ACCELERATING LATTICE-BASED CRYPTOSYSTEMS

by
KEMAL DERYA

Submitted to the Graduate School of Engineering
in partial fulfilment of
the requirements for the degree of Master of Sciences

Sabancı University
July 2022

ABSTRACT

ACCELERATING LATTICE-BASED CRYPTOSYSTEMS

KEMAL DERYA

ELECTRONICS ENGINEERING M.S. THESIS, JULY 2022

Thesis Supervisor: Prof. Erkay Savaş

Keywords: NTT, PQC, Polynomial Multiplication, Parametric, Hardware

Lattice-based cryptography has become important over the last couple of years since it gives resistance against quantum attacks that disable current security systems. The polynomial multiplication process is the most time-consuming operation in lattice-based cryptosystems. Number Theoretic Transform (NTT) facilitates efficient polynomial multiplication that is needed for key generation, encryption, and decryption operations. A design needs to offer configurability to work with different NTT parameters, as this would be an asset for developing different versions of the basic design for different cryptosystems. This thesis introduces a configurable design that can generate unified and parametric NTT-based polynomial multipliers. This design supports a broad range of parameters of lattice-bassed cryptosystems, specifically post-quantum cryptography (PQC) schemes. The *unified* butterfly unit composes the critical block of the design, and it can perform NTT and inverse NTT operations. Unique application areas need different performance goals, and this unit plays a critical role in accomplishing them. The design uses the number of butterfly units as input to achieve specific area and throughput demands and gives an optimized NTT-based polynomial multiplier hardware as output. For scheme parameters, the design offers run-time configurability. Additionally, it provides compile-time configurability for throughput and area demands. As far as we know, this design constitutes the *the first* NTT-based polynomial multiplier with *run-time* and *compile-time* configurability options. The advanced configurability options slightly affect the area and timing results, as indicated by the implementation results. This design has different sub-blocks, such as integer multiplier and reduction unit, and we present the design philosophy of each sub-block with the configurability and performance results.

ÖZET

LATTICE TABANLI KRIPTO-SISTEMLER HIZLANDIRICISI

KEMAL DERYA

ELEKTRONİK MÜHENDİSLİĞİ YÜKSEK LİSANS TEZİ, TEMMUZ 2022

Tez Danışmanı: Prof. Dr. Erkay Savaş

Anahtar Kelimeler: NTT, PQC, Polinom Çarpması, Parametrik, Donanım

Kafes-tabanlı şifreleme, güncel güvenlik sistemlerini etkisiz kılan kuantum saldırılarına karşı savunma sağladığı için, son yıllarda önemli hale geldi. Polinom çarpma işlemi kafes-tabanlı kriptosistemlerde en çok zaman harcanan operasyondur. Sayılar Teorisi Dönüşümü (NTT) anahtar üretimi, şifreleme ve şifre çözme operasyonları için gerekli olan polinom çarpma işlemini kolaylaştırmaktadır. Bir tasarım farklı NTT parametreleriyle çalışabilmesi için konfigurasyona ihtiyaç duymaktadır. Böylece eldeki basit dizayn değerli bir nitelik kazanarak farklı kriptosistemler için farklı bir şekilde geliştirilebilir. Bu tez konfigure edilebilir bir tasarım sunarak birleşik ve parametrik NTT-tabanlı polinom çarpıcısı üretmektedir. Bu tasarım kafes-tabanlı kriptosistemlerinin, özellikle kuantum-sonrası kriptosistemlerinin (PQC), geniş bir aralıkta parametrelerini desteklemektedir. Birleşik kelebek ünitesi tasarımın önemli bir bloğunu oluşturmaktadır ve NTT ve ters-NTT operasyonlarını gerçekleştirmektedir. Özgün uygulama alanları farklı performans hedeflerine ihtiyaç duymaktadır ve kelebek ünitesi bunun gerçekleşmesinde önemli bir rol oynamaktadır. Sunulan tasarım belirli alan ve çıktı ihtiyaçlarını karşılamak için kelebek ünitesinin sayısını girdi olarak kullanır ve optimize edilmiş NTT-tabanlı polinom çarpıcısını çıktı olarak verir. Şema parametreleri için dinamik olarak konfigurasyon ve alan ve çıktı ihtiyaçları için derleme anında konfigurasyon sunulmaktadır. Bildiğimiz kadarıyla, bu tasarım dinamik ve derleme anında konfigure edilebilir NTT-tabanlı ilk polinom çarpıcısıdır. İleri seviye konfigurasyon seçenekleri alan ve zaman sonuçlarını, uygulama sonuçlarında görüleceği üzere, çok az etkilemektedir. Bu dizayn tamsayı çarpıcısı ve indirgeme ünitesi gibi alt bloklara sahiptir ve her bir alt bloğun dizayn felsefesi konfigurasyon ve performans sonuçlarıyla sunulmaktadır.

