EFFICIENT IMPLEMENTATION OF LATTICE-BASED CRYPTOGRAPHIC SCHEMES FOR INTERNET OF THINGS APPLICATIONS

WONG ZHENG YAN

MASTER OF SCIENCE (COMPUTER SCIENCE)

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

DECEMBER 2021

EFFICIENT IMPLEMENTATION OF LATTICE-BASED CRYPTOGRAPHIC SCHEMES FOR INTERNET OF THINGS APPLICATIONS

By

WONG ZHENG YAN

A dissertation submitted to

Faculty of Information and Communication Technology,

Universiti Tunku Abdul Rahman,

in partial fulfillment of the requirements for the degree of Master of Science (Computer Science)

December 2021

ABSTRACT

EFFICIENT IMPLEMENTATION OF LATTICE-BASED CRYPTOGRAPHIC SCHEMES FOR INTERNET OF THINGS APPLICATIONS

Wong Zheng Yan

Lattice-based cryptography (LBC) is one of the most widely studied post-quantum cryptography (PQC) candidates to date. Polynomial multiplication (PM) and generation of error samples are two main bottlenecks in LBC. PM can be implemented through schoolbook polynomial multiplication algorithm (SPMA) and Number Theoretic Transform (NTT).

The SPMA has always been the simplest form of performing PM, and often can be implemented through very light weight designs, but it suffers from low throughputs. NTT on the other hand, requires vast hardware utilization to cope with the high parallelism of the multiplication process, although capable of completing the PM process in a much shorter timeframe. Moreover, NTT requires special ring structure to operate, which may not be found in all LBC schemes.

Karatsuba algorithm, being another candidate between these two extremes, are not widely studied for LBC scheme implementation in FPGA. Karatsuba algorithm can be used to speed up the PM process, while keeping

the hardware utilization moderately lightweight. This fills in the gap between SPMA and NTT, creating a robust and packed polynomial multiplier, especially for IoT applications that requires higher security with hardware constraints.

The main focus of this work is to develop a high-speed hardware architecture to improve the performance of PM in LBC schemes such as Ring Learning-with Error (R-LWE) and Learning-with-Errors (LWR) (Saber). This research work implemented a 1-layer Karatsuba architecture to improve the throughput of PM for R-LWE, and a 4-layer Karatsuba architecture to improve the throughput of PM for SABER.

By breaking the polynomials into smaller sub-polynomials for multiplication, along with efficient data scheduling specifically for the Karatsuba algorithm, the throughput of PM is improved drastically. Furthermore, multiplicands are also stacked up to double the throughput in both R-LWE and SABER implementations. Last but not least, the negacyclic operations are integrated into the post-processing of Karatsuba, saving additional memory elements for storing the intermediate results, and reducing the time consume for computing the PM results.

Experimental results show a speed up of 2.09× in throughput along with a 6.52% improvement in throughput-per-slice for the R-LWE polynomial multiplier. For the Saber polynomial multiplier, experimental results show a speed up of 2.17× in throughput along with a 73.55% improvement in throughput-per-slice.

IoT applications require the sensor nodes to transmit sensor data frequently to the nearby gateway device. This implies that the implementation of public key scheme used in protecting such communication must achieve sufficient throughput in order to ensure a timely response. The proposed Karatsuba-based architecture allows high throughput performance, at the same time do not consume extremely large hardware area; this shows great potential to be used in IoT applications

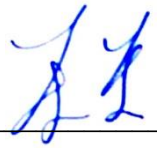
ACKNOWLEDGEMENT

First, I would like to thank the university for funding this project. This project was funded under the MoHE fundamental research grant scheme (FRGS) with the grant number FRGS/1/2018/STG06/UTAR/03/1 and UTAR top up scheme. Next, I would like to thank my supervisors Dr. Denis Wong Chee Keong, Dr. Lee Wai Kong and Mr. Mok Kai Ming for their continuous support and encouragement. I would also like to show appreciation to my postgraduate friends and seniors that aid me along the way. Furthermore, I would like to thank my friends and family for encouraging and standing by my side along the way. Finally, I would like to thank Ms. CY who was always there for me in times of despair and struggle, providing me abundance of support and inspirations.

APPROVAL SHEET

This dissertation entitled "**EFFICIENT IMPLEMENTATION OF LATTICE-BASED CRYPTOGRAPHIC SCHEMES FOR INTERNET OF THINGS APPLICATIONS**" was prepared by WONG ZHENG YAN and submitted as partial fulfillment of the requirements for the degree of Master of Science (Computer Science) at Universiti Tunku Abdul Rahman.

Approved by:



(Dr. Denis Wong Chee Keong) Date:…25/10/21…….
Supervisor
Department of Mathematical and Actuarial Sciences
Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

(Dr. Lee Wai Kong) Date:…… 25/10/21…….
Co-supervisor
Department of Computer Engineering
Gachon University

(Mr. Mok Kai Ming) Date:…… 25/10/21…….
Co-supervisor
Department of Computer and Communication Technology
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: ____25/10/21_____

SUBMISSION OF DISSERTATION

It is hereby certified that **WONG ZHENG YAN** (ID No: **20ACM00703**) has completed this dissertation entitled "**EFFICIENT IMPLEMENTATION OF LATTICE-BASED CRYPTOGRAPHIC SCHEMES FOR INTERNET OF THINGS APPLICATIONS** " under the supervision of **DR. DENIS WONG CHEE KEONG** (Supervisor) from the Department of Mathematical and Actuarial Sciences, Lee Kong Chian Faculty of Engineering Sciences , and **DR. LEE WAI KONG** (Co-Supervisor) from the Department of Computer Engineering, Gachon University, and **MR. MOK KAI MING** (Co-Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my dissertation in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

(**WONG ZHENG YAN**)

DECLARATION

I hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

(WONG ZHENG YAN)

Date _____25/10/21_____

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

FFT	Fast Fourier Transform
LBC	Lattice-based Cryptography
NIST	National Institute of Science and Technology
NTT	Number Theoretic Transform
PM	Polynomial Multiplication
PQC	Post-quantum Cryptography
SPMA	Schoolbook Polynomial Multiplication Algorithm

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

Public key cryptography (PKC), also known as asymmetric key cryptography, uses a different key to encrypt and decrypt data. The PKC scheme basically uses two mathematically related keys namely a public key and a private key. The function of the public key is to encrypt the plaintext into ciphertext, whereas the private key is used to decrypt the cipher text back to its original plaintext. A few well-regarded PKC such as the RSA encryption algorithm, Diffie-Hellman key exchange protocol and Paillier cryptosystem are used for various purposes, ranging from digital signatures, encryption, authentication, non-repudiation, and data integrity.

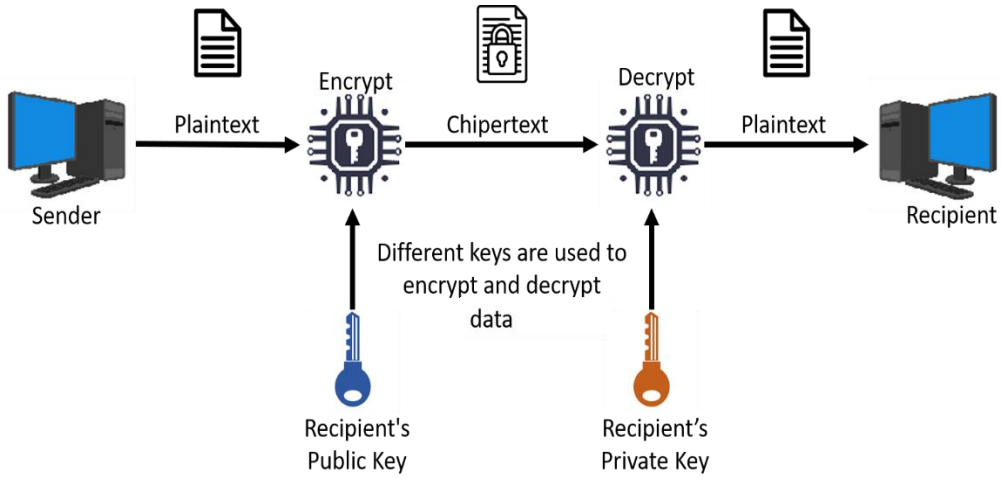


Figure 1.1 Public Key Encryption Example

For years, the public-key encryption (PKE) and public-key signature scheme cryptosystem has kept data safe, but with the emerging numbers of quantum computers, this poses a big threat to the existing cryptosystem that runs on PKE. Unlike conventional computers which use bits that can only be in two states (either *1s* or *0s*), quantum computers on the other hand, use qubits, typically subatomic particles, *i.e.*, photons and electrons. The two most significant quantum properties of qubits namely superposition and entanglement enable a quantum machine to leap exponentially in terms of computing power. An encryption algorithm that takes years to be compromised by a classical computer, may only need minutes in quantum computer, rendering many existing PKE obsolete.

Moreover, introduction of Shor's algorithm in (Shor 1997) makes the existing PKE even more vulnerable to quantum computers. The holy grail of existing cryptosystems is based on prime factorization and finding discrete logarithms, which are proven extremely difficult to be solved by classical computer. However, by cleverly utilizing both classical computer and quantum computers, computation problems like performing reversible computation, prime factorization and Fourier transform on a quantum computer can be solved. In other words, cracking prime factorization on quantum computers is nothing but a piece of cake.

1.1.1 Post-quantum Cryptography

Post-quantum cryptography (PQC) refers to the development of new kinds of cryptographic algorithms secure and resistant against attacks from quantum computers, implemented using today's conventional computers. A report by the US National Institute of Standards and Technology (NIST) states that by the year 2030, quantum machines would render the widely used RSA algorithm insecure. Hence by that same year, NIST has initiated a process to develop and standardize one or more quantum-resistant PQC algorithms (NIST 2016).

Since traditional PKE mostly rely on mathematical problems such as integer factorization and discrete logarithm, quantum machines and algorithms can efficiently solve these types of traditional number-theoretic-based cryptosystems. Although no large-scale quantum machines are yet to exist for at least a decade or more, this should be considered as a warning and steps should be taken in order to prevent quantum computers from completely rendering cryptosystem useless. Researches on PQC mainly focus on six approaches, where all six are significantly varied, namely:

i. Lattice-based Cryptography
ii. Multivariate Cryptography
iii. Hash-based Cryptography
iv. Code-based Cryptography
v. Super Singular Elliptic Curve Isogeny Cryptography
vi. Symmetric Key Quantum Resistance

This project focuses on multiple schemes in lattice-based cryptography.

1.1.2 Lattice-based Cryptography

Lattice-based cryptography (LBC) refers to the cryptography that relates their security to hard mathematics problems around lattices, and is considered one of the most widely studied post-quantum cryptography (PQC) candidates to date. In the round 3 of National Institute of Standards and Technology (NIST) post-quantum cryptography (PQC) standardization process (NIST, 2021), five out of seven selected finalists are LBC. Many tests were done on LBC and some lattice-based structures seem to resist attacks from both classical and quantum computers very well. Hence, LBC is regarded to be a prime candidate for quantum-secure PKC (Regev 2006).

There are several reasons behind choosing LBC to be the main focus of this proposal. Firstly, one of the unique properties in LBC is that it can be used to construct very reliable cryptographic primitives. Furthermore, the computations involved in implementing LBC are not sophisticated; it usually only requires matrix/polynomial multiplication and small modular arithmetic. Lastly, LBC is chosen given its efficiency and practical advantage while performing encryption on a low-cost device.

Polynomial multiplication (PM) and generation of error samples are two main bottlenecks in LBC; the focus of this project is to propose a high-speed hardware architecture to improve the performance of PM. PM can be implemented through schoolbook polynomial multiplication algorithm (SPMA) and Number Theoretic Transform (NTT). SPMA is simple and straightforward to implement in hardware, while NTT requires complicated pre-computation and array re-ordering to achieve high performance.

The SPMA has always been the simplest form of performing PM, and often can be implemented through very light weight designs. For older schemes such as the Ring-Learning with Errors (R-LWE), SPMA works well in performing the PM. The multiplicator is often small in size, but suffer from very slow speeds, in other words, extremely low to low throughputs. However, given its simplicity, the R-LWE scheme which dictates the polynomial multiplication be done in a negacyclic fashion, can be integrated and weaved into the multiplication process. NTT on the other hand, requires vast hardware utilization to cope with the high parallelity of the multiplication process. Being on the other end of the spectrum, NTT is capable of completing the PM process in a much shorter timeframe. That being said, NTT highly depends on large framework and hardware utilization, hence it is often not suitable for IoT applications that are targeted for a more lightweight and low-powered implementation. Moreover, NTT requires special ring structure to operate, which may not be found in all LBC schemes.

In newer schemes such as SABER, which is one of the NIST round three finalist amongst the other PQC candidates, many papers have already explored the implementation utilizing SPMA, since SABER cannot be implemented using NTT. Despite that, both the R-LWE and SABER schemes operate on the same polynomial ring equation (Pöppelmann, 2013), which requires a negacyclic operation for the PM process.

Karatsuba algorithm, being another candidate between these two extremes, are not widely studied for R-LWE and SABER implementation in FPGA. Since the PM process has always been the bottleneck of the cryptographic process, Karatsuba algorithm can be used as a catalyst to speed up the PM process, while keeping the hardware utilization moderately lightweight. This fills in the gap between SPMA and NTT, creating a robust and packed polynomial multiplier, especially for IoT applications that requires higher security with hardware constraints.

In this project, we proposed a 1-Layer Karatsuba architecture to improve the throughput of PM for R-LWE, and a 4-Layer Karatsuba architecture to improve the throughput of PM for SABER. Due to the nature of the Karatsuba algorithm, which involves splitting and combining the polynomial during the PM process, it forbids the negacyclic operations to be done along with the polynomial multiplication such as in SPMA. Hence, one of the main challenges is implementing Karatsuba algorithm with negacyclic efficiently.

1.1.3 Cryptography in IoT

The Internet of Things (IoT) has developed drastically over the past decade. IoT devices such as sensor nodes are deployed ubiquitously to transmit, collect and exchange information, which poses many vulnerabilities against malicious attacks from adversaries.

