1-1-2019

Hardware Implementation of Bit-Parallel Finite Field Multipliers Based on Overlap-free Algorithm on FPGA

Meitong Pan
University of Windsor

Hardware Implementation of Bit-Parallel Finite Field Multipliers Based on Overlap-free Algorithm on FPGA

By

Meitong Pan

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfilment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2019

Hardware Implementation of Bit-Parallel Finite Field Multipliers Based on Overlap-free Algorithm on FPGA

By

Meitong Pan

APPROVED BY:

S. Cheng
Department of Civil & Environmental Engineering

B. Balasingam
Department of Electrical & Computer Engineering

H. Wu, Co-Advisor
Department of Electrical & Computer Engineering

M. Mirhassani, Advisor
Department of Electrical & Computer Engineering

December 18^{th}, 2019

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

Cryptography can be divided into two fundamentally different classes: symmetric-key and public-key. Compared with symmetric-key cryptography, where the complexity of the security system relies on a single key between receiver and sender, public-key cryptographic system using two separate but mathematically related keys. Finite field multiplication is a key operation used in all cryptographic systems relied on finite field arithmetic as it not only is computationally complex but also one of the most frequently used finite field operations.

Karatsuba algorithm and its generalization are most often used to construct multiplication architectures with significantly improved in these decades. However, one of its optimized architecture called Overlap-free Karatsuba algorithm has been mention by fewer people and even its implementation on FPGA has not been mentioned by anyone. After completion of a detailed study of this specific algorithm, this thesis has proposed implementation of modified Overlap-free Karatsuba algorithm on Xilinx Spartan-605. Applied this algorithm and its specific architecture, reduced gates or shorten critical path will be achieved for the given value of n.

Optimized multiplication architecture, generated from proposed modified Overlap-free Karatsuba algorithm and applied on FPGA board,over NIST recommended fields ($n = 128$), are presented and analysed in detail. Compared with existing works with sub-quadratic space and time complexities, the proposed modified algorithm is highly recommended module and have improved on both space and time complexities. At last, generalization of proposed modified algorithm is suitable for much larger size of finite fields, and improvements of FPGA itself have been discussed.

To my family

my grandparents

my parents

my fiancé

for their unconditional love

and

support

Acknowledgments

I wish to express my sincere gratitude to my supervisor Dr.Mitra Mirhassani and my co-supervisor Dr. Huapeng Wu, for their patience, motivation and immense knowledge throughout my graduate study.

I would like to thank my family members, my mum, dad and my fiancé, for their constant support and continuous encouragement during the time of completing my further study.

I would like to thank my committee members, Dr. Huapeng Wu, Dr. Bala Balasingam and Dr. Shaohong Cheng.

I would also like to thank my colleagues at Uwindsor's Faculty of Electrical and Computer Engineering, especially Andria Ballo,for their help and support.

Table of Contents

List of Figures

List of Abbreviations

DES Data Encryption Standard

AES Advanced Encryption Standard

ECC Elliptic Curve Cryptography

KOA Karatsuba-Ofman's algorithm

VLSI Very-Large-Scale Integration

CPF Component Polynomial Formation

NIST National Institute of Standards and Technology

FPGA Field-Programmable Gate Array

CLB Configurable Logic Blocks

LUT Look-up Table

FF Flip-Flop

MUX Mutiplexer

IOB Input / Output Blocks

HDL Hardware Description Language

RTL Register Transfer Level

ISE Integrated Synthesis Environment

KA Karatsuba Algorithm

ETP Even-Term Polynomial

RKA Recursive Karatsuba Algorithm

MKM Modified Karatsuba Multiplier

Chapter 1

Introduction

With the rapid development of computer network technology, the application of the Internet has become more extensive. The openness of the Internet brings unprecedented amount of information and and the freedom of the Internet has also created the possibility of private information and data destroyed or invaded. The security of network information has become increasingly important and has been used in various fields of society. In order to protect the data being transmitted over the high risk Internet, cryptographic services have been widely used in communication, government, military and many other fields.

1.1 Motivation

Cryptography can be divided into two fundamentally different categories: symmetric keys and public key ,which also known as asymmetric key. In symmetric key encryption, both sides of the communication, sender and receiver, use the same key for both encryption and decryption process. Data Encryption Standard (DES), RC5 and Advanced Encryption Standard (AES) can be called the most famous symmetric key arithmetic. The security of this mechanism determines

the symmetric key, which only known for senders and receivers. However, it is difficult for the two parties to exchange keys without compromising the security of the keys themselves, which in return will hazard data confidentiality and data authentication. The second question is assumed that symmetric cryptography is called a key management problem. Supposing that a communication medium is shared between n users, and each pair of users needs a different key to establish their own secure communication. So $n(n-1)/2$ different keys will be provided, even in medium-sized networks, it is hard to manage.

Public key cryptography is a solution to the problem of key distribution. Instead of using a single key, the public key encryption system uses two separate but mathematically related keys: a *public key* and a *private key*. The public key is not confidential and can be freely distributed in the user's network and used for encryption purposes. On the other hand, the private key cannot be shared by both parties, but is held by only one party and is used during the decryption process only. The pair of public keys and their corresponding private keys must be used together and they have a mutual relationship so that the key pair can be used together to obtain the same result as using a symmetric key twice. It should also be noted that public key cryptography has an advantage over symmetric key cryptography because it provides additional security services such as key exchange, digital signatures, authentication, and message integrity verification.

So far, based on the concept of public key cryptography, three different types of cryptosystems have been proposed, RSA [1], ELGamal [2], and Elliptic Curve Cryptography (ECC) [3, 4]. The security of each of these cryptosystems depends on the a difficult mathematical problem, which called the one-way function. ECC is much more security for the following reasons:

- The ECC keys are obviously smaller than those of RSA and ELGamal for any given level of security.

- The ratio of key sizes of ECC is much more higher than the other two public key schemes, which means that the higher security is required, the more efficient ECC becomes.

- The key length of ECC are twice as long as those of symmetric algorithms for the same level of security, which illustrates the higher comutational complexity of the public key schemes.

In addition, in such a fast developing digital society, the speed of computing and network transmission continues to increase, and public key cryptography has played an increasingly important role. As more and more business activities begin to penetrate into the Internet,and the potential threat posed by quantum computers, this situation will expand to reliable security services,which cover people's social lives. However, the intensive computing required in public key cryptosystems is a major problem faced by the promotion of such systems. Therefore, in recent years, extensive algorithms and effective implementation of public key cryptography have been extensively researched.

1.2 Objective

Two common used classed of finite fields in cyptography are prime fields of degree one $GF(\mathrm{p})$ and binary extension fields of degree greater than one $GF(2^n)$. The latter is a subclass of a more generalized group of finite fields known as finite prime extension fields $GF(p^n)$, where the parameter p is equal to two and the extension degree is greater than one. Binary fields are more attractive for high speed cryptosystem applications. Because the basic field operations addition and multiplication in the underlying field $\mathbb{F}_2$ can be readily realized by a bit-wise XOR and a bit-wise AND operations, respectively.

Different architectures for finite field multipliers can generally be divided into bit-serial, bit-parallel and digit-level architectures. Given a binary extension field of degree n, bit-serial multipliers need n clock cycles to finish a full multiplication operation. Although they need the maximun number of clock cycles for computing the product coordinates, the provide the optimal area utilization and power consumption. On the other hand, bit-parallel multipliers utilize the highest level of parallelism, multiplication operation is performed fast and only need one clock cycle. Digit-level architecture, finally, fill the gap between bit-serial and bit-parallel design styles to keep a balance between space and delay complexities.

Since the extension of the Karatsuba algorithm (a "divide and conquer" technique for efficient integer multiplication) to finite field multiplication with quadratic space complexity, many improvements have been made to this method over the past few years. Specifically, these improvements can be summarized into two sub-areas: one attempts to improve the Karatsuba architecture through an optimized re-factoring process, and another attempt focuses on summarizing the Karatsuba formula by reducing the number of sub-multiplications, which will be introduced in Chapter 3 in detail.

To satisfy both speed and high-precision computation requirements, reconfigurable hardware is increasingly being considered. In field programmable gate arrays (FPGA), a large amount of flexible hardware resources are available for paralleling algorithms, with the further advantage of flexibility in the data path. Further more, implementing every polynomial algorithm with a dedicated custom circuit would obviously incur high development and engineering costs. While the cost of FPGA development is much more lower, and this remains true even when amortized for moderate manufacturing volumes. Although many designs with KA polynomial evaluation have been implemented in FPGA, recent articles have not focus on Overlap-free KA algorithm. In this thesis, this method will be thoroughly analysed and will be implemented on FPGA board in Chapter 4.

1.3 Organization of Thesis

- Chapter 2 In this chapter, mathematics fundamental of abstract algebra are first introduced. Binary finite extension fields has been illustrated as a special class of finite field. In the last of this chapter, arithmetic operations in $GF(2^n)$ and architecture of multipliers have been discussed in detail with their different types.

- Chapter 3 In this chapter contains two parts, including four different kinds of multiplication algorithms and their comparison based on NIST recommended $GF(2^n)$ fields. We briefly introduce original Karatsuba, Overlap-free Karatsuba, Reconstruction Karatsuba and Improved Reconstruction by Bernstein multiplication algorithms. We also arrange their recursive function describing each method's space and time complexity. Finally, we analyse the result of this four algorithms applied in different field and we also achieve the main algorithm which can efficiently apply into the $GF(2^{128})$.

- Chapter 4 In this chapter, we introduce the fundamental technology information, including FPGA, Verilog HDL and ISE software. Then we analyse the code corresponding to algorithm mentioned in chapter 3 clearly. At last of this chapter, we apply our proposed solution to make a comparison with published articles and achieve a considerable result.

- Chapter 5 In this chapter, it is a summary of our proposed contribution and future works on how to speed up FPGA its own reading and writing speed are suggested.

Chapter 2

Preliminary

In this chapter, mathematics fundamental of abstract algebra including group, rings and field are first introduced. Binary finite extension fields has been illustrated as a special class of finite field in this thesis. In the last of this chapter, arithmetic operations in $GF(2^n)$ and architecture of multipliers have been discussed in detail with their different types.

2.1 Mathematics Fundamental

In this section, three briefly definitions about group, rings and fields will be illustrated.
Definition 1: A group is a set G together with a binary operation $(\star)$ on G, such that the following three properties [5]:

- $(\star)$is associative, that is, for any a,b,c$\in$ G

$$a \star (b \star c) = (a \star b) \star c$$

- There is an identity (or unity) element in G such that for all a$\in$ G

$$a \star e = e \star a = a$$

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

$$a \star a^{-1} = a^{-1} \star a = e$$

- There is an identity (or unity) element e in G such that for all a$\in$ G

$$a \star e = e \star a = a$$

- If the group also satisfies for all a,b$\in$ G

$$a \star b = b \star a$$

  then the group is called abelian (or commutative).

