]$ with degrees smaller than 1000. Besides, many multiplier designers like to use irreducible trinomials, i.e. polynomials with only three non-zero terms, because trinomials can minimize the number of XOR operations during the modular reduction. The irreducible trinomials are more abundant. For example, there are over 1000 irreducible trinomials in $\mathbb{F}_2[x]$ with degrees smaller than 1000. When no irreducible trinomial is available for a given degree, pentanomials with exactly five non-zero terms are used as a substitute. Empirically, one can always find an irreducible pentanomial to construct $GF(2^m)$ with a sufficiently big m.

A direct implementation of the multiplication over $GF(2^m)$ includes two steps: (1) to multiply two polynomials together in $\mathbb{F}_2[x]$ and (2) to reduce the product modulo the generating polynomial of $GF(2^m)$. Nowadays, there are three typical schemes for $GF(2^m)$ multiplication: (1) the Montgomery scheme, (2) the Karatsuba scheme and (3) the Mastrovito scheme. Originally used for facilitating integer modular multiplication, the Montgomery multiplication [14] was later modified for performing multiplications on $GF(2^m)$ [15][19][20]. Suppose $f(x)$ generates $GF(2^m)$, the Montgomery multiplication is generally expressed by $A(x)B(x)R^{-1}(x) \bmod f(x)$, where $A(x), B(x) \in GF(2^m)$, $R(x)$ is a power of x and $R^{-1}(x)$ is the inverse of $R(x)$ in $GF(2^m)$. This multiplication is suitable for both hardware and software implementations.

The Karatsuba scheme is a divide-and-conquer algorithm that transforms a polynomial multiplication into three sub-polynomial multiplications with extra polynomial additions. This algorithm is often recursively applied in building sub-quadratic $GF(2^m)$ multipliers [33]. Elia [23] has constructed a bit-parallel multiplier by using only one recursion of the Karatsuba algorithm, Li [24] and Cho [22] later developed variations of

Elia's algorithm.

The Mastrovito scheme [11] carries out a multiplication in $GF(2^m)$ as a matrix-vector multiplication with elements in $GF(2)$. Suppose $C(x) = A(x)B(x) \bmod f(x)$, a matrix $\mathbf{M}$ can be constructed from $A(x)$ and $f(x)$. Then, $\mathbf{C} = \mathbf{MB}$, where $\mathbf{C}$ and $\mathbf{B}$ represent the coefficient vectors of $C(x)$ and $B(x)$, respectively. This scheme is especially suitable for constructing bit-parallel multipliers, and has attracted a lot of attentions [7][8][12][13][25].

1.2 Motivation

Modern cryptosystems, such as Elliptic curve cryptography, heavily rely on the $GF(2^m)$ arithmetic. In fact, the size of $GF(2^m)$ is usually larger than 2^{160}. This may cause efficiency problems with the arithmetic operations over $GF(2^m)$, especially in embedded systems with limited computing power and small area capacity. Thus, it is crucial to design efficient algorithms for implementing arithmetic operations over $GF(2^m)$.

When designing a multiplier, we may have to deal with the relationship between the size, i.e. space complexity, and the delay, i.e. time complexity, of the circuit. On one hand, an improvement in the speed of the circuit generally requires an increase in the number of logic gates. On the other hand, if we want to save logic gates, we may face the risk of increasing the delay of the circuit. Thus, many schemes are more efficient only for the space or for the time. To construct a multiplier with a better balance between the size and the speed, we have to deal with the problem of "space-time trade-off" as reflected by the title of this thesis. We aim to design a new multiplication algorithm that costs fewer gates than the others while maintaining a relatively high speed.

1.3 Contribution of the Thesis

This thesis presents a new divide-and-conquer $GF(2^m)$ multiplication algorithm based on the PCHS algorithm proposed by S. Park, K, Chang, D. Hong, and C. Seo[21]. The PCHS algorithm was originally used for constructing multipliers over the Galois fields generated by two special types of irreducible pentanomials. Nevertheless, the PCHS

algorithm is somewhat complicated when performing squaring operations that involve transformations between the weak dual basis (WDB) and the polynomial basis (PB) of $GF(2^m)$. By adopting the multiplicand partition rule in the PCHS algorithm and the Montgomery squaring formulae studied by Wu [19], we propose a new bit-parallel Montgomery multiplication algorithm on trinomial-generated Galois fields. We also make an explicit analysis of the new algorithm on its space and time complexities.

The main contributions of our work are as follows:

1. This new multiplier is about 25% more efficient in space complexity than the previous trinomial-based Montgomery multipliers or trinomial-based Mastrovito multipliers on $GF(2^m)$ that is big enough. Its space complexity is very close to that of the Karatsuba multiplier proposed by Elia.

2. This new multiplier is at most two T_X larger in time complexity than the fastest bit-parallel multiplier, where T_X is the XOR gate delay.

3. Let F denote the set of 1405 Galois fields generated by

$$\{f(x) = x^m + x^k + 1 \mid 100 \leq m \leq 1203,\ 1 \leq k \leq m/2,\ f(x) \text{ is irreducible}\},$$

   our multiplier has a time complexity of $T_A + (2 + \lceil \log_2 m \rceil)T_X$ on the 1061 Galois fields in F, and a time complexity of $T_A + (3 + \lceil \log_2 m \rceil)T_X$ on the rest of the Galois fields in F, where T_A is the AND gate delay.

4. We compare our multiplier with the other multipliers in the space and time complexities on the two fields generated by $x^{233} + x^{74} + 1$ and $x^{409} + x^{87} + 1$, respectively, as recommended by the National Institute of Standards and Technology (NIST). Our multiplier has a more-than-15% reduced space complexity than the other Montgomery or Mastrovito multipliers by paying at most one more T_X in time complexity. Moreover, our multiplier is one T_X faster than the Elia's multiplier at the cost of a less-than-1% increase in space complexity.

1.4 Outline and Organization

This thesis aims to obtain an improved Montgomery multiplication algorithm for the Galois fields generated by trinomials. The new algorithm uses a divide-and-conquer technique and relies on the squaring operation in $GF(2^m)$. This thesis is divided into five chapters. After a background introduction in Chapter 1 as mentioned above, Chapter 2 gives a brief introduction of $GF(2^m)$ and its arithmetic, followed by a discussion about some typical $GF(2^m)$ multiplication schemes including the ordinary algorithm, the Montgomery algorithm, the Karatsuba algorithm, and the PCHS algorithm. Chapter 3 presents the new algorithm step-by-step, providing a detailed analysis of the space and time complexities of the new multiplier on one type of Galois field, followed by a summary of the space and time complexities on the other types of Galois fields. Chapter 4 compares the time and space complexities of our algorithm with those of the other typical algorithms. Finally, Chapter 5 summarizes our research and proposes a plan for possible theoretical improvement and hardware implementation in the future.

CHAPTER 2

PRELIMINARIES

2.1 Galois Field GF(2^m) and Its Arithmetic

We first introduce the concepts of a group, a ring and a field, especially the Galois field $GF(2^m)$ where the Montgomery multiplication algorithms are used. We first present the definition of a group:

Definition 2.1.1 ([3]). *A nonempty set of elements G is said to form a group if in G there is defined a binary operation, called the product and denoted by $\cdot$, such that*

1. $a,\ b \in G$ *implies that* $a \cdot b \in G$ *(closed).*
2. $a,\ b,\ c \in G$ *implies that* $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ *(associative law).*
3. *There exists an element e in G such that* $e \cdot a = a \cdot e = a$ *for all* $a \in G$ *(the existence of an identity element in G).*
4. *For every* $a \in G$ *there exists an element* $a^{-1} \in G$ *such that* $a \cdot a^{-1} = a^{-1} \cdot a = e$ *(the existence of inverses in G).*

If the operation of a group G is such that $a \cdot b = b \cdot a$ for every a, b in G, then we call G a commutative group.

Next, we introduce the definition of a (associative) ring. A ring is a set that has two operational rules called addition and multiplication. In a ring, all the elements form a commutative group under addition, while the nonzero elements do not necessarily form a group under multiplication:

Definition 2.1.2 ([3]). *A nonempty set R is said to be an associative ring if in R there are defined two operations, denoted by + and $\cdot$ respectively, such that for all a, b, c in R:*

1. $a+b$ *is in R.*
2. $a+b = b+a$.
3. $(a+b)+c = a+(b+c)$.

4. *There is an element* 0 *in* R *such that* $a+0=a$ *(for every* a *in* R*).*
5. *There exists an element* $(-a)$ *in* R *such that* $a+(-a)=0$.
6. $a \cdot b$ *is in* R.
7. $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
8. $a \cdot (b+c) = (a \cdot b) + (a \cdot c)$ *and* $(b+c) \cdot a = (b \cdot a) + (c \cdot a)$ *(the two distributive laws).*

If the multiplication of a ring R is such that $a \cdot b = b \cdot a$ for every a, b in R, then we call R a commutative ring. Besides, a ring is said to be a division ring if its nonzero elements form a group under multiplication. Finally, we introduce the concept of a field:

Definition 2.1.3 ([3])**.** *A field is a commutative division ring.*

A field is said to be finite if it contains finitely many elements. $GF(2)$ is the smallest finite field with the following definition:

Definition 2.1.4 ([6])**.** $GF(2)$*, also known as* $\mathbb{F}_2$*, is a field which contains only two elements: 0 and 1. In* $GF(2)$*, the modulo-2 addition or XOR is defined as:* $0+1=1$*,* $1+1=0$*,* $0+0=0$*,* $1+0=1$*, and the multiplication or AND is defined as:* $0 \cdot 1 = 0$*,* $1 \cdot 1 = 1$*,* $0 \cdot 0 = 0$*,* $1 \cdot 0 = 0$*. Therefore, the additive inversion operation is* $-0=0$ *and* $-1=1$*, and the multiplicative inversion operation is* $1^{-1}=1$.

The *ring of polynomials* [3] over the field $\mathbb{F}_2$ in the indeterminate x, denoted by $\mathbb{F}_2[x]$ or $GF(2)[x]$, is the set of all symbols $a_0 + a_1x + \cdots + a_nx^n$, where n can be any non-negative integer and where the coefficients $a_1,\ a_2, \cdots,\ a_n$ are all in $\mathbb{F}_2$. We consider two $(m-1)$-degree polynomials, say $A(x)$ and $B(x)$, in $\mathbb{F}_2[x]$:

$$A(x) = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0,$$

$$B(x) = \sum_{i=0}^{m-1} b_i x^i = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_2x^2 + b_1x + b_0.$$

In $\mathbb{F}_2[x]$, the equality of two polynomials is defined by:

$$A(x) = B(x) \Longleftrightarrow \forall i \in [0,\ m-1],\ a_i = b_i,$$

and the rules for adding and multiplying two polynomials are:

$$\begin{aligned} A(x)+B(x) &= (a_{m-1}+b_{m-1})x^{m-1} + (a_{m-2}+b_{m-2})x^{m-2} + \cdots \\ &\cdots + (a_2+b_2)x^2 + (a_1+b_1)x + (a_0+b_0) \\ &= \sum_{i=0}^{m-1} (a_i+b_i)x^i, \end{aligned}$$

and

$$\begin{aligned} A(x)B(x) &= (a_{m-1}b_{m-1})x^{2m-2} + (a_{m-1}b_{m-2}+a_{m-2}b_{m-1})x^{2m-3} + \cdots \\ &\cdots + (a_2b_0+a_1b_1+a_0b_2)x^2 + (a_1b_0+a_0b_1)x + a_0b_0 \\ &= \sum_{i=0}^{m-1} (\sum_{j=0}^{i} a_j b_{i-j})x^i + \sum_{i=m}^{2m-2} (\sum_{j=i-m+1}^{m-1} a_j b_{i-j})x^i, \end{aligned}$$

respectively. Based on the above rules, it can be verified that the axioms defining a ring hold true for $\mathbb{F}_2[x]$. Moreover, the following division algorithm holds over $\mathbb{F}_2[x]$:

Lemma 2.1.5 ([3])**.** *Given two unique polynomials $p(x)$ and $f(x) \neq 0$ in $\mathbb{F}_2[x]$, then there exist two polynomials $h(x)$ and $r(x)$ in $\mathbb{F}_2[x]$ such that $p(x) = h(x) \cdot f(x) + r(x)$, where $r(x) = 0$ or $\deg r(x) < \deg f(x)$.*

In the above lemma, we call $r(x)$ the remainder of $p(x)$ modulo $f(x)$, i.e. $r(x) = p(x) \bmod f(x)$.

Definition 2.1.6. *Suppose a polynomial $p(x) \in \mathbb{F}_2[x]$, then $p(x)$ is called an irreducible polynomial over $\mathbb{F}_2$ if and only if $p(x)$ is not divisible by any polynomial in $\mathbb{F}_2[x]$ except for 1 and $p(x)$ itself.*

Based on an irreducible polynomial $f(x) \in \mathbb{F}_2[x]$, a field $\mathbb{F}_{f(x)}$ is constructed from $R_{\mathbb{F}_2,m} = \{r(x) | r(x) = p(x) \bmod f(x),\ \forall p(x) \in \mathbb{F}_2[x]\}$:

Lemma 2.1.7 ([6]). *If $f(x)$ is an irreducible polynomial of degree m over $\mathbb{F}_2$, then the set of remainder polynomials $R_{\mathbb{F}_2,m}$ with mod-$f(x)$ arithmetic forms a finite field $\mathbb{F}_{f(x)}$ with 2^m elements.*

In the above theorem, the finite field $\mathbb{F}_{f(x)}$ is a Galois field with 2^m elements, usually denoted by $GF(2^m)$; and $f(x)$ is called the generating polynomial of $GF(2^m)$. For any two elements $A(x),\ B(x) \in GF(2^m)$, the addition and multiplication in $GF(2^m)$ are defined as:

$$\begin{aligned}&\text{Addition: } A(x)+B(x),\\&\text{Multiplication: } A(x)\cdot B(x) \bmod f(x),\end{aligned}$$

respectively, where "$\cdot$" and "$+$" represent the multiplication and the addition in $\mathbb{F}_2[x]$, respectively. It is clear that $GF(2^m)$ is a commutative ring. To show that $GF(2^m)$ is also a division ring, it remains to prove that every non-zero element in $GF(2^m)$ has its inverse. For this, we make use of the Euclidean division algorithm, which provides us with $a(x), b(x) \in \mathbb{F}_2[x]$ for a nonzero element $p(x) \in GF(2^m)$ such that

$$a(x)p(x)+b(x)f(x) = \mathrm{GCD}(p(x), f(x)) = 1 \text{ (Bezout Identity)},$$

where $\deg(a(x)) \leq m-1$. Thus, $GF(2^m)$ is indeed a field.

In $GF(2^m)$, an inversion operation can be treated as a power operation and can be thus calculated through iterative multiplications [3]. Therefore, the multiplication is one of the most fundamental operations in $GF(2^m)$.

2.2 Some Concepts in Circuit Layout and Computation

In this section, we illustrate some basic concepts about logic gates and circuits. These concepts will be used in the remainder of the thesis.

2.2.1 The AND Gate and the XOR Gate

The AND gate and the XOR gate implement the addition and multiplication of $\mathbb{F}_2$, respectively. The AND gate is a component that receives two bits as signal source and outputs the product of the input bits in $\mathbb{F}_2$. The XOR gate is a component that receives two bits as signal source and outputs the sum of the input bits in $\mathbb{F}_2$. We represent these components in Figure 2.1 and 2.2, respectively.

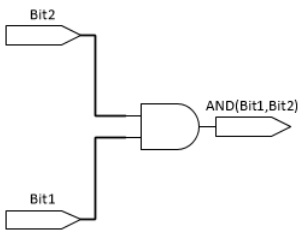


Figure 2.1 – The AND gate symbol

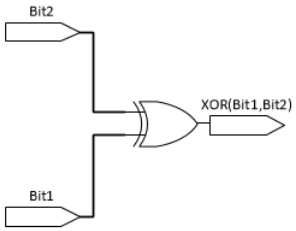


Figure 2.2 – The XOR gate symbol

Figure 2.3 – The simplified XOR gate symbol

In the next chapter, a XOR gate will be illustrated by a simplified figure in which the input and output bits are represented by circles, as shown in Figure 2.3.

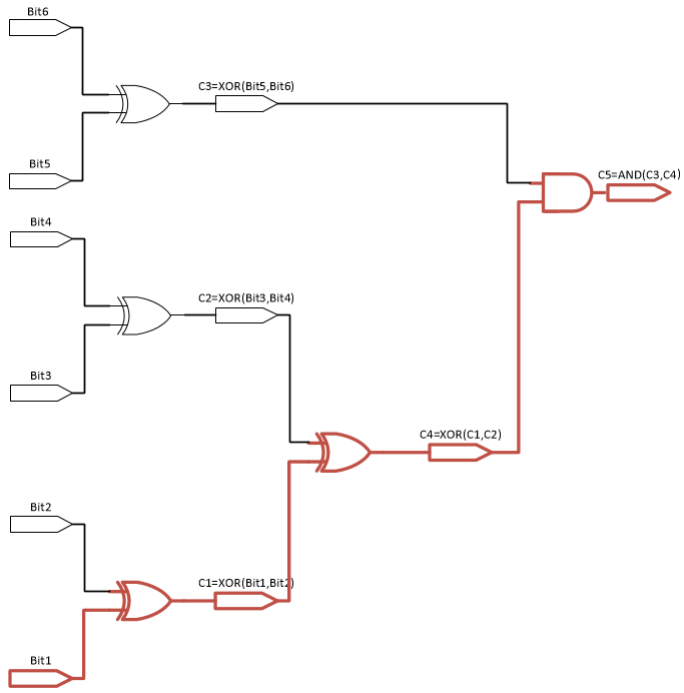
2.2.2 The Space Complexity and the Time Complexity

It takes some time for a circuit to perform a designated task. The time is determined by the longest path of the circuit from the input to the output. We call such paths the critical paths, and the length of such paths the depth of circuit.

