2009

Low Complexity Finite Field Multiplier for a New Class of Fields

Seyed Shahabi
University of Windsor

Low Complexity Finite Field Multiplier for a New Class of Fields

by

Seyed Mohammad Ali Shahabi

A Thesis

Submitted to the Faculty of Graduate Studies through Electrical and Computer Engineering in Partial Fulfillment

of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada

2009

Low Complexity Finite Field Multiplier for a New Class of Fields

by

Seyed Mohammad Ali Shahabi

APPROVED BY:

Dr. K. Li
Odette School of Business,

Dr. Kamal Tepe
Department of Electrical and Computer Engineering,

Dr. Huapeng Wu, Advisor
Department of Electrical and Computer Engineering,

Dr. R. Rashidzadeh, Chair of Defense
Department of Electrical and Computer Engineering,

Sept 23, 2009

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Finite fields is considered as backbone of many branches in number theory, coding theory, cryptography, combinatorial designs, sequences, error-control codes, and algebraic geometry. Recently, there has been considerable attention over finite field arithmetic operations, specifically on more efficient algorithms in multiplications. Multiplication is extensively utilized in almost all branches of finite fields mentioned above.

Utilizing finite field provides an advantage in designing hardware implementation since the ground field operations could be readily converted to VLSI design architecture. Moreover, due to importance and extensive usage of finite field arithmetic in cryptography, there is an obvious need for better and more efficient approach in implementation of software and/or hardware using different architectures in finite fields.
This project is intended to utilize a newly found class of finite fields in conjunction with the Mastrovito algorithm to compute the polynomial multiplication more efficiently.

To my wife, the love of my life.

Acknowledgments

There are several people who deserve my sincere thanks for their generous contributions to this project. I would first like to express my sincere gratitude and appreciation to Dr. Huapeng Wu, my supervisor for his invaluable guidance and constant support throughout the course of this thesis work.

In addition to my advisor, I would like to thank the rest of my thesis committee: Dr. Kamal Tepe from the electrical engineering department for his participation in my seminars, reviewing my thesis, and his constructive comments.

A special thanks goes to Dr. Ashkan Hosseinzadeh Namin, for proofreading this thesis.

Finally, my deepest gratitude goes to my family for their unconditional love, support and encouragement.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

OZP	One Zero Polynomial
BP	Bit-parallel architectures
BS	Bit-serial architectures
WL	Word-level architectures
AEDS	AND-Efficient Digit-Serial.
ASIC	Application-Specific Integrated Circuit.
BPWS	Bit-Parallel Word-Serial.
ESP	Equally Spaced Polynomial.
CAD	Computer Aided Design.
AOP	All One Polynomial.
GF	Galois Field.
IC	Integrated Circuit.
IEEE	Institute of Electrical and Electronics Engineers.
ISO	International Organization for Standardization.
LUT	Look-Up-Table.
NB	Normal Basis.
NIST	National Institute of Standards and Technology.
ONB	Optimal Normal Basis.
RB	Redundant Basis.
RNB	Reordered Normal Basis.
VLSI	Very-Large-Scale Integration.
XEDS	XOR-Efficient-Digit-Serial.

1 Introduction

1.1 Motivation

Until modern times cryptography referred almost exclusively to encryption, which is the process of converting ordinary information (plaintext) into unintelligible gibberish (i.e., ciphertext).[2] Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A cipher (or cypher) contains a pair of algorithms which create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a key. This is a secret parameter (ideally known only to the communicants) for a specific message exchange context. Keys are important, as ciphers without variable keys are trivially breakable and therefore less than useful for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.[19]

Modern cryptosystems could be classified into two categories, symmetric key encryption (where both parties use the same secret key) and public key encryption (where each party has a pair of keys: public key and private key). The first public key was invented in 1978 by Diffie and Hellman [3]. Although public key systems are computational intensive, slow and costly, they have advantages over symmetric key systems in that the former can provide security services such as key distribution/management and digital signature. Therefore, public key systems have attracted more attentions and been adopted into many security related standards.

Below is the list of security services using schemes introduced by cryptography.(X.800) [6]

- Authentication
  - Peer entity authentication
  - Data Origin Authentication
- Access Control
- Data Confidentiality
- Data integrity
- Non repudiation

Elliptic Curve and ElGamal are two of the most common public key cryptosystems, and both the systems can provide encryption, digital signature and key establishment. It is noted that both Elliptic curve and ElGamal systems are based on finite field computations. [6]

Since the security services provided by the cryptosystems require intensive finite field computation, this necessity sparked a need for scientists and engineers to come up with algorithms and architectures to perform finite field arithmetic more efficiently.

Each arithmetic operation used in finite fields has been the subject to improvements. Most importantly is effective and efficient computation of multiplication, since multiplication is the most used arithmetic operation in all cryptographic systems that finite fields are involved.

There are two major complexity measures we have adopted for discussion and comparison of various architectures for finite field arithmetic: Space complexity and critical time delay. Space complexity can be the number of logic gates required for a designed circuit, while critical time delay is usually measured by in the unit of delay caused by one gate.

Current research in this area has been focusing on finding new parallel finite field multiplication algorithms and architectures in order to further speed up the computation demanded by the various security services. It is important to mention that the fields of characteristic two are often chosen for hardware implementation because the ground field operations addition and multiplication can be readily implemented with VLSI XOR and AND gate, respectively.

1.2 Thesis Outline

The organization of the rest of this thesis is as follows:

Chapter 2 emphases are on a brief introduction of finite filed theory. In this section the basic definitions, elementary properties of finite fields and arithmetic algorithms will be discussed. In addition irreducible polynomials will be introduced, and a list of most used irreducible polynomial will be reviewed. Different multiplication algorithms one step and two step multiplication and their advantages and disadvantages will be demonstrated. In Chapter 3 two major architectures of multiplication will be reviewed in detail and compared. Chapter 4 will have an introduction to One Zero polynomials (OZP), in addition to in-depth usage of Mastrovito multiplication algorithm using OZP.

In Chapter 5 a brief comparison between most used irreducible polynomials in Mastrovito, is conducted. In this chapter the comparisons are conducted side by side and the improvements gained using one Zero Polynomials are indicated. In Chapter 6 conclusions and comparisons are made between the proposed multiplier and those based on pentanomials already in the literature. A few concluding remarks are also given.

2 Mathematical Background

2.1 Fundamental concepts

This section is intended to review the mathematical background on basic theorems and arithmetical functions in finite fields. In this section a brief review of ring, group and field will be conducted. In addition in this section it is intended to show most used basis and polynomial representation and their usages.

2.2 Finite Fields

Finite field or Galois field (named in honor of Évariste Galois) is a class of fields that contain only finitely many elements. The finite fields are classified by their size [4].

From a mathematical point of view, a finite field is a set of finite elements where one can add, subtract, multiply, and divide such that properties of associativity, distributivity, and commutativity are satisfied [1].

Most common representations of finite fields are:

- $GF(2^m)$, Binary extension field presentation
- GF(p) , Prime field presentation

Binary extension field representation is most desirable due to its proximity and ease of conversion to digital hardware implementation.

There exist different architectures in hardware and/or software for implementation of finite field multipliers.

Multipliers have gain special focus, due to their vital role and extensive use, in Cryptography and most algorithms in number theory. Ecommerce and credit based banking solely rely on safety and security established encryption provided by finite fields.

2.3 Groups, Rings and Fields

Definition 2.1.1. [5] A group (G,*) is a set G together with a binary operation * on G such that the following three properties hold:

1. The binary operator $*$ is associative; that is, for any $a, b, c \in$ G,

$$a * (b * c) = (a * b) * c$$

2. There is an identity (or unity) element e in G such that for all

a $\in$ G,

$$a * e = e * a = a$$

3. For each a $\in$ G, there exists an inverse element a^{-1} in G such that

$$a * a^{-1} = a^{-1} * a = e$$

If for all $a, b \in G$, $a * b = b * a$, then G is referred to as an Abelian or commutative group. A group with a finite number of elements is referred to as a finite group.

Definition 2.1.2. [5] A *ring* (r, +, $*$) is a set R together with two binary operations, denoted by + and $*$, such that the following three properties hold:

1. R is an abelian group with respect to +.

2. The binary operator $*$ is associative, which means for all

$$a, b, c \in R$$

$$a * (b * c) = (a * b) * c$$

3. The distribution law holds, which means for all $a, b, c \in R$

$$a * (b + c) = a * b + a * c$$

and

$$(b + c) * a = b * a + c * a$$

The identity element of the abelian group R with respect to + is called the *zero element*, while the identity element with respect to $*$ (if it exists) is called the *identity element*. A ring is called *commutative* if the binary operator $*$ is commutative.

Definition 2.1.3. [24] A *field* (f, +, $*$) is a set F together with two binary operations, denoted by + and $*$, such that the following two properties hold:

1. F is a commutative ring under + and $*$.
2. Nonzero elements of F from a group with the binary operation $*$.

A field with a finite number of elements is referred to as a *finite field*. The order of a finite field is the number of elements in the field. There exists a finite field F of order q if and only if q is a prime power, that is

$q = p^m$ where p is a prime number referred to as the *characteristic* of F and m is a positive integer [6].

For any prime power q, there is essentially only one finite field of order q.

This means that any two finite fields of order q are structurally the same, except that the labelling used to represent the field elements may be different. We say that any two finite fields of order q are *isomorphic*, and denote such a field by Fq^m or $GF(q^m)$ (GF stands for Galois Field, in honor of Evariste Galois, a French mathematician who is known for his work on the theory of equations and abelian integrals).[7]

2.4 Binary Fields and Bases

As mentioned in the introduction, there are several diverse representations of finite field basis, depending on the arithmetic necessities of the scheme utilized; one can choose the basis best fit.

In the next section; three most used bases will be demonstrated and their advantages/disadvantages will be explained in more detail.

It is possible to convert one form of basis to another with a cost.

2.5 Comparison of Bases

This section is intended to provide more information about the different Field representation, the structure and their most likely usage. Also a brief introduction on why and where each representation is used. The focus of this thesis is primarily dealing with polynomial basis.

In the next section of this thesis, more in-depth information is provided about the polynomial basis.