Definition 2: A ring is a set R, together with two binary operations denoted by $(+)$ and $(\cdot)$, such that [5]:

- R is an abelian group with respect to $(+)$
- $(\cdot)$ is associative, for all a,b,c$\in$ R

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

- The distribute laws hold

$$\begin{aligned} a \cdot (b + c) &= a \cdot b + a \cdot c \\ (b + c) \cdot a &= b \cdot a + c \cdot a \end{aligned}$$

Definition 3: A field, is a set F, together with two binary operations denoted by $(+)$ and $(\cdot)$, such that [5]:

- F, is a ring in tern of $(\cdot)$ and $(+)$ operation
- For any elements a,b$\in$ F, $(\cdot)$ is commutative

$$a \cdot b = b \cdot a$$

- Nonzero elements of F respect to $(\cdot)$operation form an abelian group

2.2 Finite Field

Finite field,is also called Galois field, is a set of finite number of elements, where addition and multiplication are defined.

- The finite field is an addictive group under the addition operation.
- All the non-zero elements in a finite field form a multiplicative group under multiplication operation.
- When we say the order of a field element, it means that the order of the element in the multiplicative group.

It is commonly denoted finite field as GF(p) or $\mathbb{F}_p$, where p is the number of elements in this field. The characteristic x of a finite field GF(p) is defiend as the least positive integer x and $ax = 0$ for any element a $\in$ GF(p).
There are two different kinds of finite field as below [5]:

- Prime fields, GF(p), is a set of $\{0, 1, 2, ..., p-1\}$, where p is a prime number. InGF(p) , the binary operator$(\cdot)$ refers to mod-p multiplication and

(+) refers to mod-p addition.

- Binary extended finite field, $GF(p^n)$ is a set of polynomials of degree up to $n-1$ with coefficients according to GF(p). In GF(p), the variety of those polynomials is a root of irreducible polynomial $f(x) = \sum_{i=0}^{n} f_i x^i$, for $f_i \in GF$(p). It is noted that p is a prime number and n is a positive integer, which is greater than 1. In $GF(p^n)$, the binary operator($\cdot$) refers to mod-$f(x)$ and mod-p multiplication and (+) refers to mod-p addition.

The irreducible polynomial in finite field can not be factorized into a factor, which degree between 1 and $n-1$ in the same field, just like a prime number. In this thesis, the irreducible polynomial is fixed over the basic field $GF(2^{128})$ and will be discussed in detail in the following sections.

2.3 Arithmetic Operation in Finite Field $GF(2^n)$

Binary extension field, denoted as $GF(2^n)$, is a special class of finite extension fields with element 2. The arithmetic in $GF(2^n)$ is very suitable for hardware implementation. This is mostly because the ground field operations, addition and multiplication in $GF(2)$, can be directly implemented with the AND and XOR logic gate, respectively. In fact, the class of binary extension finite fields $GF(2^n)$ has roughly the most popular applications, which is the important reason. Before we discuss the finite field arithmetic, we can talk about the complex numbers and their arithmetic operation.

2.3.1 Arithmetic operation in complex number field

The complex number field C, is denoted as

$$\mathbb{C} = \{a + bi | a, b \in \mathbb{R}, i = \sqrt{-1}\} = \{a + bi | a, b \in \mathbb{R}, i^2 + 1 = 0\},$$

where the set of real numbers is referred to R. Let $A = a_0 + a_1 i$, $B = b_0 + b_1 i$, and $a_0, a_1, b_0, b_1 \in \mathbb{R}$, then addition and multiplication operations are as follow:

$$\begin{aligned} A + B &= (a_0 + b_0) + (a_1 + b_1)i \\ A \times B &= (a_0 + a_1 i) \times (b_0 + b_1 i) mod(i^2 + 1) \\ &= (a_0 b_0 - a_1 b_1) + (a_1 b_0 + a_0 b_1)i \end{aligned}$$

Because the equation $i^2 + 1 = 0$ does not have a root in real number field, so it is called the irreducible polynomial in real number field.

The procedure of the complex number in field $\mathbb{C}$ and its arithmetic in the real number field $\mathbb{R}$ can be summarized as below:

1. Find a quadratic equation $i^2 + 1 = 0$ that has no root in R, which we also called irreducible polynomial in real number field $\mathbb{R}$.
2. Use the root of equation $i^2 + 1 = 0$ be i and coin the expression $a + bi$, where a,b $\in \mathbb{R}$. And get the representation of the complex field numbers $\mathbb{C}$.
3. Then get arithmetic operation in $\mathbb{C}$.

2.3.2 Arithmetic operation in Finite Field $GF(2^n)$

Similar to the case of complex number C and its arithmetic, we can easily derive $GF(2^n)$ and its arithmetic as follows:

1. Elements in this fields can be generated with an irreducible polynomial $f(\mathrm{x})$ of degree n. If x is the root of $f(\mathrm{x})$, a polynomial base can be represented as $\{1, x, x^2, ..., x^{n-1}\}$

2. Find an irreducible degree-n polynomial $f(\mathrm{x})$ over $GF\ (2^n)$.

3. Use x as the root of $F(x) = 0$. Then $GF(2^n) = \{a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_0 | a_i \in GF(2), f(x) = 0\}$

4. Arithmetic operations in $GF(2^n)$. For A,$B \in GF(2^n)$, and $A = \sum_{i=0}^{n-1} a_i x^i$, $B = \sum_{i=0}^{n-1} b_i x^i$, then we get

$$\begin{aligned} A + B &= (\sum_{i=0}^{n-1}(a_1 + b_i)x^i) mod 2 \\ A \times B &= (\sum_{i=0}^{n-1} a_i x^i \times \sum_{i=0}^{n-1} b_i x^i) mod 2 mod f(x) \end{aligned}$$

   Note that the product of the multiplication operation must be modular reduced to no higher than $n - 1$.

2.4 Multiplication Architectures

Time and space complexities are applied to measure the efficiency of $GF(2^n)$ multipliers. In $GF(2)$, polynomial addition can be implemented by a 2-input XOR gate and multiplication can be used by a 2-input AND gate. According to this rule, the space complexity can be represented by the total number of AND gates and XOR gates, and the time complexity can be measured by the delays occur in one AND gate and XOR gate. So we use $S\oplus$ and $S\otimes$ to denote the number of XOR and AND gates, respectively. We also use T_A and T_X to represent the delay of AND and XOR gates, respectively.

In this section, we illustrate two structures of polynomial multiplication in $GF(2^n)$, the bit-parallel multiplication and bit-serial multiplication, which usually give a lower time and space complexity, respectively.

2.4.1 Bit-parallel multiplication

Bit-parallel multipliers are recommended to apply with a requirement of large performances because it has a larger output and generate result within one clock cycle.

The classical method to calculate polynomial multipliers is a typical parallel structure. In this method, all inputs are entered and calculated in parallel. Although the classic method is a fast structure for $GF(2^n)$ multipliers, its application is limited for its large space complexity. While recently, this method combine with other methods such as non-recursive KA [6], Chinese reminder theorem [7], and Mastrovito matrix [8]. And then the new combination multiplication a highly proposed in the literature to optimize the construct quadratic space complexity multipliers, because it gives a same asymptotic time complexity with a obvious decrease in space gate cost.

2.4.2 Bit-serial multiplication

Compare with the feature to bit-parallel, bit-serial multiplication has a lower space cost, which makes it competitive in application in constrained resources. Based on the input and output sequences, bit-parallel multiplication can be divided into four types, as follows [9]:

- BL-SISO: bit-level serial input and serial output
- BL-SIPO: bit-level serial input and parallel output

- BL-PISO: bit-level parallel input and serial output
- BL-PIPO: bit-level parallel input and parallel output

In this thesis, we focus on the hardware implementation of bit-parallel binary polynomial multiplication and analyse the result.

Chapter 3

An Overview of Bit-Parallel Multiplication for $GF(2^n)$ and Comparison

In this chapter contains two parts, including four different kinds of multiplication algorithms and their comparison based on NIST recommended $GF(2^n)$ fields. First, we briefly introduce original Karatsuba, Overlap-free Karatsuba, Reconstruction Karatsuba and Improved Reconstruction by Bernstein multiplication algorithms. We also arrange their recursive function describing each method's space and time complexity. After that, we analyse the result of this four algorithms applied in different field and we also achieve the main algorithm which can efficiently apply into the $GF(2^{128})$.

3.1 Karatsuba Algorithm

In early 1960, the first sub-quadratic integer multiplication algorithm was invented by A.A.Karatsuba for fast multiplication of multi-place numbers [10]. After that,

Karatsuba-Ofman's algorithm (KOA), published in 1962 [11], was a new integer multiplication method which broke the quadratic complexity barrier in positional number systems. Due to its simplicity, the current improved works mainly focus on using more efficient polynomial multiplication algorithms or structures based on Karatsuba formulas.

Let $A = \sum_{i=0}^{n-1} a_i x^i$ and $B = \sum_{i=0}^{n-1} b_i x^i$ be two $GF(2^n)$ elements. To explain the KOA easily, we will assume that $n = 2m = 2^k (k > 1)$ in the following [12].

First, the previous KOA implementations split polynomials A and B into the "most significant half" and the "least significant half" as follows:

$$A = \sum_{i=0}^{n-1} a_i x^i = x^m \sum_{i=0}^{m-1} a_{m+i} x^i + \sum_{i=0}^{m-1} a_1 x^i = x^m A_H + A_L$$
$$B = \sum_{i=0}^{n-1} b_i x^i = x^m \sum_{i=0}^{m-1} b_{m+i} x^i + \sum_{i=0}^{m-1} b_1 x^i = x^m B_H + B_L$$

where $A_H = \sum_{i=0}^{m-1} a_{m+i} x^i$, $A_L = \sum_{i=0}^{m-1} a_i x^i$,$B_H$ and B_L are defined similarly.

Then the product AB is computed recursively using

$$AB = A_H B_H x^{2m} + \{[(A_H + A_L)(B_H + B_L)] - [A_H B_H + A_L B_L]\} x^m + A_L B_L \quad (3.1)$$

we note that in $GF(2)$ "-" is the same as "+", where means that a 2-input XOR gate is needed. For VLSI implementation of (3.1), the expression in the two square brackets are computed confluently, and one XOR gate delay $1\,T_x$ is required. As we mentioned, "-" operation is also performed at a cost of $1\,T_x$. Therefore, two XOR gate delays $2\,T_x$ are needed to calculate the three part products $A_H B_H$, $A_L B_L$ and $(A_H + A_L)(B_H + B_L)$.

In order to calculate exact complexities of the above binary polynomial KOA, we introduce some symbols [13]. Let S and T represent for "Space" and "Delay", respectively. And we use $S_\otimes(n)$ and $S_\oplus(n)$ to denote the numbers of AND and

XOR gates, $T_{\otimes}(n)$ and $T_{\oplus}(n)$ to denote the delays produced by AND and XOR gates, respectively.