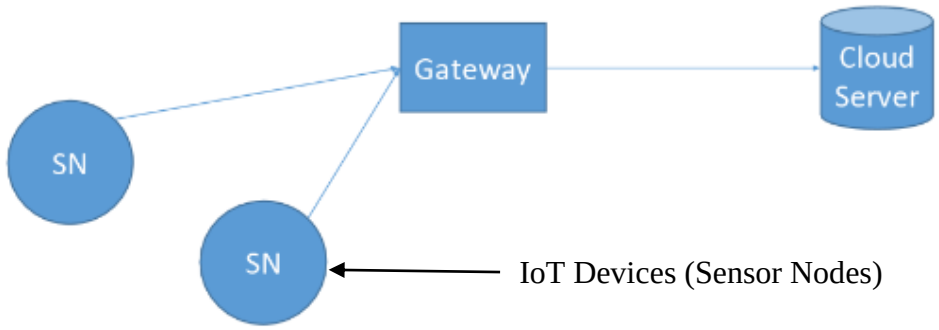


Figure 1.2 IoT framework

Figure 1.2 displays a simple IoT framework. The sensor node(s) (SN) read the IoT data and transmit them to the gateway. To encrypt the IoT data using common block cipher (e.g., AES), the sensor nodes need to randomly generate symmetric keys and transmit it to the gateway and cloud server for decryption purpose. The Key Encapsulation Mechanism (KEM) is used to perform such key exchange operations.

However, in real life implementations, the number of sensor nodes connected to the gateway are massive. When side channel attacks are performed on the sensor nodes, the symmetric key used for encrypting IoT data may be compromised. Hence, the symmetric key needs to be generated for each communication session and transmitted frequently to the gateway/cloud server. Due to this reason, the KEM must also be implemented efficiently, so that the security of the IoT framework can be protected without compromising the response time.

Despite that, previous KEM implementations for IoT devices are mostly built upon conventional public key encryption schemes such as RSA. As discussed earlier, conventional PKC is becoming obsolete due to the emergence of Shor's algorithm and more advanced quantum computers. Hence, post-quantum schemes such as the LBC are used to replace the conventional PKC in KEM. Due to the small key sizes and good performance on various hardware platforms, lattice-based KEM can be computed with high throughput and moderate amount of hardware, which is especially suitable for IoT devices. The goal in this dissertation is to propose more efficient polynomial multiplier architectures to further improve the throughput of selected lattice-based KEM.

1.2 Problem Statements

1) The SPMA polynomial multiplier architectures used in R-LWE schemes suffer from very slow speed performance. This is inherently limited by the SPMA algorithm that comes with a high complexity. Some IoT applications cannot utilize architectures that are too slow.
2) Existing Karatsuba-based polynomial multiplier architectures for LBC schemes are having poor area-time efficiency. They achieved high throughput in expense of large area consumption. The data processing mechanism in handling various Karatsuba levels are also not fully optimized.

1.3 Objectives

1) Design a Karatsuba-based polynomial multiplier architectures to speed up the speed performance of polynomial multiplication in RLWE schemes.
2) Develop a Karatsuba-based polynomial multiplier architectures with minimal additional hardware. The developed architectures should maintain a high throughput in order to cater for IoT applications, at the same time achieve better area-time efficiency compared to existing work.

1.4 Contribution

Contributions of this dissertation are as follows:

1) An efficient polynomial multiplier utilizing 1-Layer Karatsuba for R-LWE scheme. The multiplier is developed with DSP utilization algorithm for Karatsuba and integrated negacyclic operations in Karatsuba post-processing. This design is catered for IoT devices which uses relatively low hardware resources while having a high throughput.
2) A high speed shifter-based polynomial multiplier implementation utilizing 4-Layer Karatsuba for Saber key encapsulation mechanism (KEM). The grid-based parallel data input for Karatsuba pre-processing and post-process result mapping with integrated negacyclic enables an optimal polynomial multiplication process utilizing Karatsuba, eventually improving the overall speed performance. At the same time, by implementing techniques to reuse registers in the pre-processing and direct result mapping in post-processing, the hardware resources consumption is drastically decreased achieving a more area-time balanced design.

1.5 Dissertation Organization

The dissertation commenced with Chapter 1, which serves as a top down introductory for the cryptographic schemes researched, while addressing the vulnerabilities of existing cryptographic schemes. In Chapter 2, study is conducted on the existing implementation of different LBC schemes utilizing various algorithms such as SPMA, NTT and Karatsuba. Chapter 3 provides further discussion and background information on the algorithms utilized to implement polynomial multipliers in the LBC scheme, along with the introduction to negacyclic convolution requirements for certain LBC schemes.

Chapter 4 focuses on the design of polynomial multiplier for the R-LWE scheme, whereas Chapter 5 focuses on the design of polynomial multiplier for the Saber scheme. Chapter 6 provides and discusses the experimental results, where it is divided into two parts. The first part is for the R-LWE scheme, and the second part is for the LWR scheme (Saber). Finally, Chapter 7 concludes the research work, along with suggestions for potential future research ideas and directions.

CHAPTER 2

LITERATURE REVIEW

Post-quantum cryptography is vital for the future of cryptography in the imminent advancement of quantum computers. A substantial amount of quantum cryptography researches has been conducted in recent years in conjunction with NIST's search for PQC's to be standardize. Among the NIST round 3 finalist for the public-key encryption schemes are candidates such as Saber, Kyber, NTRU and Classic McEliece. The R-LWE scheme on the other hand is an older LBC scheme that is widely researched too, due to its efficiency.

Recent implementations for the various schemes aforementioned, be it on FPGA's, ARM or Intel's processors, algorithms such as SPMA and NTT are very popular, often utilized to perform the polynomial multiplication. However, since NTT is not applicable for schemes that has modular primes that are not on the prime field such as the Saber scheme, or when the targeted device has a hardware consumption limitation (commonly in IoT devices), the Karatsuba algorithm comes into play and can be leveraged to improve the performance of such schemes compared to their SPMA counterparts.

In the following sections, several researches utilizing SPMA and NTT have been reviewed to give a better understanding on how the Karatsuba algorithm can fill in the gap between both spectrums of the area-time aspect.

2.1 SPMA

2.1.1 Optimized Schoolbook Polynomial Multiplication for Compact Lattice-Based Cryptography on FPGA (Liu et. el., 2019)

Liu et. al. (2019) implements an improved SPM algorithm to perform polynomial multiplication for the R-LWE scheme. They introduced a novel method for modular reduction and performing bit reduction.

The multiplication of 13-bit integer and 6-bit integer is transform to 13-bit multiplied by 5-bit integer by applying reduced bit-width representation. The most significant bit (MSB) is removed from the 6-bit value since it only indicates the sign. To perform modulo on the 18-bits product, a modular reduction algorithm is used rather than building a hardware to perform division. The 18-bits integer undergoes a Barrett reduction algorithm, which only uses shifting, addition and subtraction, which is much more cost and speed effective.

The second improvement made is by fitting two multiplications within one DSP block on the FPGA. In Xilinx 7 series FPGAs, a single DSP block can support multiplications up to 25×18 bits. Since the reduced size of multiplication required is only 13×5 bits, one DSP block on the FPGA can be fitted with two multiplicands, hence performing two multiplications at the same time.

The hardware cost of implementation with parameters $n = 256$ and $q = 7681$ requires 898 LUTs, 1 DSP block, 3 8K BRAMs, and 815 registers, which is easily scalable due to its small size but very low in terms of efficiency.

2.1.2 Lightweight Hardware implementation of R-LWE Lattice-Based Cryptography (Fan et. al., 2019)

Fan et al. (2019) proposes a lightweight hardware implementation of a R-LWE cryptosystem without the risk of compromising the security. In other words, although only a small amount of resources is used and the performance slightly decreased, the security of the cryptosystem is guaranteed.

To achieve a lightweight R-LWE cryptosystem hardware architecture design, a low cost and fast Gaussian sampler based on cumulative distribution table (CDT) method is used. Besides, the method used for polynomial multiplication in this paper is SPM algorithm. To further enhance the efficiency, the polynomial multiplier is pipelined into multiple stages to cater the latency of the process, hence reducing the clock cycles to achieve higher frequency for the overall cryptosystem.

Compared to (Liu et al., 2019), this paper directly implements multiplication of 13×13 bits without a bit reduction scheme. However, Barrett reduction scheme is also used in this paper for modular reduction to reduce resources usage while increasing efficiency.

The hardware cost of implementation with parameters $n = 256$ and $q = 7681$ requires 1098 LUTs, 1 DSP block, 0 BRAMs, and 407 registers, which is extremely scalable due to its small size but very low in terms of efficiency. The performance results and resources used in the papers and journal are shown in the table below.

2.1.3 An Efficient and Parallel R-LWE Cryptoprocessor (Zhang et. al., 2020)

Zhang et al. (2020) proposes implementation of extra DSPs blocks into the work of (Liu et al., 2019) by tweaking the DSP utilization algorithm to cater for the extra multiplication per cycle. Furthermore, the error term's bit width is also adjusted paired which is crucial in determining the overall throughput of the entire polynomial multiplier.

Moreover, the thrifty reuse of hardware resources including the schoolbook polynomial multiplier and the polynomial adder (PA) results in the reduction of area consumption, which has not been undertaken in earlier reported implementations. They exploit the operational features of polynomial multiplication to implement an efficient and parallel SPM structure design for the most critical operation of polynomial multiplication in R-LWE.

The iteration clock cycle is reduced by factor of 4 and results in 1.8× speedup and 1.4× TPS. Furthermore, the reuse of the most critical parts in R-LWE encryption and decryption hardware enables a 14% reduced area with 1.7× throughput improvement.

The hardware cost of implementation with parameters $n = 256$ and $q = 7681$ requires 699 LUTs, 2 DSP block, and 705 registers. Compared to lightweight implementations, this polynomial multiplier outweighs them with a much better throughput-per-slice (TPS).

2.1.4 High-speed Instruction-set Coprocessor for Lattice-based Key Encapsulation Mechanism: Saber in Hardware (Roy and Basso (2020))

Roy and Basso (2020) proposed a lightweight cryptoprocessor implementing SPMA; this can be regarded as the state-of-the-art SPMA-based polynomial multiplier for Saber. They introduced an optimized coefficient-wise modular multiplier, which is essentially the multiply-and-accumulate (MAC) module. The MAC operations are carried out through simple shift and add operations, consequently requiring no DSP blocks and substantial saving in hardware resources. They also proposed a novel data loading technique, reducing the cycle count of the loading of operands and the reading of polynomial multiplication results. The size of the buffer is greatly reduced by using the least common denominator between the coefficient size and the size of the memory storing the coefficient, requiring almost 20% fewer registers.

The building blocks are integrated after optimization to realize an instruction-set coprocessor hardware architecture that is able to computes all KEM operations such as key generation, encapsulation and decapsulation.

The hardware cost of implementation for both Saber768 and Saber unified with parameters parameters $n = 256$ and $q = 8192$ requires 17429 LUTs and 5083 registers.

2.2 NTT

2.2.1 Open-Source FPGA Implementation of Post-Quantum Cryptographic Hardware Primitives (Agrawal et. al., 2019)

Agrawal et. al. (2019) proposed a FPGA-tailored implementation of the R-LWE cryptographic primitives along with novel algorithmic proposals such as oblivious transfer (OT) and zero-knowledge proof (ZKP). By using an efficient implementation of a n-point NTT algorithm, a high-speed polynomial multiplier is developed.

The prime modular q is fixed at 12,289 in this paper, since a realistic implementation of the R-LWE based PKE cryptosystem requires q to be larger than 10,000 to ensure that the implemented algorithm has at least 112 bits in terms of security level, as stated by the security standards specify by NIST. The length of the polynomial in this paper however, is parameterizable, one of the few contributions of this paper, with n (order of polynomial) ranging from 128 ~ 1024.

By optimizing and parameterizing the butterfly NTT method proposed by Chen et al. (2015), an efficient hardware design is created ready to implement. The hardware cost of implementation with parameters n=256 and $q = 12{,}289$ requires 9152 LUTs, 26 DSP blocks, 3.5 BRAMs and 396 registers, which is not the most practical implementation method for IoT devices.

2.2.2 Compact Ring-LWE Crypto-processor (Roy et. al., 2014)

Roy et el. (2014) proposed a compact but efficient crypto-processor for a R-LWE based encryption scheme. The method used for polynomial multiplication is NTT, along with three optimizations for the algorithm.

The NTT algorithm is improved by reducing the twiddle factor computation cost, deriving an efficient memory access scheme, and avoiding pre-computation during forward NTT, hence increasing the utilization of all

memory blocks and arithmetic components on the FPGA. They also devised an optimal pipeline strategy to increase the overall operating speed and frequency of the processor, which is based on two observations according to the algorithm implemented. By splitting the critical path with highest delay or latency into different stages, hence creating a balanced-delay stage pipeline processor.

The hardware cost of implementation with parameters n = 256 and q = 7681 requires 1349 LUTs, 1 DSP block, 3 18K BRAMs and 860 registers.

2.2.3 A Resource-Efficient and Side-Channel Secure Hardware Implementation of Ring-LWE Cryptographic Processor (Liu et. al., 2018)

Liu et al. (2018) proposed a practical hardware implementation of R-LWE cryptosystem design on resources constrained devices, thus making it more feasible and versatile for embedded or IoT devices.

In this paper, fast number theoretic transform (FNTT) is used for performing polynomial multiplication. “FNTT is fundamentally an FFT defined in a finite field without inaccurate floating point or complex arithmetic”, hence performing polynomial multiplication much more efficiently.

Besides, to enhance the performance of the whole system, this paper also proposes a universal module for modular arithmetic called modular

processing element (MPE). This highly reduces the need for hardware resources while performing both encryption and decryption, thus achieving a resource-efficient cryptosystem design.

The hardware cost of implementation with parameters n=256 and $q = 7681$ requires 1307 LUTs, 0 DSP block, 1 18K BRAM, 3 8K BRAMs and 889 registers. This design has a high efficiency but failed to utilize efficiently the resource and did not achieve equilibrium in area-time performance.

2.2.4 High-Throughput Ring-LWE Crypto-processors (Renteria-Mejia and Velasco-Medina, 2017)

Renteria-Mejia and Velasco-Medina (2017) proposes a R-LWE crypto-processor using NTT method with a high throughput. The paper mainly focuses on throughput performance rather than achieving balance in area-time performance.

This paper also implements Barrett reduction scheme in the modular reduction algorithm in the operation of addition and multiplication of polynomials over the ring. Inverse transform method was used while designing the gaussian sampler that is responsible for generating a narrow gaussian distribution of errors.

The cryptosystem architecture design introduced in this paper targets high security and speed, but suffers from high cost and high resources usage when compared with design from other papers.