A circuit in parallel architecture performs multiple operations in every time unit. As time goes on, the operations produce intermediate results that form successive layers corresponding to different time units. The number of such layers is equal to the depth of circuit.

Here, we give an example of bit-parallel operation in Figure 2.4. All the bits are processed pairwise on each layer. Layer 0 consists of $Bit1$, $Bit2$, $Bit3$, $Bit4$, $Bit5$ and $Bit6$ as the input; Layer 1 consists of $C1$, $C2$ and $C3$; Layer 2 consists of $C4$ and $C3$; Layer 3 contains only $C5$ as the output. One of the longest paths is $Bit1 \Rightarrow C1 \Rightarrow C4 \Rightarrow C5$, as

Figure 2.4 – Illustration of circuit delay



highlighted in red. Therefore, the depth of the example circuit is 3. Let $T^{\oplus}$ denote the total delay caused by XOR operations, and $T^{\otimes}$ the total delay by AND operations. The example circuit has an overall delay of $2T_X + T_A$ with $T^{\oplus} = 2T_X$ and $T^{\otimes} = T_A$, where T_X and T_A denote the XOR gate delay and the AND gate delay, respectively. Formally, the circuit delay is called the time complexity of circuit.

Normally, each layer of the circuit contains only one type of logic gates, i.e. either AND gates or XOR gates. It is only the XOR gates that are used when designing a circuit for evaluating the XOR of all input bits. The process of parallel XOR operation can be illustrated as an inverted binary tree with bits XORed pairwise on each layer. Each pair of bits, together with their sum in $\mathbb{F}_2$, corresponds to a XOR gate. Suppose there are n bits to XOR together, then the corresponding binary XOR tree has a depth of $\lceil \log_2 n \rceil$, and it will take a delay of $\lceil \log_2 n \rceil T_X$ to compute the XOR in parallel.

The area complexity, or the space complexity, is the total number of logic gates in a circuit. In the rest of the thesis, $S^{\otimes}$ denotes the number of AND gates in a circuit, and

$S^{\oplus}$ denotes the number of XOR gates in a circuit. For example, there are four XOR gates plus one AND gate in Figure 2.4, hence the space complexity of the example circuit is $S^{\oplus} = 4$ and $S^{\otimes} = 1$.

In this thesis, we analyse the space and time complexities of a multiplier by treating the multiplicands as two $(m-1)$-degree polynomials in $GF(2^m)$.

2.3 The Ordinary Multiplication Scheme

The ordinary method requires m^2 AND gates and near m^2 XOR gates [9]. Consider the case of an even number m and an irreducible polynomial $x^m + x^k + 1$ where $0 \leq k \leq \frac{m}{2}$, and let $GF(2^m)$ be generated by this polynomial. We calculate $A(x) \cdot B(x) \in GF(2^m)$ where $A(x),\ B(x) \in GF(2^m)$ and $\deg A(x) = \deg B(x) = m-1$. For clarity, we express $A(x)$ and $B(x)$ as

$$A(x) = A_1(x)x^{\frac{m}{2}} + A_0(x),$$
$$B(x) = B_1(x)x^{\frac{m}{2}} + B_0(x),$$

respectively, where $A_0(x),\ A_1(x),\ B_0(x),\ B_1(x)$ are all of degree $\frac{m}{2} - 1$. Thus,

$$\begin{aligned}
A(x)B(x) &= (A_1(x)x^{\frac{m}{2}} + A_0(x))(B_1(x)x^{\frac{m}{2}} + B_0(x)) \\
&= A_1(x)B_1(x)x^m + (A_1(x)B_0(x) + B_1(x)A_0(x))x^{\frac{m}{2}} + A_0(x)B_0(x) \\
&= A_1(x)B_1(x)(1+x^k) + (A_1(x)B_0(x) + B_1(x)A_0(x))x^{\frac{m}{2}} + A_0(x)B_0(x) \\
&= \underbrace{(A_1(x)B_0(x) + B_1(x)A_0(x))x^{\frac{m}{2}} + A_1(x)B_1(x)x^k}_{GF(2^m)\text{ modular reduction required}} + (A_0(x)B_0(x) + A_1(x)B_1(x)).
\end{aligned}$$

Based on the above expression, the corresponding parallel multiplication is shown in Table 2.I, with a summary of the space and time complexities.

During the past thirty years, some new algorithms have been designed to achieve higher speed or smaller circuit for implementing $GF(2^m)$ multiplication on both software and hardware.

Table 2.I – The procedure of the ordinary multiplication method

Stages	Polynomials	#AND	#XOR	Delay
Stage I	$P_0(x) = A_0(x)B_0(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	$T_A + \lceil \log_2 \frac{m}{2} \rceil T_X$
	$P_1(x) = A_0(x)B_1(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	
	$P_2(x) = A_1(x)B_0(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	
	$P_3(x) = A_1(x)B_1(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	
Stage II	$P_4(x) = P_0(x) + P_3(x) + P_3(x)x^k$	-	$2m+k-3$	$2T_X$
	$P_5(x) = (P_1(x) + P_2(x))x^{\frac{m}{2}}$	-	$\frac{3}{2}m-3$	
Stage III	$P_4(x) + P_5(x)$	-	$m-1$	T_X

2.4 The Montgomery Multiplication Scheme

The Montgomery multiplication scheme was initially introduced by Peter L. Montgomery [14] in 1985 for the integer modular multiplications. It was later in 1998 that the Montgomery multiplication scheme was adapted by Koç and Acar [15] for performing the multiplications over $GF(2^m)$. Let $f(x)$ be the generating polynomial of $GF(2^m)$, $R(x) = x^m$, $f'(x)$, $R^*(x)$ be polynomials in $\mathbb{F}_2[x]$ with degrees smaller than m such that $R(x)R^*(x) + f(x)f'(x) = 1$, i.e. $R^*(x)$ is the inverse of $R(x)$ in $GF(2^m)$. Suppose $A(x),\ B(x) \in GF(2^m)$, the first Montgomery multiplication algorithm over $GF(2^m)$ is given in Algorithm 1:

Algorithm 1 The classic Montgomery multiplication algorithm over $GF(2^m)$

Input: $A(x),\ B(x),\ R(x),\ f(x),\ f'(x)$
Output: $A(x)B(x)R^*(x) \bmod f(x)$
1: $q(x) = A(x)B(x)f'(x) \bmod R(x)$.
2: Return $(A(x)B(x) - q(x)f(x))/R(x)$.

$R(x)$ is called the Montgomery factor. As we can see, the classic Montgomery multiplication algorithm transforms a modulo-$f(x)$ reduction into a modulo-x^m reduction plus a by-x^m division, which enables simple and easy implementation. The following example, showing the application of Algorithm 1, may better help us understand the usage of

the Montgomery multiplication:

Example:

Suppose $g(x),\ f(x) \in \mathbb{F}_2[x]$, $g(x) = x^2 + x + 1$ and $f(x) = x^3 + x^2 + 1$ ($f(x)$ is irreducible), calculate $g(x)^5 \bmod f(x)$ by using Algorithm 1.

Solution:

The outline of our solution is: (1) $g(x)$ is transformed as $g_1(x)$. (2) Three Montgomery multiplications are consecutively applied. (3) The result $O_3(x)$ is transformed.

We calculate $g_1(x) = g(x)R(x) \bmod f(x) = 1$. We calculate $R^*(x) = x^2 + x + 1$ and $f'(x) = x^2 + 1$ such that $R(x)R^*(x) + f(x)f'(x) = 1$. Then, based on the binary form of the exponent $(5)_2 = (101)$, we use Algorithm 1 to calculate the following three Montgomery multiplications in a line:

$$
\begin{aligned}
O_1(x) &= (g_1(x))^2 R^*(x) \bmod f(x),\\
O_2(x) &= (O_1(x))^2 R^*(x) \bmod f(x),\\
O_3(x) &= O_2(x) g_1(x) R^*(x) \bmod f(x).
\end{aligned}
$$

It can be verified that $O_3(x) = x^2 = g^5(x)R(x) \bmod f(x)$. Thus,

$$
\begin{aligned}
g^5(x) \bmod f(x) &= O_3(x)R^*(x) \bmod f(x)\\
&= x^2 \cdot (x^2 + x + 1) \bmod x^3 + x^2 + 1\\
&= x^2 + x.
\end{aligned}
$$

The explicit proof for Algorithm 1 can be found in [15]. This algorithm has a lower time complexity than the ordinary method when performing a large series of multiplications. As a basic form of multiplication in $GF(2^m)$, the Montgomery multiplication has attracted more and more attentions world-wide today.

Initially, the Montgomery factor was selected as x^m for the efficient implementation of bit-serial Montgomery multipliers [15]. In 2002, Wu slightly generalized the method in [15] and proposed a bit-parallel Montgomery multiplier based on irreducible trinomials [19]. Wu showed that using the mid-term of the expression $f(x) = x^m + x^k + 1$,

i.e. x^k, as the Montgomery factor can help design more efficient bit-parallel multipliers. Fast and parallel, Wu's algorithm inspired Hariri and Reyhani-Masoleh to formulate a set of strategies for designing trinomial-based multipliers as well as pentanomial-based multipliers [20]. Hariri and Reyhani-Masoleh have proved that choosing the Montgomery factor $R(x)$ as x^{m-1} is optimum for constructing bit-serial Montgomery multipliers, while $R(x) = x^k$ or $R(x) = x^{k-1}$ is the best for constructing bit-parallel Montgomery multipliers. The algorithm of Hariri and Reyhani-Masoleh has the lowest time complexity reported in the literature [9]. It is an interesting coincidence that Fan [8] proposed a shifted polynomial basis multiplier based on a similar idea to Wu's [19], which supports Wu's choice of the Montgomery factor. Besides, the Montgomery multiplication algorithms have been investigated with respect to some hardware implementation issues such as the reliability [17] and the efficiency-area ratio [18]. The former is the chance of well-functioning against errors in bit-parallel architectures, the latter is the ratio of the speed to the amount of logic gates in the circuit.

When $A(x) = B(x) = \sum_{i=0}^{m-1} a_i x^i$, the Montgomery multiplication of $A(x)$ and $B(x)$ becomes the Montgomery squaring operation of $A(x)$, expressed as:

$$A^2(x)x^{-u} = x^{-u} \sum_{i=0}^{m-1} a_i x^{2i} \tag{2.1}$$

Wu [19] has done a thorough research on the trinomial-based Montgomery squaring operation. For example, suppose both m and k are odd numbers, $1 \leq k \leq \frac{m-3}{2}$, $f(x) = x^m + x^k + 1$ generates $GF(2^m)$, then the coefficient expression of $A^2(x)x^{-k} = \sum_{i=0}^{m-1} a'_i x^i$ is found as:

$$a'_i = \begin{cases} a_{\frac{i}{2}} + a_{\frac{m+k+i}{2}} & i = 0, 2, \cdots, k-1, \\ a_{\frac{m+k+i}{2}} & i = k+1, k+3, \cdots, m-k-2, \\ a_{\frac{k-m+i}{2}} & i = m-k, m-k+2 \cdots, m-1, \\ a_{\frac{k+i}{2}} & i = 1, 3, \cdots, k-2, \\ a_{\frac{k+i}{2}} + a_{\frac{m+i}{2}} & i = k, k+2, \cdots, m-2, \end{cases} \tag{2.2}$$

where the fourth sub-expression, i.e. that for $i = 1, 3, \cdots, k-2$, is neglected when $k = 1$.

It is worthwhile to note that the Montgomery squaring is much easier to implement than the Montgomery multiplication [19].

2.5 The Divide-and-Conquer Multiplication Schemes

2.5.1 The Karatsuba Algorithm in $\mathbb{F}_2[\mathbf{x}]$

The Karatsuba algorithm (KA) is one of the most famous divide-and-conquer multiplication algorithms. The bit-parallel architecture of KA over $\mathbb{F}_2[x]$ is widely used for designing bit-parallel multipliers over $GF(2^m)$. Here, we briefly introduce the bit-parallel KA over $\mathbb{F}_2[x]$.

To be concise, let m be an even integer, $A(x)$ and $B(x)$ two $(m-1)$-degree polynomials in $\mathbb{F}_2[x]$. The Karatsuba scheme partitions $A(x)$ and $B(x)$ into two sub-polynomials, respectively, such that

$$A(x) = A_1(x)x^{\frac{m}{2}} + A_0(x),$$
$$B(x) = B_1(x)x^{\frac{m}{2}} + B_0(x),$$

where $A_0(x)$, $A_1(x)$, $B_0(x)$, $B_1(x)$ are all of degree $\frac{m}{2}-1$.

Hence, the product of $A(x)$ and $B(x)$ is:

$$\begin{aligned} A(x)B(x) &= (A_1(x)x^{\frac{m}{2}} + A_0(x))(B_1(x)x^{\frac{m}{2}} + B_0(x)) \\ &= A_1(x)B_1(x)x^m + (A_1(x)B_0(x) + B_1(x)A_0(x))x^{\frac{m}{2}} + A_0(x)B_0(x) \\ &= \alpha(x)x^m + \beta(x)x^{\frac{m}{2}} + \gamma(x), \end{aligned}$$

where $\alpha(x) = A_1(x)B_1(x)$, $\gamma(x) = A_0(x)B_0(x)$, $\beta(x) = (A_0(x)+A_1(x))(B_0(x)+B_1(x)) - \alpha(x) - \gamma(x)$.

It should be noted that the operation "$-$" is the same as "$+$" in $\mathbb{F}_2$. According to the above formula, it is clear that $A(x)B(x)$ can be computed with three $(\frac{m}{2}-1)$-degree sub-polynomial multiplications at the cost of $2m-1$ additional XOR operations.

For a $m > 0$, let the two multiplicands be of degree $m-1$. Let $S^{\otimes}(m)$, $S^{\oplus}(m)$

denote the number of AND operations and the number of XOR operations, respectively. Let $T^{\otimes}(m)$, $T^{\oplus}(m)$ denote the delay caused by AND operations and the delay by XOR operations, respectively. Then we have the following recurrence relation formulae [5] for the bit-parallel KA:

$$\begin{cases} S^{\otimes}(2) = 3, \\ S^{\otimes}(m) = 3S^{\otimes}(m/2), \end{cases} \qquad \begin{cases} T^{\otimes}(2) = 1, \\ T^{\otimes}(m) = T^{\otimes}(m/2), \end{cases}$$

and

$$\begin{cases} S^{\oplus}(2) = 4, \\ S^{\oplus}(m) = 3S^{\oplus}(m/2) + 4m - 4, \end{cases} \qquad \begin{cases} T^{\oplus}(2) = 2, \\ T^{\oplus}(m) = T^{\oplus}(m/2) + 3. \end{cases}$$

For simplicity, suppose $m = 2^t$ for some $t > 1$, then KA can be recursively applied to the sub-polynomial multiplications of $A(x)$ and $B(x)$, giving rise to the following space and time complexities for computing $A(x)B(x)$:

$$\begin{cases} S^{\otimes}(m) = m^{\log_2 3}, \\ S^{\oplus}(m) = 6m^{\log_2 3} - 8m + 2, \\ T^{\otimes}(m) = 1, \\ T^{\oplus}(m) = 3\log_2 m - 1, \end{cases}$$

which shows that a sub-quadratic space complexity can be achieved by paying a logarithmic time complexity.

Elia et al. [23] adapted KA into $GF(2^m)$ multiplication and developed an efficient trinomial-based bit-parallel algorithm, as shown in Table 2.II. Elia's algorithm is nearly a quarter less than the other contemporary algorithms in space complexity, but is at least two T_X slower than the fastest algorithms [7] or [20], and is one T_X slower than the ordinary multiplier described in Table 2.I.

2.5.2 The PCHS Algorithm

The PCHS algorithm is similar to the Karatsuba algorithm. Nonetheless, it transforms the multiplicand expressions to make them contain polynomial squaring opera-

Table 2.II – Elia's algorithm on $GF(2^m)$ generated by $x^m + x^k + 1$, m even, $0 < k \leq \frac{m}{2}$

Stages	Polynomials	#AND	#XOR	Delay
Stage I	$P_0(x) = A_0(x) + A_1(x)$	-	$\frac{m}{2}$	T_X
	$P_1(x) = B_0(x) + B_1(x)$	-	$\frac{m}{2}$	
Stage II	$P_2(x) = A_0(x)B_0(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	$T_A + \lceil \log_2 \frac{m}{2} \rceil T_X$
	$P_3(x) = A_1(x)B_1(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	
	$P_4(x) = P_0(x)P_1(x)$	$\frac{m^2}{4}$	$(\frac{m}{2}-1)^2$	
	$P_5(x) = P_2(x) + P_3(x)$	-	$m-1$	
Stage III	$P_6(x) = P_4(x) + P_5(x)$	-	$m-1$	T_X
	$P_7(x) = x^k P_3(x)$	-	$k-1$	
Stage IV	$P_8(x) = x^{\frac{m}{2}} P_6(x)$	-	$\frac{m}{2}-2$	T_X
	$P_9(x) = P_5(x) + P_7(x)$	-	$m-2$	
Stage V	$P_8(x) + P_9(x)$	-	m	T_X

tions, as squaring operations can be done very efficiently in general.