2.5.1 Polynomial Basis

Polynomial base is the most popular form of basis used, Due to their inherent proximity to digital logic make it possible for easy conversion to hardware and software implementations.

"In mathematics, the polynomial basis is a basis for finite extensions of finite fields.

Let $\alpha \in GF(p^m)$ be the root of a primitive polynomial of degree m over GF(p). The polynomial basis of $GF(p^m)$ is then

$\{0,1,\alpha,\ldots\ldots,\alpha^{m-1}\}$, The set of elements of GF(p^m) can then be represented as [3]: $\{0,1,\alpha,\alpha^2\ldots\ldots,\alpha^{p^{m-1}}\}$

A polynomial p(x) with the degree of m over GF(2), could be represented in polynomial form as:

$A(x) = \alpha_m x^m + \alpha_{m-1} x^{m-1} + \cdots + \alpha_2 x^2 + \alpha_1 x + \alpha_0$

In this polynomial the coefficients are all members of Galios Fields or GF(2)={0,1}

2.5.2 Normal Basis

Squaring operation is conducted effortlessly in some applications. Normal Basis is advantageous regarding this since squaring operation is trivial. Therefore in situations and architectures which require extensive squaring, Normal Basis are favored.

As indicated before this property allows for hardware efficient multipliers designed. The normal basis representation of $GF(2^4)$ is given in Appendix A.

" The normal basis theorem states that any Galois extension of fields has a normal basis. In the case of finite fields, this means that each of the basis elements is related to any one of them by applying the pth power mapping repeatedly, where p is the characteristic of the field. Let GF(p^m) be a field with pm elements, and β an element of it such that the m elements $\{\beta,\beta^p,\beta^{p^2},\ldots\ldots,\beta^{p^{m-1}}\}$ are linearly independent. Then this set forms a normal basis for GF(p^m).

This basis is frequently used in cryptographic applications that are based on the discrete logarithm problem such as elliptic curve cryptography. Hardware implementations of normal basis arithmetic typically have far less power consumption than other bases. "[5]

2.5.3 Dual Basis

Some architectures have been designed based on the Dual Basis, but most require extensive conversion in basis prior to any implementation. The Dual basis representation of $GF(2^4)$ is given in Appendix A.

"In linear algebra, a dual basis is a set of vectors that forms a basis for the dual space of a vector space. "[2]

2.5.4 Triangular Basis

There have been few algorithms which utilize the dual basis; however, using the Dual Basis in most cases requires additional base conversion. The algorithms that use Dual Basis employ the efficiency of this presentation for Bit-Serial in finite field multipliers.

2.5.5 Redundant basis

Redundant Basis algorithms utilize the squaring operations inherent to their Basis at no cost. In addition forgoing the modular reduction at a cost of expanding in a larger ring than the underlying field. Size of Cyclotomic ring underlying the field dictate the efficiency for redundant basis.

2.6 Irreducible Polynomials

Definition A polynomial p(x) over GF(2) of degree m is irreducible if p(x) is not divisible by any polynomial over GF(2) of degree less than m and greater than zero.[17]
On another note: A polynomial which cannot be factored or is not result of multiplication of two polynomials over the same field is considered to be an irreducible polynomial over that field.[8]
For example in finite field of Q[x]={A(x),B(x),.....C(x)}, f(x) is considered to be an irreducible polynomial if first f(x) could not be factored in and also there should not exist two polynomials in Q[x] where:

$f(x)=A(x)B(x)$

Similarly, in the finite field GF(2), x^2+x+1 is irreducible.

But x^2+1 is not, since

$(x+1)(x+1)= x^2+1 \pmod 2$.[8]

Calculation of irreducibility could become extensive which with aid of specific software programs this task could be accomplished with ease.

For the purpose of this thesis, the availability of irreducible polynomial over a new family of polynomials were compiled, this code is appended to Appendix B .

In addition to the code, the result is tabularized and attached to this project as Appendix C .

Feasibility of any multiplication algorithm relies heavily on the extent of domain it can be applied to. As indicated multiplications using Trinomials is the most efficient approach but it has limited domain.

2.7 Polynomial Basis

The emphasis of this project is on polynomial basis. in polynomial basis there are few classes of fields that efficient multiplier can be implemented, most common presentation could be listed as follows:

- All-One-Polynomials (AOP)

  An AOP of degree m has all terms from x^m to x^0 with coefficients of 1, and can be written as

$$AOP_m(x) = \sum_{i=0}^{m} x^i$$

  Or

$$AOP_m(x) = x^m + x^{m-1} + \cdots + x + 1$$

- Equally Spaced Polynomials (ESP)

  An ESP could be represented as:

$$ESP(x) = \sum_{i=0}^{m} x^{si}$$

  For

  $i = 0,1, \ldots., m$ or

$$ESP(x) = x^{sm} + x^{s(m-1)} + \cdots \ldots + x^s + 1$$

- Trinomials

  Trinomial is a polynomial consisting of three terms

  $f(x) = x^m + x^n + 1,$

  $m > n$

- Pentanomials

  Pentanomials is a polynomial consisting of five terms

  $F(x) = x^{k1} + x^{k2} + x^{k3} + x^{k4} + 1$,

  $m > K1 > k2 > k3 > k4$

As mentioned earlier extensive research has been conducted to come up with efficient multiplication scheme in polynomial families listed above.

Generally, there are two aspects in introduction of a new method for polynomial multiplication,

- Complexity (Number of AND and OR gates used to achieve the multiplication)
- Time complexity, (The time required to accomplish this task)

It is intended to compare the efficiency both in time and space complexity enhancement realized in this new approach.

In conclusion, this thesis will demonstrate that, employment of this approach will provide modest improvement over current most efficient multiplication schemes in addition to a wide availability.

Most efficient multipliers were constructed using Trinomials, AOP and ESP.

For the classes of fields that the Trinomials do not exist, Pentanomials are considered next best in class.

3 Previous Work in Finite Field Multiplier

3.1 Multiplier classes

In general, the multiplier implementation could be categorized in three different classes.

- Bit-parallel architectures
  - A Bit-parallel architecture is the fastest architecture possible, which multiplies two inputs in one clock cycle. Its main draw backs are large area utilization and high power consumption.

- Bit-serial architectures
  - A Bit-Serial architecture multiplier in a field of size m, takes m clock cycles to finish one multiplication operation. The main advantage of this class of multipliers is their low power consumption and area requirements. The main disadvantage is the time complexity.

- Word-level architectures

Out of the three mentioned architectures Word-Level multipliers offer the most architectural flexibility and the best advantage in regards to performance and possibility of materializing VLSI implementation, therefore most practical.
There are many approaches in multiplication of two polynomials in
GF (2^m) effectively and efficiently.

3.2 Conventional approach

In this section a conventional approach will be demonstrated using two simple polynomials. Also a high level hardware manipulation will be demonstrated.

It is important to mention conventional approach is considered a two step multiplication. Next section will only deal with the first step of multiplication which is only the term by term multiplication, to complete this task modular reduction needs to be implemented.

In the conventional, approach each member of the polynomial would be multiplied to the next polynomial, and then after the modularization would take place on the result of multiplication.
This approach also known as pen and pencil approach should be used only as academic basis not real problem solving approach.

Example:

Two-step Multiplication in $GF(2^m)$

Let m=3, f(x)=x^3+x+1, and A, B be two elements in $GF(2^3)$

$$A(x) = (a_2x^2 + a_1x + a_0),$$
$$B(x) = (b_2x^2 + b_1x + b_0),$$
$$and$$
$$C = A \times B \bmod f(x) = (c_2x^2 + c_1x + c_0)$$

Since,

$$A \times B = \boxed{(a_2b_2)x^4 + (a_1b_2 + a_2b_1)x^3} + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + (a_0b_1 + a_1b_0)x + a_0b_0$$

So the product C is solved as:

$$c_2 = a_2 b_2;$$
$$c_1 = a_2 b_2 + a_1 b_2 + a_2 b_1 + a_0 b_2 + a_1 b_1 + a_2 b_0;$$
$$c_0 = a_1 b_2 + a_2 b_1 + a_0 b_0$$

$$C = A \times B \bmod f(x)$$
$$C = A \times B \,|_{f(x)=0} = A \times B \,|_{x^3 = x+1, x^4 = x^2 + x}$$

$$= (a_2 b_2)(x^2 + x) + (a_1 b_2 + a_2 b_1)(x+1) + (a_0 b_2 + a_1 b_1 + a_2 b_0)x + a_0 b_0$$

$$= (a_2 b_2)x^2 + (a_2 b_2 + a_1 b_2 + a_2 b_1 + a_0 b_2 + a_1 b_1 + a_2 b_0)x + (a_1 b_2 + a_2 b_1 + a_0 b_0)$$

Visual Calculation of complexity

$$= (a_2b_2)x^2 + (a_2b_2 + a_1b_2 + a_2b_1 + a_0b_2 + a_1b_1 + a_2b_0)x + (a_1b_2 + a_2b_1 + a_0b_0)$$