The hardware cost of implementation with parameters n=256 and $q = 12289$ requires 25614 LUTs, 20 DSP block, 223 BRAMs and 27129 registers. While dealing with embedded or IoT devices having many hardware and cost constrain, this design is somewhat impractical and unrealizable.

2.2.5 Area-optimized Lattice-based Cryptographic Processor for Constrained Devices (Liu et. al., 2017)

Liu et al. (2017) proposed an area optimized R-LWE crypto-processor suitable for devices with resource constraints. Furthermore, the crypto-processor is also designed to be resistance against side channel attacks.

A constant time gaussian sampler is used due to its resistance against side channel timing attacks since it has a inconsistent runtime if generating samples. To further increase the security of the system, an additional finite state machine (FSM) shuffler is designed and added to the crypto-processor. The purpose of the shuffler FSM is to further randomize the generated samples by fetching and storing the samples in a completely random fashion. Iterative

NTT is used as the method to perform polynomial multiplication. Without using any DSP blocks, a compact and lightweight crypto-processor is designed.

The hardware cost of implementation with parameters n=256 and $q = 7681$ requires 1787 LUTs, 0 DSP block, 3 BRAMs, and 790 registers. This design is scalable and feasible for resource constrained device although a bit low in terms of efficiency compared to crypto-processor with high throughput but uses a lot of hardware or resources.

CHAPTER 3

BACKGROUND

3.1 SPMA

Schoolbook Polynomial Multiplication Algorithm (SPMA) is proven to be the most simple and straightforward approach to perform polynomial multiplication.

Schoolbook polynomial multiplication algorithm

Input:
$a(x) \in \mathbb{Z}_q[x]/(x^n + 1)$, Polynomial a, degree $= n$;
$b(x)$, Polynomial b, degree $= n$;
Output:
$acc(x) \in \mathbb{Z}_q[x]/(x^n + 1)$ Product $a(x) \cdot b(x)$, degree $= n$;

1: $acc(x) \leftarrow 0$;
2: **for** $i = 0$; $i < n$; $i = i + 1$ **do**
3: **for** $j = 0$; $j < n$; $j = j + 1$ **do**
4: $acc[j] \leftarrow acc[j] + (b[j] \cdot a[i])$ mod $\mathbb{Z}_q$
5: **end for**
6: $b \leftarrow b \cdot x$ mod R_q
7: **end for**
8: **return** acc

Figure 3.1 Schoolbook polynomial multiplication algorithm (SPMA)

Referring to Figure 3.1, the Schoolbook Polynomial Multiplication Algorithm (SPMA) has complexity $O(n^2)$ with two nested loops where multiplication is repeated $n \times n$ times, where where n is the degree of polynomial. The

multiplication results are not directly used as an output, they are stored in an accumulator (line 4) and are used to compute negacyclic convolution (ref. Chapter 3.4). Finally, the secret polynomial b is rotated by multiplying it by x in R_q (line 6), essentially shifting the polynomial by a single position to the left or right. The direction of rotation for the secret polynomial depends on the design methodology as long as all iterations are visited. When rotating left, the polynomial's MSB becomes the LSB and vice versa. Polynomial multiplication is completed when polynomial b is rotated back into its original position.

3.2 NTT

Number theoretic transform (NTT) is basically a generalization of a Fast Fourier Transform (FFT) obtained by replacing $e^{-2\pi ik/N}$ with an n^{th} primitive root of unity. When the number of elements in the transform is a composite value, a fast NTT may be constructed in the same manner as a FFT is constructed from the DFT.

However, unlike the DFT, the NTT can be used for fast convolutions and correlations like the various FFT algorithms, since it has analogous theorems, such as the convolution theorem. Despite that, this dissertation does not delve deep into the various development of NTT since it is not a focus of this research.

3.3 Karatsuba

Karatsuba algorithm for polynomial multiplication stems from (Karatsuba 1962), which involves splitting the original integer or polynomial into smaller parts, replacing some multiplication process by addition and subtraction, in turn reducing the complexity of the multiplication process. Below is the pseudocode for a single iteration of Karatsuba or a 1-Layer Karatsuba.

Karatsuba Algorithm for PM

Input:

Polynomial a, size $= n$; (multiplier)
Polynomial b, size $= n$; (multiplicand)
Degree of polynomial, n

Output:

Polynomial ab_full, size $= 2n-1$; (multiplication result)

1: $new\_n \leftarrow \frac{n}{2}$;
2: $a\_low \leftarrow a[\frac{n}{2} - 1 : 0]$;
3: $a\_high \leftarrow a[n-1 : \frac{n}{2}]$;
4: $a\_mid \leftarrow a[n-1 : \frac{n}{2}] + a[\frac{n}{2} - 1 : 0]$;
5: $b\_low \leftarrow b[\frac{n}{2} - 1 : 0]$;
6: $b\_high \leftarrow b[n-1 : \frac{n}{2}]$;
7: $b\_mid \leftarrow b[n-1 : \frac{n}{2}] + b[\frac{n}{2} - 1 : 0]$;
8: $ab\_low \leftarrow a\_low \times b\_low$;
9: $ab\_high \leftarrow a\_high \times b\_high$;
10: $ab\_mid \leftarrow a\_mid \times b\_mid$;
11: $ab\_newmid \leftarrow ab\_mid - ab\_high - ab\_low$;
12: $ab\_full[2n-2 : n] \leftarrow ab\_high$;
13: $ab\_full[n-2 : 0] \leftarrow ab\_low$;
14: $ab\_full[n + \frac{n}{2} - 2 : \frac{n}{2}] \leftarrow ab\_newmid + ab\_full[n + \frac{n}{2} - 2 : \frac{n}{2}]$;
15: **return** ab_full

Figure 3.2 1-Layer Karatsuba algorithm for polynomial multiplication (PM)

The Karatsuba algorithm uses a divide and conquer approach, in which the polynomials are divided into smaller sub-polynomials to perform multiplication efficiently. These sub-polynomials are combined to form the final result. The algorithm can be divided into three main parts, pre-processing, polynomial multiplication and post-processing.

Referring to Figure 3.2, every time the algorithm is called, input polynomial *a* is split into two half-sized sub-polynomials *a_high* and *a_low* (lines 2-3). They are then added up to produce the third half-sized sub-polynomial *a_mid* (line 4). The input polynomial *b* undergoes the same process of splitting (lines 5-7), resulting in another three half sized sub-polynomials, *b_high*, *b_low* and *b_mid*.

After completing pre-processing, the sub-polynomials are then multiplied in pairs (lines 8-10). Finally, Karatsuba post-processing (combination of sub-polynomial multiplication results) can be done. The sub-polynomials *ab_low* and *ab_high* are subtracted from *ab_mid*, producing a *ab_newmid* sub-polynomial (line 11). Lastly, the sub-polynomials are stacked and added accordingly with respect to the index of n (lines 12-14).

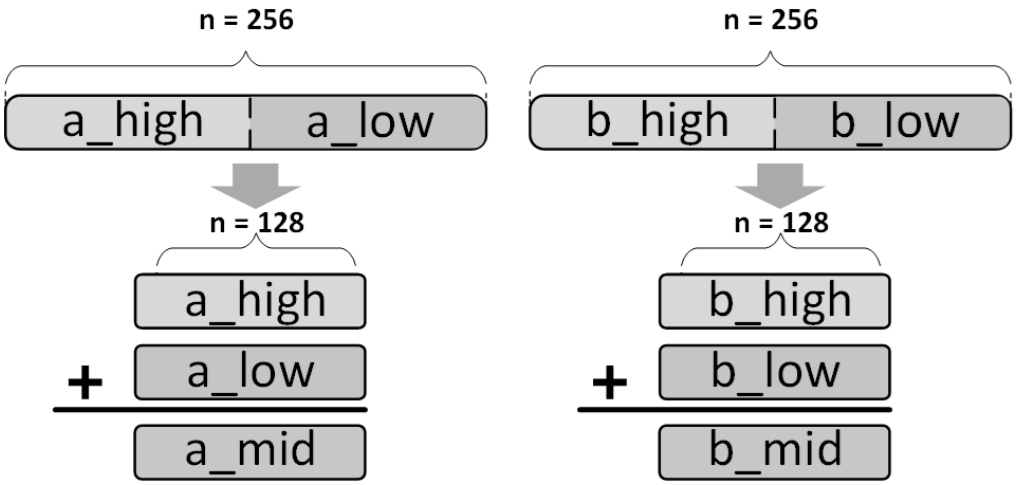


Figure 3.3 Karatsuba pre-processing (splitting)

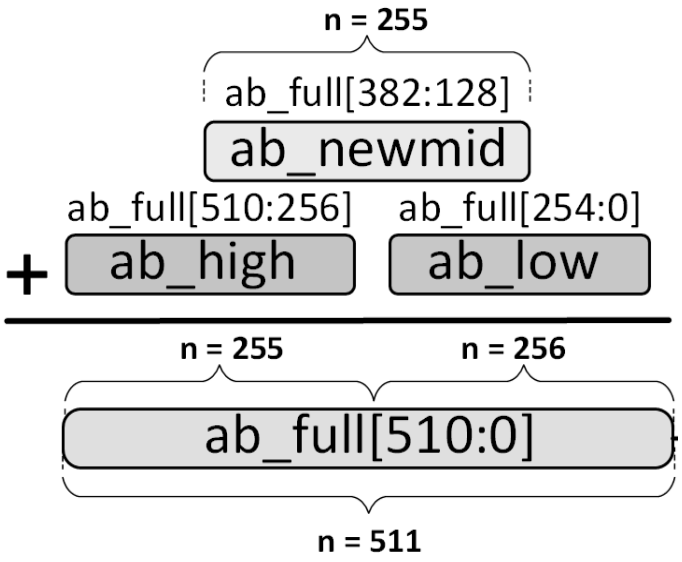


Figure 3.4 Karatsuba post-processing (combination)

3.4 R-LWE

Ring-learning with errors (R-LWE) is a widely investigated algorithm that is based on a hard lattice problem. The most critical operation in the R-LWE schemes is polynomial multiplication on the ring.

The hardware resource requirements for R-LWE-based public key encryption scheme needs to cater for both encryption and decryption operations, mainly due to the polynomial multiplication process which requires a high hardware consumption. The encryption process requires polynomial multiplication and addition while encoding the plaintext to polynomial. After performing polynomial multiplication, the decryption process also involves polynomial multiplication and addition, but this time, decoding the polynomial back into plaintext.

The R-LWE scheme basically incurs a random error (noise) value into the polynomial to increase the complexity of the lattice problem, hence the name learning with errors. More details about the R-LWE scheme can be found in (Pöppelmann and Güneysu, 2013).

3.5 LWR

The Learning-with-Rounding (LWR) scheme was first proposed in (Banerjee 2012), where a de-randomization technique was proposed to enhance the conventional Learning-with-Errors (LWE) scheme. By giving a more direct construction of the pseudorandom function (PRF) families, which is based on learning and hard lattice problems, the LWR scheme achieved a higher efficiency as it is distinctly parallelizable from the implementation aspect.

The LWR scheme is much simpler compared to the LWE scheme, in which the inner product is just deterministically rounded to the relatively nearby element, giving it an arbitrary value of error in a given range. This omits the need to perform further operations such as the addition of the small random error terms, since the scheme is no longer dependant on random and independent errors. Moreover, the LWR scheme operates on polynomials with coefficients in the form of Z_{2n} (power of two), and it introduces random errors through rounding operations. As a result, the modular operations can be carried out through simple truncated shifting operations.

3.6 Saber

Saber is an LWR-based scheme submitted to NIST standardization (NIST, 2021), first introduced in (D'Anvers et al., 2018). Like all other LBC schemes, polynomial multiplication is the most computationally intensive operation in Saber. Although Saber and LWE schemes both operate on the similar polynomial ring, $Z_q[x]/(x^n + 1)$ (Pöppelmann and Güneysu , 2013), the Number Theoretic Transform (NTT) algorithm cannot be utilized to speed up the computation of polynomial convolution in Saber as the polynomial coefficients (Z_2^{10} and Z_2^{13}) are not on a prime field as they are in LWE schemes. Furthermore, although Saber operates on two different moduli (q and p), both of them are in the powers of 2 form, hence omitting the need for additional modular operations. The degree of polynomial (n) in Saber is always 256 regardless of the security level. Currently, the Saber suite supports three security levels: LightSaber, Saber768 and FireSaber. Since Saber is a module-LWR scheme, the polynomials are organized in an $l \times n$ dimension, where l = 2, 3, 4 for LightSaber, Saber768 and FireSaber respectively. In this paper, we focus on improving the efficiency of polynomial multiplier hardware architecture, which can be used to perform matrix-vector multiplication and inner product (D'Anvers et al., 2018) in Saber.

3.7 Negacyclic Convolution

In this dissertation, research is done on two different schemes namely R-LWE and Saber. However, both schemes operates on the same polynomial ring equation $Z_q[x]/(x^n + 1)$ (Pöppelmann and Güneysu , 2013). In other words, both schemes are required to perform the polynomial multiplication in a negacyclic convolution manner.

To implement this efficiently, the negacyclic operations can be integrated into the SPMA, wherein the multiplication is performed with respect to the coefficient position. Hence, after computing the polynomial multiplication, the results are accumulated in the final registers according to the position of the polynomial coefficient, Figure 3.1 (line 4). The multiplication result of each coefficient is then stored in the respective accumulator, and the next iteration of multiplication starts after polynomial b is rotated, Figure 3.1 (line 6). Hence polynomial multiplier that utilizes SPMA with negacyclic is also called a coefficient-wise polynomial multiplier. The details can also be found in (Roy and Basso, 2020).

Due to the splitting (pre-processing) and combination (post-processing) of sub-polynomials in Karatsuba algorithm, negacyclic convolution can only be applied after completing the multiplication process, as shown in Figure 3.5. The degree of product of two polynomials, each with the degree n is 2n−1.

The product polynomial is split in the higher half polynomial with the degree of n−1 and the lower half polynomial with the degree of n. Lastly, a 1-bit 0 would be concatenated to the n^{th} bit of the higher half polynomial, and then subtracted from the lower half polynomial, giving the final result of polynomial multiplication computed through negacyclic convolution.