Let $A(x) = \sum_{i=0}^{m-1} a_i x^i$ and $B(x) = \sum_{i=0}^{m-1} b_i x^i$ be two polynomials in $GF(2^m)$. With m being an odd number, the PCHS algorithm partitions $A(x), B(x)$ into

$$A(x) = A_1^2(x) + xA_2^2(x) \text{ and } B(x) = x^{-1}B_1^2(x) + B_2^2(x),$$

respectively, where

$$A_1(x) = \sum_{i=0}^{(m-1)/2} a_{2i}x^i,\ A_2(x) = \sum_{i=0}^{(m-3)/2} a_{2i+1}x^i,$$

$$B_1(x) = \sum_{i=1}^{(m-1)/2} b_{2i-1}x^i,\ B_2(x) = \sum_{i=0}^{(m-1)/2} b_{2i}x^i.$$

Then we have

$$
\begin{aligned}
A(x)B(x) &= (A_1^2(x) + xA_2^2(x)(x^{-1}B_1^2(x) + B_2^2(x)) \\
&= x^{-1}(A_1(x)B_1(x))^2 + x(A_2(x)B_2(x))^2 \\
&\quad + (A_1(x)B_2(x))^2 + (A_2(x)B_1(x))^2 \\
&= x^{-1}(A_1(x)B_1(x))^2 + x(A_2(x)B_2(x))^2 + (A_1(x)B_1(x))^2 \\
&\quad + (A_2(x)B_2(x))^2 + [(A_1(x) + A_2(x))(B_1(x) + B_2(x))]^2 \\
&= (x^{-1} + 1)(A_1(x)B_1(x))^2 + (x+1)(A_2(x)B_2(x))^2 \\
&\quad + [(A_1(x) + A_2(x))(B_1(x) + B_2(x))]^2 .
\end{aligned}
\tag{2.3}
$$

Somewhat similar to the Karatsuba algorithm, (2.3) contains three sub-polynomial multiplications, three polynomial squaring operations, and a few extra polynomial multiplications and additions. It is the usage of efficient $GF(2^m)$ squaring techniques in (2.3) that can make the PCHS algorithm efficient. In the next chapter, we present the new bit-parallel $GF(2^m)$ Montgomery multiplication algorithm that derives from the original PCHS algorithm while using the Montgomery squaring formulae.

CHAPTER 3

NEW FIELD MULTIPLICATION USING MONTGOMERY SQUARING OPERATION

In this chapter, we present a new Montgomery multiplication algorithm for the Galois fields generated by trinomials. Although this new algorithm is based on the previously introduced PCHS algorithm, it differs from the original PCHS algorithm in that it requires a simpler $GF(2^m)$ squaring operation method than that used by the original PCHS algorithm.

Wu has given explicit formulae for the ordinary squaring operation [9] and the Montgomery squaring operation [19], respectively, on the Galois fields generated by trinomials. In Wu's work, the Montgomery squaring operation is found to be simpler than the ordinary squaring operation with respect to their space and time complexities [19]. Therefore, our algorithm uses the Montgomery squaring operation.

3.1 Our Proposal

Let $f(x) = x^m + x^k + 1$ generate $GF(2^m)$. From now on, we only use the polynomial basis for representing the elements in $GF(2^n)$. Besides, we only consider $f(x) = x^m + x^k + 1$ with $1 \leq k \leq \frac{m}{2}$. We can always find such irreducible trinomials, if there exist irreducible trinomials of degree m. This is due to the so-called "Reciprocal Property" [6]: If a trinomial $x^m + x^k + 1$ with $k > \frac{m}{2}$ is irreducible, then the trinomial $x^m + x^{m-k} + 1$ is also irreducible.

Let $A(x),\ B(x) \in GF(2^m)$:

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0,$$
$$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1x + b_0,$$

where $a_i,\ b_i \in GF(2)$. Let the Montgomery factor be $R(x) = x^u$ $(1 \leq u < m)$, the Mont-

gomery multiplication of $A(x)$ and $B(x)$ over $GF(2^m)$ is:

$$A(x)B(x)x^{-u}. \tag{3.1}$$

As the PCHS algorithm was originally designed for $GF(2^m)$ with odd m, in our algorithm, we first assume the generating trinomial to be of odd degree.

We split $A(x), B(x)$ according in the way of the PCHS algorithm:

$$A(x) = A_1^2(x) + xA_2^2(x), \;\; B(x) = x^{-1}B_1^2(x) + B_2^2(x),$$

where

$$\begin{aligned} A_1(x) &= \sum_{i=0}^{\frac{m-1}{2}} a_{2i}x^i, \; A_2(x) = \sum_{i=0}^{\frac{m-3}{2}} a_{2i+1}x^i, \\ B_1(x) &= \sum_{i=1}^{\frac{m-1}{2}} b_{2i-1}x^i, \; B_2(x) = \sum_{i=0}^{\frac{m-1}{2}} b_{2i}x^i. \end{aligned}$$

The corresponding Montgomery multiplication formula is:

$$\begin{aligned} A(x)B(x)x^{-u} &= [x^{-1}(A_1(x)B_1(x))^2 + x(A_2(x)B_2(x))^2 + (A_1(x)B_1(x))^2 \\ &\quad + (A_2(x)B_2(x))^2 + (A_1(x)+A_2(x))^2(B_1(x)+B_2(x))^2]x^{-u} \\ &= (A_1(x)B_1(x))^2x^{-u}(1+x^{-1}) + (A_2(x)B_2(x))^2x^{-u}(1+x) \\ &\quad + ((A_1(x)+A_2(x))(B_1(x)+B_2(x)))^2x^{-u}, \end{aligned} \tag{3.2}$$

which contains the Montgomery squaring of $A_1(x)B_1(x)$, the Montgomery squaring of $A_2(x)B_2(x)$, and that of $(A_1(x)+A_2(x))(B_1(x)+B_2(x))$.

It is worth noting that the choice of the Montgomery factor x^u has a direct influence on the efficiency of the related Montgomery squaring operation. Hariri and Reyhani-Masoleh have found the optimum Montgomery factors for constructing a trinomial-based Montgomery multiplier [20]. We provide their conclusion in the following lemma:

Lemma 3.1.1 ([20]). *Let $f(x) = x^m + x^k + 1$ be an irreducible trinomial over $\mathbb{F}_2$. On the*

Galois field generated by $f(x)$, in order to achieve the fewest possible XOR operations required by a Montgomery multiplication, the exponent in the Montgomery factor x^u, i.e. u, should be chosen in the following way:

$$u = \begin{cases} 1, & k = 1, \\ k \text{ or } k-1, & k > 1. \end{cases} \tag{3.3}$$

One can refer to Section 5 in [20] for further details of the above lemma. For convenience, we let our Montgomery factor be x^k as used by Wu in [19].

The following symbols and definitions are crucial to the construction of the new multiplier, and will be frequently used throughout the discussions:

$$\begin{aligned}
&C(x) = A_1(x)B_1(x) = \sum_{i=0}^{m-1} c_i x^i,\ D(x) = A_2(x)B_2(x) = \sum_{i=0}^{m-1} d_i x^i, \\
&Z(x) = (C(x))^2 x^{-k} = \sum_{i=0}^{m-1} z_i x^i,\ Z'(x) = (D(x))^2 x^{-k} = \sum_{i=0}^{m-1} z'_i x^i, \\
&S_1(x) = Z(x)(1+x^{-1}) = \sum_{i=0}^{m-1} r_i x^i,\ S_2(x) = Z'(x)(1+x) = \sum_{i=0}^{m-1} s_i x^i; \\
&U(x) = A_1(x) + A_2(x) = \sum_{i=0}^{\frac{m-1}{2}} u_i x^i,\ V(x) = B_1(x) + B_2(x) = \sum_{i=0}^{\frac{m-1}{2}} v_i x^i, \\
&E(x) = U(x)V(x) = \sum_{i=0}^{m-1} e_i x^i,\ S_3(x) = (E(x))^2 x^{-k} = \sum_{i=0}^{m-1} t_i x^i.
\end{aligned} \tag{3.4}$$

In our algorithm, $C(x)$, $D(x)$, $U(x)$ and $V(x)$ are directly computed from the sub-polynomials of $A(x)$ and $B(x)$. Polynomials $C(x)$ and $D(x)$ are used for representing $S_1(x)+S_2(x)$, while $U(x)$ and $V(x)$ are used for computing $S_3(x)$. It is important to note that no polynomial modular operation is required for computing $A_1(x)B_1(x)$, $A_2(x)B_2(x)$ and $U(x)V(x)$. Also, remember that $c_0 = 0$ and $d_{m-1} = 0$.

From the expression in (3.2) and the definitions in (3.4), our Montgomery multiplication of $A(x)$ and $B(x)$ can be expressed as

$$A(x)B(x)x^{-k} = S_1(x) + S_2(x) + S_3(x).$$

We outline the new Montgomery multiplication algorithm as follows:

Algorithm 2 The new bit-parallel Montgomery multiplication algorithm

Input: $GF(2^m)$ generated by $f(x) = x^m + x^k + 1$. $A(x),\ B(x) \in GF(2^m)$.
Output: $A(x)B(x)x^{-k} \in GF(2^m)$.

1: Transform $A(x)B(x)x^{-k}$ into a sum of $S_1(x)$, $S_2(x)$ and $S_3(x)$ according to (3.2).
2: Combine $C(x)$ with $D(x)$ to partly compute $S_1(x)+S_2(x)$. In the meantime, compute $S_3(x)$ explicitly.
3: Compute $S_1(x)+S_2(x)+S_3(x)$.

In the following sections, firstly, we show how to compute $C(x)$ and $D(x)$ in parallel and then use their coefficients to express the coefficients of $S_1(x)+S_2(x)$. It is important to note that the coefficients of $S_1(x)+S_2(x)$ will not be evaluated explicitly; instead, they will be expressed as XORs of the coefficients of $C(x)$ and $D(x)$, and then be partially computed. Secondly, we discuss our method for computing $S_3(x)$ at the same time with the partial computation of $S_1(x)+S_2(x)$. Finally, we compute $S_1(x)+S_2(x)+S_3(x)$ and provide the overall computation sequence. For simplicity, we may sometimes denote the polynomials by the capital letters only, e.g. $A_1(x)$ may be sometimes denoted by A_1.

3.2 The Complexities for Computing $\mathbf{C(x)}$ and $\mathbf{D(x)}$

In this section, we analyse the time and space complexities for computing both $C(x) = A_1(x)B_1(x)$ and $D(x) = A_2(x)B_2(x)$. Following the rule of polynomial multiplication in $\mathbb{F}_2[x]$, the coefficients of $C(x)$ are expressed as:

$$c_i = \begin{cases} 0, & i = 0, \\ \sum_{j=0}^{i-1} a_{2j} b_{2(i-j)-1}, & 1 \le i \le \frac{m-1}{2}, \\ \sum_{j=i-\frac{m-1}{2}}^{\frac{m-1}{2}} a_{2j} b_{2(i-j)-1}, & \frac{m+1}{2} \le i \le m-1. \end{cases} \tag{3.5}$$

Table 3.I presents the gate numbers and time delays required for computing every coefficient of $C(x)$.

One one hand, a total of $S^{\otimes}(C(x)) = \frac{m^2-1}{4}$ AND gates and $S^{\oplus}(C(x)) = \frac{m^2-4m+3}{4}$ XOR gates are required for computing $C(x)$, as shown in the last row of Table 3.I. On

Table 3.I – The space and time complexities for computing $C(x)$

c_i	#AND	#XOR	Delay
$c_0 = 0$	0	0	-
$c_1 = a_0 b_1$	1	0	T_A
$c_2 = a_0 b_3 + a_2 b_1$	2	1	$T_A + T_X$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$c_{\frac{m-1}{2}} = a_0 b_{m-2} + \cdots + a_{m-3} b_1$	$\frac{m-1}{2}$	$\frac{m-3}{2}$	$T_A + (\lceil \log_2(\frac{m-1}{2}) \rceil) T_X$
$c_{\frac{m+1}{2}} = a_2 b_{m-2} + \cdots + a_{m-1} b_1$	$\frac{m-1}{2}$	$\frac{m-3}{2}$	$T_A + (\lceil \log_2(\frac{m-1}{2}) \rceil) T_X$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$c_{m-1} = a_{m-1} b_{m-2}$	1	0	T_A
Total	$\frac{m^2-1}{4}$	$\frac{m^2-4m+3}{4}$	$T_A + \lceil \log_2 \left(\frac{m-1}{2}\right) \rceil T_X$

the other hand, it takes two steps to compute all the coefficients of $C(x)$ in parallel. First, it takes a delay of $T^{\otimes}(C(x)) = T_A$ to compute all the $a_i b_j$ terms simultaneously. Second, it takes $T^{\oplus}(C(x)) = \lceil \log_2 \left(\frac{m-1}{2}\right) \rceil T_X$ to compute all the coefficients of $C(x)$ by XORing up the corresponding $a_i b_j$ terms in parallel. It is the computation of $c_{\frac{m-1}{2}}$ or $c_{\frac{m+1}{2}}$ that requires the largest delay among all the coefficients of $C(x)$, as shown in the last column of Table 3.I. Thus, the total delay for computing $C(x)$ is

$$\text{Delay}(C(x)) = T^{\otimes}(C(x)) + T^{\oplus}(C(x)) = T_A + \left\lceil \log_2 \left(\frac{m-1}{2}\right) \right\rceil T_X.$$

$D(x)$ is similarly computed during the computation of $C(x)$. The time and space complexities for computing both $C(x)$ and $D(x)$ are:

$$\begin{aligned} &S^{\otimes}(C(x)) \text{ or } S^{\otimes}(D(x)) \text{: } \tfrac{m^2-1}{4}, \\ &S^{\oplus}(C(x)) \text{ or } S^{\oplus}(D(x)) \text{: } \tfrac{m^2-4m+3}{4}, \\ &\text{Delay}(C(x) \text{ and } D(x)) \text{: } T_A + \lceil \log_2 \left(\tfrac{m-1}{2}\right) \rceil T_X. \end{aligned} \tag{3.6}$$

3.3 The Representation of $\mathbf{S_1(x)+S_2(x)}$ by $\mathbf{C(x)}$ and $\mathbf{D(x)}$

It is clear from (3.4) that the expressions of $S_1(x)$ and $S_2(x)$ are based on the Montgomery squarings of $C(x)$ and $D(x)$, respectively. In order to use the appropriate Montgomery squaring formulae for $C(x)$ and $D(x)$, we need to consider four cases regarding the generating polynomial $f(x) = x^m + x^k + 1$, given that m is odd:

1. k is odd, $1 \leq k \leq \frac{m-3}{2}$,
2. k is odd, $k = \frac{m-1}{2}$,
3. k is even, $1 \leq k \leq \frac{m-3}{2}$,
4. k is even, $k = \frac{m-1}{2}$.

For simplicity, we only present a detailed discussion about the new multiplier in the first case when m is odd, k is odd, and $1 \leq k \leq \frac{m-3}{2}$.

Case 1: It can be verified from (3.4) that $Z(x)$ is the Montgomery squaring of $C(x)$, hence $Z(x)$ has the following coefficient expression according to (2.2):

$$z_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1, \\ c_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2, \\ c_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1, \\ c_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2, \\ c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{3.7}$$

Since $x^m + x^k + 1 = 0$, it follows that $x^{-1} = x^{m-1} + x^{k-1}$. Then, $S_1(x)$ can be transformed as:

$$\begin{aligned}
S_1(x) &= (1+x^{-1})Z(x) \bmod f(x) \\
&= (1+x^{-1}) \sum_{i=0}^{m-1} z_i x^i \bmod f(x) \\
&= (z_{k-1} + z_k + z_0)x^{k-1} + (z_{m-1} + z_0)x^{m-1} + \sum_{\substack{0 \le i \le m-2 \\ i \ne k-1}} (z_i + z_{i+1})x^i \\
&= \sum_{i=0}^{m-1} r_i x^i.
\end{aligned}$$

Substituting (3.7) into the above expression, we get the coefficient expression of $S_1(x)$:

$$r_i = \begin{cases}
c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}}, & i = 0, 2, \cdots, k-1, \\
c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+i+1}{2}}, & i = k+1, k+3, \cdots, m-k-2, \\
c_{\frac{k-m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+i+1}{2}}, & i = m-k, m-k+2, \cdots, m-3, \\
c_{\frac{k-1}{2}} + c_{\frac{m+k}{2}}, & i = m-1, \\
c_{\frac{k+i}{2}} + c_{\frac{i+1}{2}} + c_{\frac{m+k+i+1}{2}}, & i = 1, 3, \cdots, k-2, \\
c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{m+k+i+1}{2}}, & i = k, k+2, \cdots, m-k-3, \\
c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k-m+i+1}{2}}, & i = m-k-1, m-k+1, \cdots, m-2.
\end{cases} \tag{3.8}$$

We can similarly obtain the coefficient expression of $S_2(x)$. Since $Z'(x)$ is the Montgomery squaring of $D(x)$, we have:

$$z'_i = \begin{cases} d_{\frac{i}{2}} + d_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1, \\ d_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2, \\ d_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1, \\ d_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2, \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{3.9}$$

$S_2(x)$ can be transformed as:

$$\begin{aligned} S_2(x) &= (1+x)Z(x) \bmod f(x) \\ &= (1+x)\sum_{i=0}^{m-1} z'_i x^i \bmod f(x) \\ &= (z'_k + z'_{k-1} + z'_{m-1})x^k + (z'_{m-1} + z'_0) + \sum_{\substack{1 \le i \le m-1 \\ i \ne k}} (z'_i + z'_{i-1})x^i \\ &= \sum_{i=0}^{m-1} s_i x^i. \end{aligned}$$