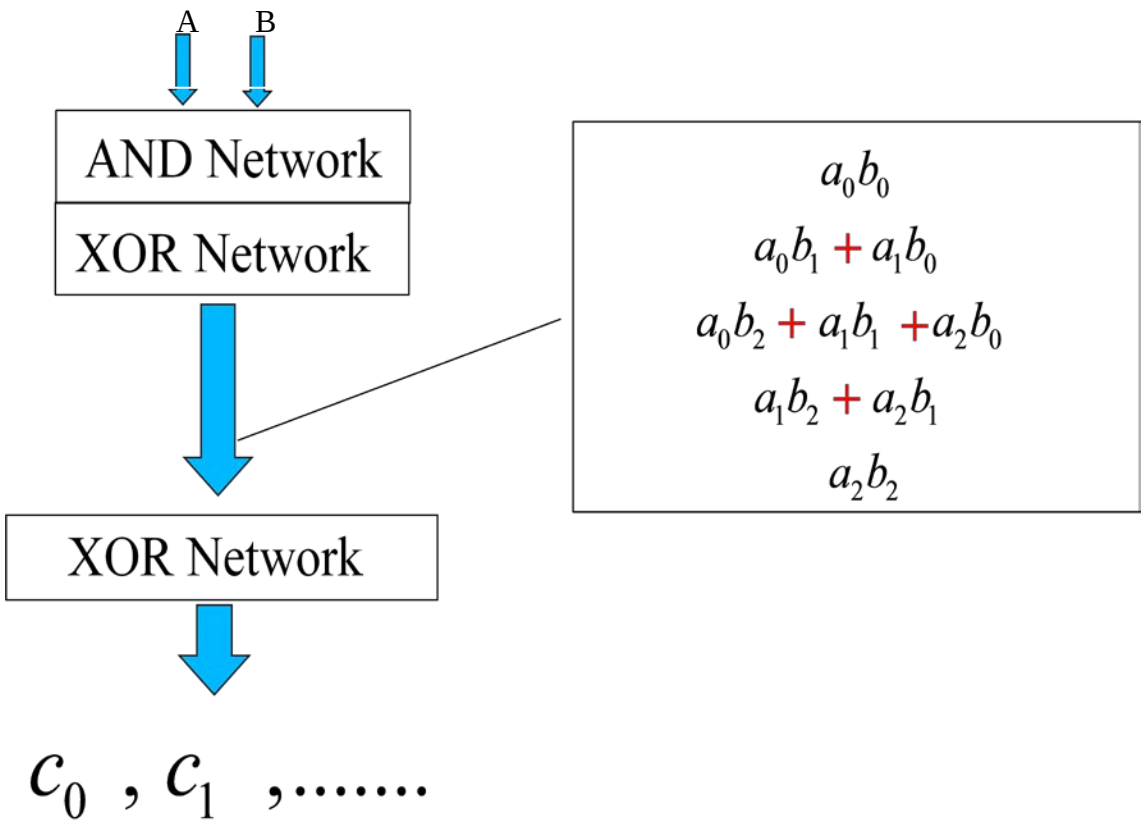

Figure 1: Graphical depiction of polynomial multiplication

3.3 Complexities for two-step multiplication

As shown in the previous section, we can conclude the complexity of multiplication algorithm for two step multiplication is subdivided to two sections:

- Space complexity (# of logic gates):
  - Step 1 requires m^2 AND gates and $(m-1)^2$ XOR gates
  - Complexities of Step 2 depends on f(x) or matrix T
- Time delay

- Step 1:

$$T_{step1} \le T_{AND} + \lceil \log_2 m \rceil T_{XOR},$$

T_{AND}, *denote the delay of an AND gate*

T_{XOR}, *denote the delay of an XOR gate*

The total time delay for the first step of multiplication, is the addition of both delays in the first step.

- Step 2: time delay again depends on f(x) the irreducible function.

3.4 Karatsuba-Ofman algorithm (KOA)

Employment of conventional multiplication approach, it would require n^2 or $\Theta(n^2)$ arithmetic operations in order to accomplish the multiplication of two multi digit numbers.(n^2 or $\Theta(n^2)$ also known as big O.

Discovered in 1960 by Dr. Karatsuba and published in a joint paper with Ofman in 1962.

KOA is a divide and conquer form algorithm, that divides the operands in two parts with less number of digits (half number of digits) and forms the final result with the help of the product of these parts.[9]

Using this approach the arithmetic operations for multiplication was reduced to:

$$\Theta(n^{\log_2 3})$$

Consider two degree 1 polynomials A(x) and B(x).

$A(x) = a_1x + a_0$

$B(x) = b_1x + b_0$

Let D_0, D_1, $D_{0,1}$ be auxiliary variables with

$D_0 = a_0b_0$

$D_1 = a_1b_1$

$D_{0,1} = (a_0 + a_1)(b_0 + b_1)$

Then the polynomial $C(x) = A(x)\ B(x)$ can be calculated in the following way:

$C(x) = D_1x^2 + (D_{0,1} - D_0 - D_1)x + D_0$

KOA could be used in recursive mode and applied for any degree m, utilizing the scheme will yield more gate savings with longer delay. KOA is most efficient if the degree of polynomials is a power of 2. KOA produces overlapping polynomial terms. These overlapping terms come from the product of the three terms in the KOA formula.

3.5 Applying Karatsuba-Ofman algorithm

KOA is defined as two step multiplication scheme, which is comprised of first polynomial multiplication and second reduction modulo of irreducible polynomial. KOA only enhances the first step of polynomial multiplication. A main advantage of KOA approach could be contributed to its recursive possibility. Only condition for using the KOA recursively is to have the polynomials with degree of n-1 and power of 2. It will be shown that for polynomials of different degrees, one can pad zeros to use the recursive function of KOA.

A brief example of KOA multiplication for A(x), B(x) in a field of GF(2^n) is shown below:

$$Let: \quad A(x) = a_{n-1}x^{n-1} + ... + a_1x^1 + a_0$$
$$B(x) = b_{n-1}x^{n-1} + ... + b_1x^1 + b_0$$

A(x),B(x) could be written as:

$$A(x) = x^{\frac{n}{2}}(a_{n-1}x^{\frac{n}{2}-1} + ... + a_{\frac{n}{2}}) + (a_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + ... + a_1x^1 + a_0) = x^{\frac{n}{2}}A_H + A_L$$

$$B(x) = x^{\frac{n}{2}}(b_{n-1}x^{\frac{n}{2}-1} + ... + b_{\frac{n}{2}}) + (b_{\frac{n}{2}-1}x^{\frac{n}{2}-1} + ... + b_1x^1 + b_0) = x^{\frac{n}{2}}B_H + B_L$$

Now let's try the multiplication using the new notation.

$$A(x)B(x) =$$

$$x^n A_H B_H + \{(A_H + A_L)(B_H + B_L) - (A_H B_H + A_L B_L)\}x^{\frac{n}{2}} + A_L B_L$$

$$\xrightarrow{Therefore}$$

$$= x^n A_H B_H + \{(A_H B_H + A_L B_L + A_L B_H + A_H B_L) - (A_H B_H + A_L B_L)\}x^{\frac{n}{2}} + A_L B_L$$

Since in finite field of degree of 2 the additions and subtractions yield the same result, it is possible to write:

$$A(x)B(x) =$$

$$x^n A_H B_H + \{(A_H B_H + A_L B_L + A_L B_H + A_H B_L) - (A_H B_H + A_L B_L)\} x^{\frac{n}{2}} + A_L B_L$$

The addition and subtraction are the same when using the Galios fields, GF(2).
Using the conventional approach the number of gates used to accomplish this task is:

$$\text{\# of AND gates} = n^2 \qquad (1)$$

$$\text{\# of XOR gates} = (n-1)^2 \qquad (2)$$

Where the two input XOR gates accomplish the coefficient's addition, and coefficient multiplication is accomplished with two input AND gates.

Employing the KOA approach reduces the complexity in multiplication, where the number of gates for the same operation is listed below:

$$\text{\# of AND gates} = \frac{3}{4}n^2 \qquad (3)$$

$$\text{\# of XOR gates} = \frac{3}{4}n^2 + n - 1 \qquad (4)$$

At the first glance it could be deduced that using KOA in polynomial multiplication reduces the number of AND gates and the expense of XOR gate, this deduction could be better quantified in higher orders and also if this scheme used in recursive mode.
In order to generalize this conclusion and also calculate the general form of KOA polynomial multiplication savings we can present the multiplication as : $A_H B_H$, $A_L B_L$ and $(A_H + A_L)(B_H + B_L)$.

Also it has been shown [13] that the KOA could be recursively applied in a polynomial multiplication.

In addition it is demonstrated that for a parallel implementation of this design in VLSI the total complexity of AND gates and XOR gates could be calculated as:

# of AND gates = $n^{\log_2 3}$ (5)

# of XOR gates = $6m^{\log_2 3} - 8m + 2$ (6)

Observing the results mentioned above, in order to benefit from the KOA recursive algorithm, we need to have at least $n \geq 2$ to improve the number of AND gate complexity and $n \geq 64$ in order to reduce the XOR complexity. [13]

In addition to improvements in space complexity, reduction in time delay will be rewarded using recursive KOA algorithm.

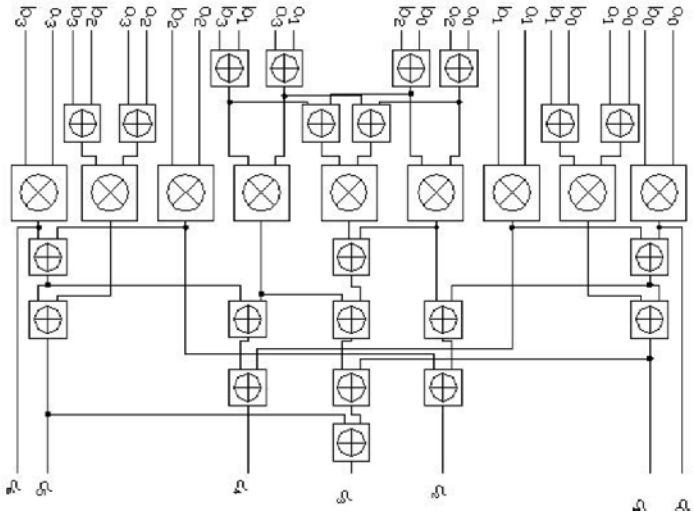

Figure 2: KOA Implementation of Multiplication of degree 4[13]

The figure 2 is depiction of implementation of This is the implementation of KOA multiplier for n=2^2. The time delay for this implementation is be calculated $6T_A$ which equals to $(3\log_2 n)T_X$.[13]

3.6 Second step (Reduction Modulo)

As indicated KOA Polynomial Multiplication is accomplished in two steps, KOA for multiplication, modulo reduction.

There are multiple algorithms introduced for modulo reduction, using matrix for modulo reduction is most used, this matrix is constructed with the irreducible function for the reduction modulo.

The efficiency of the second step (modulo reduction) directly depends on the irreducible function depicted for reduction .