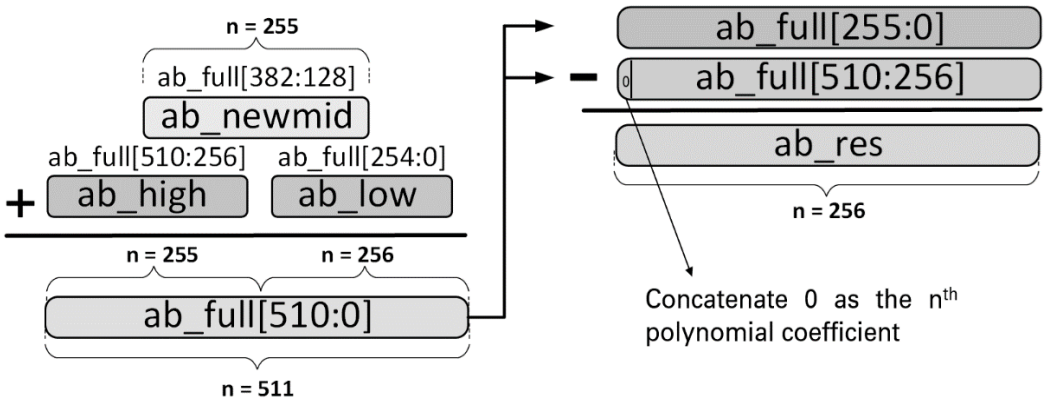


Figure 3.5 Negacyclic operation in Karatsuba post-processing

CHAPTER 4

POLYNOMIAL CONVOLUTION HARDWARE DESIGN FOR R-LWE SCHEME

A 1-Layer Karatsuba is utilized in developing the polynomial convolution hardware for R-LWE. to reduce the complexity of polynomial multiplication, wherein the core multiplication on smaller polynomial still relies on SPMA. We adopted the similar techniques from prior work to perform Barrett reduction for modular prime (Liu et al., 2019) and high-speed multiplication using DSP slice (Zhang et al., 2020). A novel technique is proposed to reduce the memory consumption in negacyclic convolution, which greatly improved the area efficiency.

Furthermore, a technique to optimize negacyclic operations in Karatsuba algorithm is proposed to reduce the memory consumption, which greatly improved the area efficiency. The proposed algorithm combines the Karatsuba combination and negacyclic operation without extra memory and latency.

4.1 Optimized Negacyclic Operations in Karatsuba Post-processing

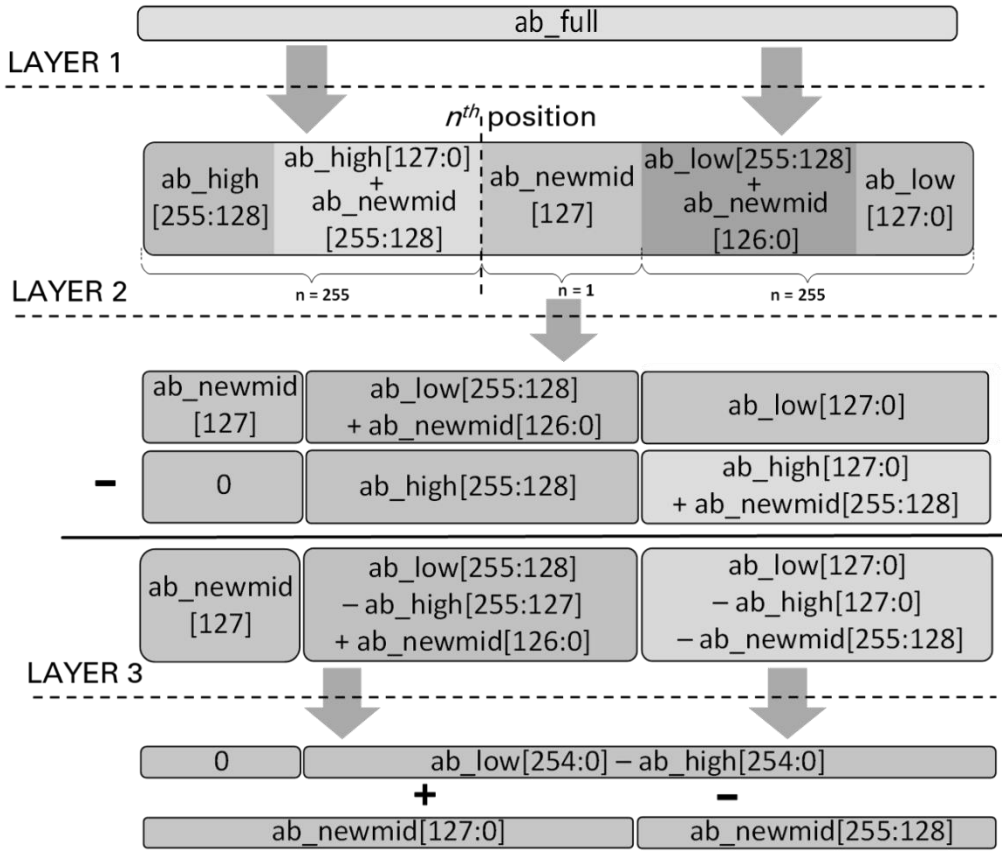


Figure 4.1 Conventional negacyclic operation for Karatsuba

To reduce time required for Karatsuba post-processing and negacyclic operations, both are combined and integrated. Referring to Figure 4.1, *ab_full* is segregated according to its sub-polynomials *ab_high*, *ab_low* and *ab_newmid* (see Layer 1) for clear view of its internal components. Layer 2 depicts the post-multiplication negacyclic operation, wherein the higher half

polynomial is subtracted from the lower half polynomial. Refer to Layer 3, by rearranging the variables after the post-multiplication negacyclic operation, we can see that the subtraction result can be simplified as such: *ab_high* is subtracted from *ab_low*, and *ab_newmid* is added to the first half of subtraction result. In addition, *ab_newmid* is also subtracted from the lower half of the subtraction result. This is the conventional way to perform negacyclic operation, which is not efficient due to the extra memory and cycles required.

Hence, a novel algorithm is proposed to combine the Karatsuba combination and negacyclic operation without extra memory and latency, shown in Figure 4.2.

Novel post-multiplication negacyclic for Karatsuba

Input:

Sub-polynomial ab_low, $n = 255$;
Sub-polynomial ab_high, $n = 255$;
Sub-polynomial ab_mid, $n = 255$;
Result polynomial (ab_res) size, $n = 256$;

Output:

Resultant polynomial ab_res, $n = 256$;

1: **for** $i = 0 : n - 1$ **do**
2: $tmp\_l1 \leftarrow ab\_low[i]$;
3: $tmp\_h1 \leftarrow ab\_high[i]$;
4: $tmp\_l2 \leftarrow ab\_low[i + \frac{n}{2}]$;
5: $tmp\_h2 \leftarrow ab\_high[i + \frac{n}{2}]$;
6: $tmp\_m \leftarrow ab\_mid[i]$;
7: $l\_sub\_h \leftarrow tmp\_l1 - tmp\_h1$;
8: $m\_sub\_hl \leftarrow tmp\_m - tmp\_h2 - tmp\_l2$;
9: **if** $i < \frac{n}{2}$ **then**
10: $ab\_res[i] \leftarrow l\_sub\_h - m\_sub\_hl$;
11: **else**
12: $ab\_res[i] \leftarrow l\_sub\_h + m\_sub\_hl$;
13: **end if**
14: **end for**
15: **return** ab_res

Figure 4.2 Optimized post-process negacyclic operation for Karatsuba

The final result can be computed directly using the resultant sub-polynomials (*ab_high*, *ab_low* and *ab_mid*) of the multiplication process, without the need to recombine them into ab full and then only perform negacyclic operation. To breakdown the algorithm in Figure 4.2, we look at the computation of the final result *ab_res*'s LSB. Referring to Layer 3 in Figure 4.1, the process requires the value of *ab_low*[0] – *ab_high*[0] – *ab_newmid*[128], whereas *ab_newmid*[128] = *ab_mid*[128] – *ab_low*[128] –

ab_high[128]. Hence, (line 2- 6) depicts the input of all required values respectively, with i = 0 and n = 256. Line 7 is the process of computing *ab_low – ab_high*, and line 8 is the process of computing *ab_newmid*. Throughout (line 9-13), if iteration index i is less than n/2 = 128, which denotes the lower half of the polynomial, *ab_newmid* is subtracted. On the other hand, *ab_newmid* is added if index is greater than n/2, denoting the higher half of the polynomial.

Compared to the conventional method which takes a total of 768 cycles, the proposed technique only takes 260 cycles. Thus, gaining a 66.15% decrease in clock cycles due to the integration of both Karatsuba combination and negacyclic operations. The memory consumption is also reduced by 100% because we no longer need to compute the final polynomial of size n = 511, hence the intermediate polynomial *ab_full* can be removed.

4.2 Hardware Design

The SPMA-Karatsuba (SK) R-LWE Polynomial Multiplier Core is segregated into three sub-modules and separated into five different stages. The functionality of sub-modules and stages are stated in Table 4.1. The module k_split_prep executes first level of Karatsuba polynomial splitting and the input preparation stage, denoted by karatsuba 1_1 and prep input respectively. Next, sk module performs spma stage. Lastly, the process ends with the first level Karatsuba polynomial combination (karatsuba 1_2) and post-multiplication negacyclic process (negacyclic), which are performed by the module k_combi_nega. The Karatsuba splitting process in karatsuba 1_1 stage will result in six sub-polynomials *a_low*, *a_high*, *a_mid*, *b_low*, *b_high* and *b_mid*. A prep_input stage is required before the spma stage to increase the parallelism of the multiplication process. This stage duplicates *b_low* to an additional BRAM to avoid memory read conflict. In all, six BRAMs are created to store *a_mid*, *b_mid*, *b_low*, *ab_high*, *ab_low* and *ab_mid*.

Module Name	Stage	Functionality
k_split_prep	karatsuba 1_1	Perform first level splitting of the polynomial into sub-polynomials
	prep_input	Preparation Stage
sk	spma	Perform Schoolbook PM
k_combi_nega	karatsuba 1_2	Perform first level combination of the polynomial into sub-polynomials
	negacyclic	Perform post-multiplication negacyclic operations

Table 4.1 Functionality of each sub modules

The architecture of sk sub-module is illustrated in Figure 4.1. It performs three sets of multiplications per cycle; in each set, one multiplier is paired with two multiplicands (i.e., a_low, b_lowE, b_lowO), whereby two multiplications (i.e., a_low × b_lowE and a_low × b_lowO) are performed through spma. Refer to Figure 4.2, all even indexed multiplicands of b_low will pass through b_lowE, and odd indexed multiplicands through b_lowO. The same input method is applied for b_high and b_mid. The sk sub-module operates using three DSP slices in parallel to achieve a higher throughput, whereby computing the three sub-polynomials ab_high, ab_low, and ab_mid in a single loop. Lastly, the proposed novel negacyclic algorithm for Karatsuba is applied in negacyclic stage, implemented using the k_combi_nega sub-module.

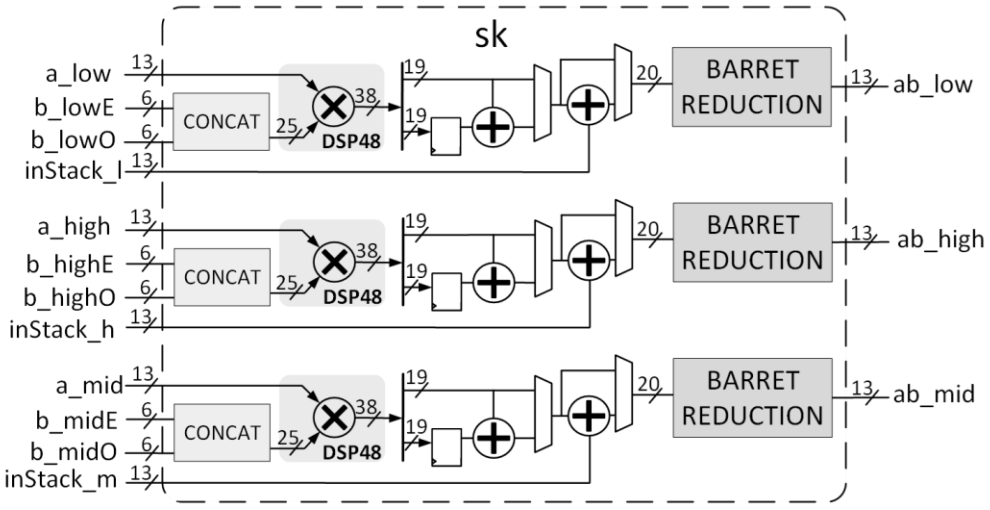


Figure 4.3 Block diagram of sk sub-module

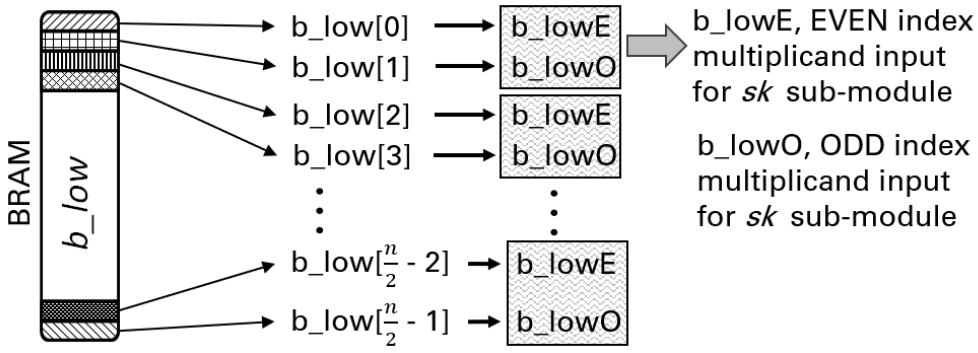


Figure 4.4 Reading polynomial data for multiplicand

CHAPTER 5

POLYNOMIAL CONVOLUTION HARDWARE DESIGN FOR SABER SCHEME

This chapter gives the architectural details, improvement over the earlier work and the the rationale for design choices of KaratSaber that employs SPMA-Karatsuba polynomial multiplier for Saber. The hierarchical Karatsuba Architecture KaratSaber uses a 4-layer Karatsuba implementation in the pre-process and post-process stage. In the earlier work by (Zhu et al., 2020), an 8-Layer Karatsuba, with four layers at the pre and post-process stages, and another four at the multiplication stage was presented introducing large pre-processing and post-processing overhead, which is far from optimal. A feasibility analysis was carried out to reach the optimized value of number of layers in Karatsuba recursion.