ACKNOWLEDGEMENTS

I am very grateful to the very great people that I have encountered throughout my master's journey. To begin with, I would like to thank my ex-supervisor Dr. Erdinç Öztürk for his support and the experience that he shared with me throughout my master's studies. I am very thankful to him for the resources that he created. Without his support, I would not be able to complete my master studies. Lastly, I am very pleased for the opportunities that he gave for my career in the future.

I also want to thank my supervisor Dr. Erkay Savaş for his support during the last stage of my master studies. His knowledge and experience have become my guidance throughout my master studies. The opportunity of working with him always gave me special feelings.

I would like to thank the members of the jury committee of my master thesis defense, Dr. Ayhan Bozkurt and Dr. Sıddıka Berna Örs Yalçın for their valuable time and comments on my work.

I would like to thank my friends in the Crytography and Information Security Group (CISEC) at Sabanci University for their support throughout my studies. In particular, I am very grateful to Ahmet Can Mert for his great efforts for my studies.

I would like to thank my friends Can, Enes, Alişah, Şeyma and others for their valuable support. I enjoyed every single moment I spent with them throughout my studies.

I want to express my thanks to my family, especially my mother Kadriye and my father Ömer. I am certainly sure that I would not accomplish anything that I have done until today without their support. From the bottom of my heart, I want to thank my family for their unconditional support.

Lastly, I would like to thank Sabanci University and the Scientific and Technological Research Council of Turkey (TÜBİTAK) for the scholarship opportunities that they have provided. This work was supported by TÜBİTAK under Grant Number 118E725.

To Lina and Alya

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Current public cryptosystems have become vulnerable to quantum computers since the development of the quantum computers showed significant progress. Integer factorization and computing discrete logarithms are the hard problems for the current cryptosystems. Shor's algorithm has shown that it is possible to break public cryptosystems by solving these hard problems with a quantum computer (Shor, 1994).

Table 1.1 Security Levels of Cryptographic Schemes

Crypto Scheme	**Key Size**	**Security Levels (in bits)**	
		Classical Computing	**Quantum Computing**
RSA-1024	1024	80	0
RSA-2048	2048	112	0
ECC-256	256	128	0
ECC-384	384	256	0
AES-128	128	128	64
AES-256	256	256	128

Table 1.1 (Mavroeidis, Vishi, Zych & Jøsang, 2018) states the security levels of common cryptosystems against classical and quantum computing. RSA and ECC algorithms show no resistance against quantum attacks i.e., 0 bit security level. The security level of AES algorithms falls by half with quantum computing. It is inevitable to switch to quantum-resistant cryptosystems in the near future. The National Institute of Standards and Technology (NIST) recently initiated a standardization process for quantum-resistant key-encapsulation and digital signature schemes, which are often referred to as post-quantum cryptography.