Below is the simplified version of The Reduction modulo for the irreducible polynomial multiplication in GF(2^m), where T is the Multiplication matrices.

$$\begin{bmatrix} x^{m} \\ x^{m+1} \\ x^{m+2} \\ . \\ . \\ x^{2m-2} \end{bmatrix}_{(m-1)\times 1} = T_{(m-1)\times m} \times \begin{bmatrix} x^{m-1} \\ x^{m-2} \\ x^{m-3} \\ . \\ . \\ 1 \end{bmatrix}_{m\times 1}$$

For $f(x) = x^m + x + 1$ is a trinomial, Then T is formulated:

$$T_{(\text{m-1})\times m} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & . & .0 & . & 0 \\ 0 & 1 & 1 & 0 & 0 & . & .0 & . & 0 \\ 0 & 0 & 1 & 1 & 0 & . & .0 & . & 0 \\ 0 & 0 & 0 & 1 & 1 & . & .0 & . & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & & \dots \\ 0 & 0 & 0 & 0 & 0 & . & .1 & . & 0 \\ 0 & 0 & 0 & 0 & 0 & . & .1 & . & 1 \end{bmatrix}_{(m-1)\times m}$$

The complexity weigh of T the transformation matrix is calculated according to the number of 1's in the transfer function. It is vital to mention that trinomials convey the highest efficiency within the transfer functions.

The complexity of modularization with trinomials could be calculated as 2(m-1) XOR gates, which is significantly fewer as compared to the general form of f(x) which required $O(m^2)$ XOR gates.

As previously indicated, the most efficient polynomials f(x) are listed below:

- Trinomial
- All-one-polynomials (AOP),
- Equally spaced polynomials (ESP),

Note that the polynomials of above forms do not exist for all degrees of m.

3.7 Issues with KOA

KOA's limitation is that it needs to be applied when the polynomial is of the degree of 2.In instances which the polynomials do not consist of even number of terms then it becomes necessary to pad 0's, if KOA is to be employed.

A second issue is the overlapping of the polynomial terms, this will cause in large gate delays. It will be shown that the number of overlapping terms using KOA could be up to 2^i-2. This overlap occurs due to multiplication of three terms in KOA.

$$A = a_3x^3 + a_2x^2 + a_1x^1 + a_0 = x^2(a_3x + a_2) + (a_1x + a_0)$$
$$B = b_3x^3 + b_2x^2 + b_1x^1 + b_0 = x^2(b_3x + b_2) + (b_1x + b_0)$$

$$\begin{aligned} C = A \times B = x^2(a_3x + a_2)(b_3x + b_2) + \\ \{((a_3x + a_2) + (a_1x + a_0))((b_3x + b_2) + (b_1x + b_0)) \\ -((a_3x + a_2)(b_3x + b_2) + (a_1x + a_0)(b_1x + b_0))\}x \\ + (a_1x + a_0)(b_1x + b_0) \end{aligned}$$

[13]

3.8 Mastrovito algorithm

Multiplication in finite field is consisted of two parts. Primarily the actual multiplication and then the modular reduction. There are few schemes that combine the two steps into one operation.

Mastrovito is one possible approach which combines the two steps into one.

Mastrovito could be briefly described as:

- One step multiplication which performs Multiplication and modular reduction in one step
- To accomplish the multiplication and reduction the goal is to write C(x) as a function of B(x), where matrix Z is what we need to solve.

$$C = \begin{bmatrix} c_0 \\ c_1 \\ . \\ . \\ c_{n-1} \end{bmatrix} = ZB = \begin{bmatrix} z_{0,0} & . & . & z_{0,n-1} \\ . & . & . & . \\ . & . & . & . \\ z_{n-1,0} & . & . & z_{n-2,n-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ . \\ . \\ b_{n-1} \end{bmatrix}$$

Each element $z_{i,j}$ *can be defined as follows*:

$$z_{i,j} = \begin{cases} a_i & ;\ j = 0\ ;\ i = 0,\ldots,m-1 \\ u(i-j)a_{i-j} + \sum_{t=0}^{j-1} q_{j-1-t}, a_{m-1-t} & ;\ j = 1,\ldots,m-1\ ;\ i = 0,\ldots,m-1 \end{cases}$$

- Where $q_{i,j}$ is defined as

$$\begin{bmatrix} x^{m} \\ x^{m+1} \\ . \\ . \\ x^{2m-2} \end{bmatrix} = \begin{bmatrix} q_{0,0} & q_{0,1} & . & q_{0,m-1} \\ q_{1,0} & q_{1,1} & . & q_{1,m-1} \\ . & . & . & . \\ . & . & . & . \\ q_{m-2,0} & q_{m-2,1} & . & q_{m-2,m-1} \end{bmatrix} \begin{bmatrix} 1 \\ x \\ . \\ . \\ x^{m-1} \end{bmatrix} \text{ mo } \quad f(x)$$

3.9 Efficient classes of fields for Mastrovito

Mastrovito algorithm efficiency and complexity is depended on the irreducible polynomial available for the field, and what irreducible polynomial is chosen. Below are the most efficient irreducible polynomials which Mastrovito could enhance.

a. Trinomials :

   i. $f(x) = x^{m} + x^{n} + 1$

b. Equally spaced polynomials (EPS)

   i. $f(x) = x^{k\Delta} + x^{(k-1)\Delta} + \cdots + x^{\Delta} + 1$

c. All one polynomials (AOP)

   i. $f(x) = x^{m} + x^{m-1} + x^{m-2} \ldots + x^{1} + 1 + 1$

d. Pentanomials

i. $f(x) = x^m + x^{k1} + x^{k2} + x^{k3} + 1$

where m > k1 > k2 > k3

3.10 Availability of irreducible polynomial

- For this purpose, a code was written to test the availability of one zero polynomial from GF(25) to GF(2800).
- The availability of one zero polynomial was found to be close to %100.
- OZP did not exist for some OZP's position of zero.

3.11 Summary of the previous related work

In this thesis, a new architecture and implementation will be proposed in a new class of binary fields for bit-parallel multiplier described on GF(2^m).

The trinomials are the best in class fields used for finite fields multiplication, but the trinomials do not cover all the degrees. Pentanomials are considered the next best in class for polynomial multiplication where the trinomials do not exist.

There also exist other classes of fields where multiplication is efficient (not as efficient as Trinomial) but also do not cover all ranges of m.

- Trinomials :
    - $f(x) = x^m + x^n + 1$
- Pentanomials
    - $f(x) = x^m + x^n + x^p + x^q + 1$ where m>n>p>q
- Equally spaced polynomials (EPS)
    - $f(x) = x^{k\Delta} + x^{(k-1)\Delta} + \ldots + x^{\Delta} + 1$
- All one polynomials (AOP)
    - $f(x) = x^m + x^{m-1} + \ldots + x^1 + 1$

As indicated a new family of finite fields will be introduced in this thesis. This new class of binary finite fields is generated with irreducible One Zero Polynomial (OZP).

Essentially this new class of finite field is the same as All One Polynomial with one coefficient set to zero.

It will be demonstrated that using Mastrovito algorithm with this class of family “OZP”, will provide more efficient multiplication algorithm with lower complexity as compared to multipliers based on irreducible Pentanomials.

Next we present a comparison regarding area complexity for previous proposals.

The Table1 briefly describes the number of XOR gates needed for implementing the multiplication using Mastrovito algorithm with different families of irreducible polynomial:

Table 1: Complexity Comparison of Mastrovito Multipliers

Polynomial	Complexity (XOR)	Reference
Trinomial	$m^2 - 1$	[10][11][13][14][15]
EST	$m^2 - \frac{m}{2}$	[15]
AOP	$m^2 - 1$	[12]
ESP	$\mathrm{m}^2 - \Delta$	[11]
Pentanomials	m2+2m-3	[11]
General	$(m-1)(m+k-1) + \sum_{j\epsilon s}(2m-1-j)$	[11]

The table above indicates that the best in class were Trinomials, EST, AOP and ESP is ranked after.

4 Main Results

4.1 Motivation

The motivation for this project is to come up with a new scheme using a new class of finite fields. This class should cover close to 100% GF(2^m) and also be more efficient than the pentanomials.

As indicated through this thesis, the new class of finite fields introduced is Called One Zero Polynomial (OZP).

Multiplication using this class of fields (OZP) is not as efficient as where trinomials exist, (Trinomials cover about 70% and Pentanomials cover the other 30%), however for the classes of fields that trinomials don't exist, OZP performance will be shown to be more efficient than Pentanomials.

4.2 One Zero Polynomial presentation

OZP, (One Zero Polynomials) is a newly found class of polynomials that have potential to replace Pentanomials for more efficient computations. The irreducible polynomial used to create OZP is defined as follows:

$$f(x) = x^m + \sum_{i=1}^{m-1} f_j x^i + 1$$

$$Where \quad f_i = \begin{cases} 0 & For\ some \quad j,\ 1 \le j < m \\ 1 & Otherwise \end{cases}$$

As it can be observed OZP irreducible polynomial is similar to the one in the All One Polynomial. The exception is that in OZP only one coefficient is equal to zero.

4.3 Availability of OZP irreducible polynomial

The main reason that any other family of polynomial is used instead of trinomials is the availability of trinomials is limited. Therefore pentanomials were being used which cover all the degrees that trinomials did not exist. Therefore, any other family of polynomials being used must possess these criteria. To show the availability of this option a code was written and simulated. The result for availability of one zero polynomials for m<800 was found to be almost 100%.

The code and the simulation results are presented in tabular format in appendix C.

4.4 Multiplication using OZP

Let the finite field F_2^m be generated with an irreducible m-term polynomial f(x).

Assuming the elements A(x) and B(x) are to be multiplied. The first step is to calculate the raw multiplication result which is called C(x). The next step is to use the irreducible polynomial to accomplish modularization.