Karatsuba Hierarchy	1-L	2-L	3-L	4-L	5-L	6-L	7-L	8-L
Clock Cycles	96	80	72	68	66	65	65	65
Sub-Polynomials	3	9	27	81	243	719	2187	6561

Table 5.1 Clock cycles for different layers of Karatsuba

Referring to Table I, after performing a four layer (4-L) hierarchical Karatsuba in the pre and post-process stages, it records the best timing performance (68 clock cycles). Increasing the number of layers to six (6- L) can offer a marginal improvement (4.4%) in performance. However, this improvement does not outweigh the hardware consumption for a more sophisticated pre-process and post-process hardware architecture to handle 719 sub-polynomials. Hence, we do not increase the Karatsuba recursion beyond four layers.

The proposed 4-layer Karatsuba architecture goes through three stages. The first stage is the pre-processing of input polynomials into sub-polynomials with a smaller degree. The Karatsuba recursion produces 3^x sub-polynomials, where x is the number of Karatsuba layers. Therefore, a 4-layer Karatsuba implementation would result in $3^4 = 81$ sub-polynomials in total, wherein each sub-polynomial has a degree of $n/2^x = 256/16 = 16$. To speed up the computation, a fully parallel grid data input technique was adopted. After pre-processing, the multiplication stage take place, wherein the 81 sets of degree-16 sub-polynomials undergo polynomial multiplication, resulting in 81 set of degree 31 sub-polynomials multiplication results. In the final stage, the multiplication results are combined and the negacyclic operations are performed. To compute the post-process and negacyclic operations in a more efficient fashion, an instruction code-based post-process mapping technique with negacyclic is proposed.

To ensure a fair comparison, we developed our multiplier with the same input bit size as (Roy and Basso, 2020). The multiplier handles polynomial *a* input in 208-bit wide (16 coefficients, size of each coefficient is 13-bit), and polynomial *b* input in 64-bit wide (16 coefficients, size of each coefficient is 4-bit). The proposed KaratSaber architecture is built upon this 4-layer hierarchical Karatsuba architecture, which includes shifter-based multipliers and several novel techniques that streamline the pre-processing and post-processing operations.

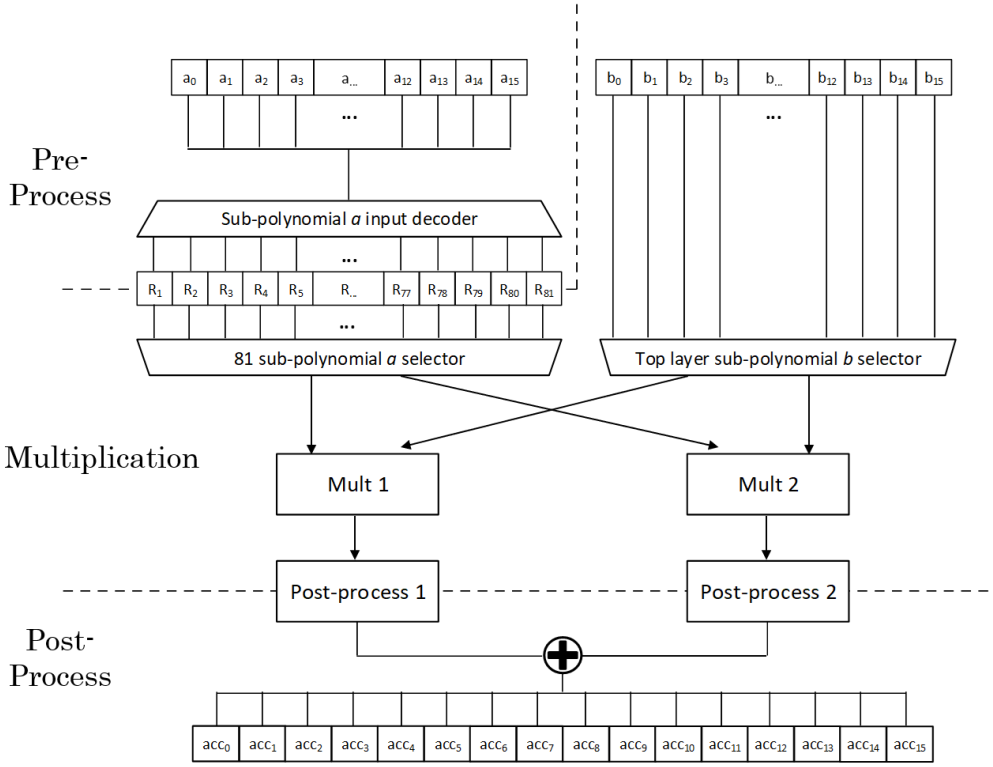


Figure 5.1 KaratSaber top level diagram

5.1 Parallel Grid Data Input

In (Roy and Basso, 2020), the compact input pre-processing is designed to only process and output a single pair of sub-polynomial for multiplication per cycle. Although some intermediate values are stored by reusing some registers, the pre-processing stage includes several no-op cycles. In other words, data dependency constraints the efficiency of pre-processing since some data requires a waiting time for the previous data to be computed and stored first, leaving some cycles empty.

To increase the parallelism of the polynomial multiplier, the pre-process stage involving the splitting of the degree-256 polynomial was performed in a parallel fashion. The 81 sub-polynomials are required to be stored in 81 parallel register sets to enable a fast data access. To achieve such performance, the data has to be ready as soon as possible. Referring to Figure 3, the polynomial can be first split into the top layer 16 sub-polynomials (a0, a1, ..., a15) each with 16 coefficients. At this moment, the top layer sub-polynomials are passed to the pre-process module as inputs for each cycle in parallel. Note that one needs to determine the register set for storing the top layer sub-polynomial. With this proposed mechanism, the data input is processed in a completely parallel manner. This enables more than one input to be accessed for multiplication per cycle through reusing the registers by exploiting the sub-polynomial grid pattern, which is explained subsequently.

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	Sub-polynomial value in register
R_1	X																a_0
R_2		X															a_1
R_3	X	X															$a_0 + a_1$
...																	
R_{25}	X		X		X		X										$a_0 + a_2 + a_4 + a_6$
R_{26}		X		X		X		X									$a_1 + a_5 + a_3 + a_7$
R_{27}	X	X	X	X	X	X	X	X									$a_0 + \ldots + a_7$
...																	
R_{52}	X		X		X		X		X		X		X		X		$a_0 + a_2 + a_4 + a_6 + a_8 + a_{10} + a_{12} + a_{14}$
R_{53}		X		X		X		X		X		X		X		X	$a_1 + a_5 + a_3 + a_7 + a_9 + a_{11} + a_{13} + a_{15}$
R_{54}	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	$a_0 + \ldots + a_{15}$
...																	
R_{79}									X		X		X		X		$a_8 + a_{10} + a_{12} + a_{14}$
R_{80}										X		X		X		X	$a_9 + a_{11} + a_{13} + a_{15}$
R_{81}									X	X	X	X	X	X	X	X	$a_8 + \ldots + a_{15}$

Figure 5.2 Parallel input data loading pattern to calculate the 81 sub-polynomials using top layer sub-polynomials (a_0, a_1..., a_{14}, a_{15})

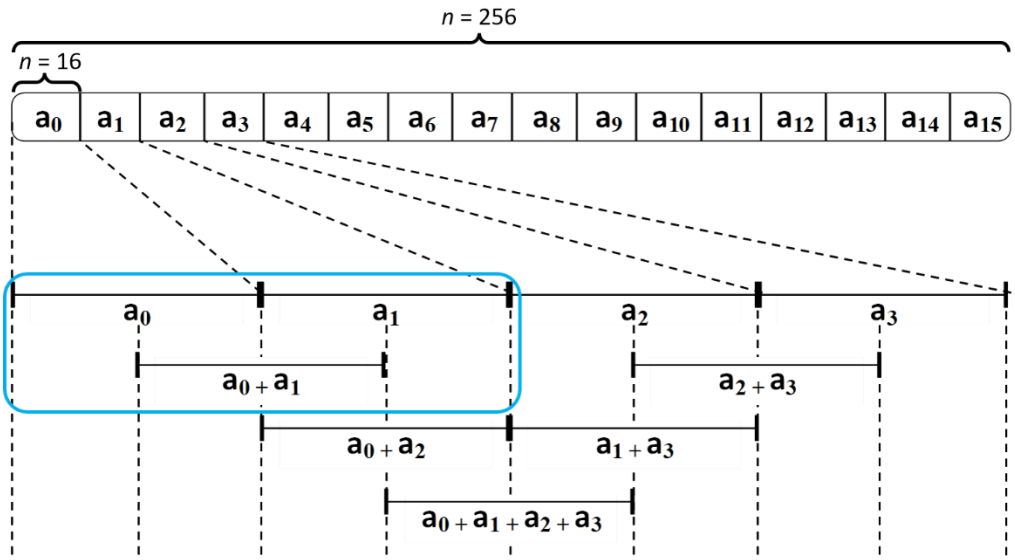


Figure 5.3 Computing sub-polynomials splitting according to layers

The sequence of sub-polynomials for consecutive layers were pre-computed prior to the implementation. Figure 5.3 shows an example of computing the first nine sub-polynomials. Continue from the top layer sub-polynomials, the second layer sub-polynomials were computed, i.e., a_0 and a_1 produces $a_0 + a_1$. The process is then repeated until all remaining sub-polynomials are computed. After 81 iterations, the results of all sub-polynomials were computed as shown in the last column in Figure 5.2.

According to the results computed, each of them is either a top layer sub-polynomial, or a combination of different top layer sub-polynomials. Hence, if the register set consist of a particular top layer sub-polynomial, it is marked. Finally, with a completed table, we can hard-code each register to either hold or add in the new value, depending of the final sub-polynomial it is supposed to store. By doing so, the pre-process can be done in only 16 cycles.

However, implementing 81 sets of registers for all sub-polynomials would require a total of 81 × 16 × 13 = 16848 registers. Referring to Figure 5.3, upon observing the full grid data input sequence, the input pattern actually repeats itself for every nine register sets across four top layer sub-polynomials, i.e., registers set $R_1 \rightarrow R_9$ and top layer sub-polynomials $a_0 \rightarrow a_3$. The next pattern repetition occurs in the register sets $R_{10} \rightarrow R_{18}$ and the top layer sub-polynomials $a_4 \rightarrow a_7$. The same pattern is repeated for a total of 16 times throughout the entire data input process. Based on this pattern, we can reduce 81 register sets to only nine by reusing them. After the first set of nine sub-polynomial multiplications are computed, the top layer sub-polynomials $a_4 \rightarrow a_7$ are added to the nine register sets, resulting in sub-polynomials $19 \rightarrow 27$. When the second set of nine polynomial multiplications are completed, top layer sub-polynomials $a_4 \rightarrow a_7$ is again loaded. At this time, the previous values in registers $R_1 \rightarrow R_9$ can be overwritten, since we no longer need the values of top layer sub-polynomials $a_0 \rightarrow a_3$; this produces sub-polynomials $10 \rightarrow 18$. The process of adding and overwriting the nine register sets is repeated until all sub-polynomials have undergone the multiplication process.

	a0	a1	a2	a3	a4	a5	a6	a7	a8 – a15
1	X								
2		X							
3	X	X							
4			X						
5				X					
6			X	X					
7	X		X						
8		X		X					
9	X	X	X	X					
10					X				
11						X			
12					X	X			
13							X		
14								X	
15							X	X	
16					X		X		
17						X		X	
18					X	X	X	X	
19	X				X				
20		X				X			
21	X	X			X	X			
22			X				X		
23				X				X	
24			X	X			X	X	
25	X		X		X		X		
26		X		X		X		X	
27	X	X	X	X	X	X	X	X	

Figure 5.4 Fully parallel grid input data loading for sub-polynomials 1-27

5.2 Partial Sub-polynomial Multiplication

Referring to Figure 5.3, R_3 should be holding the value of $(a_0 + a_1)$, and it is supposed to be multiplied with $(b_0 + b_1)$. However, pre-processing (b_0+b_1) would result in the overflow problem mentioned earlier in Chapter 3.3.

$$(a_0 + a_1)(b_0 + b_1) = (a_0 + a_1) * b_0 + (a_0 + a_1) * b_1$$

Hence, to avoid the problem of overflow, a partial multiplication technique for each sub-polynomial was proposed by performing separate multiplications between the same multiplier $(a_0 + a_1)$ and different multiplicands (b_0 and b_1) respectively. In other words, only polynomial *a* is required to undergo pre-process.

By doing so, we eliminate the overflow problem of polynomial b regardless of how many layers of Karatsuba is implemented. Although the number of multiplications increases, the hardware consumption and clock cycles are still relatively low compared to multipliers that utilize DSP48E1 blocks. This can be achieved by cleverly stacking the operations and performing them simultaneously in our shifter-based multiplier, to cater for overflowed polynomial b. Further discussion and comparison that involves the design of shifter-based multiplier are presented in Section 5.4.

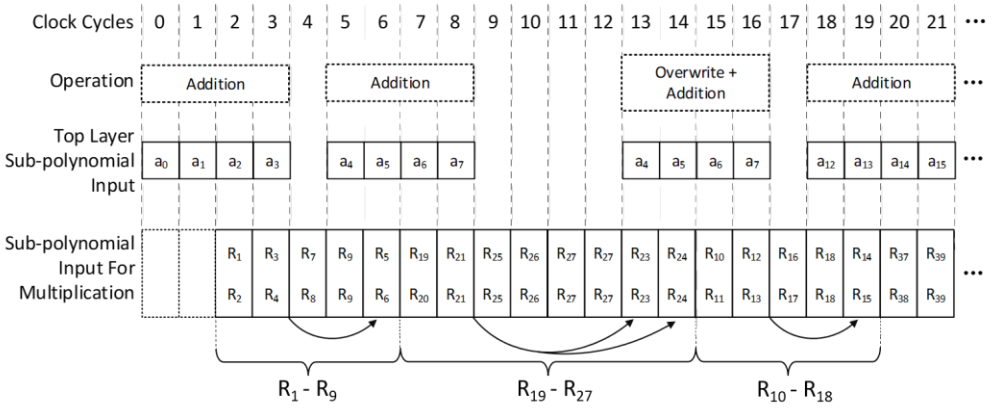


Figure 5.5 Input sequence for top layer sub-polynomials and multiplication

Figure 6 shows the sub-polynomial input sequence for multiplication. Since each set of shifter-based multipliers can handle one multiplier and two multiplicands, two sets of multiplications are carried out. For multiplications that only has a single multiplicand, i.e., $a_0 * b_0$, a value 0 is taken as the second multiplicand. Nevertheless, to avoid data dependencies and empty cycles, the top layer sub-polynomial input is carefully arranged. Referring to Figure 6, after the first two cycles, register sets R_1, R_2 and R_3 will be ready for multiplication since they only consist of a_0 and a_1. After the 4^{th} cycle, the nine register sets already has the first nine sub-polynomials (1→9) stored waiting for multiplication.