Lattice-based cryptography has emerged as one of the most promising cryptographic constructions for post-quantum cryptography as it is based on a set of hard mathematical problems, which are conjectured to be resistant to attacks by quantum computers. Five out of the seven remaining candidates in the third round of the

Algorithm 1 Algorithm of Karatsuba Multiplication

Input: $\boldsymbol{A}(x), \boldsymbol{B}(x) \in \boldsymbol{R}_q$
Input: $2^a = n, a \in \boldsymbol{Z}_q$
Output: $\boldsymbol{A}(x) \cdot \boldsymbol{B}(x) \in \boldsymbol{R}_q$
1: $A = A_1 x^{n/2} + A_0, B = B_1 x^{n/2} + B_0$
2: $Z_0 = A_0 B_0$
3: $Z_1 = (A_0 A_1)(B_0 B_1)$
4: $Z_2 = A_1 B_1$
5: **return** $Z_2 x^2 + ((Z_1 - Z_0 - Z_2)x^{n/2} + Z_0$

NIST standardization process are lattice-based cryptographic schemes (Chen, Jordan, Liu, Moody, Peralta, Perlner & Smith-Tone, 2016).

Polynomial multiplication is the most time-consuming operation in lattice-based cryptosystems. Feasible implementations of lattice-based cryptosystems need to have efficient polynomial multipliers (Pöppelmann & Güneysu, 2014). There are methods for polynomial multiplication such as schoolbook multiplication, Karatsuba multiplication, and NTT-based multiplication.

The schoolbook multiplication method is a traditional way to multiply two polynomials. $A(x)$ and $B(x)$ are polynomials with degree of $n-1$.

$$\boldsymbol{A}(x) = \sum_{i=0}^{n-1} a_i \cdot x^i, \boldsymbol{B}(x) = \sum_{i=0}^{n-1} b_i \cdot x^i \tag{1.1}$$

The output $C(x) = A(x)B(x)$ can be calculated as

$$\boldsymbol{C}(x) = \sum_{i=0}^{n-1} x^i \cdot \sum_{j+k=i}^{n-1} a_j b_k \tag{1.2}$$

This method uses n^2 multiplication operations with $(n-1)^2$ additions. Thus, it has complexity of $O(n^2)$.

Karatsuba multiplication method offers divide-and-conquer algorithm for polynomial multiplication. Alg. 1 has complexity of $O(n^{\log_2 3})$ which results in better complexity than the schoolbook method.

Furthermore, the number-theoretic transform (NTT) is a popular method utilized in the lattice-based cryptosystems to reduce the complexity of multiplication in the polynomial rings,

$$\boldsymbol{R}_q = \mathbb{Z}_q[x]/\phi(m) \tag{1.3}$$

where the coefficient modulus q and the degree of the cyclotomic polynomial

$$\phi_m(x) = x^n + 1 \tag{1.4}$$

n, are referred to as the scheme parameters.

NTT reduces complexity to $O(n \log n)$ (Mert, Karabulut, Ozturk, Savas & Aysu, 2020). When the polynomial multiplication in $\boldsymbol{R}_q$ is performed using NTT, the efficiency of the NTT operation determines the performance of the cryptosystem to a great extent. Since NTT method offers the least complexity, we opt to use it for polynomial multiplication operation.

Different application areas have different performance criteria to be fulfilled. IoT implementations need area-optimized design since they are implemented on resource-poor microdevices and rely less on throughput. On the other hand, cloud centers require high throughput rates with strict timing needs, and they need to be optimized performance-wise (Nejatollahi, Dutt, Ray, Regazzoni, Banerjee & Cammarota, 2019).

While it stays a vital goal to design polynomial multipliers efficiently for different applications, it is also crucial to have a configurable design to yield to the power, area, and throughput requirements. The configurability feature of these design parameters is specified as *compile-time configurability* (CTC).

Different PQC schemes operate using different scheme parameters, and it is possible to use the same hardware for different schemes without recompiling it. This design aspect is referred to as *run-time configurability* (RTC). Furthermore, the PQC schemes have different levels of security by using different scheme parameters. By *run-time configurability* for different scheme parameters, it is possible to adjust dynamically to different PQC schemes and the security level of the same scheme. An NTT multiplier, which can work with various n and q values that appeared in various PQC schemes without recompilation, is referred to as RTC.