Mastrovito algorithm as it was explained earlier in Chapter 2, as indicated this scheme consolidates the two steps into one. Mathematically multiplication of two elements using OZP can be defined as follows:

$$Let \quad A(x) = a_0 + a_1 x^1 + a_2 x^2 ..... + a_{m-1} x^{m-1} \quad or$$

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

$$B(x) = b_0 + b_1 x^1 + b_2 x^2 ..... + b_{m-1} x^{m-1} \quad or$$

$$B(x) = (\sum_{i=0}^{m-1} b_i x^i)$$

$$Where \quad S(x) = A(x) \times B(x)$$

$$S(x) = s_0 + s_1 x^1 + s_2 x^2 ..... + s_{m-1} x^{m-1} \quad or$$

$$S(x) = (\sum_{i=0}^{m-1} s_i x^i)$$

$$F(x) = x^m + \sum_{i=1}^{m-1} f_i x^i + 1$$

$$Irreducible\ OZP\ \ polynomials\ were\ only\ one\ \ f_i = 0$$

$$Reduction\ modulo\ the\ irreducible\ polynomial:$$

$$C(x) = \quad S(x) \quad mod \quad f(x)$$

4.5 OZP and Mastrovito

Instead of multiplying and then applying the reduction modulo to the system, in Mastrovito a Z matrix is introduced. This matrix is product of both B(x) and F(x), (one of the input operands and the irreducible polynomial).

To calculate the Z matrix there are many approaches, in next sections these algorithms will be reviewed.

4.6 Calculation of Mastrovito with OZP

$$S(x) = a(x)b(x) = \left(\sum_{i=0}^{m-1} a_i x^i\right)\left(\sum_{j=0}^{m-1} b_j x^j\right) = \sum_{i=0}^{2m-2} s_i x^i$$

Coefficient S_i can be expanded *as*:

$$S_i = \begin{cases} \sum_{j=0}^{i} a_j b_{i-j} & 0 \le i \le m-1 \\ \sum_{j=i-m+1}^{m-1} a_j b_{i-j} & m \le i \le 2m-2 \end{cases}$$

$$= \begin{cases} \sum_{j=0}^{i} a_{i-j} b_j & 0 \le i \le m-1 \\ \sum_{j=i-m+1}^{m-1} a_{i-j} b_j & m \le i \le 2m-2 \end{cases}$$

$$C(x) = S(x) \text{ mo } \mathrm{d}f(x) = \left[\sum_{i=0}^{m-1} S_i x^i + \sum_{i=m}^{2m-2} S_i x^i\right] \text{mo } \mathrm{d}f(x)$$

Further expansion of S_i we have,

$$= \sum_{i=0}^{m-1}\left(\sum_{j=0}^{i} a_{i-j} b_j\right) x^i \underbrace{\quad}_{No \bmod req.} + \left[\sum_{i=m}^{2m-2}\left(\sum_{j=i-m+1}^{m-1} a_{i-j} b_j\right) x^i\right] \bmod f(x)$$

$$= \sum_{i=0}^{m-1}\left(\sum_{j=0}^{i} a_{i-j} b_j\right) x^i + \underbrace{\left[\sum_{i=0}^{m-2}\left(\sum_{j=i+1}^{m-1} a_{i+m-j} b_j\right) \underbrace{x^{m+i}}_{\text{using this for substitution}}\right] \bmod f(x)}_{i=i-m}$$

to calculate x^{m+i}, *one should note that,*

$$\begin{bmatrix} x^{m} \\ x^{m+1} \\ \vdots \\ x^{m+i} \\ \vdots \\ x^{2m-2} \end{bmatrix}_{(m-1)\times 1} = T.\Psi = \begin{bmatrix} t_{0,0} & \cdots & t_{0,m-1} \\ t_{1,0} & \cdots & t_{1,m-1} \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ t_{m-2,0} & \cdots & t_{m-2,m-1} \end{bmatrix}_{(m-1)\times m} \begin{bmatrix} x^{m-1} \\ x^{m-2} \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix}_{m\times 1}$$

$$T.\Psi = \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_i \\ \vdots \\ T_{m-2} \end{bmatrix}_{m\times 1} .\Psi$$

where $T_i = (t_{i,0}, t_{i,1}, t_{i,2}, \cdots, t_{i,m-1})$

$$X^{m+i} = T_i.\Psi = \sum_{j=0}^{m-1} t_{i,j}.x^{m-1-j} \qquad 0 \le i \le m-2$$

$$X^{m+i} = \sum_{j=0}^{m-1} t_{i,m-1-j}.x^{j}$$

Now replacing X^{m+i}, *we have:*

$$C(x)=\sum_{i=0}^{m-1}(\sum_{j=0}^{i}a_{i-j}b_j)x^i+\underbrace{\left[\sum_{i=0}^{m-2}(\sum_{j=i+1}^{m-1}a_{i+m-j}b_j)\underbrace{x^{m+i}}_{\text{using this for substitution}}\right]}_{i=i-m}\ \text{mo}\ \ \text{d}f(x)$$

$$=\sum_{i=0}^{m-1}(\sum_{j=0}^{i}a_{i-j}b_j)x^i+\sum_{i=0}^{m-2}(\sum_{j=i+1}^{m-1}a_{i+m-j}b_j)\sum_{k=0}^{m-1}t_{i,m-1-k}.x^k$$

Exchange of i and k

$$\underset{k}{\overset{i}{\rightleftharpoons}}=\sum_{i=0}^{m-1}(\sum_{j=0}^{i}a_{i-j}b_j)x^i+\sum_{i=0}^{m-1}\left[\sum_{k=0}^{m-2}\left[\sum_{j=k+1}^{m-1}a_{m+k-j}b_j\right]t_{k,m-1-i}\right]x^i$$

$$C(X)=\sum_{i=0}^{m-1}\left[\sum_{j=0}^{i}a_{i-j}b_j+\sum_{k=0}^{m-2}(\sum_{j=k+1}^{m-1}a_{m+k-j}b_j)t_{k,m-1-i}\right]x^i=\sum_{i=0}^{m-1}c_ix^i$$

$$C_i=\underbrace{\sum_{j=0}^{i}a_{i-j}b_j}_{\textit{This part is the U matrix}}+\sum_{k=0}^{m-2}(\sum_{j=k+1}^{m-1}a_{m+k-j}b_j)t_{k,m-1-i}$$

$$Then\ the\ C_i\ coefficients\ are\ equal\ to:$$

$$C_i = \underbrace{\sum_{j=0}^{i} a_{i-j} b_j}_{This\ part\ is\ the\ U\ matrix} + \sum_{j=1}^{m-1} a_{m-j} \,.t_{0,m-1-i} \,.b_j$$

$$+ \sum_{j=2}^{m-1} a_{m+1-j} \,.t_{1,m-1-i} \,.b_j$$

$$+ \sum_{j=3}^{m-1} a_{m+2-j} \,.t_{2,m-1-i} \,.b_j$$

$$\vdots$$

$$+ \sum_{j=m-1}^{m-1} a_{2m-2-j} \,.t_{m-2,m-1-i} \,.b_j$$

As indicated the Mastrovito algorithm in matrix format is:

$$C = Z.B$$

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-2} \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{m-2} \\ z_{m-1} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{bmatrix} \qquad Where\ Z_i = \left(z_{i,0}, z_{i,1}, \cdots, z_{i,m-1} \right)$$

$$C_i = Z_i \,. \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{bmatrix} = \sum_{j=0}^{m-1} z_{i,j} b_j$$

$$Now\ \text{decomposing}\ Z\ \text{in}\ to\ Z = U + V_0 + V_1 + \cdots + V_{m-2}$$

$$C_i = \underbrace{\sum_{j=0}^{m-1} u_{i,j}.b_j}_{U} + \underbrace{\sum_{j=0}^{m-1} v_{i,j}^{(0)}.b_j}_{V_0} + \underbrace{\cdots\cdots\cdots}_{V_x} + \underbrace{\sum_{j=0}^{m-1} v_{i,j}^{(m-2)}.b_j}_{V_{m-2}}$$

Or in matrix format we have:

$$UB = \begin{bmatrix} a_0 & 0 & \cdots & \cdots & \cdots & 0 \\ a_1 & a_0 & 0 & \ddots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & 0 & 0 \\ a_{m-1} & a_{m-2} & \cdots & \cdots & a_1 & a_0 \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$$V_1 B = \begin{bmatrix} 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$i = 0,1,2$

$$c^{v_1} = \sum_{j=2}^{m-1} a_{m+1-j} \,.\, b_j = a_{m-1}.b_2 + a_{m-2}.b_3 + \cdots + a_2.b_{m-1}$$

$$for\ i = 3\ to\ m-1 \xrightarrow{all\ zeros} C^{v_1} = 0$$

$$V_2\ B \quad = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & a_{m-1} & \cdots & a_2 \\ 0 & 0 & 0 & a_{m-1} & \cdots & a_2 \\ 0 & 0 & 0 & a_{m-1} & \cdots & a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$i = 1, 2, 3$

$$c^{v_2} = \sum_{j=3}^{m-1} a_{m+2-j}\ .\ b_j = a_{m-1}\ .b_3 + a_{m-2}\ .b_4 + \cdots + a_3\ .b_{m-1}$$

$$for\ \ i = 0\ \ and\ \ 4\ to\ m-1 \xrightarrow{all\ zeros} C^{v_2} = 0$$

$$V_{m-2}B = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \cdots & \cdots & a_{m-1} \\ 0 & 0 & 0 & \cdots & \cdots & a_{m-1} \\ 0 & 0 & 0 & \cdots & \cdots & a_{m-1} \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$i = m-3, m-2, m-1$

$$c^{v_{m-2}} = \sum_{j=m-1}^{m-1} a_{2m-2-j}\ .\ b_j = a_{m-1}\ .b_{m-1}$$

$$for\ \ i = 0\ \ to\ \ m-4 \xrightarrow{all\ zeros} C^{v_{m-2}} = 0$$

$$V_0 B = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$$t_{0,m-1-i} = \begin{cases} 0 & i = 1 \\ 1 & Otherwise \end{cases}$$

$$c^{v_0} = \sum_{j=1}^{m-1} a_{m-j} \,.\, t_{0,m-1-i} \,.\, b_j$$

$$C^{v_0} = \begin{cases} 0 & i = 1 \\ \sum_{j=1}^{m-1} a_{m-j} \,.\, b_j & Otherwise \end{cases}$$

$$\xrightarrow{i \neq 1} c^{v_0} = \sum_{j=1}^{m-1} a_{m-j} \,.\, b_j = a_{m-1} \,.\, b_1 + a_{m-2} \,.\, b_2 + \cdots + a_1 \,.\, b_{m-1}$$