To ensure a smooth transition, the next top layer sub-polynomials $a_4 \rightarrow a_7$ have to be added starting from the 5^{th} cycle. However, the first nine multiplications are yet to be completed, and interfering with the registers too early may lead to the multiplier inputted with inaccurate data. To overcome this, multiplication for R_5 and R_6 is done lastly for the first nine multiplications. Referring to Figure 5.4, the arrow displays the original and modified sequence of R_5 and R_6, changing it from the 5^{th} to the 7^{th} cycle.

Among the four top layer sub-polynomials loaded, R_5 and R_6 only store the last two top layer sub-polynomials, i.e., (a_2 & a_3) or (a_6 & a_7) etc. This enables the first two top layer sub-polynomials to be added or overwritten before the current nine sub-polynomial multiplications have been fully completed without affecting the data integrity. By the third cycle, registers R_1, R_2 and R_3 are already ready for the multiplication process, therefore the pre-process stage can be overlapped and polynomial multiplication can be started at the third cycle.

5.3 Code-based Post-processing Mapping with Negacyclic

(Zhu et al., 2021) only implemented two layers of mapping, which means the post-process is separated into several parts. As soon as a degree 64 sub-polynomial multiplication result is ready, it is transferred out to an external memory element storing the intermediate results. In other words, they implemented a multiplier that depends on external infrastructure to complete the multiplication process. Furthermore, the negacyclic operations after completing the entire post-process are not documented clearly in their work. A Karatsuba mapping process that enables the sub-polynomial multiplication results to be directly transposed back into the final multiplication results was proposed, along with the necessary negacyclic operations.

After the multiplication process, we would arrive at 81 sub-polynomial multiplication results that need to undergo four layers of Karatsuba combination or post-process. Furthermore, negacyclic operations are required since Karatsuba does not support integration of negacyclic during multiplication like the case in SPMA.

We can reverse the method used in pre-processing, and map the results back into the final polynomial directly from the fourth layer. Since the final polynomial would be a degree $(2 * 256) - 1 = 511$ polynomial, a total of 32 sets of 16 coefficient registers are required to hold the final polynomial result.

The results of multiplication with degree 31 were then concatenated with a 13-bit 0's as the MSB. Finally, the concatenated degree 32 polynomial multiplication result is split into lower half, R_L, and higher half, R_H, each holding a degree-16 polynomial.

The mapping process is essentially combining the sub-polynomial multiplication results back into the full multiplication results. Conventionally, the post-process for each layer is done hierarchically, going from the lowest layer to the final layer. However, since we can directly compute the sub-polynomials such as in pre-processing, the mapping process can actually be computed by reversing the pre-process steps, the layer diagram of sub-polynomial in Figure 5.3 can be extended to the final register sets acc_i, where $i = 0 \rightarrow 31$ as shown in Figure 8. The first four sets of top layer sub-polynomials $a_0 \rightarrow a_3$ with a total degree of 64 produces a degree 128 sub-polynomial multiplication result after MSB concatenation. Hence, eight sets of registers $acc_0 \rightarrow acc_7$, each with a capacity of degree-16 polynomial is required.

To determine the sequence of mapping, all occurrence of the targeted sub-polynomial is tabulated. Referring to Figure 5.6, all sub-polynomials are always multiplied with its corresponding pair, i.e., $a_0 * b_0$, $(a_0 + a_1) * (b_0 + b_1)$. All multiplication results in Figure 5.6 are only denoted by polynomial *a*. For instance, $[a_0 * b_0]$ is denoted by a_0 and $(a_0 + a_1) * (b_0 + b_1)$ is denoted by $(a_0 + a_1)$.

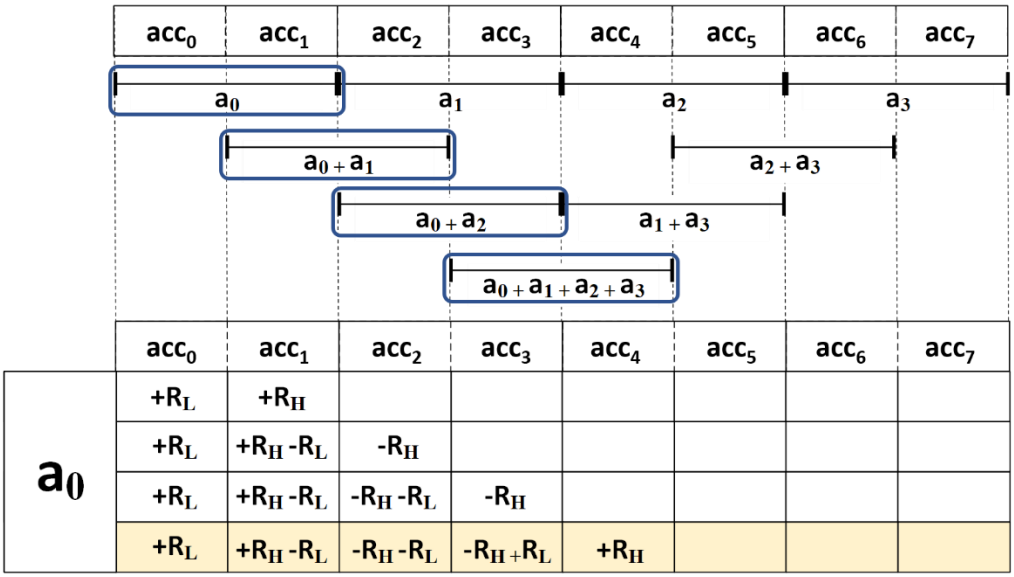


Figure 5.6 Computing sub-polynomials mapping according to layers

The first sub-polynomial result that requires mapping in our case is a_0. Based on the sub-polynomial layer diagram in Figure 5.6, to compute the full mapping sequence for a_0, all occurrence of a_0 is tabulated layer by layer. Since we are starting with a single coefficient sub-polynomial a_0, a positive result ($+R_L$, $+R_H$) is stored to the respective register sets. When the coefficient size is doubled due to addition (i.e., (a_0 + a_1) and (a_0 + a_2)), the polarity of the result is toggled and a negative result ($-R_L$, $-R_H$) is stored. Lastly, for the next subsequent doubled coefficient size (a_0 + a_1 + a_2 + a_3), the polarity is again toggled, changing it back into a positive result. In summary, regardless of the size of coefficients, the computation always starts from a positive polarity, and is toggled every time the coefficient size doubles.

The mapping sequence is done for all 81 sub-polynomials, spanning across 32 sets of registers acc_i. Before actually implementing the design, we proposed to integrate the negacyclic operations here. The upper half ($acc_{16} \rightarrow acc_{31}$) of register set acci is subtracted from the lower half register sets ($acc_0 \rightarrow acc_{15}$). For the example in Figure 5.6, the upper half ($acc_4 \rightarrow acc_7$) is subtracted from ($acc_0 \rightarrow acc_3$). Referring to Figure 5.7, the register sets required is reduced in half immediately, requiring only $acc_0 \rightarrow acc_3$ now, and for the full extend case, the total register sets required is reduced from $acc_0 \rightarrow acc_{31}$ to $acc_0 \rightarrow acc_{15}$.

Referring to Figure 5.8, the mapping sequence of first nine sub-polynomials are shown in the upper part. Without optimization, the mapping sequence has to be hard-coded and components will be fixed for ad-hoc computation. Optimization such as rearrangement of the sequence or scaling the postprocess would be impossible. Hence, we propose to transform the mapping sequence into a code-based instruction to gain flexibility in changing and modifying the sequence. This is also highly beneficial for scaling or modifying the multiplier. Referring to Table 5.2, each mode is represented by a decimal number, and the number 0 is used to fill in the empty slots, representing no operation. Finally, we would arrive at a much more orderly mapping sequence for each sub-polynomials. During implementation, each value to be stored in the final register set acc_i's is simply multiplexed using the proposed code-based instructions for each sub-polynomials. Since each row of

code-based sequence is fixed for each multiplication results, the column sequence can be simply rearranged for scalability purposes, i.e., four multiplications per cycle.

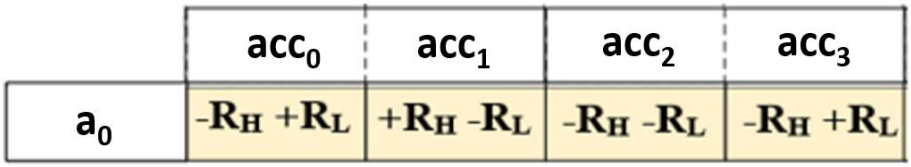


Figure 5.7 Negacyclic operation for mapping process

Mode	Code
$+R_L$	1
$+R_H$	2
$-R_L$	3
$-R_H$	4
$+R_H + R_L$	5
$+R_H - R_L$	6
$-R_H + R_L$	7
$-R_H - R_L$	8

Table 5.2 Code-based transformation

		acc_0	acc_1	acc_2	acc_3	acc_4	acc_5	acc_6	acc_7	acc_8	acc_9	acc_{10}	acc_{11}	acc_{12}	acc_{13}	acc_{14}	acc_{15}
1	α_0	$-R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$
2	α_1	$-R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$
3	$\alpha_0 + \alpha_1$	$+R_H$	$+R_L$	$+R_H$	$-R_L$	$-R_H$	$-R_L$	$-R_H$	$+R_L$	$+R_H$	$-R_L$	$-R_H$	$+R_L$	$+R_H$	$+R_L$	$+R_H$	$-R_L$
4	α_2	$-R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$
5	α_3	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H - R_L$	$-R_H + R_L$	$+R_H + R_L$	$+R_H - R_L$	$-R_H + R_L$
6	$\alpha_2 + \alpha_3$	$+R_H$	$-R_L$	$-R_H$	$-R_L$	$-R_H$	$+R_L$	$+R_H$	$+R_L$	$+R_H$	$-R_L$	$-R_H$	$+R_L$	$+R_H$	$-R_L$	$-R_H$	$-R_L$
7	$\alpha_0 + \alpha_2$	$+R_H$		$+R_L$	$+R_H - R_L$	$-R_H$		$-R_L$	$-R_H + R_L$	$+R_H$		$-R_L$	$-R_H + R_L$	$+R_H$		$+R_L$	$+R_H - R_L$
8	$\alpha_1 + \alpha_3$	$+R_H - R_L$	$-R_H$		$-R_L$	$-R_H + R_L$	$+R_H$		$+R_L$	$+R_H - R_L$	$-R_H$		$+R_L$	$+R_H - R_L$	$-R_H$		$-R_L$
9	$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3$	$-R_H$			$+R_L$	$+R_H$			$-R_L$	$-R_H$			$-R_L$	$-R_H$			$+R_L$
		acc_0	acc_1	acc_2	acc_3	acc_4	acc_5	acc_6	acc_7	acc_8	acc_9	acc_{10}	acc_{11}	acc_{12}	acc_{13}	acc_{14}	acc_{15}
1	α_0	7	6	8	7	6	7	5	6	8	7	5	6	7	6	8	7
2	α_1	7	6	7	5	6	7	6	8	7	5	6	8	7	6	7	5
3	$\alpha_0 + \alpha_1$	2	1	2	3	4	3	4	1	2	3	4	1	2	1	2	3
4	α_2	8	7	6	7	5	6	7	6	8	7	5	6	8	7	6	7
5	α_3	7	5	6	7	6	8	7	6	7	5	6	8	7	5	6	7
6	$\alpha_2 + \alpha_3$	2	3	4	3	4	1	2	1	2	3	4	1	2	3	4	3
7	$\alpha_0 + \alpha_2$	2	0	1	6	4	0	3	7	2	0	3	7	2	0	1	6
8	$\alpha_1 + \alpha_3$	6	4	0	3	7	2	0	1	6	4	0	1	6	4	0	3
9	$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3$	4	0	0	1	2	0	0	3	4	0	0	3	4	0	0	1

Figure 5.8 Code-based transformation for mapping sequence with various mode

5.4 Hardware Design

The KaratSaber Polynomial Core comprises of three major blocks: the pre-process module, the Shift-Two-Multiplicand (STM) multiplication module and the post-process module, along with a controller and a wrapper engulfing all, as referred in the Figure 5.9. A total of 512 shift-based multipliers are implemented for the polynomial multiplication. As soon as the first set of data are ready after pre-processing, the multiplication process is performed. The multiplication results are then passed as input for post processing and forwarded to the register sets storing the final polynomial multiplication results with negacyclic.

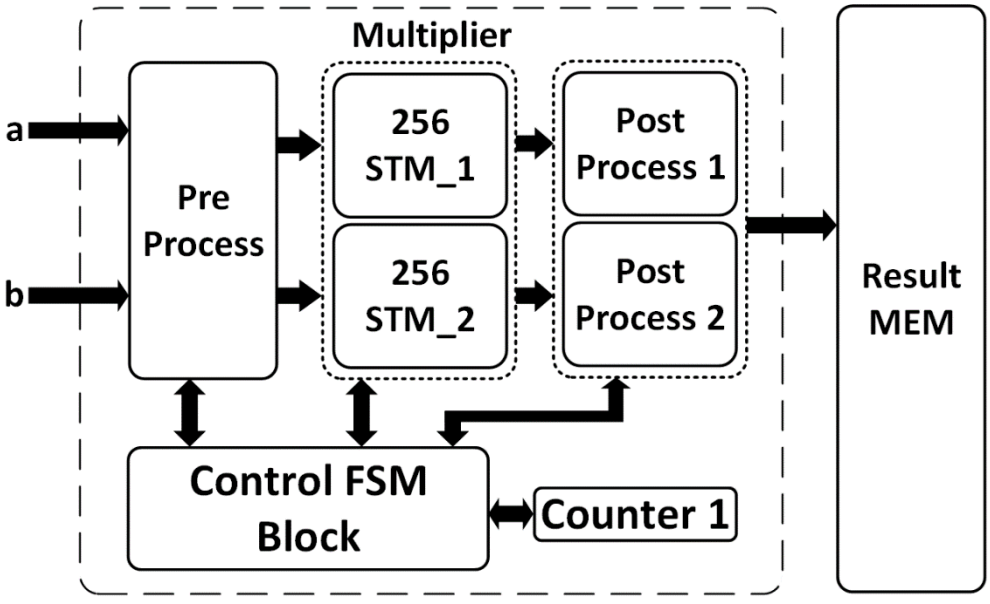


Figure 5.9 KaratSaber polynomial multiplier block diagram

5.4.1 Shift-Two-Multiplicands (STM) Module

The coefficients of secret polynomial (polynomial *b*) are randomly generated through binomial distributions with varying range of [-3,3], [-4.4] and [-5,5], depending on the level of security (LightSaber, Saber768, FireSaber). Combined with the fact that Saber operates on modulo with power of 2, a truncated modular multiplier can be developed by using shifters only. From a hardware consumption point of view, DSP48E1 blocks consume more hardware area than a shift-based multiplier. For instance, on our target FPGA device (ZCU102), one DSP-based multiplier consumes 70 Configurable Logic Block (CLB) slices, whereas a shift-based multiplier only consumes 10 CLB slices. With the vast difference in hardware consumption, we can used it as our advantage since there is much room for additional hardware targeting a throughput improvement.