The core processing unit in an NTT multiplier is the butterfly circuit, which can take different forms such as Cooley-Tukey (CT) and Gentleman and Sande (GS) designs (Chu & George, 1999). An NTT multiplier that features a butterfly unit

(BU), which implements both CT and GS is referred to as a unified multiplier. It enables the implementation of NTT and inverse NTT (INTT) in the same circuit efficiently. An NTT-based multiplier that can be recompiled with a different number of BUs for throughput or time-area efficiency is referred to as CTC.

Table 1.2 NTT-friendly Lattice-based Post-Quantum Cryptosystems

Cryptosystem	n	$\log_2(q)$	q	**Operation**	**NIST Round**	**Support**
CRYSTALS-Kyber (v1)	256	13	7681	KEM	2,3	✓
CRYSTALS-Kyber (v2)	256	12	3329	KEM	2,3	✓
NewHope-512	512	14	12289	KEM	2	✓
NewHope-1024	1024	14	12289	KEM	2	✓
CRYSTALS-Dilithium	256	23	8380417	DS	2,3	✓
Falcon-I	512	14	12289	DS	2,3	✓
Falcon-II	1024	14	12289	DS	2,3	✓
qTESLA-q-I	1024	29	343576577	DS	2	✓
qTESLA-q-III	2048	30	856145921	DS	2	–

KEM: Key Encapsulation **DS**: Digital Signature

Table 1.2 shows the PQC schemes that support NTT-based polynomial multiplication method. They operate on different scheme parameters for different operations. Thus, they offer different security levels. In our work, we offer support for all PQC schemes except qTESLA-q-III scheme.

Our motivation can be summarized as follows. We want to optimize the polynomial multiplication operation which is the drawback of lattice-based cryptosystems. We opt to use NTT-based multiplication method since it offers the best complexity. Additionally, we want to support as many as scheme parameters so that different PQC schemes can be used on the same hardware i.e., RTC. Moreover, we want to design a hardware that is suitable for different application areas that have different performance requirements i.e., CTC.

1.1 Related Works

Lattice-based cryptosystems for different platforms require diverse area, power, and timing specifications. To meet these requirements, various implementations of NTT and NTT-based polynomial multiplication operations have been developed. The works in (Alkim, Evkan, Lahr, Niederhagen & Petri, 2020; Fritzmann, Sharif, Müller-Gritschneder, Reinbrecht, Schlichtmann & Sepulveda, 2019; Fritzmann, Sigl

& Sepúlveda, 2020; Seiler, 2018) offer software implementations that support run-time configurability; therefore, they can operate on different lattice-based algorithms.

Seiler (Seiler, 2018) enhanced the NTT operation on Intel processors with AVX2 instruction set while others (Alkim et al., 2020; Fritzmann et al., 2019; Fritzmann et al., 2020) focused on hardware/software co-design on RISC-V architecture with instruction set extensions. CPUs offer a limited number of computing units that limit the level of parallelization. As a result, CPUs cannot offer more performance for timing requirements since NTT involves multiply-accumulate operations that can be parallelizable; therefore, hardware implementations that can offer more parallelization have gained more importance.