Substituting (3.9) into the above expression, we get the coefficient expression of $S_2(x)$:

$$s_i = \begin{cases} d_{\frac{i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}}, & i = 0, 2, \cdots, k-1, \\ d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = k+1, k+3, \cdots, m-k-2, \\ d_{\frac{k-m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k, m-k+2, \cdots, m-1, \\ d_{\frac{k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = 1, 3, \cdots, k-2, \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{m+k+i-1}{2}}, & i = k, k+2, \cdots, m-k-1, \\ d_{\frac{k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k-m+i-1}{2}}, & i = m-k+1, m-k+3, \cdots, m-2. \end{cases} \tag{3.10}$$

Based on (3.8) and (3.10), every coefficient of $S_1(x)+S_2(x)$ can be expressed as a XOR of certain coefficients of $C(x)$ and $D(x)$:

$$r_i+s_i=\begin{cases} c_{\frac{i}{2}}+c_{\frac{m+k+i}{2}}+c_{\frac{k+i+1}{2}}+d_{\frac{i}{2}} & \\ \quad +d_{\frac{m+k+i}{2}}+d_{\frac{k+i-1}{2}}, & i=0,2,\cdots,k-1, \\ c_{\frac{m+k+i}{2}}+c_{\frac{k+i+1}{2}}+c_{\frac{m+i+1}{2}} & \\ \quad +d_{\frac{m+k+i}{2}}+d_{\frac{k+i-1}{2}}+d_{\frac{m+i-1}{2}}, & i=k+1,k+3,\cdots,m-k-2, \\ c_{\frac{k-m+i}{2}}+c_{\frac{k+i+1}{2}}+c_{\frac{m+i+1}{2}} & \\ \quad +d_{\frac{k-m+i}{2}}+d_{\frac{k+i-1}{2}}+d_{\frac{m+i-1}{2}}, & i=m-k,m-k+2,\cdots,m-3, \\ c_{\frac{k-1}{2}}+c_{\frac{m+k}{2}}+d_{\frac{k-1}{2}}+d_{\frac{k+m-2}{2}}, & i=m-1, \\ c_{\frac{k+i}{2}}+c_{\frac{i+1}{2}}+c_{\frac{m+k+i+1}{2}} & \\ \quad +d_{\frac{k+i}{2}}+d_{\frac{i-1}{2}}+d_{\frac{m+k+i-1}{2}}, & i=1,3,\cdots,k-2, \\ c_{\frac{k+i}{2}}+c_{\frac{m+i}{2}}+c_{\frac{m+k+i+1}{2}} & \\ \quad +d_{\frac{k+i}{2}}+d_{\frac{m+i}{2}}+d_{\frac{m+k+i-1}{2}}, & i=k,k+2,\cdots,m-k-3, \\ c_{\frac{m-1}{2}}+c_{\frac{2m-k-1}{2}}+d_{\frac{m-1}{2}}+d_{\frac{2m-k-1}{2}}, & i=m-k-1, \\ c_{\frac{k+i}{2}}+c_{\frac{m+i}{2}}+c_{\frac{k-m+i+1}{2}} & \\ \quad +d_{\frac{k+i}{2}}+d_{\frac{m+i}{2}}+d_{\frac{k-m+i-1}{2}}, & i=m-k+1,m-k+3,\cdots,m-2. \end{cases}$$

When $k=1$, the above coefficient expression holds with the fifth case, i.e. that for $i=1,3,\cdots k-2$, being neglected.

We can just follow the same line as above to discuss the other three cases regarding the generating polynomial $f(x)=x^m+x^k+1$. The coefficient expressions of $S_1(x)+S_2(x)$ in the other three cases are presented in Appendix I.

3.4 The Computation of $\mathbf{S_3(x)}$

$S_3(x)$ is computed at the same time with the partial computation of $S_1(x)+S_2(x)$. In this section, we provide an explicit analysis of the space and time complexities for

computing $S_3(x) = E^2(x)x^{-k}$ on $GF(2^m)$ where m is odd, k is odd, and $1 \le k \le \frac{m-3}{2}$.

According to (3.4) and the rule of polynomial multiplication in $\mathbb{F}_2[x]$, the coefficient expression of $E(x)$ is:

$$e_i = \begin{cases} \sum_{j=0}^{i} u_j v_{i-j}, & 0 \le i \le \frac{m-1}{2}, \\ \sum_{j=i-\frac{m-1}{2}}^{\frac{m-1}{2}} u_j v_{i-j}, & \frac{m+1}{2} \le i \le m-1. \end{cases} \tag{3.11}$$

$S_3(x)$ is the Montgomery squaring of $E(x)$, with its coefficients expressed as:

$$t_i = \begin{cases} e_{\frac{i}{2}} + e_{\frac{m+k+i}{2}}, & i = 0, 2, \cdots, k-1, \\ e_{\frac{m+k+i}{2}}, & i = k+1, k+3, \cdots, m-k-2, \\ e_{\frac{k-m+i}{2}}, & i = m-k, m-k+2 \cdots, m-1, \\ e_{\frac{k+i}{2}}, & i = 1, 3, \cdots, k-2, \\ e_{\frac{k+i}{2}} + e_{\frac{m+i}{2}}, & i = k, k+2, \cdots, m-2. \end{cases} \tag{3.12}$$

3.4.1 A New Method for Computing $S_3(x)$ Using a XOR-Gate-Saving Strategy

We take a new approach to compute $S_3(x)$ without computing $E(x)$ explicitly. Without loss of generality, let us consider $3 \le k \le \frac{m-3}{2}$ first. By substituting (3.11) into (3.12), every coefficient of $S_3(x)$ is expressed as a XOR of certain $u_i v_j$ terms. In Table 3.II, in the two columns under the title "#XOR", the number of XOR operations related to every coefficient of $S_3(x)$ are listed.

Our new method takes three steps to compute $S_3(x)$: (1) to compute $\{u_i \mid 0 \le i \le \frac{m-1}{2}\}$ and $\{v_i \mid 0 \le i \le \frac{m-1}{2}\}$ at the same time, (2) to compute $\{u_i v_j \mid 0 \le i, j \le \frac{m-1}{2}\}$ all at once, and (3) to compute $\{t_i \mid 0 \le i \le m-1\}$ in parallel by XORing together the corresponding terms in $\{u_i v_j \mid 0 \le i, j \le \frac{m-1}{2}\}$, as indicated by the t_i expressions in Table 3.II.

A number of redundant XOR gates are discovered in the step (3). For example, look at the parallel computation of t_0 and t_k. Since $t_0 = e_0 + e_{\frac{m+k}{2}}$ and $t_k = e_k + e_{\frac{m+k}{2}}$, a term $e_{\frac{m+k}{2}}$ exists in both the expressions of t_0 and t_k. As expressed in (3.11), $e_{\frac{m+k}{2}}$ is

Table 3.II – The initial XOR numbers between $u_i v_j$ terms for each of the coefficients of $S_3(x)$ $(k \geq 3)$

i	t_i	#XOR	i	t_i	#XOR
0	$u_0v_0+\sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$	$\frac{m-k}{2}$	1	$\sum_{j=0}^{\frac{k+1}{2}} u_j v_{\frac{k+1}{2}-j}$	$\frac{k+1}{2}$
2	$u_0v_1+u_1v_0+\sum_{j=\frac{k+3}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}+1-j}$	$\frac{m-k}{2}$	3	$\sum_{j=0}^{\frac{k+3}{2}} u_j v_{\frac{k+1}{2}+1-j}$	$\frac{k+3}{2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k-1$	$\sum_{j=0}^{\frac{k-1}{2}} u_j v_{\frac{k-1}{2}-j}+\sum_{j=k}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}+k-j}$	$\frac{m-k}{2}$	$k-2$	$\sum_{j=0}^{k-1} u_j v_{k-1-j}$	$k-1$
$k+1$	$\sum_{j=k+1}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}+k+1-j}$	$\frac{m-1}{2}-k-1$	k	$\sum_{j=0}^{k} u_j v_{k-j}+\sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$	$\frac{m+k}{2}$
$k+3$	$\sum_{j=k+2}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}+k+2-j}$	$\frac{m-1}{2}-k-2$	$k+2$	$\sum_{j=0}^{k+1} u_j v_{k+1-j}+\sum_{j=\frac{k+3}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}+1-j}$	$\frac{m+k}{2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-k-2$	$u_{\frac{m-1}{2}} v_{\frac{m-1}{2}}$	0	$m-k-1$	$\sum_{j=0}^{\frac{m-1}{2}} u_j v_{\frac{m-1}{2}-j}+\sum_{j=\frac{m-k}{2}}^{\frac{m-1}{2}} u_j v_{m-\frac{k+1}{2}-j}$	$\frac{m+k}{2}$
$m-k$	u_0v_0	0	$m-k+1$	$\sum_{j=1}^{\frac{m-1}{2}} u_j v_{\frac{m+1}{2}-j}+\sum_{j=\frac{m-k}{2}+1}^{\frac{m-1}{2}} u_j v_{m-\frac{k-1}{2}-j}$	$\frac{m+k}{2}-2$
$m-k+2$	$u_0v_1+u_1v_0$	1	$m-k+3$	$\sum_{j=2}^{\frac{m-1}{2}} u_j v_{\frac{m+3}{2}-j}+\sum_{j=\frac{m-k}{2}+2}^{\frac{m-1}{2}} u_j v_{m-\frac{k-3}{2}-j}$	$\frac{m+k}{2}-4$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-1$	$\sum_{j=0}^{\frac{k-1}{2}} u_j v_{\frac{k-1}{2}-j}$	$\frac{k-1}{2}$	$m-2$	$\sum_{j=\frac{k-1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-1-j}+u_{\frac{m-1}{2}} v_{\frac{m-1}{2}}$	$\frac{m-k}{2}+1$

equal to $\sum_{j=\frac{k+1}{2}}^{\frac{m-1}{2}} u_j v_{\frac{m+k}{2}-j}$ which has $\frac{m-k}{2}$ addends. Without loss of generality, suppose the number of addends $\frac{m-k}{2}$ is odd, then the first step of bit-parallel XOR operation for computing t_0 and t_k is expressed as follows:

$$\begin{aligned} t_0 =& \left[u_{\frac{k+1}{2}} v_{\frac{m-1}{2}} + u_{\frac{k+3}{2}} v_{\frac{m-3}{2}}\right] + \left[u_{\frac{k+5}{2}} v_{\frac{m-5}{2}} + u_{\frac{k+7}{2}} v_{\frac{m-7}{2}}\right] \\ &+ \cdots + \left[u_{\frac{m-5}{2}} v_{\frac{k+5}{2}} + u_{\frac{m-3}{2}} v_{\frac{k+3}{2}}\right] + \left[u_{\frac{m-1}{2}} v_{\frac{k+1}{2}} + u_0 v_0\right], \end{aligned} \tag{3.13}$$

and

$$\begin{aligned} t_k =& \left[u_{\frac{k+1}{2}} v_{\frac{m-1}{2}} + u_{\frac{k+3}{2}} v_{\frac{m-3}{2}}\right] + \left[u_{\frac{k+5}{2}} v_{\frac{m-5}{2}} + u_{\frac{k+7}{2}} v_{\frac{m-7}{2}}\right] \\ &+ \cdots + \left[u_{\frac{m-5}{2}} v_{\frac{k+5}{2}} + u_{\frac{m-3}{2}} v_{\frac{k+3}{2}}\right] + \left[u_{\frac{m-1}{2}} v_{\frac{k+1}{2}} + u_0 v_k\right] \\ &+ \left[u_1 v_{k-1} + u_2 v_{k-2}\right] + \cdots + \left[u_{k-1} v_1 + u_k v_0\right]. \end{aligned} \tag{3.14}$$

Every pair of the above square brackets corresponds to a XOR gate on Layer 0 of the corresponding XOR tree. All the XORs in the above brackets are evaluated simultaneously

in a delay of one T_X.

The processes for computing t_0 and t_k are illustrated in Figure 3.1 and Figure 3.2, respectively. The coefficient t_0 is computed with the binary XOR tree (a). The $u_i v_j$ terms of (3.13) correspond to the circles on Layer 0 of (a), while the XOR outputs in the square brackets of (3.13) correspond to the circles on Layer 1 of (a). In the tree (a), all the bits on the same layer are XORed pairwise to obtain the bits on the next layer, until Layer $\lceil \log_2(\frac{m-k+2}{2}) \rceil$ is reached where t_0 is evaluated. Meanwhile, t_k is computed in the same way as illustrated in Figure 3.2.

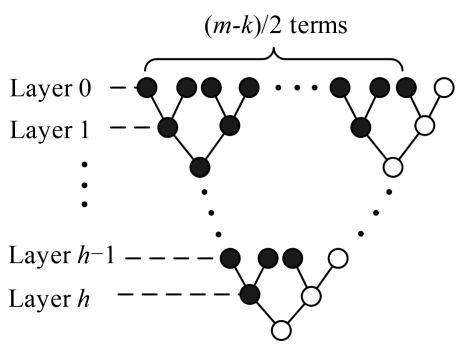


Figure 3.1 – The binary XOR tree (a) related to t_0

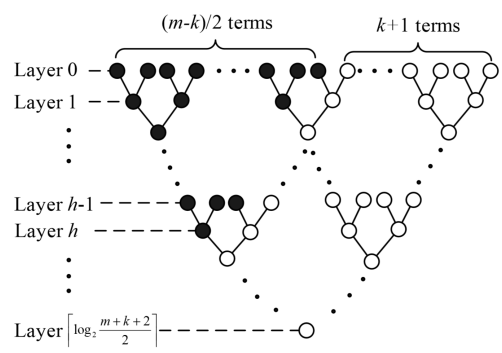


Figure 3.2 – The binary XOR tree (b) related to t_k

In the above two binary XOR trees, we use black circles to represent the overlapping bits resulting from the XOR expression of $e_{\frac{m+k}{2}}$, i.e. the common term in $t_0 = e_0 + e_{\frac{m+k}{2}}$ and $t_k = e_k + e_{\frac{m+k}{2}}$. It is clear that the two binary XOR trees have the same inputs for their respective first $\lfloor \frac{m-k}{4} \rfloor$ XOR gates on Layer 0. In other words, the first $\lfloor \frac{m-k}{4} \rfloor$ XOR gates on Layer 0 of the tree (b) are redundant. These redundant XOR gates can thus be eliminated. Consequently, we eliminate the higher-layer XOR gates in the tree (b) that are fully based on the outputs of the Layer-0 redundant XOR gate. As a result, $\lfloor \frac{m-k}{2^{\ell+2}} \rfloor$ XOR gates are eliminated on Layer ℓ of the tree (b), where $0 \le \ell \le h := \lfloor \log_2(m-k) \rfloor - 2$. Thus, a total of

$$\left\lfloor \frac{m-k}{4} \right\rfloor + \left\lfloor \frac{m-k}{8} \right\rfloor + \cdots + \left\lfloor \frac{m-k}{2^{\lfloor \log_2(m-k) \rfloor}} \right\rfloor$$

redundant XOR gates can be eliminated from the sub-circuit for t_k.

Look at an example based on $f(x) = x^{17} + x^3 + 1 \in \mathbb{F}_2[x]$ with $m = 17$ and $k = 3$. It can be verified from (3.11) and (3.12) that $t_0 = e_0 + e_{10}$ and $t_3 = e_3 + e_{10}$. Both the expressions of t_0 and t_3 contain e_{10}:

$$e_{10} = u_2v_8 + u_3v_7 + u_4v_6 + u_5v_5 + u_6v_4 + u_7v_3 + u_8v_2,$$

which is a XOR of seven bits. Since $\lfloor \frac{m-k}{4} \rfloor + \lfloor \frac{m-k}{8} \rfloor$=4, four XOR gates can be eliminated from the sub-circuit for t_3.

Figure 3.3 gives a full detail of our bit-reusing trick to the above example. There

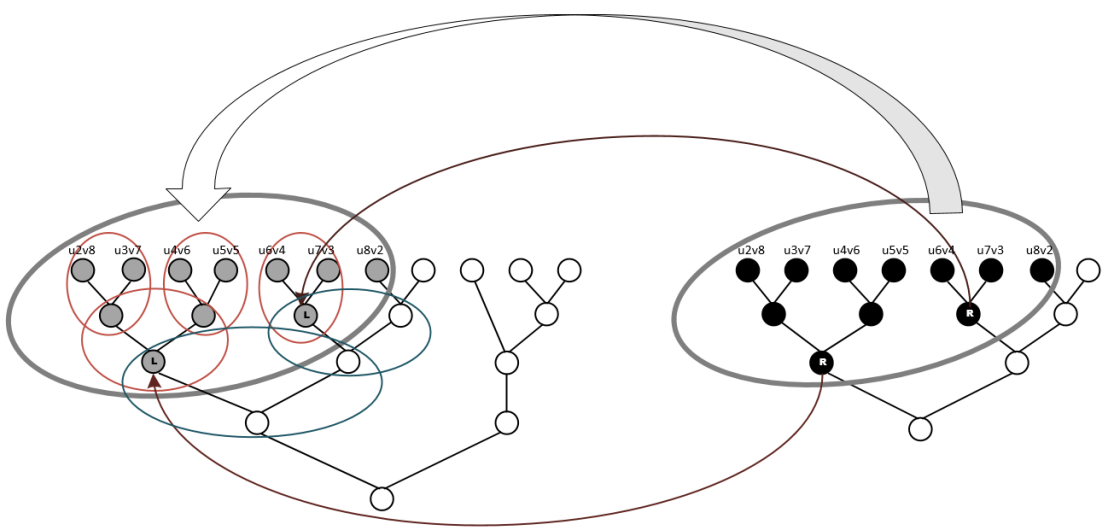


Figure 3.3 – Illustration of our bit-reusing trick between the binary XOR trees

are two binary XOR trees in this figure. The tree on the left computes t_3, while the tree on the right computes t_0. Each circle represents a bit. The seven bits u_2v_8, u_3v_7, u_4v_6, u_5v_5, u_6v_4, u_7v_3, u_8v_2 are depicted as seven dark circles, respectively, from left to right on Layer 0 of both of the trees. We have eleven dark circles encircled in both of the trees to indicate the overlapping bits. Surrounded in red circles, the four XOR gates are eliminated from the tree computing t_3. The two bits marked with "**R**", representing the values of $u_6v_4 + u_7v_3$ and $u_2v_8 + u_3v_7 + u_4v_6 + u_5v_5$, respectively, are transmitted from the right tree to the left tree. Becoming the inputs for the two XOR gates in blue circles,

the two bits marked with "**R**" take over the positions of the two bits marked with "**L**", respectively, which ensures a proper calculation of t_3. Generally, a bit can be transmitted by being rewired out as an input for another gate, at no additional cost of logic gates.