In (Roy and Basso, 2020), they proposed a Multiply-And-Accumulate shifter-based multiplier to replace conventional DSP48E1 blocks. The shifting technique for multiplication in our multiplier is similar since the process is straight forward. However, we do not need the accumulator for the multiplication results since we are not utilizing SPMA. Furthermore, we also doubled the multiplier throughput by increasing the number of multiplicands.

In this research, a shifter-based multiplier capable of processing two multiplications per cycle by pairing two multiplicands with one multiplier, namely the Shift-Two-Multiplicand (STM) module was proposed. Since the two multiplicands share the same multiplier, the same set of results are multiplexed to two different results. Referring to Figure 12, the values of multiplicands (*b1* and *b2*) are used as a multiplexer signal to choose between the same set of multiplier results.

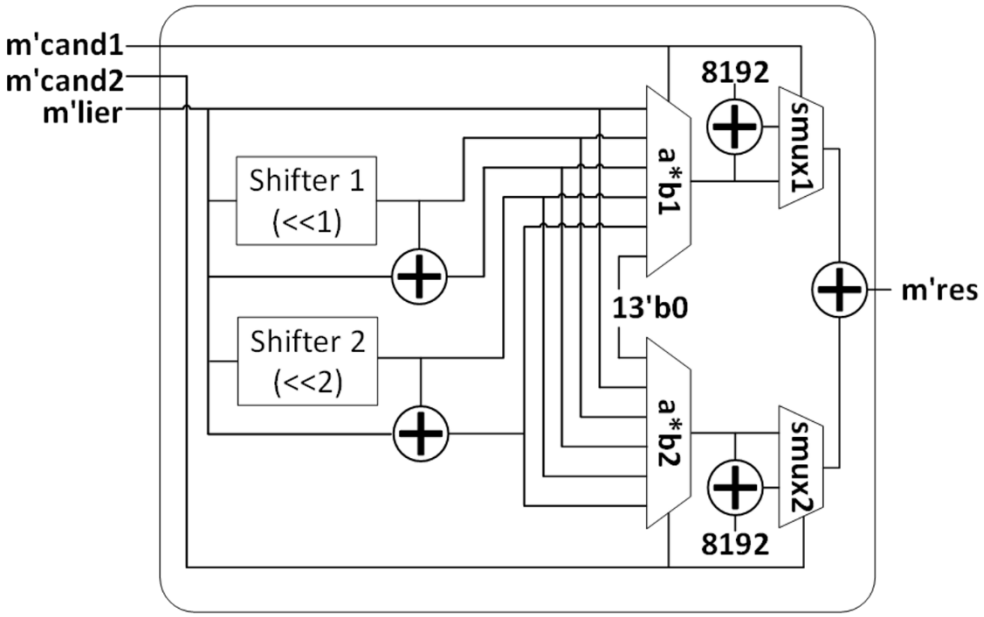


Figure 5.10 STM module block diagram

Multiplier	Slice[1]	Mult-per-cycle	Efficiency[2]
DSP48E1	70	1	1.43
(Roy and Basso, 2020)'s MAC	10	1	10.00
This work's STM	19	2	10.53

Table 5.3 Hardware consumption comparison for different multipliers

[1] Slice = FPGA Configurable Logic Block (CLB) slice

[2] Efficiency = (Multiplication-per-cycle/Number of slices)*100

Even when paired with two multiplicands, the STM module only consumes 19 CLB slices. Referring to Table 5.3, when compared to a single multiplicand shifter-based multiplier such as (Roy and Basso, 2020)'s MAC, we doubled the throughput by utilizing 90% more hardware, increasing the efficiency by 5.3%. Whereas when compared to a DSP48E1 multiplier, the proposed STM is 3.68× smaller with twice as much throughput. During implementation, the multiplier modules are often utilized in great numbers (i.e., 256) parallelly, hence even a small improvement in efficiency such as 5.3% would contribute to a large amount of reduction in hardware consumption.

CHAPTER 6

EXPERIMENTAL RESULTS AND DISCUSSION

6.1 SPMA-Karatsuba (SK) R-LWE Polynomial Multiplier Utilizing Karatsuba

The proposed architecture was implemented and synthesized on a Xilinx Kintex 7 FPGA (KC705) using Vivado 2019.2. A total of four generations of SK R-LWE Polynomial Multiplier Core were developed in this work (SK Gen 1-4) and the results are presented in Table 6.1. Other than the Karatsuba algorithm, Gen 1 also utilizes the barrett reduction from (Liu et al. 2019).

Generation	LUT/FF/Slice[1] /BRAM/DSP	Freq (MHz)	Clock Cycles	Throughput (Kbps)	TPS[2]
SK Gen 1	538/611/189 /3.0/1	299.40	50829	1507.9	7.98
Sk Gen 2	698/734/289 /8.0/1	328.95	25876	3254.4	11.26
SK Gen 3	872/881/363 /4.0/1	332.67	22565	3775.8	10.40
SK Gen 4	1125/1034/394 /3.0/3	335.80	8787	9783.2	24.83

Table 6.1 FPGA implementation results of SK R-LWE PM core

[1] Slice - FPGA Configurable Logic Block (CLB) slice,

[2] TPS - Throughput per slice, without considering BRAM and DSP

Gen 2 is essentially the Gen 1 architecture with additional DSP slice utilization algorithm implemented. Building on Gen 2 architecture, the novel negacyclic algorithm is implemented in Gen 3. Finally, Gen 4 was built with extra DSP slices shows a significant increase in throughput, taking full advantage of the Karatsuba algorithm. To ensure a fair comparison of this work with the other implementations on different FPGAs e.g., Virtex/Spartan, a conversion is performed to compute the equivalent Configurable Logic Block (CLB) slice value for the DSP48E1 slices and BRAMs.

By utilizing the built-in IP core for a Xilinx 7 series, a single DSP48E1 slice can be substituted with a 25- bit × 18-bit multiplier resulting in 128 slices. Similarly, an 18K BRAM can be substituted by 166 slices (Liu et al. 2019). However, the proposed design does not fully utilize the DSP slice and BRAMs, therefore, a weight ratio is given to accurately depict the actual number of slices utilized. The weight and equivalent slices in SK Gen 4 implemented on the FPGA are shown in detailed in Table IV.

Attribute	Utilization (BRAM/DSP)	BRAM (18K)	DSP
Equivalent CLB Slices (ECS)	1/1	166	128
Implementation ECS	6/3	996	384
Weight	6/3	0.19	0.88
Actual Implementation ECS	6/3	189.2	337.9

Table 6.2 Equivalent CLB slices (ECS) for DSP and BRAM blocks in SK GEN 4

Table 6.3 compares the proposed SK R-LWE PM core with earlier implementations of the SPMA R-LWE cryptoprocessors, which implemented the same parameters (n = 256, q = 7681). The work by (Zhang et al., 2020) only presented the results for entire R-LWE cryptoprocessors. In this paper, we have implemented the PM core (SPMA-4) based on the description of their work in order to provide a fair comparison. Since this proposed work is not implementing a fully recursive Karatsuba algorithm, the conventional complexity equation O(n) cannot be used to compute the total number of cycles. Given that the polynomial was split into three sub-polynomials for computation using SPMA with DSP utilization, the total number of cycles would be $O(n^2)$ 2 ÷ 2 + n + n 2 cycles for all stages, including 211 constant extra cycles for overhead and pipe-lining purposes.

Experimental results show that the proposed architecture achieved 2.09× higher throughput along with a 6.52% improvement in throughput-per-

slice. However, the architecture from (Liu et al., 2019) remains the most area-time efficient design, with a much lower throughput (3.92× slower than our SK Gen 4).

	Device	LUT/FF/Slice	BRAM (18K)	DSP	Freq (MHz)	Cycle	Throughput (Kbps)	ECS	Throughput/ Slice[1]
This Work	Kintex 7	1125/1034/394	3.0	3	335.80	8787	9783.2	921.1	10.62
(Liu et al., 2019) (SPMA)	Kintex 7	317/198/103	0.0	1	333.00	34177	2494.3	210.4	11.85
(Zhang et al., 2020)[2] (SPMA)	Kintex 7	699/705/265	0.0	2	300.95	16456	4681.8	469.8	9.97
(Feng et al., 2020)[3] (NTT)	Spartan 6	-/-/8680	0.5	128	235.29	220	273.8k	25125.4	10.9
(Liu et al., 2018)[4] (NTT)	Spartan 6	1307/889/406	0.5	1	80.00	72.0k	284.44	462.0	0.62

Table 6.3 Comparison with previous R-LWE implementation

[1] 3 Calculated using Equivalent CLB slices (ECS)

[2] Re-implemented by following the original paper (Zhang et al., 2020)

[3] Estimated by scaling the 21-bit modulus q to 13-bit

[4] Result of Complete R-LWE Processor in (Liu et al., 2018)

6.2 KaratSaber Saber Polynomial Multiplier Utilizing Karatsuba

The proposed KaratSaber architecture is synthesized and implemented on a Xilinx Zynq UltraScale+ FPGA (ZCU102) using Vivado 2019.2. To benchmark efficiency vs. architectural optimization, three versions of KaratSaber Polynomial Multiplier Core (Saber PM 1-3) were developed for both Saber768 and unified Saber (LightSaber, Saber768, FireSaber). KaratSaber PM 1 is the basic architecture that utilizes a 4- layer hierarchical Karatsuba. It implements the proposed fully parallel data input and instruction code-based post-process mapping with negacyclic. Only a single multiplicand shifter-based multiplier is used in this iteration. In KaratSaber PM 2, we implemented the proposed STM modules by doubling the multiplicand inputs to the shifters. Towards our aim, some additional optimizations were done, which include adding pipelines at the modified STM and control modules to increase the throughput. Data input of the STM modules were rearranged to reduce data dependencies. In KaratSaber PM 3, data input for the STM modules were rearranged to further reduce data dependencies, reducing the idle cycles from nine to three. Lastly, in KaratSaber PM 4, the proposed parallel grid data input is introduced, reducing the hardware consumption drastically. Data input for the STM modules was again rearranged (depicted in Chapter 5.2 and Figure 5.5).

Generation	LUT/ FF/ Slice[1]	Freq (MHz)	Clock Cycles	Throughput (Mbps)	TPS[2]
KaratSaber PM 1	84554/ 52060/ 12948	229.15	152	387.88	29.96
KaratSaber PM 2	115403/ 70805/ 18959	336.02	88	984.62	51.93
KaratSaber PM 3	97111/ 64128/ 16074	338.18	82	1066.67	66.35
KaratSaber PM 4	77546/ 56658/ 11916	322.16	82	1005.89	84.42

Table 6.4 KaratSaber768 PM Core Post PAR results on Zynq UltraScale+ FPGA (ZCU102)

Table 6.4 shows the implementation of this work from KaratSaber PM 1 to PM 4, all implemented with post-place and post-route. To ensure a fair comparison of this work with other implementations utilizing different Ultrascale+ FPGAs e.g., Zynq/Virtex/Kintex, a metric is used to compute the equivalent Configurable Logic Block (CLB) slices value. This metric was also used in the previous work (Liu et al., 2019), (Zhang et al., 2020), (Wong et al., 2021), for comparing results from different FPGA implementations.

The LUTs and FFs consumption reported in (Roy and Basso, 2020), (Zhu et al., 2021) and (He et al., 2021) can be converted into CLB slice values. Referring to the Xilinx Ultrascale Architecture CLB resources documentation (Ultrascale Architecture), both SLICEL and SLICEM has the same amount of eight LUTs and 16 FFs per CLB slice. Hence, we can easily convert the documented LUTs and FFs into equivalent CLB slice (ECS) values. For example, based on the hardware consumption reported by (Roy and Basso, 2020), the value of 17429 LUTs is divided by eight, and the result is rounded to the nearest integer (ceiling operation), giving us 2179 ECS. For the value of 5083 FFs, it is divided by 16, giving us 318 ECS. In total, we can arrive at a total ECS of 2496.

In most cases, the CLB slices are not fully utilized, therefore the ECS values computed through conversion may be slightly underestimated compared to the actual hardware consumption, since it also excludes some circuitries and minor some components. According to the CLB conversion and the actual CLB slices consumed in this work, we found that there is a 2.6% difference between these two values. However, we did not leverage this difference and maintained the original value computed.

Xilinx Vivado does not provide the resources utilized for DSP and BRAM blocks. To convert the DSP48E1 blocks into ECS, we can utilize the built-in IP core for a Xilinx Ultrascale series, whereby a single DSP48E1 slice

can be substituted with a 25-bit × 18-bit unsigned multiplier resulting in 70 slices. For implementations that utilize DSPs such as (Zhu et al., 2021), the DSPs are converted into ECS values, i.e., 85 × 70 = 5950.