Table 1.3 Configurability of Works in the Literature

Work	**Platform**	n	q	**BU**
(Yaman et al., 2021)	Virtex-7	Fixed 256	Fixed 12-bit	Fixed
(Xing & Li, 2021)	Artix-7	Fixed 256	Fixed 12-bit	Fixed
(Mert et al., 2019)	Virtex-7	Fixed 1024	Constant 32-bit	Fixed
(Fritzmann et al., 2020)	RISC-V	RTC up to 1024	RTC up to 32-bit	Fixed
(Fritzmann & Sepúlveda, 2019)	65 nm	RTC up to 1024	RTC up to 16-bit	Fixed
(Mert et al., 2020)	Virtex-7	RTC up to 4096	RTC up to 32-bit	Fixed
(Banerjee et al., 2019)	40 nm	RTC up to 2048	RTC up to 24-bit	Fixed
(Fritzmann et al., 2021)	Artix-7	RTC up to 4096	RTC up to 39-bit	Fixed
(Roy et al., 2014)	Virtex-6	CTC up to 512	CTC up to 13-bit	Fixed
(Wang et al., 2020)	Artix-7	CTC up to 2048	CTC up to 30-bit	Fixed
(Mert et al., 2020)	Virtex-7	CTC up to 4096	CTC up to 60-bit	CTC
(Tan et al., 2020)	32 nm	CTC up to 4096	CTC up to 71-bit	CTC
This Work	Virtex-7	RTC 256 to 1024	RTC 12-bit to 30-bit	CTC

Those in (Xing & Li, 2021; Yaman et al., 2021) offer designs specifically for Ky-

ber (v2) schemes. They have fixed scheme parameters with fixed number of BU. Therefore, they do not offer RTC or CTC.

The design in (Mert et al., 2019) offers configurability option for q parameter that is only suitable for 32-bits prime q values. It has fixed number of BU. Thus, it does not offer CTC.

The work in (Fritzmann et al., 2020) offers RISC-V extensions for NTT operations that support different scheme parameters. Even though it offers RTC for scheme parameters, it is not possible to configure for area and throughput at the compile-time.

The architecture in (Fritzmann & Sepúlveda, 2019) is impelemented on 65-nm platform. It supports different scheme parameters. Thus, it has RTC. On the other hand, it does not have CTC for area and throughput.

The design in (Mert et al., 2020) offers NTT accelerator for different scheme parameters. As a result, it does have RTC. However, it is not possible to tune it into different area requirements i.e., no CTC suppport.

A crypto-processor is implemented in (Banerjee et al., 2019) on 40-nm platform. This work supports different scheme parameters with RTC. However, it does not offer CTC for area and throughput requirements.

The work in (Fritzmann et al., 2021) offers instruction set extensions for PQC. In this work, NTT hardware supports different scheme parameters, therefore it has RTC. On the other hand, it is not possible to adjust the throughput rate.

The crypto-processor in (Roy et al., 2014) offers CTC for scheme parameters that only supports specific set of parameter after the compilation. The throughput rate of this architecture is fixed. Thus, it does not have CTC.

The work in (Wang et al., 2020) proposed hardware architecture for Tesla PQC scheme with support for multiple scheme parameters. Nevertheless, this design does not provide *RTC* since it only offers configurability for scheme parameters at the compile-time. Additionally, it does not adjust the throughput rate i.e., no CTC for area requirements.

Moreover, HLS implementations (Mert et al., 2020; Nguyen, Dang & Gaj, 2019,2) for configurable NTT architectures have room for optimizations. These designs can be optimized for design parameters by using RTL since it is not possible to have the most optimized design by using HLS.

We offer a unique design to maximize the optimization opportunities. This de-

sign offers *run-time*(RTC) and *compile-time*(CTC) configurability options for NTT-friendly PQC schemes that used in NTT-based polynomial multiplier hardware. The proposed desing offers CTC for throughput and area (i.e., the number of computing units), and RTC for scheme parameters(n and q).

The current designs in the literature are shown in Table 1.3 in terms of configurability options. If the stated work only operates on a single parameter, the aforementioned parameter is referred to as a fixed parameter. If the stated work only operates within a constant range, i.e., constant $\lceil \log_2(q) \rceil$ bit-size, the parameter is referred to as a constant parameter. The table contains only two designs that provide configurability at the compile-time for the area and performance (Mert et al., 2020; Tan et al., 2020).