It can be generalized from the above discussion that if two binary XOR trees have i input bits in common at Layer 0, then a total of

$$\left\lfloor \frac{i}{2} \right\rfloor + \left\lfloor \frac{i}{4} \right\rfloor + \cdots + \left\lfloor \frac{i}{2^{\lfloor \log_2 i \rfloor}} \right\rfloor \tag{3.15}$$

XOR gates can be saved from one of the trees. Equation (3.15) can be further simplified by using Proposition 3.4.2, which uses the notion of Hamming weight.

Definition 3.4.1. *The Hamming weight of an integer is the number of "1"s in the binary representation of the integer. For example, the integer 107 has its binary representation*

$$(107)_2 = (1101011),$$

thus 107 has a Hamming weight of 5.

Proposition 3.4.2. *Let $W(i)$ be the Hamming weight of an integer i. Suppose $i = 2^{n_1} + 2^{n_2} + \cdots + 2^{n_t}$, where $n_1 > n_2 > \cdots > n_t \geq 0$. Note that i can always be written in this form and $\lfloor \log_2 i \rfloor = n_1$. Then*

$$\left\lfloor \frac{i}{2} \right\rfloor + \left\lfloor \frac{i}{4} \right\rfloor + \cdots + \left\lfloor \frac{i}{2^{n_1}} \right\rfloor = i - W(i)$$

Proof. We can write:

$$
\begin{aligned}
\left\lfloor \frac{i}{2} \right\rfloor &= 2^{n_1-1} + 2^{n_2-1} + \cdots + 2^{n_t-1}, \\
\left\lfloor \frac{i}{4} \right\rfloor &= 2^{n_1-2} + 2^{n_2-2} + \cdots + 2^{n_t-2}, \\
&\vdots \\
\left\lfloor \frac{i}{2^{n_t}} \right\rfloor &= 2^{n_1-n_t} + 2^{n_2-n_t} + \cdots + 2^{n_{t-1}-n_t} + 2^0, \\
&\vdots \\
\left\lfloor \frac{i}{2^{n_1}} \right\rfloor &= 1.
\end{aligned}
$$

The sum of the expressions on the left of the above equality signs is:

$$
\left\lfloor \frac{i}{2} \right\rfloor + \left\lfloor \frac{i}{4} \right\rfloor + \cdots + \left\lfloor \frac{i}{2^{n_1}} \right\rfloor, \tag{3.16}
$$

and the sum of the expressions on the right of the equality signs is:

$$
\begin{aligned}
&(2^{n_1-1} + 2^{n_1-2} + \cdots + 1) + (2^{n_2-1} + 2^{n_2-2} + \cdots + 1) \\
&\quad + \cdots + (2^{n_t-1} + 2^{n_t-2} + \cdots + 1) \\
&= (2^{n_1} - 1) + (2^{n_2} - 1) + \cdots + (2^{n_t} - 1) \\
&= (2^{n_1} + 2^{n_2} + \cdots + 2^{n_t}) - \underbrace{(1 + 1 + \cdots + 1)}_{\text{The number of “1”s is W(i)}},
\end{aligned} \tag{3.17}
$$

which is $i - W(i)$ by definition. Thus, combining (3.16) and (3.17) we get

$$
\left\lfloor \frac{i}{2} \right\rfloor + \left\lfloor \frac{i}{4} \right\rfloor + \cdots + \left\lfloor \frac{i}{2^{n_1}} \right\rfloor = i - W(i).
$$

□

Suppose two binary XOR trees have i bits in common on Layer 0, according to Proposition 3.4.2, there are $i - W(i)$ XOR gates that can be eliminated from one of the binary XOR trees by applying our bit-reusing trick.

The coefficients containing redundant XOR operations are gathered in Table 3.III. Given a subscript i in the first column, there is a corresponding subscript j in the second column, which implies that t_i and t_j have common addends in their expressions. The third column shows the amount of redundant XOR gates to be eliminated from the sub-circuit for t_i. For example, the overlap between the binary XOR trees for t_k and t_0 leads to the elimination of $\frac{m-k}{2} - W(\frac{m-k}{2})$ XOR gates. It should be noted that our bit-reusing trick does not change the depth of circuit. After the removal of redundant XOR gates, the delays for computing $\{t_i \mid i = 0,\ 2, \cdots,\ k-1,\ k,\ k+2, \cdots,\ m-2\}$ are still determined by the corresponding expressions in Table 3.II, as reflected by the last column in Table 3.III.

Table 3.III – The $S_3(x)$ coefficients containing redundant XOR operations $(k \geq 3)$

i	j	#XOR saved	Delay(t_i)
0	$m-k$	$1-W(1)$	$\lceil \log_2(\frac{m-k}{2}+1) \rceil T_X$
2	$m-k+2$	$2-W(2)$	$\lceil \log_2(\frac{m-k}{2}+1) \rceil T_X$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$k-1$	$m-1$	$\frac{k+1}{2} - W(\frac{k+1}{2})$	$\lceil \log_2(\frac{m-k}{2}+1) \rceil T_X$
k	0	$\frac{m-k}{2} - W(\frac{m-k}{2})$	$\lceil \log_2(\frac{m+k}{2}+1) \rceil T_X$
$k+2$	2	$\frac{m-k}{2} - 1 - W(\frac{m-k}{2} - 1)$	$\lceil \log_2(\frac{m+k}{2}+1) \rceil T_X$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-k-1$	$m-2k-1$	$\frac{k+1}{2} - W(\frac{k+1}{2})$	$\lceil \log_2(\frac{m+k}{2}+1) \rceil T_X$
$m-k+1$	$m-2k+1$	$\frac{k-1}{2} - W(\frac{k-1}{2})$	$\lceil \log_2(\frac{m+k}{2}-1) \rceil T_X$
$m-k+3$	$m-2k+3$	$\frac{k-3}{2} - W(\frac{k-3}{2})$	$\lceil \log_2(\frac{m+k}{2}-3) \rceil T_X$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-2$	$m-k-2$	$1-W(1)$	$\lceil \log_2(\frac{m-k}{2}+2) \rceil T_X$

Based on Table 3.II and Table 3.III, an explicit analysis of the space and time complexities for computing $S_3(x)$ is given as follows.

3.4.2 Summary of the Space Complexity Analysis of $\mathbf{S_3(x)}$

On one hand, the number of AND gates used for calculating $S_3(x)$ is equal to the cardinality of $\{u_i v_j \mid 0 \le i, j \le \frac{m-1}{2}\}$:

$$\begin{aligned} S^{\otimes}(S_3) &= \left|\{u_i v_j \mid 0 \le i, j \le \frac{m-1}{2}\}\right| \\ &= \left|\{i \mid 0 \le i \le \frac{m-1}{2}\}\right| \cdot \left|\{j \mid 0 \le j \le \frac{m-1}{2}\}\right| \\ &= \frac{(m+1)^2}{4}. \end{aligned}$$

On the other hand, when $3 \le k \le \frac{m-3}{2}$, the XOR gates for calculating $S_3(x)$ are divided into two parts: (1) the $m-1$ XOR gates for computing $\{u_i \mid 0 \le i \le \frac{m-3}{2}\}$ and $\{v_j \mid 1 \le j \le \frac{m-1}{2}\}$, (2) the total XOR gates shown in Table 3.II in the column "#XOR" excluding the redundant XOR gates shown in Table 3.III in the column "#XOR saved". Therefore, the number of XOR gates used for computing $S_3(x)$ is:

$$\begin{aligned} S^{\oplus}(S_3) = (m-1) &+ \left(\frac{k+1}{2}\right)\left(\frac{m-k}{2}\right) + \sum_{i=1}^{\frac{m-1}{2}-k}\left(\frac{m-1}{2} - k - i\right) + \sum_{i=0}^{\frac{k-1}{2}} i \\ &+ \sum_{i=\frac{k+1}{2}}^{k-1} i + \left(\frac{m-2k+1}{2}\right)\left(\frac{m+k}{2}\right) + \sum_{i=1}^{\frac{k-1}{2}}\left(\frac{m+k}{2} - 2i\right) \\ &- \left(\sum_{i=1}^{\frac{k+1}{2}}(i - W(i)) + \sum_{i=1}^{\frac{m-k}{2}}(i - W(i))\right) \\ = \quad & \frac{m^2+2m-3}{4} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i). \end{aligned} \tag{3.18}$$

By neglecting the part for $t_1,\ t_3, \cdots,\ t_{k-2}$ in Table 3.II, as well as that for t_{m-k+1}, $t_{m-k+3}, \cdots, t_{m-2}$ in Table 3.III, it can be similarly verified that when $k = 1$, $S^{\oplus}(S_3) = \frac{(m+1)^2}{4} + \sum_{i=1}^{\frac{m-1}{2}} W(i)$. Thus, when $1 \le k \le \frac{m-3}{2}$, $S^{\oplus}(S_3) = \frac{m^2+2m-3}{4} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i)$.

Thus, with respect to the number of AND gates and the number of XOR gates, the

space complexity for computing $S_3(x)$ is:

$$\begin{aligned} &S^{\otimes}(S_3) : \tfrac{(m+1)^2}{4}, \\ &S^{\oplus}(S_3) : \tfrac{m^2+2m-3}{4} + \textstyle\sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i). \end{aligned} \tag{3.19}$$

3.4.3 Summary of the Time Complexity Analysis of $\mathbf{S_3(x)}$

The parallel computation of all the coefficients of $S_3(x)$ takes place in three steps. Firstly, it takes one T_X to get $\{u_i \mid 0 \le i \le \frac{m-1}{2}\}$ and $\{v_i \mid 0 \le i \le \frac{m-1}{2}\}$. Secondly, it takes one T_A to get $\{u_i v_j \mid 0 \le i, j \le \frac{m-1}{2}\}$. Finally, all the coefficients of $S_3(x)$ are computed in parallel by XORing up the corresponding $u_i v_j$ terms. Among all the coefficients of $S_3(x)$ shown in Table 3.II, $\{t_i \mid \ k \le i \le m-k-1,\ i \text{ is odd}\}$ have the XOR expressions with the most addends, i.e. $\frac{m+k+2}{2}$ bits to XOR up. Hence, a delay of $\lceil \log_2 \frac{m+k+2}{2} \rceil T_X$ is incurred by the third step. In summary, $T^{\otimes}(S_3(x)) = T_A$, $T^{\oplus}(S_3(x)) = \left(1 + \lceil \log_2 \frac{m+k+2}{2} \rceil\right) T_X$. The total delay for computing $S_3(x)$ is $T^{\otimes}(S_3(x)) + T^{\oplus}(S_3(x))$, i.e.

$$\text{Delay}(S_3(x)) : T_A + \left(1 + \lceil \log_2 \tfrac{m+k+2}{2} \rceil\right) T_X. \tag{3.20}$$

3.5 The Full Computation Sequence and the Overall Complexity Analysis

In order to complete the Montgomery multiplication $A(x)B(x)x^{-k}$, it is necessary to combine the expressions of $S_3(x)$ and $S_1(x) + S_2(x)$. In this section, we give a detailed discussion about the full computation sequence of $A(x)B(x)x^{-k}$ in the case when m is odd, k is odd and $k \le \frac{m-3}{2}$.

The overall computation sequence for the Montgomery multiplication

$$A(x)B(x)x^{-k} = S_1(x) + S_2(x) + S_3(x)$$

is shown in Table 3.IV. The whole process of our algorithm consists of two parts, with each part having three stages. Let us assume that Part I is for $S_3(x)$, and Part II is for $S_1(x) + S_2(x)$. Part I and Part II are processed in parallel from Stage I to Stage II, then,

the two parts are combined together in Stage III for completing the remainder of the XOR operations.

Table 3.IV – The computation sequence in the case when m, k is odd, $k \leq \frac{m-3}{2}$

Parts╲ Stages	I	II	III
I	$\underbrace{U = A_1 + A_2,\ V = B_1 + B_2}_{\text{Delay: } 1T_X}$	$\underbrace{S_3 = (UV)^2 x^{-k}}_{T_A + \lceil \log_2 \frac{m+k+2}{2} \rceil T_X}$	$\underbrace{[S_1 + S_2] + S_3}_{2T_X}$
II	$\underbrace{C = A_1 B_1,\ D = A_2 B_2}_{\text{Delay: } T_A + \lceil \log_2 \frac{m-1}{2} \rceil T_X}$	$\underbrace{[S_1 + S_2]}_{1T_X}$	

It should be noted that the delay for computing $S_3(x)$ is at least one T_X longer than that for computing both $C(x)$ and $D(x)$. Look at Table 3.IV. In Part I, it takes one T_X to compute $U(x)$ and $V(x)$ in Stage I, and then $T_A + \lceil \log_2 \frac{m+k+2}{2} \rceil T_X$ to compute $S_3(x) = (U(x)V(x))^2 x^{-k}$ in Stage II. Meanwhile, in Part II, it takes $T_A + \lceil \log_2 \frac{m-1}{2} \rceil T_X$ to compute $C(x)$ and $D(x)$ in Stage I. Hence, the length of Stage II of Part II is the difference between the delay for computing $S_3(x)$ and that for both $C(x)$ and $D(x)$:

$$\begin{aligned}
\text{Length(Stage II of Part II)} &= \left(T_A + T_X + \left\lceil \log_2 \frac{m+k+2}{2} \right\rceil T_X \right) - \left(T_A + \left\lceil \log_2 \frac{m-1}{2} \right\rceil T_X \right) \\
&= T_X + \left\lceil \log_2 \frac{m+k+2}{2} \right\rceil T_X - \left\lceil \log_2 \frac{m-1}{2} \right\rceil T_X \\
&\geq T_X,
\end{aligned}$$

which means Stage II of Part II is an interval at least as long as one T_X.

Therefore, in Stage II of Part II, we can take one T_X to perform a single step of parallel XOR operation between the coefficients of $S_1(x)$ and $S_2(x)$. Such operation is expressed by $[S_1 + S_2]$ in Stage II of Part II, which means to do all the XOR operations

inside the following square brackets simultaneously:

$$
r_i + s_i = \begin{cases}
[c_{\frac{i}{2}} + c_{\frac{m+k+i}{2}}] + [c_{\frac{k+i+1}{2}} + d_{\frac{i}{2}}] & \\
\quad + [d_{\frac{m+k+i}{2}} + d_{\frac{k+i-1}{2}}], & i = 0, 2, \cdots, k-1, \\
[c_{\frac{m+k+i}{2}} + c_{\frac{k+i+1}{2}}] + [c_{\frac{m+i+1}{2}} + d_{\frac{m+k+i}{2}}] & \\
\quad + [d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}], & i = k+1, k+3, \cdots, m-k-2, \\
[c_{\frac{k-m+i}{2}} + c_{\frac{k+i+1}{2}}] + [c_{\frac{m+i+1}{2}} + d_{\frac{k-m+i}{2}}] & \\
\quad + [d_{\frac{k+i-1}{2}} + d_{\frac{m+i-1}{2}}], & i = m-k, m-k+2, \cdots, m-3, \\
[c_{\frac{k-1}{2}} + c_{\frac{m+k}{2}}] + [d_{\frac{k-1}{2}} + d_{\frac{k+m-2}{2}}], & i = m-1, \\
[c_{\frac{k+i}{2}} + c_{\frac{i+1}{2}}] + [c_{\frac{m+k+i+1}{2}} + d_{\frac{k+i}{2}}] & \\
\quad + [d_{\frac{i-1}{2}} + d_{\frac{m+k+i-1}{2}}], & i = 1, 3, \cdots, k-2, \\
[c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}] + [c_{\frac{m+k+i+1}{2}} + d_{\frac{k+i}{2}}] & \\
\quad + [d_{\frac{m+i}{2}} + d_{\frac{m+k+i-1}{2}}], & i = k, k+2, \cdots, m-k-3, \\
[c_{\frac{m-1}{2}} + c_{\frac{2m-k-1}{2}}] + [d_{\frac{m-1}{2}} + d_{\frac{2m-k-1}{2}}], & i = m-k-1, \\
[c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}}] + [c_{\frac{k-m+i+1}{2}} + d_{\frac{k+i}{2}}] & \\
\quad + [d_{\frac{m+i}{2}} + d_{\frac{k-m+i-1}{2}}], & i = m-k+1, m-k+3, \cdots, m-2.
\end{cases} \tag{3.21}
$$

After such single step of parallel XOR operation in Stage II of Part II, it is evident from the above expression that every coefficient of $S_1(x) + S_2(x)$ becomes a XOR of at most three bits. Then, at the start of Stage III, with the addition of the explicitly computed $S_3(x)$, every coefficient of $S_1(x) + S_2(x) + S_3(x)$ becomes a XOR of at most four bits. Thus, it takes two T_X in Stage III to do the remainder of the parallel XOR operations for computing $A(x)B(x)x^{-k}$.