4.7 Complexity in OZP multiplication

Assume that $C(x) = A(x) \times B(x)$. *Then the relationship between* $a_i, b_i,$ *and* c_i *is as follows*:

$$c_i = \left[\left[\sum_{j=0}^{i} a_{i-j} b_j + \sum_{k=0}^{m-2} \left[\sum_{k+1}^{m-1} a_{m+k-j} b_j \right] \right] t_{k,m-1-i} \right]$$

where $t_{i,j}$ *is defind as*:

$$\begin{bmatrix} x^m \\ x^{m+1} \\ x^{m+2} \\ \vdots \\ x^{2m-2} \end{bmatrix} = \mathbf{T}_{(\mathbf{m-1})\times\mathbf{m}} \times \begin{bmatrix} x^{m-1} \\ x^{m-2} \\ x^{m-3} \\ \vdots \\ 1 \end{bmatrix}$$

or

$$\underbrace{x^{m+i} \bmod f(x) = \sum_{k=0}^{m-1} t_{i,m-1-k} x^k}_{Exchanging \quad k \quad with \quad i}$$

From Mastrovito we had,

$C = ZB \quad where\ Z = f(A(x), f(x))$:

$$c_i = Z_i \times \begin{bmatrix} b_0 \\ . \\ . \\ b_{m-1} \end{bmatrix} = \sum_{j=0}^{m-1} z_{i,j} b_j$$

Decomposing Z into:

$$Z = U + V^{(0)} + V^{(1)} + ...V^{(m-2)}$$

$$\Rightarrow c_i = \sum_{j=0}^{m-1} U_{i,j} b_j + \sum_{j=0}^{m-1} V_{i,j}^{(0)} b_j + ... + \sum_{j=0}^{m-1} V_{i,j}^{(m-2)} b_j$$

4.8 Format of U and V matrices where a_1=0

$$U = \begin{bmatrix} a_0 & 0 & . & . & . & 0 \\ a_1 & a_0 & 0 & . & 0 & 0 \\ . & . & . & . & 0 & 0 \\ a_{m-1} & a_{m-2} & . & . & a_1 & a_0 \end{bmatrix}$$

$$V_0 = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & a_{m-3} & . & a_2 \\ 0 & 0 & 0 & 0 & . & 0 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & . & a_2 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & . & a_2 \\ . & . & . & . & . & . \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & . & a_2 \end{bmatrix}$$

$$V_2 = \begin{bmatrix} 0 & 0 & 0 & . & 0 & 0 \\ 0 & 0 & 0 & a_{m-1} & . & a_2 \\ 0 & 0 & 0 & a_{m-1} & . & a_2 \\ 0 & 0 & 0 & a_{m-1} & . & a_2 \\ 0 & 0 & 0 & . & 0 & 0 \\ . & . & . & . & . & . \end{bmatrix}$$

$$V_{m-2} = \begin{bmatrix} 0 & 0 & 0 & . & 0 & 0 \\ 0 & 0 & 0 & . & 0 & 0 \\ . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & a_{m-1} \\ 0 & 0 & 0 & . & . & a_{m-1} \\ 0 & 0 & 0 & . & . & a_{m-1} \end{bmatrix}$$

4.9 Area Complexity of the Proposed Multiplier

The area complexity of each decomposed matrix (UB and V_xB_x) is calculated and then added to find the total area complexity of this proposed multiplier.

$$Z = U + V_0 + V_1 + \cdots + V_{m-2}$$

$$C = Z \ . \ B = UB + V_0 B + VB_1 + \cdots + V_{m-2} B$$

$$Complexity:$$

$$UB = \begin{bmatrix} a_0 & 0 & \cdots & \cdots & \cdots & 0 \\ a_1 & a_0 & 0 & \ddots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & 0 & 0 \\ a_{m-1} & a_{m-2} & \cdots & \cdots & a_1 & a_0 \end{bmatrix}_{m \times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix}_{m \times 1}$$

$$\begin{cases} Number\ of\ \ AND\ Gates: 1 + 2 + 3 + \cdots + m = \dfrac{m^2}{2} + \dfrac{m}{2} \\ Number\ of\ \ XOR\ Gates: 0 + 1 + 2 + 3 + \cdots + (m-1) = \dfrac{m^2}{2} - \dfrac{m}{2} \end{cases}$$

$Complexity:$

$$V_0 B = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_2 \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$$\begin{cases} Number\ of\ \ AND\ Gates: m-1 \\ Number\ of\ \ XOR\ Gates: (m-2)+(m-1) \end{cases}$$

$Complexity:$

$$V_1 B = \begin{bmatrix} 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & a_{m-1} & a_{m-2} & \cdots & a_2 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$$\begin{cases} Number\ of\ \ AND\ Gates: m-2 \\ Number\ of\ \ XOR\ Gates: (m-3)+3 \end{cases}$$

$Complexity:$

$$V_{m-2} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \cdots & \cdots & a_{m-1} \\ 0 & 0 & 0 & \cdots & \cdots & a_{m-1} \\ 0 & 0 & 0 & \cdots & \cdots & a_{m-1} \end{bmatrix}_{m\times m} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{m-1} \end{bmatrix}_{m\times 1}$$

$$\begin{cases} Number\ of\ \ AND\ Gates: 1 \\ Number\ of\ \ XOR\ Gates: 0+3 \end{cases}$$

$Total\ Complexity:$

$Total\ Number\ of\ \ AND\ gates:$

$$= UB_{(\#of\ AND GATES)} + V_0 B_{(\#of\ AND GATES)} + \cdots + V_{m-2} B_{(\#of\ AND GATES)}$$

$$\xrightarrow{Total\ \#\ AND GATES} \frac{m^2}{2} + \frac{m}{2} + (m-1) + (m-2) + \cdots + 1$$

$$= \frac{m^2}{2} + \frac{m}{2} + (\frac{m^2}{2} - \frac{m}{2}) = m^2$$

$Total\ Complexity:$

$Total\ Number\ of\ XOR\ gates:$

$$= UB_{(\#of\ XOR\ GATES)} + V_0 B_{(\#of\ XOR\ GATES)} + \cdots + V_{m-2} B_{(\#of\ XOR\ GATES)}$$

$$\xrightarrow{Total\ \#\ AND\ GATES} \left(\frac{m^2}{2} - \frac{m}{2}\right) + [(m-2) + (m-1)] + [(m-3) + 3]$$

$$+ [(m-4) + 3] + \cdots + [0 + 3]$$

$$= \left(\frac{m^2}{2} - \frac{m}{2}\right) + [1 + 2 + \cdots + (m-2)] + (m-1) + 3 \times (m-2)$$

$$= \frac{m^2}{2} - \frac{m}{2} + \left(\frac{m^2}{2} - \frac{m}{2}\right) + 3m - 6$$

$$= m^2 - m + 3m - 6$$

$$= m^2 + 2m - 6$$

$$Z = U + V_0 + V_1 + \cdots + V_{m-2}$$

$$C = Z\ .\ B = UB + V_0 B + VB_1 + \cdots + V_{m-2} B$$

4.10 Time Delay of the Proposed Multiplier

To calculate the time delay and generalize it, the vector matrices are calculate

$$C = UB + V_0B + V_1B + \ldots\ldots + V_{m-2}B$$

$$UB = \{u_0 \ , \ u_1 \ , \ u_2 \ , \ u_3 \ , \ u_4 \ ,\ldots, \ u_{m-1}\}$$

$$V_0B = \{v_0 \ , \ 0 \ , \ v_0 \ , \ v_0 \ , \ v_0 \ ,\ldots, \ v_0\}$$

$$V_1B = \{v_0 \ , \ v_0 \ , \ v_0 \ , \ 0 \ , \ 0 \ ,\ldots, \ 0\}$$

$$V_0B = \{0 \ , \ v_1 \ , \ v_1 \ , \ v_1 \ , \ 0.. \ , \ 0\}$$

$$V_0B = \{0 \ , \ 0 \ , \ v_2 \ , \ v_2 \ , \ v_2 \ ,\ldots, \ 0\}$$

.

.

$$V_{m-2}B = \{0 \ , \ 0 \ , \ 0 \ ,..,v_{m-2}, \ v_{m-2} \ , \ v_{m-2}\}$$

Since in each row consists of m terms, the total delay can be calculated as following.

$$Total\ delay = \left(\lceil \log_2(m-1) \rceil\right) \ T_x + T_A + 3\,T_x$$

$$\rightarrow Total\ delay = \left(\lceil \log_2(m-1) \rceil + 3\right) \ T_X + T_A$$

4.11 Examples of Transfer matrices

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_0$$

$$Let \;\; a_1 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 \\
 & & & & & & & 1 & 1 & & 1 \\
 & & & & & & 1 & 1 & & 1 & \\
 & & & & & 1 & 1 & & 1 & & \\
 & & & & 1 & 1 & & 1 & & & \\
 & & & 1 & 1 & & 1 & & & & \\
 & & 1 & 1 & & 1 & & & & & \\
 & 1 & 1 & & 1 & & & & & & \\
1 & 1 & & 1 & & & & & & & \\
 & & 1 & & 1 & 1 & 1 & 1 & & 1 & 1
\end{bmatrix}$$

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_1x^1 + a_0$$

$$Let \;\; a_2 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 \\
 & & & & & & & & 1 & 1 & 1 \\
 & & & & & & & 1 & 1 & 1 & \\
 & & & & & & 1 & 1 & 1 & & \\
 & & & & & 1 & 1 & 1 & & & \\
 & & & & 1 & 1 & 1 & & & & \\
 & & & 1 & 1 & 1 & & & & & \\
 & & 1 & 1 & 1 & & & & & & \\
 & 1 & 1 & 1 & & & & & & & \\
1 & 1 & 1 & & & & & & & &
\end{bmatrix}$$

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$$

$$Let \;\; a_4 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 \\
 & & & & & 1 & 1 & & & & 1 \\
 & & & & 1 & 1 & & & & 1 & \\
 & & & 1 & 1 & & & & 1 & & \\
 & & 1 & 1 & & & & 1 & & & \\
 & 1 & 1 & & & & 1 & & & & \\
1 & 1 & & & & 1 & & & & & \\
 & 1 & 1 & 1 & & 1 & & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & & 1 & & 1 & 1 & 1 & 1 & \\
 & & 1 & & 1 & & 1 & & & 1 & 1
\end{bmatrix}$$