The architecture in (Tan et al., 2020) supports a wide range of scheme parameters for homomorphic encryption. On the other hand, the work in (Mert et al., 2020) is only capable of the NTT operation. Moreover, the work in (Riazi, Laine, Pelton & Dai, 2020) supports scheme parameters ,q from 109 to 438 bits and n from 4096 to 32768, for homomorphic encryption and is not listed in Table 1.3. Our work targets lattice-based PQC schemes with small parameters and optimized designs. Therefore, the work in (Riazi et al., 2020) is not comparable with our work. Lastly, the architectures in (Mert et al., 2020; Riazi et al., 2020; Tan et al., 2020) do not provide run-time configurability for scheme parameters.

1.2 Our Contribution

The proposed design introduces a configurable polynomial multiplier that is capable of performing NTT, INTT, and NTT-based polynomial multiplication operations. The architecture specifically utilized for NTT-friendly PQC schemes including CRYSTALS-KYBER (Kyber) with old and new versions (Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler & Stehlé, 2017; Lyubashevsky & Seiler, 2019), NewHope-512/1024 (Alkim, Ducas, Pöppelmann & Schwabe, 2016), CRYSTALS-DILITHIUM (Dilithium) (Ducas, Kiltz, Lepoint, Lyubashevsky, Schwabe, Seiler & Stehlé, 2018), Falcon-I/II (Fouque, Hoffstein, Kirchner, Lyubashevsky, Pornin, Prest, Ricosset, Seiler, Whyte & Zhang, 2018), and qTESLA-q-I (Alkim, Barreto, Bindel, Longa & Ricardini, 2019).

Moreover, the proposed architecture can also perform operations for other NTT-friendly lattice-based cryptosystems with coefficients up to 30 bits and ring degree

between 256 and 1024. Thus, In SABER scheme (D'Anvers, Karmakar, Roy & Vercauteren, 2018), NTT operations can be utilized with three prime moduli as shown by Fritzmann *et al.* (Fritzmann et al., 2020). SABER PQC scheme can be performed on the proposed design without any modification. Our architecture offers cryptographic flexibility that can be beneficial for many other algorithms. It is possible to utilize some PQC scheme with different settings on this proposed architecture even though they did not make it to the final round of NIST's contest for standardization (i.e., NewHope) or they have been modified (i.e., Kyber (v1)).

Moreover, it is possible to scale this architecture to support other lattice-based applications such as homomorphic encryption. Ring degree, n, can be utilized to other applications rather than proposed PQC schemes to NIST's process. On this work, we only introduce the implementations that only suitable for PQC schemes since the homomorphic encryption applications go beyond the scope of this work.

Hardware implementations of lattice-based cryptosystems can be generated by using the intermediate blocks that generated by the proposed architecture. Thus, we deeply analyze each block and give a detailed summary of their functionalities.

We list our contribution in this work as follows:

- We propose an optimized FPGA implementation of configurable architecture of NTT-based polynomial multiplier for lattice-based cryptosystems.
- We introduce a modular multiplier block that utilizes *run-time configurable word-level Montgomery multiplication* technique. This unit can perform operations on coefficients that up to 30-bits. While this unit utilizes incomplete arithmetic (Yanik, Savas & Koc, 2002), it should be also noted that word-level Montgomery algorithm is utilized to decrease the complexity of NTT operation with NTT-friendly primes.
- We introduce the essential processing element in our design as a *unified butterfly unit*. This unit is capable of performing CT and GS butterfly operations. To perform NTT, INTT, and NTT-based polynomial multiplication operations, this unit can be configured at *run-time*.
- It is possible to configure the number of butterfly units at the compile-time to tune the throughput level of the proposed architecture. Moreover, the occupied area of the design can be configured. Thus, this architecture can be applied to different applications that operate on different circumstances by using CTC feature.
- We propose a *configurable memory control unit* that can read coefficients from

BRAMs of FPGA devices. Moreover, it can also write cofficients to BRAMs and this unit assures that the memory units operate with the processing elements properly. It is possible to change the state of the architecture among NTT, INTT, and NTT-based polynomial multiplicaton operations since this unit controls the state of the overall design. This unit can be utilized to perform with different number of butterfly units on different polynomials with degree between 256 and 1024.