As we mentioned above, the XOR gate delay $T_{\oplus}(n) = T_{\oplus}(\frac{n}{2}) + 3$. It is easy to get that $2\,T_X$ is required to compute the product of two polynomials of degree 1. Thus, we can establish the recurrence relation of the XOR gate delay, and similarly, we can obtain the recurrence relations of $S_{\otimes}(n)$, $S_{\oplus}(n)$ and $T_{\otimes}(n)$. These recurrence relations illustrate the space and time complexities of the KOA [14].

$$\begin{cases} S_{\otimes}(2) = 3 \\ S_{\otimes}(n) = 3S_{\otimes}(\frac{n}{2}) \end{cases} \qquad \begin{cases} T_{\otimes}(2) = 1 \\ T_{\otimes}(n) = T_{\otimes}(\frac{n}{2}) \end{cases}$$

$$\begin{cases} S_{\oplus}(2) = 4 \\ S_{\oplus}(n) = 3S_{\oplus}(\frac{n}{2}) + 4n - 4 \end{cases} \qquad \begin{cases} T_{\oplus}(2) = 2 \\ T_{\oplus}(n) = T_{\oplus}(\frac{n}{2}) + 3 \end{cases}$$

After solving the above recurrence relations using the formula derived in the new method [13], we obtain the following complexity results for the binary polynomial KOA [17], [14].

$$\begin{cases} S_{\oplus}(n) = 6n^{\log_2 3} \\ S_{\otimes}(n) = n^{\log_2 3} \\ T_{\oplus}(n) = 3\log_2 n - 1 \\ T_{\otimes}(n) = 1 \end{cases} \tag{3.2}$$

3.2 Overlap-free Karatsuba-Ofman Algorithm

In 2010, H.Fan have proposed a new method to implement the polynomial KOA for hardware multipliers [12]. It estimates overlaps in the previous designs so the XOR gate delay of proposed is obviously better than the original KOA. In addition

to the theoretical significance, this new method is also suitable for practical VLSI applications such as designs of hybrid $GF(2^n)$ multipliers.

From the equation (3.1), we can get that the partial polynomials $A_H B_H x^{2m}$, $\{[(A_H + A_L)(B_H + B_L)] - [A_H B_H + A_L B_L]\}x^m$ and $A_L B_L$ are XORed by adding coefficients of common exponents of x together. The VLSI module used to perform this XOR operation is called overlap module [14]. In order to explain overlaps of common exponents of x clearly, we present the following table, which shows ranges of x's exponents in these three polynomials. From the figure, it is easy to know

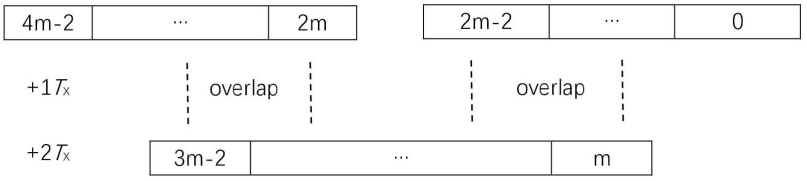


FIGURE 3.1: Ranges of x's exponents of equation (3.1)

that overlaps occur only when $n \geqslant 4$ or $m \geqslant 2$, and there is no overlap when $n = 2$ or $m = 1$.

Because of the overlaps, one more XOR gate delay is needed in the overlap module to compute the summation of the three polynomials $A_H B_H x^{2m}$, $\{[(A_H + A_L)(B_H + B_L)] - [A_H B_H + A_L B_L]\}x^m$ and $A_L B_L$. According to this, a total of 3 XOR gates delays are required in (3.1) besides the cost of the recursive computation of the three partial products.

Therefore, a new method focus on overlaps has been proposed. Instead of splitting two input operands int the "most significant half" and the "least significant half", this new method split operands according to the parity of x's exponents. So we

can rewrite A and B as follows [12]:

$$A = \sum_{i=0}^{n-1} a_i x^i = \sum_{i=0}^{m-1} a_{2i} x^{2i} + \sum_{i=0}^{m-1} a_{2i+1} x^{2i+1} = \sum_{i=0}^{m-1} a_{2i} x^{2i} + x \sum_{i=0}^{m-1} a_{2o+1} x^{2i}$$
$$B = \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{m-1} b_{2i} x^{2i} + \sum_{i=0}^{m-1} b_{2i+1} x^{2i+1} = \sum_{i=0}^{m-1} b_{2i} x^{2i} + x \sum_{i=0}^{m-1} b_{2o+1} x^{2i}$$

Now let $y = x^2$, then operands A and B can be rewritten as

$$A = A_e(y) + xA_o(y)$$
$$B = B_e(y) + xB_o(y),$$

where $A_e(y) = \sum_{i=0}^{m-1} a_{2i} y^i$, $A_o(y) = \sum_{i=0}^{m-1} a_{2i+1} y^i$, and $B_e(y)$ and $B_o(y)$ are defined similarly. Because$A_e(y)$, $A_o(y)$, $B_e(y)$ and $B_o(y)$ are polynomials in degree of y, which is less than m, multiplication operations among them may also be computed recursively. Then we can get the product of A, B as the KOA-like formula as follows

$$\begin{aligned} AB &= (A_e(y) + xA_o(y))(B_e(y) + xB_o(y)) \\ &= \{A_e(y)B_e(y) + x^2 A_o(y)B_o(y)\} + x\{A_e(y)B_o(y) + A_o(y)B_e(y)\} \\ &= \{A_e(y)B_e(y) + yA_o(y)B_o(y)\} + \\ &\quad x\{(A_e(y) + A_o(y))(B_e(y) + B_o(y)) - (A_e(y)B_e(y) + A_o(y)B_o(y))\} \end{aligned} \tag{3.3}$$

Obviously, function (3.3) also includes three partial products and in hardware implementation multiplying a polynomial by x or $y = x^2$ is equivalent to shifting its coefficients left and no extra gate is required. It is clearly to check that the expansion of $A_e(y)B_e(y) + yA_o(y)B_o(y)$ contains with even exponents x, and the expansion of $x\{(A_e(y) + A_o(y))(B_e(y) + B_o(y)) - (A_e(y)B_e(y) + A_o(y)B_o(y))$ contains with odd exponents x. Therefore, no overlap exists when computing their summation, and no gate is needed either.

Consequently, the recurrence relations describing the time and space complexities can be cited as follows:

$$\begin{cases} S_{\otimes}(2) = 3 \\ S_{\otimes}(n) = 3S_{\otimes}(\frac{n}{2}) \end{cases} \qquad \begin{cases} T_{\otimes}(2) = 1 \\ T_{\otimes}(n) = T_{\otimes}(\frac{n}{2}) \end{cases}$$

$$\begin{cases} S_{\oplus}(2) = 4 \\ S_{\oplus}(n) = 3S_{\oplus}(\frac{n}{2}) + 4n - 4 \end{cases} \qquad \begin{cases} T_{\oplus}(2) = 2 \\ T_{\oplus}(n) = T_{\oplus}(\frac{n}{2}) + 2 \end{cases}$$

Then we can get the solutions as below:

$$\begin{cases} S_{\oplus}(n) = 6n^{\log_2 3} - 8n + 2 \\ S_{\otimes}(n) = n^{\log_2 3} \\ T_{\oplus}(n) = 2\log_2 n \\ T_{\otimes}(n) = 1 \end{cases} \tag{3.4}$$

Compared with formula (3.2), the overlap-free method reduces the XOR gate delay $T_{\oplus}(n)$ from $3\log_2 n - 1$ to $2\log_2 n$, which nearly equal to 33% for $n = 2^t$ $(t > 1)$.

3.3 Reconstructed Karatsuba Algorithm

In 2009, Bernstein [15], Zhou and Michalik [16] has optimize the reconstruction part of the Karatsuba formula by factorizing some constant common terms. Bernstein also applied this optimization to the reconstruction of Karatsuba formula and then to two recursion of Karatsuba resulting in $5.46n^{\log_2(n)} + S_{\oplus}$ instead of $6n^{\log_2(n)} + S_{\oplus}$ for the original Karatsuba formula and a delay of $2.5\log_2(n)\,T_{\oplus} + T_{\otimes}$.

Let consider two degree $n-1$ polynomials $A(x) = \sum_{i=0}^{n-1} a_i x^i$ and $B(x) = \sum_{i=0}^{n-1} b_i x^i$ with $n = 2^k$. The method of Karatsuba for polynomial multiplication consists of

expressing the product $C = A \times B$ in terms of three multiplications of polynomial of half size. The detailed computations are given below:

- *Component polynomial formation*(CPF). The CPF consists of splitting A in two halves

$$A(x) = \underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_i x^i}_{A_L} + x^{\frac{n}{2}} \underbrace{\sum_{i=0}^{\frac{n}{2}-1} a_{i+\frac{n}{2}} x^i}_{A_H}$$

  and then generate three polynomials of half size $A_0^{'} = A_L$, $A_1^{'} = A_L + A_H$ and $A_2^{'} = A_H$. The same as $B = B_L + B_H x^{\frac{n}{2}}$, we generate $B_0^{'} = B_L$, $B_1^{'} = B_L + B_H$ and $B_2^{'} = B_H$.

- *Recursive products.* We perform the pairwise products of the CPF of A and B

$$\begin{aligned} C_0^{'} &= A_0^{'} B_0^{'} = A_L B_L \\ C_1^{'} &= A_1^{'} B_1^{'} = (A_L + A_H)(B_L + B_H) \\ C_2^{'} &= A_2^{'} B_2^{'} = A_H B_H \end{aligned} \tag{3.5}$$

- *Reconstruction.* We reconstruct $C = A \times B$ as

$$\begin{aligned} C &= C_0^{'}(1 + x^{\frac{n}{2}} + C_1^{'} x^{\frac{n}{2}} + C_2^{'} x^{\frac{n}{2}}(1 + x^{\frac{n}{2}}) \\ &= C_0^{'} + (C_0^{'} + C_1^{'} + C_2^{'}) x^{\frac{n}{2}} + C_2^{'} x^n \end{aligned} \tag{3.6}$$

The three half size products $C_0^{'}$, $C_1^{'}$ and $C_2^{'}$ of (3.5) are computed by applying the same method recursively. If the recursive computations are performed in parallel we get a parallel multiplier with a sub-quadratic space complexity and a logarithmic delay. And a non-recursive form of the number of XOR gates , AND

gates , the total delay shows as below:

$$\begin{cases} S_{\otimes} = 6n^{\log_2(3)} - 8n + 2 \\ S_{\oplus} = n^{\log_2(3)} \\ T = 3\log_2(n)\, T_{\oplus} + T_{\otimes} \end{cases} \tag{3.7}$$

3.4 Improved Reconstruction by Bernstein

Recently an optimized version of the Karatsuba formula, which we mentioned on previous section, have been proposed. Bernstein have reduced the complexity of the reconstruction step as follows [18]

$$\begin{array}{lll} \text{Step 1.} & R_0 = P_0 + x^{\frac{n}{2}} P_1 & (\text{Cost} = \frac{n}{2} - 1 \text{ bit additions}) \\ \text{Step 2.} & R_1 = R_0(1 + x^{\frac{n}{2}}) & (\text{Cost} = n - 1 \text{ bit additions}) \\ \text{Step 3.} & C = R_1 + P_2 x^{\frac{n}{2}} & (\text{Cost} = n - 1 \text{ bit additions}) \end{array} \tag{3.8}$$

This method reduces the number of bit additions of one recursion of the Karatsuba formula $S_{\oplus} = 7n/2 - 3 + 3S_{\oplus}(n/2)$, which gives for a full recursion $S_{\oplus} = 5.5n^{\log_2 n} - 7n + 3/2$. But this method converses a delay of $T = 3\log_2 n D_{\oplus} + T_{\otimes}$. In this result, we call the reconstruction formula (3.8) as improved reconstruction by Bernstein.