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_2x^2 + a_1x^1 + a_0$$

$$Let \;\; a_3 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 \\
 & & & & & & 1 & 1 & & & 1 \\
 & & & & & 1 & 1 & & & 1 & \\
 & & & & 1 & 1 & & & 1 & & \\
 & & & 1 & 1 & & & 1 & & & \\
 & & 1 & 1 & & & 1 & & & & \\
 & 1 & 1 & & & 1 & & & & & \\
1 & 1 & & & 1 & & & & & & \\
 & 1 & 1 & & 1 & 1 & 1 & & 1 & 1 & 1 \\
1 & 1 & & 1 & 1 & 1 & & 1 & 1 & 1 &
\end{bmatrix}$$

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$$

$$Let \;\; a_7 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & 1 & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & 1 & 1 & & & & & & & & 1 \\
1 & 1 & & & & & & & & 1 & \\
 & 1 & & 1 & 1 & 1 & 1 & 1 & & 1 & 1 \\
1 & & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & \\
1 & & 1 & & & & & & & 1 & 1 \\
1 & & & 1 & 1 & 1 & 1 & 1 & & & 1 \\
1 & 1 & 1 & & & & & 1 & 1 & & 1 \\
 & & & 1 & 1 & 1 & & & 1 & & 1 \\
 & & 1 & 1 & 1 & & & 1 & & 1 &
\end{bmatrix}$$

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$$

$$Let \;\; a_8 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & 1 & 1 & & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & 1 & 1 & & & & & & & 1 \\
 & 1 & 1 & & & & & & & 1 & \\
1 & 1 & & & & & & & 1 & & \\
 & 1 & 1 & & 1 & 1 & 1 & & 1 & 1 & 1 \\
1 & 1 & & 1 & 1 & 1 & & 1 & 1 & 1 & \\
 & 1 & & 1 & & 1 & & & & 1 & 1 \\
1 & & 1 & & 1 & & & & 1 & 1 & \\
1 & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 \\
1 & & & 1 & & & 1 & & & 1 & 1
\end{bmatrix}$$

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$$

$$Let \;\; a_9 = 0 \;\; then:$$

$$\begin{bmatrix}
1 & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & & & & & & & & & 1 \\
 & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & & \\
 & 1 & & & & & 1 & & 1 & 1 & 1 \\
1 & & & & & 1 & & 1 & 1 & 1 & \\
1 & & 1 & 1 & & 1 & & & & 1 & 1 \\
1 & 1 & & 1 & & 1 & 1 & 1 & & & 1 \\
 & & & 1 & & & & 1 & 1 & & 1
\end{bmatrix}$$

$$Let \quad F(x) = a_1\, x^1 + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$$

$$Let \;\; a_{10} = 0 \;\; then:$$

$$\begin{bmatrix}
 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & \\
1 & & & & & & & & & 1 & 1 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & \\
1 & & & & & & & 1 & 1 & 1 & 1 \\
 & 1 & 1 & 1 & 1 & 1 & & & & & 1 \\
1 & 1 & 1 & 1 & 1 & & & & & 1 & \\
1 & & & & 1 & 1 & 1 & 1 & & 1 & 1 \\
 & 1 & 1 & & & & & 1 & & & 1
\end{bmatrix}$$

Using the transfer matrix, one can calculate the V matrices with ease, this could be demonstrated as follows:

For example the transfer matrix below, from the previous example:

$$Let \quad F(x) = a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_1x^1 + a_0$$
$$Let \quad a_2 = 0 \quad then:$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 \\
 & & & & & & & & 1 & 1 & 1 \\
 & & & & & & & 1 & 1 & 1 & \\
 & & & & & & 1 & 1 & 1 & & \\
 & & & & & 1 & 1 & 1 & & & \\
 & & & & 1 & 1 & 1 & & & & \\
 & & & 1 & 1 & 1 & & & & & \\
 & & 1 & 1 & 1 & & & & & & \\
 & 1 & 1 & 1 & & & & & & & \\
1 & 1 & 1 & & & & & & & & 1
\end{bmatrix}$$

Using the transfer function and transposing it we can have, (for referral purposes this matrix is named as Ω):

$$\begin{bmatrix}
1 & 1 & & & & & & & & & \\
 & 1 & 1 & & & & & & & & \\
1 & 1 & 1 & 1 & & & & & & & \\
1 & & 1 & 1 & 1 & & & & & & \\
1 & & & 1 & 1 & 1 & & & & & \\
1 & & & & 1 & 1 & 1 & & & & \\
1 & & & & & 1 & 1 & 1 & & & \\
1 & & & & & & 1 & 1 & 1 & & \\
1 & & & & & & & 1 & 1 & 1 & \\
1 & & & & & & & & 1 & 1 & 1 \\
1 & & & & & & & & & 1 & 1
\end{bmatrix} = \Omega$$

Using the transposed 'Ω' matrix one can calculate the V matrices(m*m) as such:

For V_0 the first column of Ω is used, for V_0 the first column is all zeros and then for every entry of "1" in the Ω a row is added to V_0, from a_{m-1} to the last item in each row with a_{m-1} decrementing.

$$V_0 = \begin{bmatrix} 0 & a_1 & a_0 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{10} & a_9 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_1 & a_0 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a_1 & a_0 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_1 & a_0 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \end{bmatrix}_{m\times m}$$

For V_1 the first column of Ω is used, for V_0 the first two columns are all zeros and then for every entry of "1" in the Ω a row is added to V_1, from a_{m-1} to the last item in each row with a_{m-1} decrementing.

$$V_1 = \begin{bmatrix} 0 & 0 & a_1 & a_0 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & a_{10} & a_9 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & a_1 & a_0 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{m\times m}$$

For V_{m-1} *the* $(m-1)^{th}$ *column of* Ω *is used, for* V_{m-1} *the first* $(m-2)$ *columns are all zeros and then for every entry of* "1" *in the* Ω *a row is added to* V_{m-1}, *from* a_{m-1} *to the last item in each row with* a_{m-1} decrementing.

$$V_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{10} \end{bmatrix}_{m \times m}$$

Additional example of calculation of matrices directly :

$$A(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$

$$B(x) = b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

$$F(x) = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$$

$$UB = \begin{bmatrix} a_0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_1 & a_0 & 0 & 0 & 0 & 0 & 0 \\ a_2 & a_1 & a_0 & 0 & 0 & 0 & 0 \\ a_3 & a_2 & a_1 & a_0 & 0 & 0 & 0 \\ a_4 & a_3 & a_2 & a_1 & a_0 & 0 & 0 \\ a_5 & a_4 & a_3 & a_2 & a_1 & a_0 & 0 \\ a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

$$V_0B = \begin{bmatrix} 0 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \\ 0 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

$$VB_1 = \begin{bmatrix} 0 & 0 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & a_6 & a_5 & a_4 & a_3 & a_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

$$V_2B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_6 & a_5 & a_4 & a_3 \\ 0 & 0 & 0 & a_6 & a_5 & a_4 & a_3 \\ 0 & 0 & 0 & a_6 & a_5 & a_4 & a_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

$$V_3B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_6 & a_5 & a_4 \\ 0 & 0 & 0 & 0 & a_6 & a_5 & a_4 \\ 0 & 0 & 0 & 0 & a_6 & a_5 & a_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

$$V_4B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_6 & a_5 \\ 0 & 0 & 0 & 0 & 0 & a_6 & a_5 \\ 0 & 0 & 0 & 0 & 0 & a_6 & a_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

$$V_5B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_6 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

5 Complexity Comparison

This chapter is intended to compare the pentanomials and One Zero polynomials using Mastrovito algorithm.

As previously mentioned Pentanomials are considered the next best in class where the trinomials AOP or ESP do not exist.

Having One Zero polynomials exhibit vast coverage in finite field and improve space and time complexity over pentanomials will make this new class more efficient than the pentanomials without the compromise of coverage.

Table 2: Complexity Comparison with OZP

Polynomial	Complexity (XOR)	Reference
Trinomial	$m^2 - 1$	[10][11][13][14][15]
EST	$m^2 - \frac{m}{2}$	[15]
AOP	$m^2 - 1$	[12]
ESP	$m^2 - \Delta$	[11]
Pentanomials	m^2+2m-3	[11]
OZP	$m^2 + 2m - 6$	
General	$(m-1)(m+k-1) + \sum_{j\epsilon s}(2m-1-j)$	[11]

In addition to the space complexity improvement shown in Table 2, the time complexity is shown in Table 3.

Table 3 Time Delay comparison

Polynomial	Time Complexity	Reference
Trinomial	$\left(\lceil \log_2(n) \rceil + 1\right) T_X + T_A$	[10][11][13][14][15]
EST	$\left(\lceil \log_2(n) \rceil + 1\right) T_X + T_A$	[15]
AOP	$\left(\lceil \log_2(n-1) \rceil + 1\right) T_X + T_A$	[12]
ESP	$\left(\lceil \log_2(n) \rceil + 1\right) T_X + T_A$	[11]
Pentanomials	$\left(\lceil \log_2(n-1) \rceil + 4\right) T_X + T_A$	[11]
OZP	$\left(\lceil \log_2(n-1) \rceil + 3\right) T_X + T_A$	
General	$(m-1)(m+k-1) + \sum_{j \epsilon s} (2m-1-j)$	[11]

Therefore it is proven that the One Zero Polynomials provide both shorter time delay and less space complexity.

6 Conclusion

6.1 A Summary of Contribution

The emphasis in this project is on the Polynomial basis and their one step and two step variation of multiplication algorithm.

For the two step multiplication a conventional approach is demonstrated in detail (KOA and some of its variation.). Their efficiency and area of usage, as well as their advantages and disadvantages were described.

The second step of the two step multiplication is the modularisation which is computed by means of transfer matrix. The complexity of this step in is directly dependent to the number of ones in that multiplication matrix.