Therefore, with reference to (3.20), the total time complexity of our multiplier is:

$$\begin{aligned}
\text{Delay} &= \text{Length(Stage I} \bigcup \text{Stage II)} + \text{Length(Stage III)} \\
&= \text{Delay}(S_3(x)) + 2T_X \\
&= T_A + \left(1 + \left\lceil \log_2 \frac{m+k+2}{2} \right\rceil\right) T_X + 2T_X \\
&= T_A + (2 + \lceil \log_2(m+k+2) \rceil) T_X.
\end{aligned}$$

Now, let us examine the total space complexity of our multiplier. On one hand, all the AND gates are used for computing $C(x)$, $D(x)$ and $S_3(x)$. With reference to (3.6) and (3.19), the number of AND gates needed for computing $A(x)B(x)x^{-k}$ is:

$$\begin{aligned}
S^{\otimes} &= S^{\otimes}(C(x)) + S^{\otimes}(D(x)) + S^{\otimes}(S_3(x)) \\
&= \frac{m^2-1}{4} \times 2 + \frac{(m+1)^2}{4} \\
&= \frac{3m^3+2m-1}{4}.
\end{aligned}$$

On the other hand, let us consider the XOR gates. Among all the XOR gates, some are used for computing $C(x)$, $D(x)$ or $S_3(x)$, and the rest are used for XORing the coefficients of $C(x)$, $D(x)$ and $S_3(x)$ together. First, with reference to (3.6) and (3.19), the number of XOR gates used for computing $C(x)$, $D(x)$ and $S_3(x)$ is:

$$\begin{aligned}
S^{\oplus}(C(x)) + S^{\oplus}(D(x)) + S^{\oplus}(S_3(x)) &= \frac{m^2-4m+3}{4} \times 2 + \frac{m^2+2m-3}{4} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i) \\
&= \frac{3m^2-6m+3}{4} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i).
\end{aligned}$$

Then, let us check the number of XOR gates used for XORing up the coefficients of $C(x)$, $D(x)$ and $S_3(x)$. Such "combination" happens in Stage II of Part II as well as in Stage III. For clarity, we give the following coefficient expression of $S_1(x) + S_2(x) + S_3(x)$ based

on the coefficients of $C(x)$, $D(x)$ and $S_3(x)$:

$$r_i+s_i+t_i=\begin{cases} c_{\frac{i}{2}}+c_{\frac{m+k+i}{2}}+c_{\frac{k+i+1}{2}}+d_{\frac{i}{2}} & \\ \quad +d_{\frac{m+k+i}{2}}+d_{\frac{k+i-1}{2}}+t_i, & i=0,2,\cdots,k-1, \\ c_{\frac{m+k+i}{2}}+c_{\frac{k+i+1}{2}}+c_{\frac{m+i+1}{2}}+d_{\frac{m+k+i}{2}} & \\ \quad +d_{\frac{k+i-1}{2}}+d_{\frac{m+i-1}{2}}+t_i, & i=k+1,k+3,\cdots,m-k-2, \\ c_{\frac{k-m+i}{2}}+c_{\frac{k+i+1}{2}}+c_{\frac{m+i+1}{2}}+d_{\frac{k-m+i}{2}} & \\ \quad +d_{\frac{k+i-1}{2}}+d_{\frac{m+i-1}{2}}+t_i, & i=m-k,m-k+2,\cdots,m-3, \\ c_{\frac{k-1}{2}}+c_{\frac{m+k}{2}}+d_{\frac{k-1}{2}}+d_{\frac{k+m-2}{2}}+t_i, & i=m-1, \\ c_{\frac{k+i}{2}}+c_{\frac{i+1}{2}}+c_{\frac{m+k+i+1}{2}}+d_{\frac{k+i}{2}} & \\ \quad +d_{\frac{i-1}{2}}+d_{\frac{m+k+i-1}{2}}+t_i, & i=1,3,\cdots,k-2, \\ c_{\frac{k+i}{2}}+c_{\frac{m+i}{2}}+c_{\frac{m+k+i+1}{2}}+d_{\frac{k+i}{2}} & \\ \quad +d_{\frac{m+i}{2}}+d_{\frac{m+k+i-1}{2}}+t_i, & i=k,k+2,\cdots,m-k-3, \\ c_{\frac{m-1}{2}}+c_{\frac{2m-k-1}{2}}+d_{\frac{m-1}{2}} & \\ \quad +d_{\frac{2m-k-1}{2}}+t_i, & i=m-k-1, \\ c_{\frac{k+i}{2}}+c_{\frac{m+i}{2}}+c_{\frac{k-m+i+1}{2}}+d_{\frac{k+i}{2}} & \\ \quad +d_{\frac{m+i}{2}}+d_{\frac{k-m+i-1}{2}}+t_i, & i=m-k+1,m-k+3,\cdots,m-2, \end{cases} \tag{3.22}$$

where it can be verified that the total number of XOR operations among the coefficients of $C(x)$, $D(x)$ and $S_3(x)$ is $6m-4$. Moreover, because the above expression contains $c_0=0$ at $i=0,\ m-k$, and $d_{m-1}=0$ at $i=m-k-2,\ m-2$, we can further save four XOR gates from the $6m-4$ XOR gates. Thus, $6m-8$ XOR gates are needed for the combination of the coefficients of $C(x)$, $D(x)$ and $S_3(x)$ in order to compute $S_1(x)+S_2(x)+S_3(x)$.

Therefore, the total number of XOR gates used for computing $A(x)B(x)x^{-k}$ is:

$$\begin{aligned} S^{\oplus} &= S^{\oplus}(C(x)) + S^{\oplus}(D(x)) + S^{\oplus}(S_3(x)) + 6m - 8 \\ &= \frac{3m^2 - 6m + 3}{4} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i) + 6m - 8 \\ &= \frac{3m^2 + 18m - 29}{4} + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i). \end{aligned}$$

In summary, the total space and time complexities of our multiplier is:

$$\begin{aligned} &S^{\otimes} : \tfrac{3m^2+2m-1}{4}, \\ &S^{\oplus} : \tfrac{3m^2}{4} + \tfrac{9m}{2} + \textstyle\sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i) - \tfrac{29}{4}, \\ &\text{Delay} : T_A + (2 + \lceil \log_2(m+k+2) \rceil)\, T_X. \end{aligned} \tag{3.23}$$

By the way, it should be noted that $T_A + (2 + \lceil \log_2(m+k+2) \rceil)\, T_X$ may be equal to

$$T_A + (2 + \lceil \log_2 m \rceil)\, T_X,$$

when k is small. This means that our multiplier can be one T_X faster on certain fields than on the other fields.

For simplicity, we do not present detailed analyses for the other three cases mentioned at the beginning of Section 3.3. One can use the same strategy as described above to develop the corresponding odd-m multipliers whose space and time complexities are summarized in the first three rows of Table 3.V. The last two rows of Table 3.V are for the even-m multipliers discussed in the next section. For further illustration, a schematic of our multiplier based on $x^7 + x + 1$ is given in Appendix II.

3.6 The Case When m Is Even

We briefly discuss our algorithm for the remaining case when m is even. In such case, we make some small changes to the multiplicand partition method in the PCHS

Table 3.V – Complexities of the new Montgomery multiplier in the other cases

Case	#AND	#XOR	Delay
m odd, k even $0 < k \leq \frac{m-1}{3}$	$\frac{3m^2+2m-1}{4}$	$\frac{3m^2}{4} + \frac{9m}{2} + \sum_{i=1}^{\frac{k}{2}} W(i) + \sum_{i=1}^{\frac{m-k-1}{2}} W(i) - \frac{29}{4}$	$T_A + (2 + \lceil \log_2(2m-3k-2) \rceil) T_X$
m odd, k even $\frac{m-1}{3} < k \leq \frac{m-3}{2}$			$T_A + (2 + \lceil \log_2 3k \rceil) T_X$
m odd, k even $k = \frac{m-1}{2}$	$\frac{3m^2+2m-1}{4}$	$\frac{3m^2}{4} + \frac{9m}{2} + \sum_{i=1}^{\frac{m-1}{4}} W(i) + \sum_{i=1}^{\frac{m-1}{4}} W(i) - \frac{29}{4}$	$T_A + (1 + \lceil \log_2(3m-3) \rceil) T_X$
m odd, k odd $k = \frac{m-1}{2}$	$\frac{3m^2+2m-1}{4}$	$\frac{3m^2}{4} + \frac{9m}{2} + \sum_{i=1}^{\frac{m+1}{4}} W(i) + \sum_{i=1}^{\frac{m+1}{4}} W(i) - \frac{29}{4}$	$T_A + (1 + \lceil \log_2(3m+3) \rceil) T_X$
m even, k odd $k = \frac{m}{2}$	$\frac{3m^2}{4}$	$\frac{3m^2}{4} + 4m + \sum_{i=1}^{\frac{m+2}{4}} W(i) + \sum_{i=1}^{\frac{m-6}{4}} W(i) - 4$	$T_A + (2 + \lceil \log_2 m \rceil) T_X$
m even, k odd $k < \frac{m}{2}$	$\frac{3m^2}{4}$	$\frac{3m^2}{4} + 4m + \sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k-3}{2}} W(i) - 4$	$T_A + (2 + \lceil \log_2 m \rceil) T_X$

algorithm[21]. We split $A(x)$, $B(x)$ into two parts as follows:

$$A(x) = A_1^2(x) + xA_2^2(x), \;\; B(x) = B_1^2(x) + xB_2^2(x),$$

where

$$A_1(x) := \sum_{i=0}^{\frac{m}{2}-1} a_{2i}x^i, \; A_2(x) := \sum_{i=0}^{\frac{m}{2}-1} a_{2i+1}x^i,$$
$$B_1(x) := \sum_{i=1}^{\frac{m}{2}-1} b_{2i}x^i, \; B_2(x) := \sum_{i=0}^{\frac{m}{2}-1} b_{2i+1}x^i.$$

Suppose the Montgomery factor is x^{-u}, the Montgomery multiplication formula can be written as:

$$\begin{aligned} A(x)B(x)x^{-u} &= \left[(A_1^2(x) + xA_2^2(x))(B_1^2(x) + xB_2^2(x))\right] x^{-u} \\ &= \left[(A_1^2(x) + xA_2^2(x))(x^{-1}B_1^2(x) + B_2^2(x))\right] x^{-u+1} \\ &= [x^{-1}(A_1(x)B_1(x))^2 + x(A_2(x)B_2(x))^2 \\ &\quad + (A_1(x)B_2(x))^2 + (A_2(x)B_1(x))^2]x^{-u+1} \\ &= (A_1(x)B_1(x))^2 x^{-u+1}(1+x^{-1}) + (A_2(x)B_2(x))^2 x^{-u+1}(1+x) \\ &\quad + ((A_1(x) + A_2(x))(B_1(x) + B_2(x)))^2 x^{-u+1}. \end{aligned} \tag{3.24}$$

The above expression is nearly the same as 3.2 in Section 3.1. In order to use Wu's Montgomery squaring formula, we let $u = k+1$.

We now define the following symbols and equations like we did in (3.4):

$$
\begin{aligned}
&C(x) = A_1(x)B_1(x) = \sum_{i=0}^{m-2} c_i x^i,\ D(x) = A_2(x)B_2(x) = \sum_{i=0}^{m-2} d_i x^i,\\
&Z(x) = (C(x))^2 x^{-k} = \sum_{i=0}^{m-1} z_i x^i,\ Z'(x) = (D(x))^2 x^{-k} = \sum_{i=0}^{m-1} z'_i x^i,\\
&S_1(x) = Z(x)(1+x^{-1}) = \sum_{i=0}^{m-1} r_i x^i,\ S_2(x) = Z'(x)(1+x) = \sum_{i=0}^{m-1} s_i x^i;\\
&U(x) = A_1(x) + A_2(x) = \sum_{i=0}^{\frac{m-2}{2}} u_i x^i,\ V(x) = B_1(x) + B_2(x) = \sum_{i=0}^{\frac{m-2}{2}} v_i x^i,\\
&E(x) = U(x)V(x) = \sum_{i=0}^{m-2} e_i x^i,\ S_3(x) = (E(x))^2 x^{-k} = \sum_{i=0}^{m-1} t_i x^i.
\end{aligned}
$$

The coefficients of $C(x)$ and $D(x)$ are given by

$$
c_i = \begin{cases} \sum_{j=0}^{i} a_{2j} b_{2(i-j)}, & 0 \le i \le \frac{m}{2} - 1,\\ \sum_{j=i-\frac{m}{2}+1}^{\frac{m}{2}-1} a_{2j} b_{2(i-j)}, & \frac{m}{2} \le i \le m-2. \end{cases} \tag{3.25}
$$

and

$$
d_i = \begin{cases} \sum_{j=0}^{i} a_{2j+1} b_{2(i-j)+1}, & 0 \le i \le \frac{m}{2} - 1,\\ \sum_{j=i-\frac{m}{2}+1}^{\frac{m}{2}-1} a_{2j+1} b_{2(i-j)+1}, & \frac{m}{2} \le i \le m-2, \end{cases} \tag{3.26}
$$

where $c_{m-1} = 0$ and $d_{m-1} = 0$. Now consider the expression of $S_1(x) + S_2(x)$. It should be noted that when m is even, k has to be odd to make $f(x) = x^m + x^k + 1$ irreducible in $\mathbb{F}_2[x]$. Therefore, we only discuss two cases with respect to k:

Case 1: m is even and k is odd, $m > 2k$. We give the coefficient expressions of $S_1(x)$ and $S_2(x)$ in (3.27) and (3.28), respectively, followed by the coefficient expression of $S_1(x)+S_2(x)$ in (3.29):

$$r_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+k+i+1}{2}} + c_{\frac{k+i+1}{2}}, & i = 0,2,\cdots,k-3, \\ c_0 + c_{\frac{k-1}{2}} + c_{\frac{m+2k}{2}} + c_k, & i = k-1, \\ c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+k+i+1}{2}}, & i = k+1,k+3,\cdots,m-k-3, \\ c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{k-m+i+1}{2}}, & i = m-k-1,m-k+1,\cdots,m-2, \\ c_{\frac{k-1}{2}} + c_{\frac{m+k-1}{2}} + c_0, & i = m-1, \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}}, & i = 1,3,\cdots,k-2, \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{m+i+1}{2}}, & i = k,k+2,\cdots,m-k-2, \\ c_{\frac{k+i}{2}} + c_{\frac{k-m+i}{2}} + c_{\frac{m+i+1}{2}}, & i = m-k,m-k+2,\cdots,m-3, \end{cases} \tag{3.27}$$

$$s_i = \begin{cases} d_{\frac{i}{2}} + d_{\frac{m+k+i-1}{2}} + d_{\frac{k+i-1}{2}}, & i = 0,2,\cdots,k-1, \\ d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = k+1,k+3,\cdots,m-k-1, \\ d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{k-m+i-1}{2}}, & i = m-k+1,m-k+3,\cdots,m-2, \\ d_{\frac{k+i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{i-1}{2}}, & i = 1,3,\cdots,k-2, \\ d_{\frac{k+i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{m+i-1}{2}}, & i = k,k+2,\cdots,m-k-2, \\ d_{\frac{k+i}{2}} + d_{\frac{k-m+i}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k,m-k+2,\cdots,m-1, \end{cases} \tag{3.28}$$

$$
r_i + s_i = \begin{cases}
c_{\frac{i}{2}} + c_{\frac{m+k+i+1}{2}} + c_{\frac{k+i+1}{2}} + d_{\frac{i}{2}} & \\
\quad + d_{\frac{m+k+i-1}{2}} + d_{\frac{k+i-1}{2}}, & i = 0, 2, \cdots, k-3, \\
c_0 + c_{\frac{k-1}{2}} + c_{\frac{m+2k}{2}} + c_k + d_{\frac{k-1}{2}} & \\
\quad + d_{\frac{m+2k-2}{2}} + d_{k-1}, & i = k-1, \\
c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{m+k+i+1}{2}} & \\
\quad + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{m+k+i-1}{2}}, & i = k+1, k+3, \cdots, m-k-3, \\
c_{m-\frac{k+1}{2}} + c_{\frac{m}{2}} + c_0 + d_{m-\frac{k+1}{2}} + d_{\frac{m}{2}-1}, & i = m-k-1, \\
c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} + c_{\frac{k-m+i+1}{2}} & \\
\quad + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}} + d_{\frac{k-m+i-1}{2}}, & i = m-k+1, m-k+3, \cdots, m-2, \\
c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}} & \\
\quad + d_{\frac{k+i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{i-1}{2}}, & i = 1, 3, \cdots, k-2, \\
c_{\frac{k+i}{2}} + c_{\frac{m+k+i}{2}} + c_{\frac{m+k+i+1}{2}} & \\
\quad + d_{\frac{k+i}{2}} + d_{\frac{m+k+i}{2}} + d_{\frac{m+i-1}{2}}, & i = k, k+2, \cdots, m-k-2, \\
c_{\frac{m-1}{2}} + c_{\frac{2m-i+1}{2}} + d_{\frac{m-1}{2}} + d_{\frac{2m-k-1}{2}}, & i = m-k-1, \\
c_{\frac{k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k-m+i+1}{2}} & \\
\quad + d_{\frac{k+i}{2}} + d_{\frac{k-m+i}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k, m-k+2, \cdots, m-3, \\
c_{\frac{k-1}{2}} + c_{\frac{m+k-1}{2}} + c_0 + d_{\frac{m+k-1}{2}} + d_{\frac{k-1}{2}}, & i = m-1.
\end{cases} \tag{3.29}
$$