3.5 Comparison

From the previous sections, we have summarized four different kinds of bit-parallel multiplication algorithms,including original KOA, overlap-free KOA, reconstruction Karatsuba and improved reconstruction by Bernstein. Therefore, we collect all these four algorithms results and briefly make a comparison, including space complexity (the number of AND gates and XOR gates)and time complexity. It shows in the form of table as follows: For more specific digital comparison, we

	Space Complexity		Time Complexity (Critical Path Delay)
	#AND $\otimes$	#XOR $\oplus$	
General KA	$n^{\log_2 3}$	$6n^{\log_2 3} - 8n + 2$	$(3\log_2(n) - 1)\,T_\oplus + T_\otimes$
Overlap-free	$n^{\log_2 3}$	$6n^{\log_2 3} - 8n + 2$	$2\log_2(n)\,T_\oplus + T_\otimes$
Reconstruction KA	$n^{\log_2 3}$	$6n^{\log_2 3} - 8n + 2$	$3\log_2(n)\,T_\oplus + T_\otimes$
Reconstruction by Bernstein	$n^{\log_2 3}$	$5.5n^{\log_2 3} - 7n + \frac{3}{2}$	$3\log_2(n)\,T_\oplus + T_\otimes$

FIGURE 3.2: Comparison time and space complexities of four different multiplication algorithms

set several examples of these four architectures later on NIST(National Institute of Standard and Technology) recommended fields. The corresponding time and space complexities and their comparison are given as well. Each kind of algorithm is applied to build efficient polynomial multiplication over NIST recommended fields $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$ and $GF(2^{409})$. Some detailed number of the time and space complexities will also be presented.

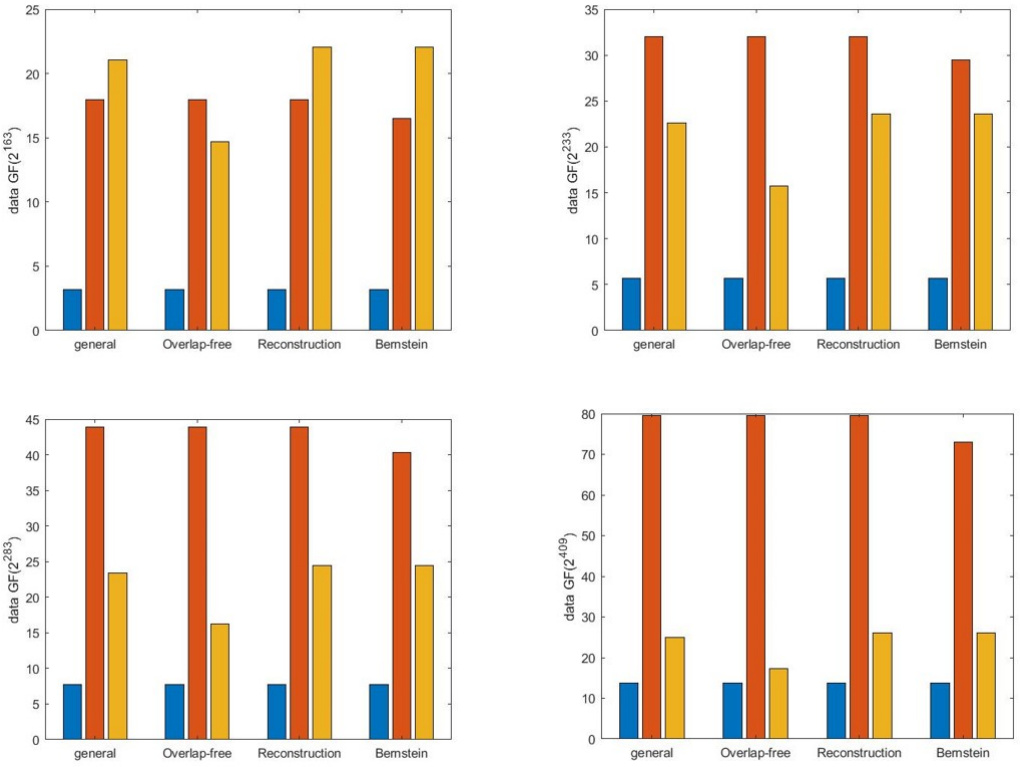
First, according to figure 3.2, we analyse the data in horizon direction, which means that we compare three concepts among four multiplication algorithms, including #AND (the number of AND gates), #XOR (the number of XOR gates) and time complexity.

where we use blue, orange and yellow column to represent #AND, #XOR and time complexity, respectively.

From figure 3.3 we can achieve some disciplines:

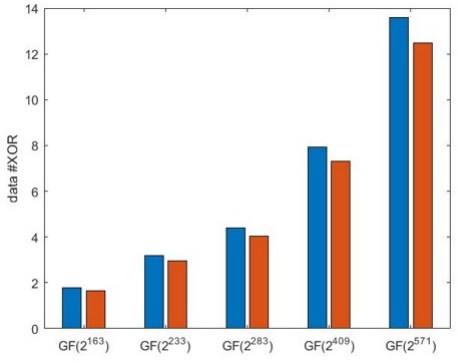
- All the methods have approximately same number of AND gates
- Using the Improved Reconstruction by Bernstein algorithm can achieve lowest number of XOR gates
- Using the Overlap-free Karatsuba algorithm can achieve lowest time complexity

FIGURE 3.3: Horizon direction comparison



Because of the number of AND gates, we only make the vertical comparison two concepts among these four algorithms, including #XOR and time complexity. where red and blue column represent the Improved Reconstruction by Bernstein

FIGURE 3.4: Comparison in the number of XOR gates

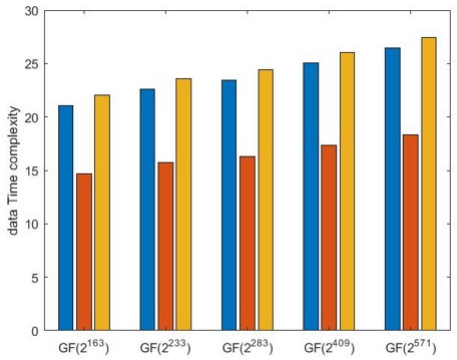


and the other three multiplication algorithms, respectively.

From figure 3.4 we can obtain that

- Improved Reconstruction by Bernstein multiplication algorithm only reduce slight number of XOR gates. The gap between Improved Reconstruction by Bernstein multiplication algorithm and others may obvious with n increasing.

FIGURE 3.5: Comparison in time complexity



where blue, red and yellow column represent the original KOA, Overlap-free Karatsuba and Reconstruction Karatsuba (or Improved Reconstruction by Bernstein), respectively.

From figure 3.5 we can read that

- The apparent gap between Overlap-free Karatsuba and other three algorithms always exists no matter how the value of n changing.

Above these figures and analyses, we can settle that we will just focus on the Overlap-free Karatsuba algorithm and its hardware implement in the following chapters in this thesis. Although the Improved Reconstruction by Bernstein algorithm can do well in the space complexity, especially for the number of XOR

gates, this result is the consequence of the huge value of m. For the limit of the input and output number in the FPGA (Field- Programmable Gate Array) board, which will be mentioned in the next chapter, we will design the hardware implementation when $n = 128$. And in this case, the Improved Reconstruction by Bernstein algorithm does not have a better layout in the comparison of the number of XOR gates. Therefore, we only do the research on Overlap-free Karatsuba multiplication algorithm as the following chapter. We will also compare the proposed hardware implementation with other methods or other published data in space and time complexities in detail.

Chapter 4

Proposed Hardware Implementation of Modified Overlap-free Karatsuba Multiplication Algorithm for $\boldsymbol{GF}(2^n)$

In this chapter, we first introduce the fundamental technology information, including FPGA and its internal architecture, Verilog HDL and ISE software in detail. Then we illustrate the meaning of each code correspond to its function in algorithm. Finally, we compare the proposed module implementation with published article in $GF(2^4)$, $GF(2^8)$ and $GF(2^{16})$ and then achieve a considerable result.

4.1 Fundamental Technology Background

In this section, we briefly introduce the fundamental technology that we need throughout hardware implementation. First, we present what is FPGA and its internal architecture. In order to program it, we explain the reason why we choose Verilog as HDL and ISE as the simulator.

4.1.1 FPGA and their internal architecture

Field Programmable Gate Array (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects [19]. Typical internal structure of FPGA (figure 4.1) comprises of three major elements:

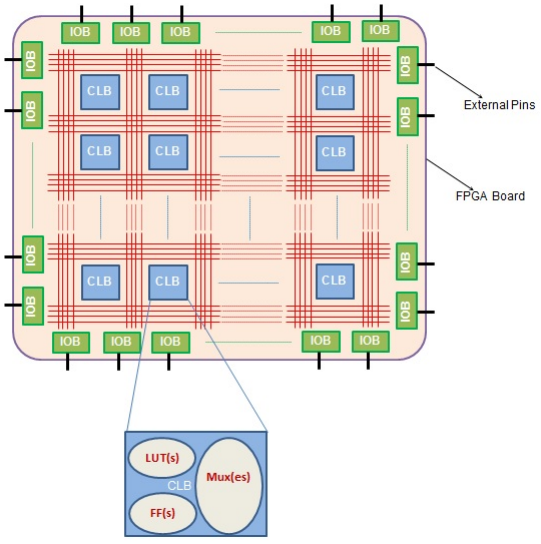


FIGURE 4.1: Internal architecture of a typical FPGA

- Configurable logic blocks (CLBs), shown as blue boxes in figure 4.1, are the resources of FPGA meant to implement logic functions. Each CLB is comprised of a set of slices which are further decomposable into a definite number of look-up tables (LUTs), flip-flops (FFs) and multiplexers (MUXes).

- Input/Output Blocks (IOBs) available at FPGA's periphery facilitate external connections. These programmable blocks carry signals 'to' or 'from' FPGA chip. Figure 4.1 shows IOBs as a set of rectangular boxes enclosed within the FPGA boundary.

- Switch Matrix, shown as red-coloured lines in figure 4.1, is an interconnecting wire-like arrangement within FPGA. These offer connectivity for the CLBs or provide dedicated low impedance, minimum delay path such as global clock line [20].

In general, FPGAs are more flexible than ASICs as they are able to programmed easily to desired functions or applications, with the emphasis on the ease of re-programmability. This is the feature that makes such devices suitable for building processing units for polynomials which are likely to have to adapt to parameter changes from time to time. The fundamental building block of a FPGA is its logic cells. Despite the different hardware used to realize the logic cell functions and different input widths provided by various FPGA vendors, they can be mapped to certain logic functions with the help of the synthesis and mapping tools.

Xilinx Spartan-605 FPGA cells contain a 6-input LUTs improving performance and minimize power in a certain degree. Each CLB in Spartan-605 FPGA consists of two slices, arranged side-by-side as part of two vertical columns. There are three types of CLB slices in the Spartan-605 architecture: SLICEM, SLICEL and SLICEX. Each slice contains four LUTs, eight FFs, and miscellaneous logic. The LUTs are for general-purpose combinatorial and sequential logic support. Synthesis tools take advantage of these highly efficient logic, arithmetic and memory features [21],[22].

4.1.2 Verilog HDL and ISE Design Suite

FPGAs are much more than just a bunch of gates. Although it is possible to build logic circuits of any complexity simply by arranging and connecting logic gates, it is just not practical and efficient. So we need a way to express the logic in some easy to use format that can be converted to an array of gates eventually. And HDL will be focused throughout this thesis. A Hardware Description Language (HDL) is a software programming language used to model the intended operation of a piece of hardware. There are two aspects to the description of hardware that an HDL facilities; the abstract behaviour modelling and hardware structure modelling.