- We present a code sample that includes the proposed design on the Github repository (`https://github.com/kemalderya/pqc-param-ntt`). It should be noted that hardware-based NTT designs may not be found as open-source designs. To help the future developers in this area, this open-source hardware can be useful. Furthermore, the proposed architecture can be integrated into other researchers design by adjusting a single parameter to fix tje performance.

The sections of this work is explained as follows. In Section II, the background information is introduced. Section III proposes the hardware architecture. Section IV introduces the implementation results with comparison the prior works. Section V concludes the paper.

2. BACKGROUND

This chapter introduces the notation used in this work and the background information about lattice-based cryptography and NTT-based polynomial multiplication.

2.1 Notation

The ring $\mathbb{Z}_q$ uses q as the modulus and modular addition and modular multiplication are defined on the ring with integers in $[0, q)$. The unique irreducible polynomial is represented by $\phi_m(x)$ which is the cyclotomic polynomial. $\phi_m(x) = x^n + 1$ forms the cyclotomic polynomial where n is a power of two. The polynomial ring reduced with $\phi_m(x)$ over $\mathbb{Z}_q$ is represented by $\boldsymbol{R}_q = \mathbb{Z}_q[x]/\phi_m(x)$. The coefficients in $\mathbb{Z}_q$ of the polynomials are reduced with $\phi_m(x)$. The degree of the ring is referred to as n in our terminology.

The lowercase letters (i.e., a) represent the integers and the boldface lowercase letters (i.e., $\boldsymbol{a}$ or $\boldsymbol{a}(x)$) represent the polynomials. The polynomial $\boldsymbol{a}$ in the NTT domain is represented by $\overline{\boldsymbol{a}}$ (i.e., $\overline{\boldsymbol{a}} = \mathrm{NTT}(a)$). Let $\boldsymbol{a} \leftarrow \boldsymbol{R}_q$ and $\boldsymbol{a} \leftarrow \boldsymbol{D}_{\mu,\sigma}$ represent polynomial $\boldsymbol{a}$ is uniformly sampled from $\boldsymbol{R}_q$ and the distribution $\boldsymbol{D}_{\mu,\sigma}$ with mean μ and std. dev. σ. Finally, $\cdot$ represents the integer multiplication. $\times$ represents the polynomail multiplication. The NTT-domain multiplication operation is represented by $\odot$.

2.2 Lattice-based Cryptography

The security of current cryptosystems is based on the hardness of the underlying mathematical problems such as integer factorization problem. Current computer systems are not capable of solving these problems in polynomial time. However, it is shown that quantum computers can break some of the current cryptosystems by solving these problems in polynomial-time by Shor's algorithm (Chen et al., 2016).

In the recent years, the progress in the quantum computers has increased and the interest in PQC has risen on both academical and industrial ends. Thus, cryptographic systems such as lattice-based cryptography (Alkim et al., 2019,1; Ducas et al., 2018) and code-based cryptography (McEliece, 1978) are among PQC schemes that proposed to fit for this purpose. Lattice-based cryptographic schemes are said to be more useful, straightforward, and parallelizable than other schemes (Chen et al., 2016). They can provide security even under worst-case occasions.