Table 6.5 compares the proposed Saber768 and Saber unified polynomial multiplier core with earlier Saber implementations that utilize both SPMA and Karatsuba. For clarity, we take our cores throughput as reference (1×) and compare with all other cores throughput in comparison (e.g., n times lower depicted as ↓ n×). Similarly, a TPS percentage difference (as ±m%) of our cores compared to the reported results is also presented. Experimental results prove that the proposed Saber768 architecture achieved 7.47× and 20.04× higher throughput when compared to Saber polynomial multipliers developed utilizing SPMA in (Roy and Basso, 2020) and (He et al., 2021) respectively. Although (He et al., 2021) remains the most area-time efficient architecture with the highest throughput per slice (TPS), the throughput is extremely low, which may not be suitable for applications that require a timely response. When compared to the Karatsuba implementation for Saber768 in (Zhu et al., 2021), the proposed polynomial multiplier achieved a 2.04× higher throughput along with a 26.98% improvement in TPS. On the other hand, the proposed Saber unified (LightSaber, Saber768, FireSaber) polynomial multiplier achieved 8.10×, 27.01× and 3.40× higher throughput when compared to (Roy and Basso, 2020), (He et al., 2021) and (Zhu et al., 2021) respectively.

Variation	Implementation	LUT/ FF/ Slice	DSP	Freq (MHz)	Cycle		Throughput (Mbps)	Speed up (x)		ECS		Throughput per Slice[1]	Improve-ment (%)
Saber768	This work[2]	77546 / 56658 / 11916	0	322.16	82		1005.89	**1.00**		11916		84.42	**100**
	(Zhu et al., 2021)	13735 / 4486 / 1998	85	160.00	83		493.45	↓2.04		7948		62.08	-26.98
	(Roy and Basso, 2020)	17429 / 5083 / 2496	0	250.00	297		134.74	↓7.47		2570		53.98	-46.04
	(He et al., 2021)	2231 / 1737 / 492	0	250.00	1279		50.20	↓20.04		492		102.02	+22.73

Table 6.5 KaratSaber768 Comparison with Previous Saber768 Implementations

[1] Calculated using Equivalent CLB slices (ECS)

[2] This work's KaratSaber768 PM 4

[3] This work's KaratSaber unified PM 4

Variation	Implementation	LUT/ FF/ Slice	DSP	Freq (MHz)	Cycle		Throughput (Mbps)	Speed up (x)		ECS		Throughput per Slice[1]	Improve-ment (%)
Saber Unified	This work[3]	89222 / 56657 / 14082	0	335.68	82		1024.00	**1.00**		14082		74.41	**100**
	(Zhu et al., 2021)	13735 / 4486 / 1998	85	100.00	83		308.43	↓3.40		7948		38.81	-84.62
	(Roy and Basso, 2020)	17429 / 5083 / 2496	0	150.00	297		129.19	↓8.10		2570		51.80	-38.32
	(He et al., 2021)	2231 / 1737 / 492	0	250.00	1647		38.79	↓27.01		492		78.84	+9.12

Table 6.6 KaratSaber unified Comparison with Previous Saber unified Implementations

[1] Calculated using Equivalent CLB slices (ECS)

[2] This work's KaratSaber768 PM 4

[3] This work's KaratSaber unified PM 4

The proposed KaratSaber architecture can be used as a co-processor in FPGA-based IoT processor (Kiat et al., 2020) in various IoT applications. The high TPS achieved by KaratSaber indicates that it is capable in handling high throughput KEM operations, with a moderate hardware consumption. This is an important feature when it is used in an IoT sensor node that needs to update the cloud server frequently, or it is used in a gateway device that needs to communicate with hundreds of sensor nodes in a timely manner. On the other hand, applications that has very stringent hardware area consumption may opt for extremely small architecture like (He et al., 2021), which greatly sacrifices the throughput performance.

6.3 Utilizing Research Output in IoT Applications

Application of cryptography in IoT covers a wide array of devices targeting different usage. For the work in this dissertation, the targeted implementation falls in the category of IoT applications that requires relatively high throughput, i.e., communications and database.

In the encryption techniques for secure communication analysis (Sanap and More, 2021), they compared various symmetric encryption algorithms for secure communication. To hinder misuse and alteration of sensitive information, an efficient cryptosystem with robust security is relatively important. Algorithms compared in this analysis (Sanap and More, 2021) includes DES, 3DES and AES, which are symmetric encryption schemes that are widely used in IoT applications. The symmetric keys need to be refreshed and encapsulated by the sensor nodes frequently before transmitting them to the cloud servers. The high throughput hardware architectures developed in this dissertation can be used to accelerate this process (KEM), which benefit many IoT applications that require timely response. Due to the area-time efficient design proposed in this dissertation, the area consumption is also reasonable for many IoT applications.

Furthermore, implementations such as database security (Zaw et. al., 2019), cloud storage (Deepthi et. al., 2021) and medical images encryption (Benssalah et. al., 2018) involves transmitting, computing and storing

immense quantity of sensitive data that requires robust security and high throughput. This research work can also be integrated into such applications inhibit attacks from advanced quantum computers.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this research work, a high-speed polynomial multiplier utilizing Karatsuba algorithm was developed for R-LWE and Saber schemes. As discussed in earlier chapters, when implementing the Karatsuba algorithm, the overhead (hardware consumption) increases proportional to the layers of Karatsuba. To avoid sacrificing too much hardware resulting in imbalance in the area-time aspect, the Karatsuba algorithm implemented in this research is never in a fully recursive fashion. However, due to the nature of the Karatsuba algorithm, it actually can be aid algorithms like SPMA to perform the polynomial multiplication in s shorter timeframe, in other words, increasing the overall throughput buy using a moderate amount of additional hardware. Hence, Karatsuba can be regarded as a catalyst algorithm to speed up the polynomial multiplication process, since it still relies on SPMA to complete the multiplication operation. Despite that, the throughput advantage of the Karatsuba algorithm far outweighs the hardware scarification.

In summary, this dissertation has provided answers to the following research challenges:

1) Karatsuba algorithm can be used to speed up the polynomial multiplication in LBC schemes, which is more advantageous compared to SPMA. It is able to achieve better area-time balance compared to pure SPMA implementations.

2) Promising PQC schemes like LWR that inhibits the use of NTT implementation (e.g., Saber) can be accelerated by utilizing the Karatsuba algorithm, enabling it to achieve higher throughput while maintaining a balance area-time aspect

3) Negacyclic operations for Karatsuba can be easily integrated into the Karatsuba post-processing stage, despite the number of implemented Karatsuba layers (recursiveness). This enables the efficiency of Karatsuba implementation for LBC schemes by omitting the additional cycles required for negacyclic operations.

4) While implementing Karatsuba, the challenge of polynomial *b* overflowing can be solved by constraining the layers of Karatsuba implemented, or on the other hand, perform separate

multiplications simultaneously. The latter increases the flexibility and scalability of the polynomial multiplier.

7.2 Future Work

The Karatsuba polynomial multiplier developed for both R-LWE and Saber scheme can also be implemented to cater for other schemes such as Scabbard, (Mera et al., 2021). Since the bottleneck for LBC scheme is polynomial multiplication, the shift-based multiplicator that has a high efficiency and low hardware consumption can also be migrated into different LBC schemes for future works. The shift-based multiplicator can also be customized to cater for additional numbers of multiplicands in different use cases, hence widening the possibilities for future implementations.

Furthermore, different layers of Karatsuba implementation can also be researched. Algorithms such as Toom-Cook algorithm is also a viable research direction where different layers and parameters can be tweaked to develop an efficient polynomial multiplier. Since the Toom-Cook algorithm is able to split the original polynomial into unsymmetrical sizes, this opens many possibilities and combinations for implementing the algorithm.

Besides, the Toom-Cook algorithm is also more suitable for asymptotical cases where it can efficiently compute large polynomial multiplication. In other words, the conquer and dividing techniques displayed in this research can be implemented on other PQC schemes that has larger parameters (requires higher security) using the Toom-Cook algorithm.

Lastly, the energy consumption of the polynomial multiplier can be further minimized for applications such as IoT devices using system level power management technique (Tan et al., 2021) is another promising direction.

LIST OF PUBLICATIONS

1. Wong, Z.-Y., Wong, D.C.-K., Lee, W.-K. and Mok, K.-M., 2021. High-Speed RLWE-Oriented Polynomial Multiplier Utilizing Karatsuba Algorithm. In: *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 6, pp. 2157-2161.

2. Wong, Z.-Y., Wong, D.C.-K., Lee, W.-K., Mok, K.-M., Yap, W.-S. and Khalid, A., 2021. KaratSaber: New Speed Records for Saber Polynomial Multiplication using Efficient Karatsuba FPGA Architecture. [Currently under review. Submitted to IEEE Transaction on Circuits and System I: Regular Papers]

BIBLIOGRAPHY

Agrawal, R., Bu, L., Ehret, A. and Kinsy, M., 2019. Open-Source FPGA Implementation of Post-Quantum Cryptographic Hardware Primitives. 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, pp.211–217.

Banerjee, A., Peikert, C. and Rosen, A., 2012. Pseudorandom functions and lattices, in *Advances in Cryptology*—EUROCRYPT, Pointcheval, D. and Johansson, T., Eds. Berlin, Germany: Springer, 2012, pp. 719–737.

Benssalah, M., Rhaskali, Y. and Azzaz, M. S., 2018. Medical Images Encryption Based on Elliptic Curve Cryptography and Chaos Theory, *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pp. 222-226

D'Anvers, J.P., Karmakar, A., Roy, S.S. and Vercauteren, F., 2018. *Saber: Module-LWR based key exchange, CPA-secure encryption and CCA secure KEM*, National Institute of Standards and Technology (NIST). [Online] Available: <https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/SABER-Round3.zip>.

Deepthi, B., Ramani, G., Deepika, R. and Shabbeer., M., 2021. Hybrid Secure Cloud Storage data based on improved Encryption Scheme, *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pp. 776-779.

Fan, S., Liu, W., Howe, J., Khalid, A. and O'Neill, M., 2018. "Lightweight Hardware Implementation of R-LWE Lattice-Based Cryptography," *in 2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2018, pp. 403-406

He, P., Lee, C.-Y. and Xie, J., 2021. Compact Coprocessor for KEM Saber: Novel Scalable Matrix Originated Processing, in *Third PQC Standardization Conference*. [Online] Available at: <https://csrc.nist.gov/CSRC/media/Events/third-pqc-standardization-conference/documents/accepted-papers/xie-compact-coprocessor-pqc2021.pdf>.

Karatsuba, A. and Ofman Y., "Multiplication of Many-Digital Numbers by Automatic Computers," in *Proceedings of the USSR Academy of Sciences*. 145: 293–294. Translation in the academic journal Physics-Doklady, 7 (1963), pp. 595–596, 1962

Kiat, W. P., Mok, K. M., Lee, W. K., Goh, H. G. and Achar, R., 2020. An energy efficient FPGA partial reconfiguration based micro-architectural technique for IoT applications, in *Microprocessors and Microsystems*, vol. 73, pp. 102966 – 102975.

Liu, D., Cong, Z. and Hui, L., 2017. Area-optimized Lattice-based cryptographic processor for constrained devices. Midwest Symposium on Circuits and Systems, 2017–August, pp.277–280.

Liu, D., Zhang, C., Lin, H., Chen, Y. and Zhang, M., 2018. A Resource-Efficient and Side-Channel Secure Hardware Implementation of Ring-LWE Cryptographic Processor. IEEE Transactions on Circuits and Systems I: Regular Papers, 66(4), pp.1474–1483.

Liu, W., Fan, S., Khalid, A., Rafferty, C., O'Neill, M., 2019. Optimized Schoolbook Polynomial Multiplication for Compact Lattice-Based Cryptography on FPGA, in *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp.2459–2463.

Mera, J. M. B., Karmakar, A., Kundu, S., Verbauwhede, I., 2021. Scabbard: a suite of efficient learning with rounding key-encapsulation mechanisms", IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(4), 474–509.

NIST, 2016, Post-Quantum Cryptography Standardization, [online] Available at: <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization>.

NIST, 2021. *Post-Quantum Cryptography Standardization: Round 3 Submissions*. [online] Available at: <https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-3-Submissions>.

Pöppelmann, T. and Güneysu, T., 2013. Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware. In: *Proc. Int. Conference of Selected Areas Cryptography*, pp. 68–85.

Regev, O. 2006. Lattice-Based Cryptography, C. Dwork (Ed.) CRYPTO 2006. LCNS, vol. 4117, pp. 131-141.

Renteria-Mejia, C.P. and Velasco-Medina, J., 2017. High-Throughput Ring-LWE Crypto-processors. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(8), pp.2332–2345.

Roy, S.S. and Basso, A., 2020. High-speed Instruction-set Coprocessor for Lattice-based Key Encapsulation Mechanism: Saber in Hardware, in *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4), 443-466.

Roy, S.S., Vercauteren, F., Mentens, N., Donglong, D. and Verbauwhede, I., 2014. Compact ring-LWE cryptoprocessor. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8731, pp.371–391.

Sanap. S. D. and More, V., 2021. Analysis of Encryption Techniques for Secure Communication," *2021 International Conference on Emerging Smart Computing and Informatics (ESCI),* pp. 290-294.

Shor, P.W. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, *SIAM J. Comput*., 26 (5), pp. 1484–1509.

Tan, B. L., Mok, K. M., Chang, J. J., Lee, W. K. and Hwang, S. O., 2021. RISC32- LP: Low-power FPGA-based IoT Sensor Nodes with Energy Reduction Program Analyzer, in IEEE Internet of Things Journal, in press.

UltraScale Architecture Configurable Logic Block User Guide. [online] Available at: <https://www.xilinx.com/support/documentation/user guides/ug574- ultrascale-clb.pdf>.

Wong, Z.-Y., Wong, D.C.-K, Lee, W.-K. and Mok, K.-M, 2020. High speed RLWE-Oriented Polynomial Multiplier Utilizing Karatsuba Algorithm, in *IEEE Transactions on circuits and Systems II: Express Briefs*, vol. 68, no. 6, pp. 2157-2161.

Zaw, T. M., Thant, M. and Bezzateev, S.V., 2019. Database Security with AES Encryption, Elliptic Curve Encryption and Signature, *2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF),* pp. 1-6

Zhang, Y., Wang, C., Kundi, D.E.S., Khalid, A., O'Neill, M. and Liu, W., 2020. An Efficient and Parallel R-LWE Cryptoprocessor, in *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 67, no. 5, pp. 886-890, 2020.

Zhu, Y., Zhu, M., Yang, B., Zhu, W., Deng, C., Chen, C., Wei, S. and Liu, L., 2021. LWRpro: An Energy-Efficient Configurable Crypto-Processor for Module-LWR, in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1146-1159.