- *Abstract behaviour modelling.* A hardware description language is declarative in order to facilitate the abstract description of hardware behaviour for specification purposes. This behaviour is not prejudiced by structural or design aspects of the hardware intent.
- *Hardware structure modelling.* Hardware structure is capable of being modelled in a hardware description language irrespective of the design's behaviour.

The behaviour of hardware may be modelled and represented at various levels of abstraction during the design process. Higher level models describe the operation of hardware abstractly, while lower level models include more detail, such as inferred hardware structure.

Verilog, standardized as IEEE 1364, is a HDL, which can be used to describe digital circuits in a textual manner [23]. It is most commonly used in the design and verification of digital circuits at the register transfer level (RTL) of abstraction. It is also used in the verification of analog circuits and mixed-signal circuits, as well as in the design of genetic circuits. Verilog gained a strong foothold among advanced, high-end designers for the following reasons:

- The behavioural constructs of Verilog could describe both hardware and test stimulus.
- The Verilog simulator is fast, especially at the gate level.
- The Verilog simulator is an "interpreter", which interpretive software executes source code directly instead of pre-compiling the source code into intermediate "object" code.

According to these features, we choose Verilog as HDL in this thesis to write the code and program the FPGA board.

Simulation is the fundamental and essential part of the design process for any electronic based product; not just FPGA devices. For FPGA devices, simulation is the process of verifying the function characteristics of models at any level or behaviour, that is, from high levels of abstraction down to low levels. The basic arrangement for simulation is shown in Figure 4.2.

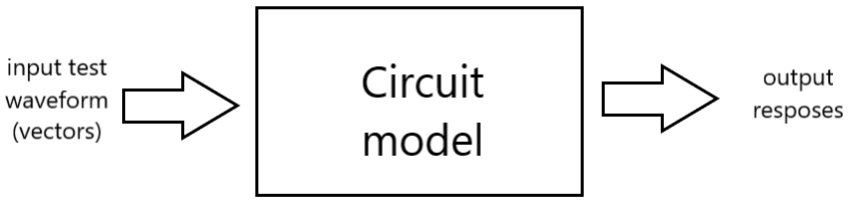


FIGURE 4.2: Basic simulation arrangement

In this thesis, we choose Xilinx ISE software as the simulator to finish the FPGA board hardware simulation. The Xilinx ISE (Integrated Synthesis Environment) produced by Xilinx for synthesis and analysis of HDL design. The ISE software controls all aspects of the design flow [24]

- synthesis or compile its design

- perform timing snaysis
- examine RTL diagrams
- simulate a design's reaction to different stimuli
- configure the target device with the programmer

Through the Project Navigator interface (shown in figure 4.3), you can access all

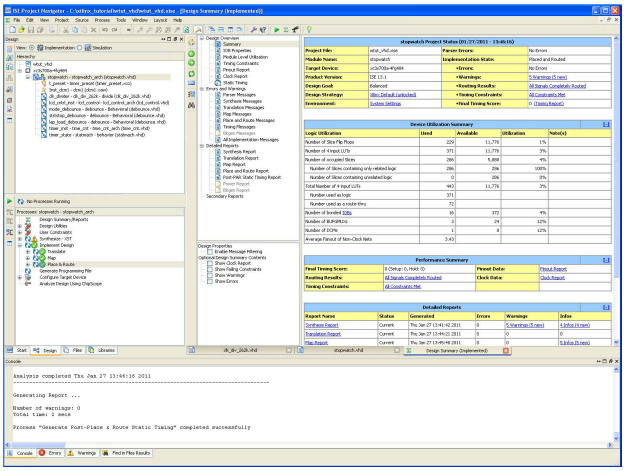


FIGURE 4.3: Project Navigator Interface [24]

of the design entry and design implementation tools. You can also access the files and documents associated with your project.

4.2 Hardware implementation of Modified Overlap-free Karatsuba algorithm for $GF(2^n)$ on FPGA

In this section, we first present the complexity analysis by applying 1-step Overlap-free KA (Karatsuba) for even-term polynomials (ETP). Then we apply the proposed modified algorithm into FPGA and achieve the results for $GF(2^{128})$.

4.2.1 Fundamental Multiplication Modules for $GF(2^4)$

We now convey an example to compare the proposed modified method with the original KOA. We assume $n = 4$ and then let

$$A = a_3x^3 + a_2x^2 + a_1x + a_0 = A_Hx^2 + A_L$$
$$B = b_3x^3 + b_2x^2 + b_1x + b_0 = B_Hx^2 + B_L$$

where $A_H = a_3x + a_2$, $A_L = a_1x + a_0$, $B_H = b_3x + b_2$ and $B_L = b_1x + b_0$ are the polynomials of degree 1 in x. Then the original KOA computes the product AB using

$$AB = A_HB_Hx^4 + \{[(A_H + A_L)(B_H + B_L)] + [A_HB_H + A_LB_L]\}x^2 + A_LB_L \quad (4.1)$$

there are three products of polynomials of degree 1 in (4.1), and they can be computed recursively using the KOA at a cost of $2\,T_x$.

To show the role of the overlap in 4.1, we group the three products in 4.1 and write them as polynomials of degree 2 in x as follows:

$$\begin{aligned} d_2x^2 + d_1x + d_0 &= A_H + B_H \\ e_2x^2 + e_1x + e_0 &= [(A_H + A_L)(B_H + B_L)] + [A_HB_H + A_LB_L] \\ f_2x^2 + f_1x + f_0 &= A_LB_L \end{aligned}$$

Then we have

$$\begin{aligned} AB &= (d_2x^2 + d_1x + d_0)x^4 + (e_2x^2 + e_1x + e_0)x^2 + (f_2x^2 + f_1x + f_0) \\ &= d_2x^6 + d_1x^5 + (d_0 + e_2)x^4 + e_1x^3 + (e_0 + f_2)x^2 + f_1x + f_0 \end{aligned} \quad (4.2)$$

Obviously, one XOR gate delay $1\,T_x$ is required to compute the overlap summations $(d_0 + e_2)$ and $(e_0 + f_2)$. Because we need $2\,T_x$ to perform the XOR operations in

the curly bracket of (4.1), the total number of XOR gate delays of the original KOA is $2 + 1 + 2 = 5$.

Let $y = x^2$, the proposed method in Chapter 2 function computes AB as follows [13]

$$\begin{aligned} AB &= (A_e(y) + xA_o(y))(B_e(y) + xB_o(y)) \\ &= \{A_e(y)B_e(y) + x^2A_o(y)B_o(y)\} + x\{A_e(y)B_o(y) + A_o(y)B_e(y)\} \\ &= \{A_e(y)B_e(y) + yA_o(y)B_o(y)\}+ \\ &\quad x\{(A_e(y) + A_o(y))(B_e(y) + B_o(y)) - (A_e(y)B_e(y) + A_o(y)B_o(y))\} \end{aligned} \tag{4.3}$$

where $A_e(y) = a_2y + a_0$, $A_o(y) = a_3y + a_1$, $B_e(y) = b_2y + b_0$ and $B_o(y) = b_2y + b_1$ are polynomials of degree 1 in y, where we will do modified into the architecture.

Then we define four polynomials of degree 2 in y as belows:

$$\begin{aligned} p_2y^2 + p_1y + p_0 &= A_e(y)B_e(y) \\ q_2y^2 + q_1y + q_0 &= A_o(y)B_o(y) \\ r_2y^2 + r_1y + r_0 &= (A_e(y) + A_o(y))(B_e(y) + B_o(y)) \\ s_2y^2 + s_1y + s_0 &= A_e(y)B_e(y) + A_o(y)B_o(y) \end{aligned}$$

We need $1\,T_x$ to perform "+" operations in the last two equations. We also need $2\,T_x$ to compute the three products of polynomials of degree 1 in $y$in the above four equations. Then we have the product AB can be shown as follows:

$$\begin{aligned} AB &= \{(p_2y^2 + p_1y + p_0) + y(q_2y^2 + q_1y + q_0)\}+ \\ &\quad x\{(r_2y^2 + r_1y + r_0) + (s_2y^2 + s_1y + s_0)\} \\ &= q_2x^6 + (p_2 + q_1)x^4 + (p_1 + q_0)x^2 + p_0+ \\ &\quad (r_2 + s_2)x^5 + (r_1 + s_1)x^3 + (r_0 + s_0)x \end{aligned} \tag{4.4}$$

Evidently, one XOR gate delay is needed to obtain the summations in the five brackets. Therefore the total number of XOR gate delay is 4, and $1\,T_x$ has been saved compared to the original KOA.

Figure 4.4 shows the multiplier architecture by applying one step Overlap-free KA algorithm as above example, if $m = n$ is even. The multiplier includes three stages: the splitting stage, the sub-multiplier stage and the alignment stage, where three sub-multiplier operate in parallel.

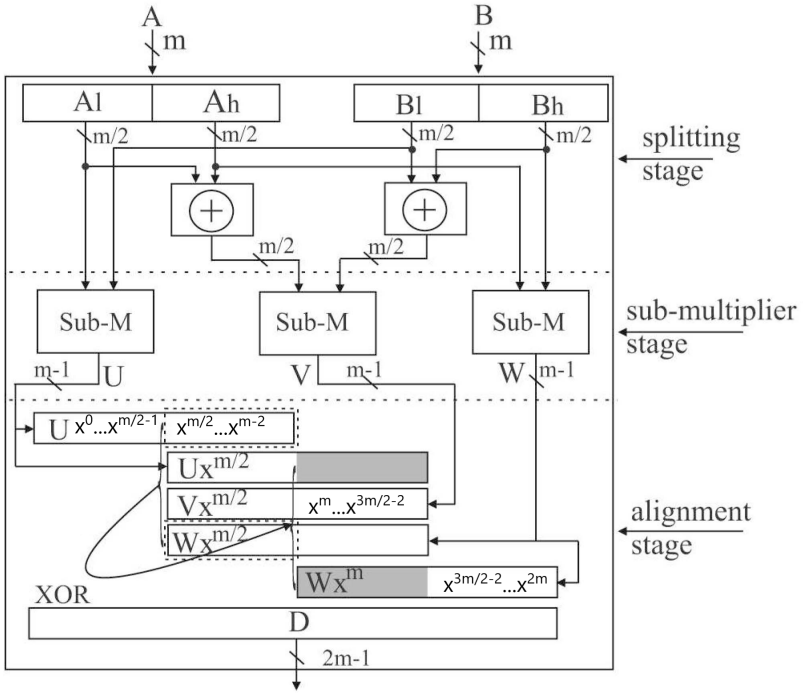


FIGURE 4.4: Multiplier Architecture by applying Overlap-free KA

In this architecture [16], we can efficiently define which part's function. The splitting stage requires m XOR gates to generate the inputs for the middle multiplier, which compute the product of $A_e(y) + A_o(y)$ and $B_e(y) + B_o(y)$. The alignment stage merges the output of sub-multipliers according to their degrees. Both in figure 4.4 and (4.5), common sub-expressions are found when calculating $D_{\frac{m}{2}\ldots m-2}$

and $D_{m...\frac{3m}{2}-2}$ [25]

$$\begin{cases} D_{\frac{m}{2}...m-2} = [U_{\frac{m}{2}...m-2} + W_{0...\frac{m}{2}-2}] + U_{0...\frac{m}{2}-2} + V_{0...\frac{m}{2}-2} \\ D_{m...\frac{3m}{2}-2} = [U_{\frac{m}{2}...m-2} + W_{0...\frac{m}{2}-2}] + W_{\frac{m}{2}...m-2} + V_{\frac{m}{2}...m-2} \end{cases} \tag{4.5}$$

Using this architecture and proposed modified Overlap-free Karatsuba algorithm, we can implement it on the typical FPGA board and analyse its features.