Case 2: m is even and k is odd, $m = 2k$. Here, we only give the coefficient expression of $S_1(x) + S_2(x)$:

$$r_i + s_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{m+2i+2}{4}} + c_{\frac{3m+2i+2}{4}} + d_{\frac{i}{2}} & \\ \quad + d_{\frac{3m+2i-2}{4}} + d_{\frac{m+2i-2}{4}}, & i = 0, 2, \cdots, \frac{m}{2} - 3, \\ c_{\frac{m}{2}} + c_{\frac{m-2}{4}} + d_{\frac{m}{2}-1} + d_{\frac{m-2}{4}}, & i = \frac{m}{2} - 1, \\ c_{\frac{m+i}{2}} + c_{\frac{m+2i+2}{4}} + c_{\frac{2i-m+2}{4}} & \\ \quad + d_{\frac{m+i}{2}} + d_{\frac{m+2i-2}{4}} + d_{\frac{2i-m-2}{4}}, & i = \frac{m}{2} + 1, \frac{m}{2} + 3, \cdots, m - 2, \\ c_{\frac{m}{2}} + c_{\frac{3m+2}{4}} + c_0 + d_{\frac{m}{2}} & \\ \quad + d_0 + d_{\frac{3m-2}{4}}, & i = \frac{m}{2}, \\ c_{\frac{m+2i}{4}} + c_{\frac{3m+2i}{4}} + c_{\frac{i+1}{2}} & \\ \quad + d_{\frac{m+2i}{4}} + d_{\frac{2i-m}{4}} + d_{\frac{m+i-1}{2}}, & i = 1, 3, \cdots, \frac{m}{2} - 2, \\ c_{\frac{m+2i}{4}} + c_{\frac{2i-m}{4}} + c_{\frac{m+i+1}{2}} & \\ \quad + d_{\frac{m+2i}{4}} + d_{\frac{2i-m}{4}} + d_{\frac{m+i-1}{2}}, & i = \frac{m}{2} + 2, \frac{m}{2} + 4, \cdots, m - 3, \\ c_{\frac{m-2}{4}} + c_{\frac{3m-2}{4}} + d_{\frac{m}{2}} + d_{\frac{m-2}{4}}, & i = m - 1. \end{cases} \tag{3.30}$$

In the meantime, we compute $S_3(x)$ with the previously described bit-reusing trick. As the degrees of $A_1(x)$, $A_2(x)$, $B_1(x)$ and $B_2(x)$ are at most $\frac{m}{2} - 1$, it follows that $0 \leq \deg U(x), \deg V(x) \leq \frac{m}{2} - 1$. In Case 1, the coefficient expressions of $E(x)$ and $S_3(x)$ are

$$e_i = \begin{cases} \sum_{j=0}^{i} u_j v_{i-j}, & 0 \leq i \leq \frac{m}{2} - 1, \\ \sum_{j=i-\frac{m}{2}+1}^{\frac{m}{2}-1} u_j v_{i-j}, & \frac{m}{2} \leq i \leq m - 2, \end{cases} \tag{3.31}$$

and

$$t_i = \begin{cases} e_{\frac{i}{2}}, & i = 0, 2, \cdots, k - 1, \\ e_{\frac{m+i}{2}}, & i = k + 1, k + 3, \cdots, m - 2, \\ e_{\frac{k+i}{2}} + e_{\frac{m+k+i}{2}}, & i = 1, 3, \cdots, m - k - 2, \\ e_{\frac{k+i}{2}} + e_{\frac{k-m+i}{2}}, & i = m - k, m - k + 2, \cdots, m - 1, \end{cases} \tag{3.32}$$

respectively. The computation of $S_3(x)$ is based on the same idea as that in the odd-m case. The computation sequence in Case 1 is shown in Table 3.VI:

Table 3.VI – The computation sequence in the case when m is even and $m > 2k$

Parts\ Stages	I	II	III
I	$\underbrace{U = A_1 + A_2,\ V = B_1 + B_2}_{\text{Delay: } 1T_X}$	$\underbrace{S_3 = (UV)^2 x^{-k}, \{c_0 x^{k-1}\} \text{ inserted}}_{T_A + \lceil \log_2 \frac{m}{2} \rceil T_X}$	$\underbrace{[S_1 + S_2] + S_3}_{2T_X}$
II	$\underbrace{C = A_1 B_1, D = A_2 B_2}_{\text{Delay: } T_A + \lceil \log_2 \frac{m}{2} \rceil T_X}$	$\underbrace{[S_1 + S_2], \{c_0 x^{k-1}\} \text{ removed}}_{1T_X}$	

In addition, we need to mention a few points about the time complexity of our algorithm in Case 1 when m is even and $m > 2k$:

1. The computation of both $C(x)$ and $D(x)$ requires a delay of $T_A + \lceil \log_2 \frac{m}{2} \rceil T_X$.
2. By substituting (3.31) into (3.32), each t_i becomes a XOR of at most $\frac{m}{2}$ terms in $\{u_i v_j \mid 0 \leq i, j \leq \frac{m}{2} - 1\}$. Since it takes $T_X + T_A$ to compute all the $u_i v_j$ terms in advance, the delay for computing $S_3(x)$ is at most $T_A + T_X + \lceil \log_2 \frac{m}{2} \rceil T_X$.
3. In the coefficient expression of $S_1(x)$, i.e. (3.27), we notice that r_{k-1} is a XOR of four bits. Hence, $r_{k-1} + s_{k-1}$ becomes a XOR of four bits at the end of Stage II, which leads to a delay of $3T_X$ for computing $S_1(x) + S_2(x) + S_3(x)$ in Stage III. To make Stage III shorter, we need to delete c_0 from the expression of r_{k-1} and insert c_0 into the expression of t_{k-1}. Since t_{k-1} is a XOR of $\frac{k+1}{2}$ bits at the beginning of Stage II of Part I, t_{k-1} becomes a XOR of $\frac{k+3}{2}$ bits after it incorporates $c_0 = a_0 b_0$ as an extra addend. Because $m > 6$ in practice and $m > 2k$, it will take no more than $T_A + \lceil \log_2 \frac{m}{2} \rceil T_X$ to compute the modified expression of t_{k-1} in Stage II of Part I. In this way, we keep the length of Stage III within a delay of $2T_X$ while leaving the delay for computing $S_3(x)$ unaffected.

The space and time complexities of our multiplier on even-m $GF(2^m)$ are summarized in Table 3.V in the last two rows.

CHAPTER 4

COMPARISON AND DISCUSSION

In the previous chapter, we have expressed the space complexities of the new multiplier with sums of Hamming weight. In order to facilitate the comparison between our algorithm and the others in terms of space complexity, we need to simplify the Hamming weight sum expression.

First, we compute the upper bound of $\sum_{i=1}^{N} W(i)$, $\forall N \in \mathbb{Z}^+$. Since the binary length of N is $\lfloor \log_2 N \rfloor + 1$, all the positive integers smaller than N have a binary length equal to or smaller than $\lfloor \log_2 N \rfloor + 1$. Suppose $M = \{i \mid 1 \leq i \leq N,\ i \in \mathbb{Z}^+\}$, $G = \{j \mid 1 \leq L(j) \leq \lfloor \log_2 N \rfloor + 1,\ j \in \mathbb{Z}^+\}$ where $L(j)$ is the length of $(j)_2$, it follows that $M \subseteq G$, hence $\sum_{i \in M} W(i) \leq \sum_{j \in G} W(j)$.

More generally, suppose $G' = \{j \mid 1 \leq L(j) \leq L_0,\ j \in \mathbb{Z}^+\}$ where $L_0 \in \mathbb{Z}^+$, let us calculate $\sum_{j \in G'} W(j)$. We can partition G' into L_0 non-overlapping subsets, i.e. $G' = \bigcup_{k=1}^{L_0} G'_k$ and $\bigcap_{k=1}^{L_0} G'_k = \emptyset$, such that

$$G'_k = \{n \mid 1 \leq L(n) \leq L_0,\ W(n) = k\}.$$

There is a bijection between G'_k and the set of L_0-bit-long binary strings that have k bits equal to "1" and the other $L_0 - k$ bits equal to "0". Hence $\left|G'_k\right| = \binom{L_0}{k}$.

Then, we have the following transformation:

$$\begin{aligned}
\sum_{j\in G'} W(j) &= \sum_{k=1}^{L_0} \sum_{n\in G'_k} W(n) \\
&= \sum_{k=1}^{L_0} \sum_{n\in G'_k} k \\
&= \sum_{k=1}^{L_0} \binom{L_0}{k} k \\
&= \left(\frac{d}{dx} \left(\sum_{k=1}^{L_0} \binom{L_0}{k} x^k \right) \right) \Bigg|_{x=1} \\
&= \left(\frac{d}{dx} \left(\sum_{k=1}^{L_0} \binom{L_0}{k} x^k \cdot 1^{L_0-k} \right) \right) \Bigg|_{x=1},
\end{aligned}$$

which can be further transformed by using the binomial theorem

$$\sum_{k=0}^{L_0} \binom{L_0}{k} a^k b^{L_0-k} = (a+b)^{L_0}$$

as

$$\begin{aligned}
\left(\frac{d}{dx} \left(\sum_{k=1}^{L_0} \binom{L_0}{k} x^k \cdot 1^{L_0-k} \right) \right) \Bigg|_{x=1} &= \left(\frac{d}{dx} \left((x+1)^{L_0} - 1 \right) \right) \Bigg|_{x=1} \\
&= L_0 (x+1)^{L_0-1} |_{x=1} \\
&= L_0 2^{L_0-1}.
\end{aligned}$$

Thus, $\sum_{j\in G'} W(j) = L_0 2^{L_0-1}$. By replacing L_0 with $\lfloor 1+\log_2 N \rfloor$, we get

$$\sum_{i=1}^{N} W(i) \leq \sum_{j\in G} W(j) = \lfloor 1+\log_2 N \rfloor 2^{\lfloor \log_2 N \rfloor},$$

which is a tight upper bound for $\sum_{i=1}^{N} W(i)$. For simplicity, we may use the following upper bound:

$$\sum_{i=1}^{N} W(i) \leq N + N \log_2 N.$$

Here, $k+1 \leq \frac{m-1}{2}$ and $m-k \leq m-1$, so the Hamming weight sum in (3.18) is:

$$\begin{aligned}\sum_{i=1}^{\frac{k+1}{2}} W(i) + \sum_{i=1}^{\frac{m-k}{2}} W(i) &\leq \frac{k+1}{2}(1+\log_2 \frac{k+1}{2}) + \frac{m-k}{2}(1+\log_2 \frac{m-k}{2}) \\ &< \frac{m-1}{4}(1+\log_2 \frac{m-1}{4}) + \frac{m-1}{2}(1+\log_2 \frac{m-1}{2}) \\ &= \frac{3m-3}{4}\log_2(m-1) - \frac{m-1}{4}.\end{aligned}$$

Thus, the total space complexity in (3.23) is less than $\frac{3m^2+17m-28}{4} + \frac{3m-3}{4}\log_2(m-1)$. By the same process, we were able to simplify the space complexity expressions in Table 3.V.

The efficiency of our new algorithm in terms of speed was analysed in the previous chapter. As we have seen, given a Galois field generated by a trinomial $f(x) = x^m + x^k + 1$, according to Table 3.V, when m is odd and $k \leq \frac{m-1}{2}$, the time complexity of our multiplier is:

$$\text{Delay} \leq T_A + (3 + \lceil \log_2 m \rceil) T_X,$$

and when m is even and $k \leq \frac{m}{2}$, the time complexity of our multiplier is:

$$\text{Delay} = T_A + (2 + \lceil \log_2 m \rceil)\, T_X.$$

Actually, for certain values of odd m and k, the time complexity of our multiplier can be even smaller than $T_A + (3 + \lceil \log_2 m \rceil) T_X$. We have inspected all the 786 irreducible trinomials with odd degrees between 101 and 1203 inclusively. Among these trinomials, we found that about 56% of them can generate the Galois fields where our multiplier runs with a time complexity of $T_A + (2 + \lceil \log_2 m \rceil) T_X$.

In Table 4.I, we provide a comparison in space and time complexities between our algorithm and several other bit-parallel $GF(2^m)$ multiplication algorithms. The comparisons are split in three cases for the trinomial $x^m + x^k + 1$: when $k = 1$, when $k = \frac{m}{2}$ and m is even, and when $1 < k < \frac{m}{2}$. Notice that the first two cases may have a time complexity one T_X lower than that in the general case $1 < k < \frac{m}{2}$ and k is odd. Com-

pared with the fastest multipliers [7] [20], our algorithm saves about 25% logic gates by choosing k appropriately, while being only at most $2T_X$ slower in speed. Moreover, compared with Elia's Karatsuba-based multiplier [23], our algorithm maintains a nearly equal space complexity while having the same or less time complexity.

Let us use examples to illustrate the merit of the new multiplier. Among the five irreducible polynomials recommended for the elliptic curve digital signature algorithm (ECDSA) by the National Institute of Standards and Technology (NIST) [26], two trinomials,

$$p_1(x) = x^{233} + x^{74} + 1 \text{ and } p_2(x) = x^{409} + x^{87} + 1,$$

generate the fields on which the new algorithm has the same time complexity as that of the Mastrovito multipliers in [11] [12] [13] or that of the Montgomery multiplier in [19], but requires less space complexity.

To facilitate description, in Table 4.II, we only compare our multiplier with three typical multipliers presented in [20], [23] and [22], respectively. Compared with the fastest algorithm by Hariri et al. [20], our algorithm is one T_X slower, while saving about 24.8% AND gates plus 22.5% XOR gates over $p_1(x)$, and 24.9% AND gates plus 23.5% XOR gates over $p_2(x)$. Compared with the bit-parallel Karatsuba variant proposed by Cho et al. [22], our algorithm has the same speed but saves 16.4% AND gates plus 13.9% XOR gates over $p_1(x)$, and 21.3% AND gates plus 19.9% XOR gates over $p_2(x)$. Compared with the algorithm of Elia [23] which uses the fewest gates, our algorithm requires the same number of AND gates plus a little more XOR gates (about 0.9% over $p_1(x)$ and 0.6% over $p_2(x)$), but is one T_X faster. We illustrate the above algorithms with the delay-by-gate-number graphs in Figure 4.1 and Figure 4.2, respectively.

In Figure 4.1 and Figure 4.2, it can be observed that while the total gate number increases, the corresponding circuit delay draws near to the red dotted line representing the lowest possible delay. Remember the so-called "space-time trade-off" presented in the title of this thesis. For a $GF(2^m)$ generated by $x^m + x^k + 1$, we may use more gates for constructing faster multipliers. Nevertheless, we cannot reduce the general delay of

Table 4.I – Comparison of several bit-parallel multipliers based on irreducible trinomials

Multiplier	# AND	# XOR	Delay
$x^m + x + 1$			
[11][12][13]	m^2	$m^2 - 1$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$
Wu[19]	m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 (m-2) \rceil) T_X$
[7][20][25]	m^2	$m^2 - 1$	$T_A + \lceil \log_2 (2m-1) \rceil T_X$
Cho[22]	$m^2 - 1$	$m^2 - 1$	$T_A + (2 + \lceil \log_2 (m-4) \rceil) T_X$
Proposed	$\frac{3m^2+2m-1}{4}$	$< \frac{3m^2+18m-25}{4} + \frac{m-1}{2} \log_2 (m-1)$ (m odd)	$\leq T_A + (2 + \lceil \log_2 (m+3) \rceil) T_X$
	$\frac{3m^2}{4}$	$< \frac{3m^2+16m-12}{4} + \frac{m-4}{2} \log_2 (m-4)$ (m even)	$T_A + (2 + \lceil \log_2 m \rceil) T_X$
$x^m + x^k + 1$ $(1 < k < \frac{m}{2})$			
[11][12][13][19]	m^2	$m^2 - 1$	$T_A + (2 + \lceil \log_2 m \rceil) T_X$
Petra[25]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 (2m + 2k - 3) \rceil) T_X$
Fan [7]Hariri[20]	m^2	$m^2 - 1$	$T_A + \lceil \log_2 (2m - k - 1) \rceil T_X$
Elia [23]	$\frac{3m^2+2m-1}{4}$	$\frac{3m^2}{4} + 4m + k - \frac{23}{4}$ (m odd)	$T_A + (3 + \lceil \log_2 (m-1) \rceil) T_X$
	$\frac{3m^2}{4}$	$\frac{3m^2}{4} + \frac{5m}{2} + k - 4$ (m even)	
Li[24]	$\frac{m^2}{2} + (m-k)^2$	$\frac{m^2}{2} + (m-k)^2 + 2k$	$T_A + (2 + \lceil \log_2 (m-1) \rceil) T_X$
Cho[22]	$m^2 - k^2$	$m^2 + k - k^2 - 1 (1 < k < \frac{m}{3})$	$\leq T_A + (2 + \lceil \log_2 m \rceil) T_X$
		$m^2 + 4k - k^2 - m - 1 (\frac{m}{3} \leq k < \frac{m-1}{2})$	
		$m^2 + 2k - k^2 (k = \frac{m-1}{2})$	
Proposed	$\frac{3m^2+2m-1}{4}$	$< \frac{3m^2+17m-30}{4} + \frac{3m-5}{4} \log_2 (m+1)$ (m odd)	$\leq T_A + (3 + \lceil \log_2 m \rceil) T_X$
	$\frac{3m^2}{4}$	$< \frac{3m^2+14m-10}{4} + \frac{3m-10}{4} \log_2 m$ (m even)	$T_A + (2 + \lceil \log_2 m \rceil) T_X$
$x^m + x^{\frac{m}{2}} + 1$			
[11][12][13][19]	m^2	$m^2 - \frac{m}{2}$	$T_A + (1 + \lceil \log_2 (m-1) \rceil) T_X$
[7][20][25]	m^2	$m^2 - \frac{m}{2}$	$T_A + \lceil \log_2 \frac{3m}{2} \rceil T_X$
Shen [34]	$\frac{3m^2}{4}$	$\frac{3m^2}{4} + m + 1$	$T_A + (1 + \lceil \log_2 (m-1) \rceil) T_X$
Shou [35]	$\frac{3m^2}{4}$	$\frac{3m^2}{4} + m + 1$	$T_A + (3 + \lceil \log_2 (m-1) \rceil) T_X$
Cho[22]	$\frac{3m^2}{4}$	$\frac{3m^2}{4} + m + 1$	$T_A + (1 + \lceil \log_2 (m-2) \rceil) T_X$
Proposed	$\frac{3m^2}{4}$	$< \frac{3m^2+14m-12}{4} + \frac{m-2}{2} \log_2 (m+2)$	$T_A + (2 + \lceil \log_2 m \rceil) T_X$

Table 4.II – Complexities of several bit-parallel multipliers on the two Galois fields recommended by the National Institute of Standards and Technology

	# AND	# XOR	Delay
$x^{233}+x^{74}+1$			
Fan[7]Hariri[20]	54289	54288	T_A+9T_X
Elia[23]	40833	41717	T_A+11T_X
Cho[22]	48813	48886	T_A+10T_X
This thesis	40833	42091	T_A+10T_X
$x^{409}+x^{87}+1$			
Fan[7]Hariri[20]	167281	167280	T_A+10T_X
Elia[23]	125665	127178	T_A+12T_X
Cho[22]	159712	159798	T_A+11T_X
This thesis	125665	127974	T_A+11T_X

$GF(2^m)$ multiplication to lower than $T_A+\lceil\log_2 m\rceil T_X$. Thus, there should be an optimum number of gates, say $OG_{m,k}$, that is used to construct a multiplier with a short delay, say $OD_{m,k}$. When designing a multiplication circuit, a reasonable increase in gate number can help bring a delay of $OD_{m,k}+T_X$ down to a delay of $OD_{m,k}$, but it might be a much larger increase in gate number that can remove another T_X from the delay of $OD_{m,k}$. On the aforementioned two fields recommended by NIST, it is likely that our algorithm is closer to the optimum in gate number than the others.