For one step multiplication the Mastrovito algorithm was described in detail and examples were provided.

Mastrovito constructs a Z matrix form an operand and the irreducible polynomial, which this Z Matrix will multiply with the first operand. This one multiplication will result in both the multiplication and modulo reduction.

Efficiency of Mastrovito multiplication greatly depends on the irreducible polynomial available for the field. Most efficient polynomials were listed with their limitations.

Different variations of using Mastrovito was demonstrated in detail with their examples.

For the degrees of m where the most efficient irreducible functions do not exist, (Such as Trinomial, AOP and ESP), the next best in class where considered to be the Pentanomials.

In this thesis a new family of irreducible polynomials were introduced and it was proven to cover all the degrees m. In addition to the coverage it was proven that newly introduced polynomial (One Zero Polynomial), reduces the time and space complexity in multiplication.

6.2 Future work

Additional research can be conducted in using One Zero Polynomials and hardware implementation. Also using One Zero Polynomials with alternative algorithms could yield better results and more efficient multiplication algorithm.

Using One Zero Polynomial did improve Mastrovito, but calculations for KOA was not conducted, there are indications that using One Zero polynomials could potentially improve the time and space complexity if used in modularization in two step multipliers.

More research could be applied into which coefficient of One Zero Polynomial being zero would yield most efficiency and under what circumstance.

References

[1] R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications. Cambridge University Press, Cambridge, England, 1997.

[2] A. J. Menezes et. al, Applications of Finite Fields, Kluwer, 1993.

[3] D.G. Cantor, "On Arithmetical Algorithms over Finite Fields," J. Combinatorial Theory, Series A 50, pp. 285-300, 1989.

[4] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.

[5] R. Lidl and H. Niederreiter, Introduction to Finite Fields and Their Application, Cambridge University Press, 1986.

[6] D. Hankerson, A. Menezes, and S. Vanstone. Guide to Elliptic Curve Cryptography. Springer, New York, NY, 2003.

[7] Ashkan Hosseinzadeh Namin, H.Wu, and M.Ahmadi, High Speed World Level Finite Field Multipliers in $F(2^m)$, IEEE Trans on VLSI, To appear, 2009

[8] Karatsuba A.; Ofman Y. Multiplication of multidigit numbers by automata, Soviet Physics-Doklady 7, p.595-596, 1963

[9] D.E. Knuth, "The Art of Computer Programming," Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981

[10] E.D. Mastrovito "VLSI Architecture for multiplication over finite field GF(2m)," Applied Algebra, Algebraic ALGORITHNS AND Error-Correcting Codes, T. Mora, ed., PP297-309, Berlin: Springer-Verlag 1988.

[11] E.D. Mastrovito "VLSI Architecture for Computation in Galios field GF(2m)," PHD thesis, Dep. Of electrical Eng.,Linkoping Univ., Linkoping, Sweden, 1991.

[12] C.K. Kos and B. Sunar, "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields". IEEE Trans. Computers , vol.47, no, 3. Pp.353-356, Mar. 1998.

[13] C. Paar, "Efficient VLSI Architecture for Bit-Parallel Computation in Galios Fields." PhD Thesis, Universitat GH Essen, VDI Verlog, 1994.

[14] C. Paar, "A new architecture for a parallel Finite Fields Multiplier with low complexity based on composite fields ." IEEE Trans. Computers , vol.45, no, 7. Pp.856-861, July. 1996.

[15] C.K. Kos and B. Sunar, "Mastrovito Multiplier for all Trinomials". IEEE Trans. Computers , vol.48, no, 5. PP.522-527, May. 1999.

[16] J.L. Massey, J.K. Omura, "Computational method and apparatus for finite field arithmetic", U.S. Patent Aplication, submitted 1981.

[17] Ernest Jamro, "THE DESIGN OF A VHDL BASED SYNTHESIS TOOL FOR BCH CODECS," School of Engineering,The University of Huddersfield. September 1997

[18] M. Leone, "A New Low Complexity Parallel Multiplier for a Class of Finite Fields",

Proc. Cryptographic Hardware and Embedded Systems (CHES 2001),.

[19] E.D. Mastrovito, “VLSI Architectures for Multiplication Over Finite Field GF(2m)”, Proc. Sixth Int'l Conf., AAECC-6, T. Mora, ed., pp. 297-309, Rome, Jul.1988.

[20] P.K. Meher, Y. Ha, C.Y. Lee, “An Optimized Design for Serial-Parallel Finite Field Multiplication over GF(2m) Based on All-One Polynomials”, Proc. 2009 Conf. on Asia and South Pacific Design Automation, Yokohama, Japan, pp. 201-225, 2009.

[21] H. Wu, “Bit-Parallel Polynomial Basis Multiplier for New Classes of Finite Fields”, IEEE Trans. Computers, Vol. 57, No. 8, pp. 1023-1031, Aug 2008

[22] H.Wu, M.A. Hasan, I.F. Blake, “New low-complexity bit-parallel finite field multipliers using weakly dual bases”, IEEE Trans. Computers, Vol.51, No. 11, pp. 1223-1234, Nov. 2002.

[23] H. Wu, M.A. Hasan, I.F. Blake, and S. Gao, “Finite Field Multiplier Using Redundant Representation,” IEEE Trans. Computers, Vol. 51, No. 11, pp. 1306-1316, Nov. 2002.

[24] T. Zhang, K. Parhi, “Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials”, IEEE Trans. Computers, Vol. 50, No. 7, pp. 734-749, Jul. 2001.

[25] G. H. Golub and C. E Van Loan, Matrix Computations, Johns Hopkins University Press,
3rd edition, 1996.

[26] M. A. Hasan, M. Z. Wang, and V. K. Bhargava, “Modular construction of low complexity parallel multipliers for a class of finite fields GF(2”)”, IEEE Trans. on Computers, vol.41, pp. 962-971, Aug. 1992.

[27] J.-C. Bajard, L. Imbert, and G.A. Jullien, “Parallel Montgomery Multiplication in GF(2^k) Using Trinomial Residue Arithmetic,” Proc. 17th IEEE Symp. Computer Arithmetic (ARITH ’05), 2005.

[28] T.C. Bartee and D.I. Schneider, Computations with Finite Fields, Inform. Contr., vol. 6, pp. 79-98, Mar. 1963.

[29] R. W. K. Odoni, &&Zeros of random polynomials over "nite "elds," in Mathematical Proceedings of the Cambridge Philosophical Society, Vol. 111, March 1992, Part, 2. pp. 193}197.

[30] W. M. Schmidt, ‘Equations over Finite Fields, An Elementary Approach," Lectures Notes in Math., Vol. 536, Springer-Verlag, 1976.

[31] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.

[32] D.G. Cantor and E. Kaltofen, “On Fast Multiplication of Polynomials over Arbitrary Algebras,” Acta Informatica, vol. 28, pp. 693-701, 1991.

[33] S. S. Erdem. Improving the Karatsuba-Ofman Multiplication Algorithm for Special Applications. PhD thesis, Department of Electrical and Computer Engineering, Oregon State University, November 2001.

[34] K. Posch and R. Posch, “Modulo Reduction in Residue Number Systems,” IEEE Trans. Parallel and Distributed Systems, vol. 6, no. 5, pp. 430-461, 1995.

[35] J. Bajard, L. Didier, and P. Kornerup, “Modular Multiplication and Base Extensions in Residue Number Systems,” Proc. 15th IEEE Symp. Computer Arithmetic (ARITH ’01), pp. 59-65, 2001.

Appendix A

Polynomial, dual and normal basis representations of GF(24), generated by the irreducible polynomial $p(x)= 1 + x + x^4$.[17]

power of α	Standard basis $1, \alpha, \alpha^2, \alpha^3$	Dual basis $1, \alpha^3, \alpha^2, \alpha$	Normal basis $\alpha^3, \alpha^6, \alpha^{12}, \alpha^9$
-	0000	0000	0000
0	1000	1000	1111
1	0100	0001	1001
2	0010	0010	1100
3	0001	0100	1000
4	1100	1001	0110
5	0110	0011	0101
6	0011	0110	0100
7	1101	1101	1110
8	1010	1010	0011
9	0101	0101	0001
10	1110	1011	1010
11	0111	0111	1101
12	1111	1111	0010
13	1011	1110	1011
14	1001	1100	0111

APPENDIX B

Maple code to calculate the availability of OZP in different degrees

```
> interface(rtablesize=2):
> poly:=x^4+x^3+x^2+x+1:
> poly_orig:=poly:
> i:=1:
> FileTools[Text][Open]("Table1.txt", create=true,
overwrite=true):
> for m from 0 to 800 do
>                     A[m]:=poly_orig + x^m:
>           for n from 1 to m do
>                               B[m]:=A[m] + x^n:
>                               C[m]:=B[m] mod 2:
>                               J:=irreduc(C[m]):
                              k:=J ;
                              l:=cat( "The value of M= ",
m ,"     The  irreducible OZP exists=   ", k, "." );
>                               #D[m,(irreduc(C[m]))];
>                          for y from 1 to m do
                              D[m]:=B[m]-x^y:
                              E[m]:=D[m] mod 2:
>                               J:=irreduc(D[m]):
                              k:=J ;
                              l:=cat( "The value of M= ",
m ,"     The  irreducible OZP exists=   ", k, "." );
>                               #D[m,(irreduc(D[m]))];
                          end do:
>    #FileTools[Text][WriteInteger]( "Table1.txt",  m );
   FileTools[Text][WriteString]( "Table1.txt",  l );
FileTools[Text][WriteLine]( "Table1.txt");
>           end do:
>                      poly_orig:=poly_orig + x^m:
>
> #D[m,(irreduc(C[m]))];
>
>
> end do:
> FileTools[Text][Close]("Table1.txt");
```

VITA AUCTORIS

NAME:	Seyed Mohammad Ali Shahabi
PLACE OF BIRTH:	Tehran, Iran
YEAR OF BIRTH:	1975
EDUCATION:	Motahary High School, Tehran, Iran
	1990-1994
	University of Western Ontario, London, Canada
	1996-2001 B.Sc.
	University of Windsor, Windsor, Ontario
	2006-2009 M.A.Sc.