4.2.2 Implementation of proposed modified algorithm for $GF(2^n)$ on FPGA

In this part, we will combine the proposed modified Overlap-free KA algorithm with the Multiplier architecture, and use Verilog HDL to complete the implementation of proposed modified Overlap-free KA algorithm for $GF(2^n)$, where $n = 128$, on Xilin Spartan-605 board.

In order to make easier understand, we first make the module when $n = 2$ as an example. The detail Verilog HDL code has been shown in table 4.1.

TABLE 4.1: Verilog HDL $n = 2$ module

```
1 module mul_2_module(
2 input [1:0] A,
3 input [1:0] B,
4 output[3:0] mul_2
5 );
6  assign mul_2[0]=A[0]&B[0];
7  assign mul_2[2]=A[1]&B[1];
8  assign mul_2[3:0]=
9  {A[1]&B[1],
10 (A[0]^A[1])&(B[0]^B[1])^mul_2[0]^mul_2[2],A[0]&B[0]};
11 endmodule
```

Because of the value of n, in Chapter 3 we have mentioned, no overlap will occur at this time. To analyse the table 4.1 more clearly, we show some typical steps explanation as follows:

```
assign mul_2[0]=A[0]&B[0]
```

equal to function $A_L B_L$

```
assign mul_2[2]=A[1]&B[1]
```

equal to function $A_H B_H$

```
assign A[0]^A[1])&(B[0]^B[1])^mul_2[0]^mul_2[2]
```

equal to function $[(A_H + A_L)(B_H + B_L)] + [A_H B_H + A_L B_L]$

Then we extend the value of n from 2 to 4, which Verilog HDL shows in table 4.2. Since value 4 is exact double size of 2, we use nested and transferred statement to finish the module. For this value, overlap occurs during the alignment stage and then we apply the proposed algorithm in this part, which also shows in the table 4.2, the specific code as below:

```
assign d7=d2^d1^d0;
assign mul_4[7:0]={d2[3:2],(d2[1:0]^d7[3:2]),
 (d0[3:2]^d7[1:0]),d0[1:0]}
```

therefore, we can extend the value of n until 128. The detail Verilog HDL code has been shown in Appendix in the end of this thesis.

TABLE 4.2: Verilog HDL $n = 4$ module

```
1 module mul_4_module(
2 input [3:0]A,
3 input [3:0]B,
4 output[7:0]mul_4
5 );
6 wire[3:0] d0,d1,d2,d7;
7 mul_2_module u0((A[1:0]),(B[1:0]),(d0));
8 mul_2_module u1((A[1:0]^A[3:2]),(B[1:0]^B[3:2]),(d1));
9 mul_2_module u2((A[3:2]),(B[3:2]),(d2));
10 assign d7=d2^d1^d0;
11 assign mul_4[7:0]={d2[3:2],(d2[1:0]^d7[3:2]),
12 (d0[3:2]^d7[1:0]),d0[1:0]};
13 endmodule
```

Following the nested and transferred statement, we finally get the module of $n = 128$ in Appendix A. Then we use the simulator, Xilinx ISE software, to complete

the simulation of all the huge module. From the simulator, we first implement this module in to the typical board, Xilinx Spartan-605. And then complete implement design part, including translate, map and place & route. After that, we also generate programming file and from the simulation part, we achieve the RTL schematic in figure 4.5.

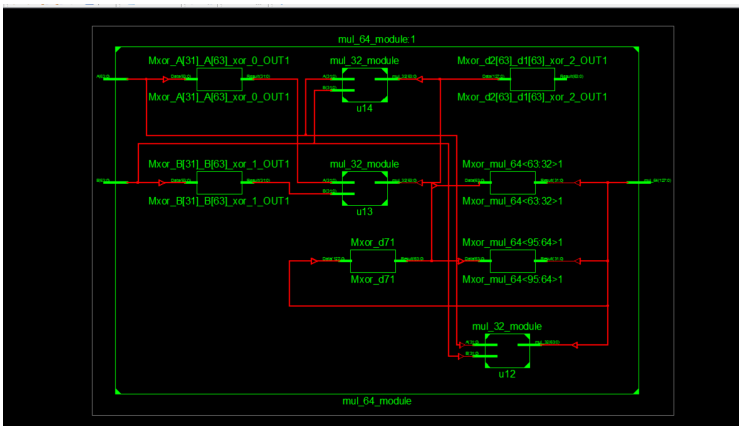


FIGURE 4.5: RTL Schematic

In figure 4.5, we can directly know that our module exactly follows multiplier architecture, in figure 4.4. There are several CLBs shown in the RTL scheme, including the input, output and the name of the block, which also illustrates the steps.

In each CLB, when we check in it, it shows the kinds of LUTs, FFs and MUX. And we summarise the exact kinds of LUTs, in figure 4.6 which occurs in the whole module.

Internally, LUTs comprises of 1-bit memory cells and a set of multiplexers. One value among these SRAM bits will be available at the LUT's output depending on the value(s) fed to the control line(s) of the multiplexers. For these features, LUTs is an important cell in CLB. If we can design the LUT's structure, we may optimize the speed of input and output, which reflects on chips is the speed of reading and writing information.

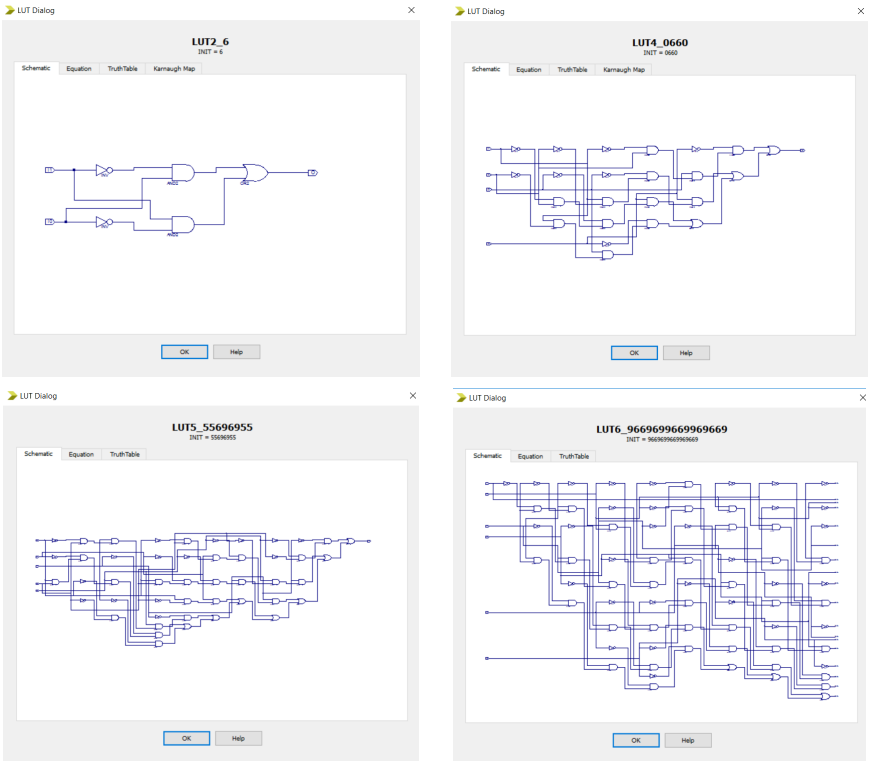


FIGURE 4.6: 2, 4, 5 and 6-input LUTs

We can also get ISm simulation in the simulator, shown in figure 4.7 and 4.8 without and with input respectively. We can control the value of each input, directly achieve the output value, analyse the time delay and get wave changes if we design the clockwise.

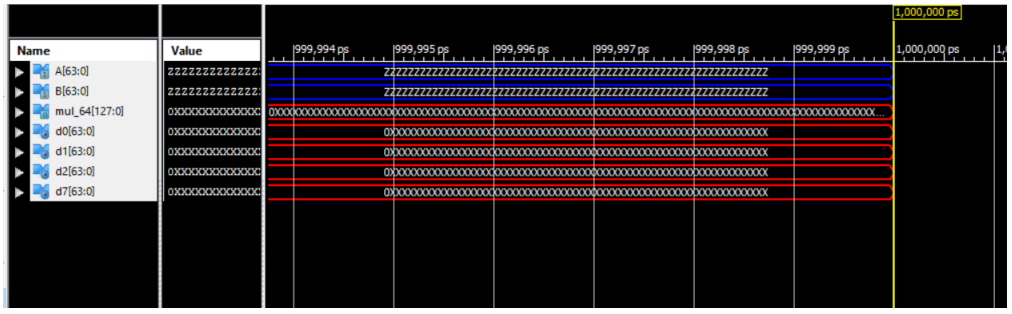


FIGURE 4.7: ISim simulation without input module

In the next section, we will discuss the time delay and the comparison value of output using the ISm simulation.

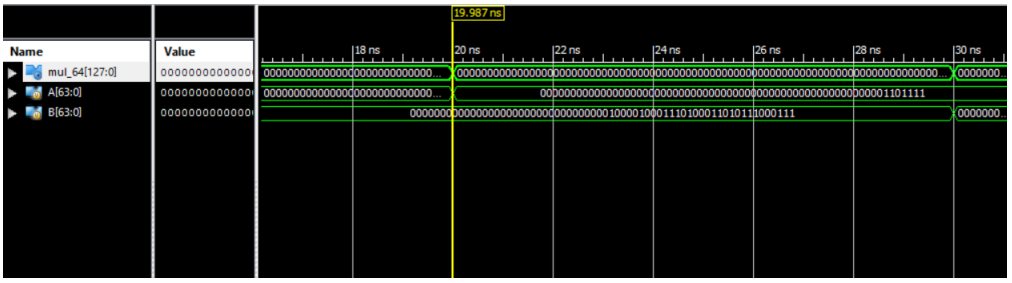


FIGURE 4.8: ISm simulation with proposed input module

4.3 Complexity Comparison

In this section, we first present the simulation results of proposed modified module in Verilog code and ISE system. Then compare it with other published multipliers, for $GF(2^4)$,$GF(2^8)$ and $GF(2^{16})$ field, referencing specific paper.

4.3.1 Synthesis results

First, we take the simulation results of proposed modified module using overlap-free Karatsuba multiplication algorithm for $GF(2^4)$ as an example, which has been shown in following codes and figures.

The proposed modified module has been coded in Verilog in Appendix B. From the code, the first two inputs have been settled 001, 001 respectively and the system needs to wait 100ns for global reset to finish. Then the value of B, which is one of inputs, has changed from 001 to 111 every 1ns. And using the simulation system we can achieve the following figure.