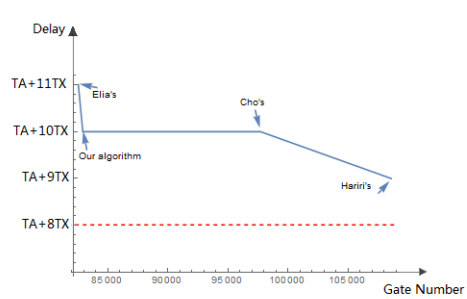


Figure 4.1 – A comparison over $p_1(x)$

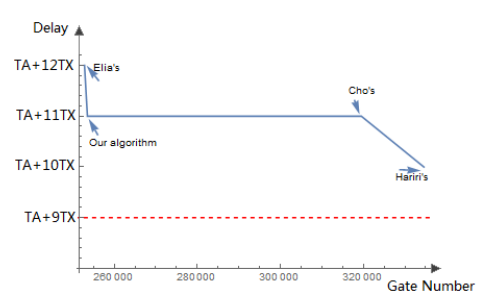


Figure 4.2 – A comparison over $p_2(x)$

Such good property with respect to the space-time trade-off enables our multiplier to satisfy spatial constraints inherent to small electronic devices (such as smart card, RFID tags etc.) while maintaining a time complexity very close to the best multipliers.

CHAPTER 5

CONCLUSIONS

5.1 Summaries and Conclusions

This thesis proposes a new bit-parallel Montgomery multiplication algorithm based on trinomials. This algorithm has been developed from the PCHS algorithm and Wu's Montgomery squaring formulae. The proposed new algorithm has a significantly lower space complexity than most of the previous algorithms while maintaining a low time complexity. Our work is especially interesting for the implementation of ECDSA on space-constrained devices. An integrated analysis of both space complexities and time complexities over the two trinomial-based Galois fields recommended by NIST shows that our algorithm reaches a better balance between the space complexity and the time complexity than the other typical algorithms.

5.2 Future Research

We are working on improving our multiplier over some special Galois fields. We are using the shifted polynomial basis to further reduce our multiplier's time complexity at the cost of a slight increase in its space complexity. Moreover, some circuit simulations will be done for the high-degree trinomials recommended by NIST. Such simulation should be carried out with the support of a professional software (e.g. Proteus). It is our hope that this new algorithm may be implemented on a real chip in the near future. Finally, we intend to develop an algorithm for the pentanomial-generated Galois fields by modifying the current trinomial-based algorithm.

BIBLIOGRAPHY

[1] A.J. Menezes, I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone and T. Yaghoobian. *Applications of Finite Fields*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1993.

[2] I. Blake, G. Seroussi and N. Smart. Elliptic curves in cryptography. *London Mathematical Society Lecture Note Series*, 265, Cambridge University Press, 1999.

[3] I. N. Herstein. *Topics in Algebra, 2nd Edition*. John Wiley & Sons, USA, 1975.

[4] M. Morales-Sandoval, C. Feregrino-Uribe and P. Kitsos. Bit-serial and digit-serial $GF(2^m)$ Montgomery multipliers using linear feedback shift registers. *Computers & Digital Techniques, IET*, 5(2):86-94, 2011.

[5] J. von zur Gathen and J. Shokrollahi. Efficient FPGA-based Karatsuba multipliers for polynomials over $\mathbb{F}_2$. *Selected Areas in Cryptography*, *Lecture Notes in Computer Science*, 3897:359-369, 2006.

[6] R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, New York, NY, USA, 1996.

[7] H. Fan, M.A. Hasan. Fast bit parallel-shifted polynomial basis multipliers in $GF(2^n)$. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 53(12):2606-2615, 2006.

[8] H. Fan, Y. Dai. Fast bit-parallel $GF(2^n)$ multiplier for all trinomials. *IEEE Transactions on Computers*, 54(4):485-490, 2005.

[9] H. Wu. Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basis. *IEEE Transactions on Computers*, 51(7):750-758, 2002.

[10] A. Cilardo. Fast parallel $GF(2^m)$ polynomial multiplication for all degrees. *IEEE Transactions on Computers*, 62(5):929-943, 2013.

[11] B. Sunar, Ç.K. Koç. Mastrovito multiplier for all trinomials. *IEEE Transactions on Computers*, 48(5):522-527, 1999.

[12] A. Halbutogullari, Ç.K. Koç. Mastrovito multiplier for general irreducible polynomials. *IEEE Transactions on Computers*, 49(5):503-518, 2000.

[13] T. Zhang, K.K. Parhi. Systematic design of original and modified mastrovito multipliers for general irreducible polynomials. *IEEE Transactions on Computers*, 50(7):734-749, 2001.

[14] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519-521, 1985.

[15] Ç. K. Koç and T. Acar. Montgomery multiplication in $GF(2^k)$. *Designs, Codes and Cryptography*, 14(1):57-69, 1998.

[16] J.C. Bajard, L. Imbert, and C. Negre. Arithmetic operations in finite fields of medium prime characteristic using the Lagrange representation. *IEEE Transactions on Computers*, 55(9):1167-1177, 2006.

[17] C. Chiou, C. Lee, A. Deng, and J. Lin. Concurrent error detection in Montgomery multiplication over $GF(2^m)$. *IEICE Transactions Fundamentals of Electronics Communications and Computer Sciences*, 89(2):566-574, 2006.

[18] M. Morales-Sandoval, C. Feregrino-Uribe, P. Kitsos, and R. Cumplido. Area/performance trade-off analysis of an FPGA digit-serial $GF(2^m)$ Montgomery multiplier based on LFSR. *Computer Electrical Engineering Journal*, 39(2):542-549, 2013.

[19] H. Wu. Montgomery multiplier and squarer for a class of finite fields. *IEEE Transactions on Computers*, 51(5):521-529, 2002.

[20] A. Hariri and A. Reyhani-Masoleh. Bit-serial and bit-parallel Montgomery multiplication and squaring over $GF(2^m)$. *IEEE Trans. Computers*, 58(10):1332-1345, 2009.

[21] S. Park, K, Chang, D. Hong, and C. Seo. New efficient bit-parallel polynomial basis multiplier for special pentanomials. *Integration, the VLSI Journal*, 47(1):130-139, 2014.

[22] Y. Cho, N. Chang, C. Kim, Y. Park, and S. Hong. New bit parallel multiplier with low space complexity for all irreducible trinomials over $GF(2^n)$. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(10):1903-1908, 2012.

[23] M. Elia, M. Leone, and C. Visentin. Low complexity bit-parallel multipliers for $GF(2^m)$ with generator polynomial $x^m + x^k + 1$. *Electronic Letters*, 35(7):551-552, 1999.

[24] Y. Li, G. Chen, and J. Li. Speed-up of bit-parallel Karatsuba multiplier in $GF(2^m)$ generated by trinomials, *Information Processing Letters*, 111(8):390-394, 2011.

[25] N. Petra, D. De Caro, and A.G.M. Strollo. A novel architecture for Galois fields $GF(2^m)$ multipliers based on mastrovito scheme. *IEEE Transactions on Computers*, 56(11):1470-1483, 2007.

[26] Recommended Elliptic Curves for Federal Government Use, `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`, July, 1999.

[27] A. Reyhani-Masoleh and M.A. Hasan. A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$. *IEEE Transactions on Computers*, 51(5):511-520, 2001.

[28] R.C. Mullin, I.M. Onyszchuk, S.A, Vanstone, and R.M. Wilson. Optimal normal basis in $GF(p^n)$. *Discrete Applied Mathematics*, 22(2):149-161, 1988/1989.

[29] I.S. Hsu, T.K. Truong, H.M. Shao, and L.J. Deutsch. A comparison of VLSI architecture of finite field multipliers using dual, normal or standard basis. *IEEE Transactions on Computers*, 37(6):735-739, 1988.

[30] S.T.J. Fenn, M. Benaissa and D. Taylor. $GF(2^m)$ multiplication and division over the dual basis. *IEEE Transactions on Computers*, 45(3):319-327, 1996.

[31] Y. Li, G. Chen, and X. Xie. Low complexity bit-parallel $GF(2^m)$ multiplier for all-one polynomials. *eprint.iacr.org*, 2012.

[32] F. Rodiguez-Henriquez and Ç.K. Koç. Parallel multipliers based on special irreducible pentanomials. *IEEE Transactions on Computers*, 52(11), 2003.

[33] H. Fan and M.A. Hasan. A new approach to sub-quadratic space complexity parallel multipliers for extended binary fields. IEEE Transactions on Computers, 56(2):224-233, 2007.

[34] H. Shen and Y. Jin. 2008, Low complexity bit parallel multiplier for $GF(2^m)$ generated by equally-spaced trinomials. *Information Processing Letters*, 107(6):211-215, 2008.

[35] G. Shou, Z. Mao, Y. Hu, Z. Guo and Z. Qian. Low complexity architecture of bit parallel multipliers for $GF(2^m)$. *Electronics Letters*, 46(19):1326-1327, 2010.

APPENDIX I

The Expressions of $\mathbf{S_1(x)+S_2(x)}$ in Case 2, Case 3 and Case 4 When m Is Odd

Case 2: When both m and k are odd, $k=\frac{m-1}{2}$, we have:

$$r_i+s_i=\begin{cases} c_{\frac{i}{2}}+c_{\frac{3m+2i-1}{4}}+c_{\frac{m+2i+1}{4}}+d_{\frac{i}{2}} & \\ \quad +d_{\frac{3m+2i-1}{4}}+d_{\frac{m+2i-3}{4}}, & i=0,2,\cdots,\frac{m-3}{2}, \\ c_{\frac{2i-m-1}{4}}+c_{\frac{m+2i+1}{4}}+c_{\frac{m+i+1}{2}} & \\ \quad +d_{\frac{2i-m-1}{4}}+d_{\frac{m+2i-3}{4}}+d_{\frac{m+i-1}{2}}, & i=\frac{m+1}{2},\frac{m+3}{2},\cdots,m-3, \\ c_{\frac{m-3}{4}}+c_{\frac{3m-1}{4}}+d_{\frac{m-3}{4}}+d_{\frac{3m-5}{4}}, & i=m-1, \\ c_{\frac{m+2i-1}{4}}+c_{\frac{i+1}{2}}+c_{\frac{3m+2i+1}{4}} & \\ \quad +d_{\frac{m+2i-1}{4}}+d_{\frac{i-1}{2}}+d_{\frac{3m+2i-3}{4}}, & i=1,3,\cdots,\frac{m-5}{2}, \\ c_{\frac{m-1}{2}}+c_{\frac{3m-1}{4}}+d_{\frac{m-1}{2}}+d_{\frac{3m-1}{4}}, & i=\frac{m-1}{2}, \\ c_{\frac{m+2i-1}{4}}+c_{\frac{m+i}{2}}+c_{\frac{2i-m+1}{4}} & \\ \quad +d_{\frac{m+2i-1}{4}}+d_{\frac{m+i}{2}}+d_{\frac{2i-m-3}{4}}, & i=\frac{m+3}{2},\frac{m+7}{2},\cdots,m-2. \end{cases}$$

The discussions about Case 1 and Case 2 provide the coefficient expressions of $S_1(x)+S_2(x)$ when k is odd. The results for the cases when k is even are presented as follows:

Case 3: When m is odd and k is even, $0 < k \leq \frac{m-3}{2}$, we have:

$$r_i + s_i = \begin{cases} c_{\frac{i}{2}} + c_{\frac{k+i}{2}} + c_{\frac{m+k+i+1}{2}} & \\ \quad + d_{\frac{i}{2}} + d_{\frac{k+i}{2}} + d_{\frac{m+k+i-1}{2}}, & i = 0, 2, \cdots, k-2, \\ c_{\frac{k+i}{2}} + c_{\frac{m+k+i+1}{2}} + c_{\frac{m+i+1}{2}} & \\ \quad + d_{\frac{k+i}{2}} + d_{\frac{m+k+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = k, k+2, \cdots, m-k-3, \\ c_{\frac{m-1}{2}} + c_{m-\frac{k}{2}} + d_{\frac{m-1}{2}} + d_{m-\frac{k}{2}-1}, & i = m-k-1, \\ c_{\frac{k+i}{2}} + c_{\frac{k-m+i+1}{2}} + c_{\frac{m+i+1}{2}} & \\ \quad + d_{\frac{k+i}{2}} + d_{\frac{k-m+i-1}{2}} + d_{\frac{m+i-1}{2}}, & i = m-k+1, m-k+3, \cdots, m-3, \\ c_{\frac{m+k+i}{2}} + c_{\frac{i+1}{2}} + c_{\frac{k+i+1}{2}} & \\ \quad + d_{\frac{m+k+i}{2}} + d_{\frac{i-1}{2}} + d_{\frac{k+i-1}{2}}, & i = 1, 3, \cdots, k-1, \\ c_{\frac{m+k+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} & \\ \quad + d_{\frac{m+k+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}}, & i = k+1, k+3, \cdots, m-k-2, \\ c_{\frac{k-m+i}{2}} + c_{\frac{m+i}{2}} + c_{\frac{k+i+1}{2}} & \\ \quad + d_{\frac{k-m+i}{2}} + d_{\frac{m+i}{2}} + d_{\frac{k+i-1}{2}}, & i = m-k, m-k+2, \cdots, m-2, \\ c_{\frac{m+k-1}{2}} + c_{\frac{k}{2}} + d_{\frac{m+k+1}{2}} + d_{\frac{k}{2}-1}, & i = m-1. \end{cases}$$

Case 4: When m is odd and k is even, $k=\frac{m-1}{2}$, we have:

$$r_i+s_i=\begin{cases} c_{\frac{i}{2}}+c_{\frac{m+2i-1}{4}}+c_{\frac{3m+2i+1}{4}} & \\ \quad +d_{\frac{i}{2}}+d_{\frac{m+2i-1}{4}}+d_{\frac{3m+2i-3}{4}}, & i=0,2,\cdots,\frac{m-5}{2}, \\ c_{\frac{m-1}{2}}+c_{\frac{3m+1}{4}}+d_{\frac{m-1}{2}}+d_{\frac{3m-3}{4}}, & i=\frac{m-1}{2}, \\ c_{\frac{m+2i-1}{4}}+c_{\frac{2i-m+1}{4}}+c_{\frac{m+i+1}{2}} & \\ \quad +d_{\frac{m+2i-1}{4}}+d_{\frac{2i-m-3}{4}}+d_{\frac{m+i-1}{2}}, & i=\frac{m+3}{2},\frac{m+7}{2},\cdots,m-3, \\ c_{\frac{3m-3}{4}}+c_{\frac{m-1}{4}}+d_{\frac{3m-7}{4}}+d_{\frac{m-5}{4}}, & i=m-1, \\ c_{\frac{3m+2i-1}{4}}+c_{\frac{i+1}{2}}+c_{\frac{m+2i+1}{4}} & \\ \quad +d_{\frac{3m+2i-1}{4}}+d_{\frac{i-1}{2}}+d_{\frac{m+2i-3}{4}}, & i=1,3,\cdots,\frac{m-3}{2}, \\ c_{\frac{2i-m-1}{4}}+c_{\frac{m+i}{2}}+c_{\frac{m+2i+1}{4}} & \\ \quad +d_{\frac{2i-m-1}{4}}+d_{\frac{m+i}{2}}+d_{\frac{m+2i-3}{4}}, & i=\frac{m+1}{2},\frac{m+5}{2},\cdots,m-2. \end{cases}$$

APPENDIX II

A Schematic of Our Multiplier Based On $x^7 + x + 1$, With $\{S\_i \mid 0 \le i \le 6\}$ As the Coefficients of Product