The figure shows the binary equivalent of multiplication of two 4-bit numbers to give the product. Ports A and B are the input ports that accept the numbers to be multiplied. The port mul_4 is the output port, where the product of the two aforesaid numbers are obtained. For example, the product of 0001 and 0001 (binary equivalents), specified at the ports A and B respectively, is obtained at port mul_4, output port, as 00000001. Similarly, products of other specified finite

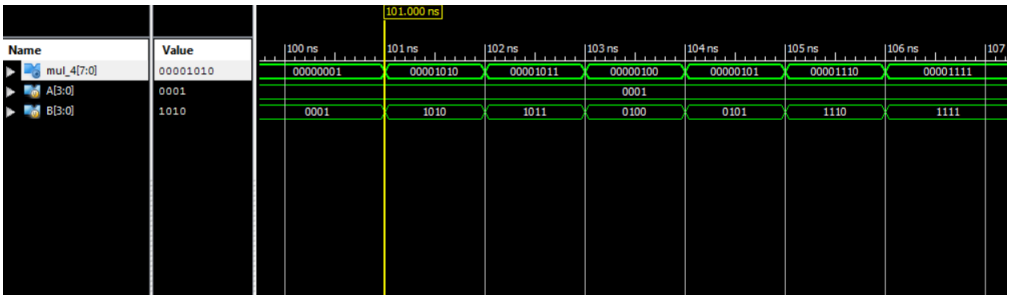


FIGURE 4.9: Simulation result of proposed modified module for $GF(2^4)$

field $GF(2^8)$ and $GF(2^{16})$ are obtained, shown as figure 4.10 and 4.11 respectively.

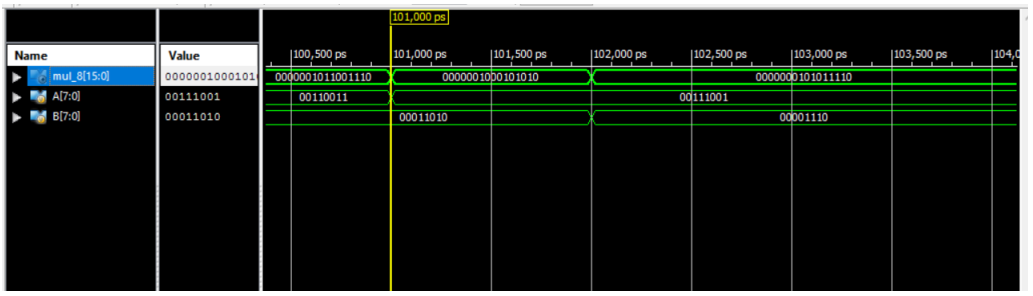


FIGURE 4.10: Simulation result of proposed modified module for $GF(2^8)$

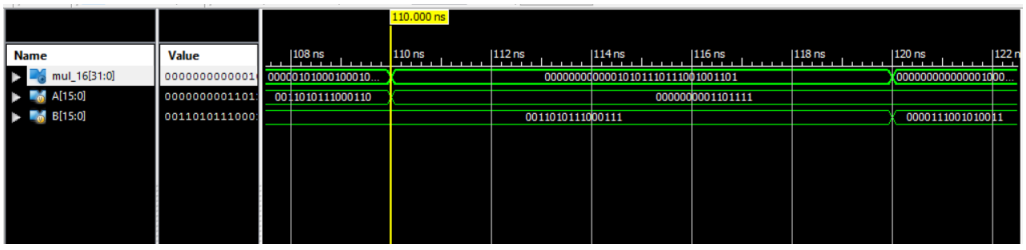


FIGURE 4.11: Simulation result of proposed modified module for $GF(2^{16})$

4.3.2 Comparison

According to the simulation results, we reference the paper called *FPGA Based Modified Karatsuba Multiplier* [32] because it has valuable kinds of finite field multipliers. We have studied the performance of each multiplier over $GF(2^4)$, $GF(2^8)$ and $GF(2^{16})$ employing the Xilinx ISE simulation tool. All multipliers

are implemented on Spartan-605 device. These multipliers are compared based on number of slices, 4-input LUTs, bonded I/O blocks and maximum combinational path delay.

Different GF Multiplier	No.of slices (out of 6822)	No.of 4-input LUTs (out of 27288)	No.of bonded IOBs (out of 296)	Max combinational Path delay(ns)
Mastrovito [10]	7	12	12	13.195
Paar-Roelse [11]	7	12	12	13.083
Massy Omura [12]	7	13	12	14.932
Hasan Masoleh [13]	7	12	12	13.271
Berlekamp [14]	8	15	12	12.985
Karatsuba [15]	9	15	12	14.790
Modified Karatsuba [16]	6	11	12	13.057
Proposed Overlap-free	6	16	12	10.101

FIGURE 4.12: Comparison of device utilization and combinational path delay of proposed modified multipliers and other multipliers for $GF(2^4)$

Table in figure 4.12 shows the result of device utilization and combinational path delay of various types of $GF(2^4)$ multipliers. The number of slices and combinational path delay for proposed modified multiplier are 6 out of 6822 and 10.101 ns respectively. Whereas, the minimum number of slices and combinational path delay for Modified Karatsuba multiplier are 6 out of 6822 and 13.057 ns respectively. Although they have the same number of slices, the combinational path delay for proposed modified multiplier is 23.4% lower than the one for Modified Karatsuba, which is the minimum combinational path delay among the other multipliers.

In order to make the comparison clearer, we only implement the polynomial multiplication part, which will be research further in Chapter 5. So we compare Karatsuba, Modified Karatsuba and proposed modified Overlap-free algorithm multiplication, in the following comparison for $GF(2^8)$ and $GF(2^{16})$.

Tables 4.3 and 4.4 illustrate the result of device utilization and combinational path delay of three types multipliers for $GF(2^8)$ and $GF(2^{16})$ respectively. The combinational path delays for proposed modified Overlap-free multiplier are 13.425

TABLE 4.3: Comparison of device utilization and combinational path delay for $GF(2^8)$

Different GF Multipliers	No. of slices (out of 6822)	No.of 4-input LUTs (out of 27288)	No.of boned IOBs (out of 296)	Max combinational path delay(ns)
Karatusba[31]	66	115	24	20.028
Modified Karatsuba[32]	36	62	24	17.035
Proposed modified Overlap-free	60	74	24	13.425

TABLE 4.4: Comparison of device utilization and combinational path delay for $GF(2^{16})$

Different GF Multipliers	No. of slices (out of 6822)	No.of 4-input LUTs (out of 27288)	No.of boned IOBs (out of 296)	Max combinational path delay(ns)
Karatusba[31]	252	395	52	27.012
Modified Karatsuba[32]	130	230	52	24.413
Proposed modified Overlap-free	248	254	52	18.277

ns and 18.277 ns respectively. For $GF(2^8)$, the combinational path delay for proposed modified Overlap-free multiplier is 32.97% lower than that for Karatsuba multiplier and 21.19% lower than the one for Modified Karatsuba multiplier. For $GF(2^{16})$, the combinational path delay for proposed modified Overlap-free multiplier is 32.34% and 25.13% lower than that for Karatsuba multiplier and Modified Karatsuba multiplier respectively. Although the number of slices occupied of proposed modified Overlap-free multiplier is not obviously less than the other two methods, the max combinational path delay of proposed modified Overlap-free multiplier has a significant reduction among these three methods.

In conclusion, proposed modified multiplier module has less hardware space complexity and time complexity than other finite field multipliers. And this result proves the comparison made in Chapter 3, that Overlap-free Karatsuba algorithm

multiplication has lower time delay comparing with other kinds of finite field multipliers.

Chapter 5

Conclusion

In this chapter, we summarize the main contribution in this thesis and propose the future work in related implementation.

5.1 Summary of Contribution

Bit-parallel multiplication applied with modified Overlap-free Karatsuba algorithm has been investigated in this thesis when n is presented for NIST recommended fields. Our main contribution is summarized as follows:

- Compared Overlap-free Karatsuba algorithm with other existing popular algorithm, such as Karatsuba algorithm, reconstruction Karatsuba algorithm and improved reconstruction by Bernstein, and achieve the advantage of proposed algorithm in the max combinational path delay.
- Implement the proposed modified Overlap-free Karatsuba algorithm multiplication on FPGA board, simulate in the ISE Xilinx system and achieve the synthesis result in NIST recommended field $n = 128$.

- Compared proposed modified Overlap-free Karatsuba algorithm multiplication with especially published multiplications (Karatsuba and Modified Karatsuba multiplications),for $GF(2^4)$, $GF(2^8)$ and $GF(2^{16})$. The results of the comparison have confirmed that proposed modified Overlap-free Karatsuba algorithm multiplication provides a obvious reduction on the max combinational path delay.

5.2 Future Work

Proposed modified Overlap-free Karatsuba algorithm multiplication effects the most research efforts on parallel finite field multiplications. In this thesis, it talks about multiplication part of a bit-parallel polynomial basis multiplier without the reduction modulo of the irreducible polynomial. So the potential work will discusses implementation of the irreducible polynomial.

There are two steps to implement a bit-parallel polynomial basis multiplier in $GF(2^n)$: polynomial multiplication and reduction modulo [33]. In this thesis, we finish the first step, and define that proposed modified Overlap-free Karatsuba algorithm polynomial multiplication is the best in critical path among the other methods. In the optimization work, to make this result more persuasive, we will choose a irreducible polynomial to reduce modulo in the result of $\mathbf{A}(\mathrm{x})$ and $B(\mathrm{x})$ production $D(\mathrm{x})$. The most significant $m-1$ terms of $D(\mathrm{x})$ are iteratively reduced to polynomials with degree less than m by using the irreducible polynomial $F(\mathrm{x})$ [25]. The reduction operation usually costs a small number of gates compared with KOMs because $F(\mathrm{x})$ typically has low weight as recommended by the NIST in [34] and the SECG in [35]. So adding the reduction modulo, will not effect the recent solution.

TABLE 5.1: Complexity for modular reduction operations[25]

m	113	128	163	193	233	283
# XOR	232	527	665	398	537	1159

Table 5.1 shows the number of XOR gates for the finite field with the irreducible given in equation 5.1.

$$\begin{cases} GF(2^{113}) : F(x)_{113} = x^{113} + x^9 + 1 \\ GF(2^{128}) : F(x)_{128} = x^{128} + x^8 + x^7 + x^2 + x + 1 \\ GF(2^{163}) : F(x)_{163} = x^{163} + x^7 + x^6 + x^3 + 1 \\ GF(2^{193}) : F(x)_{193} = x^{193} + x^{15} + 1 \\ GF(2^{233}) : F(x)_{233} = x^{233} + x^{74} + 1 \\ GF(2^{283}) : F(x)_{283} = x^{283} + x^{12} + x^7 + x^5 + 1 \end{cases} \tag{5.1}$$

$F(x)_{128}$ is adopted for GHASH function in the AES-GCM standard [36], and other polynomials are recommended for elliptic curve crypto-systems by NIST FIPS-186-2 standard [34] or the SECG domain parameters in [35].

In conclusion, the future work implementation of proposed modified Overlap-free Karatsuba algorithm multiplication for $GF(2^n)$, where $n = 128$, can be concurrently applied polynomial multiplication and reduction modulo with the function of $F(x)_{128} = x^{128} + x^8 + x^7 + x^2 + x + 1$.

Bibliography

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, Feb 1978.

[2] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory* , vol. 31, pp. 469–472, September 2006.

[3] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comp.*, vol. 48, no. 177, pp. 203–209, 1987.

[4] V. S. Miller, Use of Elliptic Curves in Cryptography, pp. 417–426. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986.

[5] R. Lidl and H. Niederreiter, "Introduction to finite fields and their applications", *Cambridge university press*, 1994.

[6] Y. Li, Y. Zhang, and X. Guo, "Efficient non-recursive bit-parallel Karatsuba multiplier for a special class of trinomials," VLSI Design, vol. 2018, 2018.

[7] H. Fan,"A Chinese remainder theorem approach to bit-parallel GF(2^n) polynomial basis multipliers for irreducible trinomials", *IEEE Transactions on Computers*, no. 1, pp. 1-1, 2016.

[8] Y. Li, X. Ma, Y. Zhang, and C. Qi, "Mastrovito form of non-recursive Karatsuba multiplier for all trinomials," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1573-1584, 2017.

[9] M. Imran and M. Rashid, "Architectural review of polynomial bases finite field multipliers over $GF(2^m)$," in Communication, Computing and Digital Systems (C-CODE), *International Conference on. IEEE*, pp. 331-336, 2017.

[10] A. A. Karatsuba, "The complexity of computations", *Proceedings of the Steklov Institute of Mathematics Interperiodica Translation*, vol. 211, pp. 169-183, 1995.

[11] Karatsuba, A., and Ofman Y., "Multiplication of Multidigit Numbers on Automata", *Soviet Physics-Doklady (English translation)*, vol. 7, no. 7, pp. 595-596, 1963.

[12] H. Fan, J. Sun, M. Gu, and K.-Y. Lam, "Overlap-free KaratsubaOfman polynomial multiplication algorithms," *IET Information security*, vol. 4, no. 1, pp. 8-14, 2010.

[13] Fan, H., and Hasan, M. A., "A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields",*IEEE Transactions on Computers*, vol. 56, no. 2, pp. 224-233, Feb. 2007.

[14] Gathen, J. V. Z., and Shokrollahi, J., "Efficient FPGA-based Karatsuba Multipliers for Polynomials over F_2", *Proc. 12th Workshop on Selected Areas in Cryptography (SAC 2005)*, LNCS 3897 pp.359-369, 2006.

[15] D. J. Bernstein, "Batch binary Edwards," in *Advances in Cryptology - CRYPTO, 29th Annual International Cryptology Conference*, pp. 317-336, 2009.

[16] G. Zhou and H. Michalik, "Comments on a new architecture for a parallel finite field multiplier with low complexity based on composite field",*IEEE Transactions on Computers*, vol. 59, no. 7, pp. 10071008, 2010.

[17] Paar, C., "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields",*IEEE Transactions on Computers*, vol. 45, no. 7, pp. 856-861, July 1996

[18] C. Negre, "Efficient binary polynomial multiplication based on optimized Karatsuba reconstruction," *Journal of Cryptographic Engineering*, vol. 4, no. 2, pp. 91-106, 2014.

[19] X. Inc., "Field programmable gate array (fpga)", [Online], Available: http://www.xilinx.com/training fpga/fpga-eld-programmable-gate-array.htm, 2013.

[20] Sneha H.L., "Purpose and Internal Functionality of FPGA Look-Up Tables", [Online], Available:X. Inc. (2013) Field programmable gate array (fpga). [Online]. Available: https://www.allaboutcircuits.com/technical-articles/purpose-and-internal-functionality-of-fpga-look-up-tables/

[21] X. Inc., "Spartan-6 FPGA Configurable Logic Block", *User Guide*, UG384(v1.1), February 23, 2010.

[22] X. Inc., "Spartan-6 Family Overview", *Product Specification*, DS160(v2.0), October 25, 2011.

[23] Nielsen AA, Der BS, Shin J, Vaidyanathan P, Paralanov V, Strychalski EA, Ross D, Densmore D, Voigt CA, "Genetic circuit design automation",*Science*, vol. 352 (6281), 2016.

[24] X. Inc., "ISE In-Depth Tutorial", UG695(v13.3), October 19, 2011.

[25] Gang Zhou, Harald Michalik, and László Hinsenkamp, "Complexity analysis and efficient implementations of bit parallel finite field multipliers based on Karatsuba-Ofman algorithm on FPGAs", *IEEE Transactions Very Large Scale Integration (VLSI) System*, vol. 18, no. 7, July 2010.

[26] T. Zhang and K.K. Parhi, "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 734-749, July 2001.

[27] C. Paar, P. Fleischmann, and P. Roeise, "Efficient Multiplier Architectures for Galois Fields GF(2^{4n})", *IEEE Trans. Computers*, vol. 47, no. 2, pp. 162-170, Feb. 1998.

[28] C. A. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in GF(2^m)", *IEEE Transactions on Computers*,34(8):709- 717, Aug 1985.

[29] A. Reyhani-Masoleh and M.A. Hasan, "A New Construction of Massey-Omura Parallel Multiplier over GF(2^m)", *IEEE Trans. Computers*, vol. 51, no. 5, pp. 511-520, May 2002.

[30] Berlekamp, E. R., "Bit-Serial Reed-Solomon Encoder", *IEEE Trans. Inform. Theory*, Vol. IT-28, pp. 869-874 (1982).

[31] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers", in *Doklady Akad. Nauk SSSR*, vol. 145, no. 293-294, pp. 85, 1962

[32] Jagannath Samanta, Razia Sultana, Jaydeb Bhaumik, "FPGA based modified Karatsuba multiplier", *International Conference on VLSI and Signal Processing (ICVSP)*, vol. 10-12, January 2014.

[33] H. Wu, "Bit-parallel finite field multiplier and squarer using polynomial basis," *IEEE Transactions on Computers*, vol. 51, no. 7, pp. 750758, 2002.

[34] *Digital Signature Standard (DSS)*, FIPS PUB 186-2, NIST, 2000.

[35] Certicom Research, ON, Canada, "SEC 2: Recommended ellipltic curve domain parameters", 2000.

[36] D.A.McGrew and J. Viega. "The Galois/counter mode of operation (GCM)", NIST, May 2005.

Appendix A

Proposed Modified Overlap-free KA Algorithm in $GF(2^{128})$ Verilog code

```
module mul_2_module(
input   [1:0] A,
input   [1:0] B,
output[3:0] mul_2
);
        assign mul_2[0]=A[0]&B[0];
        assign mul_2[2]=A[1]&B[1];
        assign mul_2[3:0]={A[1]&B[1],(A[0]^A[1])&(B[0]^B[1])^mul_2[0]^mul_2[2],A[0]&B[0]};
        endmodule

module mul_4_module(
input [3:0]A,
input [3:0]B,
output[7:0]mul_4
);
wire[3:0] d0,d1,d2,d7;
mul_2_module u0((A[1:0]),(B[1:0]),(d0));
mul_2_module u1((A[1:0]^A[3:2]),(B[1:0]^B[3:2]),(d1));
mul_2_module u2((A[3:2]),(B[3:2]),(d2));
assign d7=d2^d1^d0;
assign mul_4[7:0]={d2[3:2],(d2[1:0]^d7[3:2]),(d0[3:2]^d7[1:0]),d0[1:0]};
endmodule

module mul_8_module(
input [7:0]A,
input [7:0]B,
output [15:0]mul_8
```

```
);
wire[7:0]d0,d1,d2,d7;
mul_4_module u3((A[3:0]),(B[3:0]),(d0));
mul_4_module u4((A[3:0]^A[7:4]),(B[3:0]^B[7:4]),(d1));
mul_4_module u5((A[7:4]),(B[7:4]),(d2));
assign d7=d2^d1^d0;
assign mul_8[15:0]={d2[7:4],(d2[3:0]^d7[7:4]),(d0[7:4]^d7[3:0]),d0[3:0]};
endmodule

module mul_16_module(
input [15:0]A,
input [15:0]B,
output[31:0]mul_16
);
wire[15:0]d0,d1,d2,d7;
mul_8_module u6((A[7:0]),(B[7:0]),(d0));
mul_8_module u7((A[7:0]^A[15:8]),(B[7:0]^B[15:8]),(d1));
mul_8_module u8((A[15:8]),(B[15:8]),(d2));
assign d7=d2^d1^d0;
assign mul_16[31:0]={d2[15:8],(d2[7:0]^d7[15:8]),(d0[15:8]^d7[7:0]),d0[7:0]};
endmodule

module mul_32_module(
input [31:0]A,
input [31:0]B,
output[63:0]mul_32
);
wire[31:0]d0,d1,d2,d7;
mul_16_module u9((A[15:0]),(B[15:0]),(d0));
mul_16_module u10((A[15:0]^A[31:16]),(B[15:0]^B[31:16]),(d1));
mul_16_module u11((A[31:16]),(B[31:16]),(d2));
assign d7=d2^d1^d0;
assign mul_32[63:0]={d2[31:16],(d2[15:0]^d7[31:16]),(d0[31:16]^d7[15:0]),d0[15:0]};
endmodule

module mul_64_module(
input [63:0]A,
input [63:0]B,
output[127:0]mul_64
);
wire[63:0]d0,d1,d2,d7;
mul_32_module u12((A[31:0]),(B[31:0]),(d0));
mul_32_module u13((A[31:0]^A[63:32]),(B[31:0]^B[63:32]),(d1));
mul_32_module u14((A[63:32]),(B[63:32]),(d2));
assign d7=d2^d1^d0;
```

```
assign mul_64[127:0]={d2[63:32],(d2[31:0]^d7[63:32]),(d0[63:32]^d7[31:0]),d0[31:0]};
endmodule

module mul_128_module(
input[127:0] A,
input[127:0] B,
output[255:0] mul_128
);
wire[127:0] d0,d1,d2,d7;
mul_64_module mul_641((A[63:0]),(B[63:0]),(d0));
mul_64_module mul_642((A[63:0]^A[127:64]),(B[63:0]^B[127:64]),(d1));
mul_64_module mul_643(A[127:64],B[127:64],(d2));
assign d7 = d1^d2^d0;
assign mul_128[255:0] = {d2[127:64],((d2[63:0])^(d7[127:64])),((d0[127:64])^(d7[63:0])),d0[63:0]
endmodule
```

Appendix B

Simulation code of proposed modified module using Overlap-free Karatsuba multiplication algorithm for $\boldsymbol{GF}(2^4)$

```
1  module test_sim;
2  //Inputs
3  reg [3:0] A;
4  reg [3:0] B;
5  //Outputs
6  wire [7:] mul_4;
7  //Instantiate the Unit Under Test(UUT)
8  test uut(
9  .A(A),
10 .B(B),
11 .mul_4(mul_4)
12 );
13 initial begin
14 //Initialize Inputs
15 A=001;
16 B=001;
17 //wait 100 ns for global reset to finish
18 #100
19 //Add stimulus here
20 #1 B=010;
21 #1 B=011;
22 #1 B=100;
23 #1 B=101;
24 #1 B=110;
25 #1 B=111;
```

```
26 end
27 endmodule
```

Vita Auctoris

NAME:	Meitong Pan
PLACE OF BIRTH:	Shenyang, Liaoning, China
YEAR OF BIRTH:	1995
EDUCATION:	Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu, China 2013-2017, Bachelor of Science Optoelectronic Engineering
	University of Windsor, Windsor, Ontario, Canada 2017-2019, Master of Applied Science Electrical and Computer Engineering