The Learning with Errors (LWE) problem underlies the most of the lattice-based cryptosystems. By using a sequence of random linear equations on s, the LWE problem aims to find a secret $s \in \boldsymbol{Z}_q^n$ (Regev, 2010).

$$
\begin{aligned}
14\boldsymbol{s_1} + 15\boldsymbol{s_2} + 5\boldsymbol{s_3} + 2\boldsymbol{s_4} &\approx 8 \pmod{17} \\
13\boldsymbol{s_1} + 14\boldsymbol{s_2} + 14\boldsymbol{s_3} + 6\boldsymbol{s_4} &\approx 16 \pmod{17} \\
6\boldsymbol{s_1} + 10\boldsymbol{s_2} + 13\boldsymbol{s_3} + 1\boldsymbol{s_4} &\approx 3 \pmod{17} \\
&\vdots \\
6\boldsymbol{s_1} + 7\boldsymbol{s_2} + 16\boldsymbol{s_3} + 2\boldsymbol{s_4} &\approx 3 \pmod{17}
\end{aligned}
\tag{2.1}
$$

These equations can be solved with the Gaussian elimination algorithm if there is no error additive introduced. When there is an error additive, the Gaussian elimination algorithm amplifies the error so it is not possible to solve the equations. As a result the equations become significantly difficult to solve with the error additive (Regev, 2010).

It is possible to reduce the LWE problem into easier variants. LWE problem needs to have larger key size because it needs n vectors $\boldsymbol{Z}_q^n$. If we use n samples from $\boldsymbol{Z}_q^n$, we can have $O(n)$ elements from $\boldsymbol{Z}_q^n$, instead of $O(n)^2$. The ring, $\mathbb{Z}_q[x]/\phi(m)$ can be used instead of $\boldsymbol{Z}_q^n$ which allows us to perform fast Fourier transform on the vectors. Thus, the problem now requires smaller key with faster computation (Regev, 2010).

The equation

$$\boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{s} + \boldsymbol{e} \pmod{q} \tag{2.2}$$

formulates the R-LWE problem. $\boldsymbol{a}$, $\boldsymbol{b} \in \boldsymbol{R}_q$ form the public parameters. The secret key is referred to as $\boldsymbol{s} \in \boldsymbol{R}_q$. The error polynomial is formulated by $\boldsymbol{e} \leftarrow \boldsymbol{D}_{0,\sigma}$ in $\boldsymbol{R}_q$ where zero mean and (a small) σ standard deviation is used for normal distribution of the coefficients.

Ring-LWE (R-LWE) used in NewHope (Alkim et al., 2016) and Module-LWE (M-LWE) used in Kyber (Bos et al., 2017; Lyubashevsky & Seiler, 2019) are variants of LWE problem that offer superior performance.

The search problem and the decision problem are two hard problems of R-LWE. The search problem uses the given pair $(\boldsymbol{a}, \boldsymbol{b})$ to find the value $\boldsymbol{s}$. The decision problem aims to differentiate between the pair $(\boldsymbol{a}, \boldsymbol{b})$ and a random pair sampled from a uniform distribution over $\boldsymbol{R}_q$. On the other hand, M-LWE problem utilizes matrices of ring elements (i.e., polynomials in $\boldsymbol{R}_q$) rather than the ring elements of R-LWE (D'Anvers et al., 2018).

2.3 NTT-based Polynomial Multiplication

Discrete Fourier transform can be utilized on integers defined over $\boldsymbol{R}_q = \mathbb{Z}_q[x]/\phi_m(x)$. This approach makes polynomial multiplication operation faster since NTT simplifies fast convolutions over polynomials. NTT of a $n-1$ degree

$$\boldsymbol{a}(x) = \sum_{i=0}^{n-1} a_i \cdot x^i \tag{2.3}$$

polynomial defined over $\boldsymbol{R}_q$ can be represented by a $n-1$ degree polynomial

$$\overline{\boldsymbol{a}}(x) = \sum_{i=0}^{n-1} \overline{a}_i \cdot x^i \tag{2.4}$$

defined over $\boldsymbol{R}_q$ in NTT domain. Eqn. 2.5 is used for calculating the coefficients $\overline{a}_i$.

$$\overline{a}_i = \sum_{j=0}^{n-1} a_j \cdot \omega^{i \cdot j} \pmod{q} \text{ for } i = 0, 1, ..., n-1 \tag{2.5}$$

Eqn. 2.5 computes the coefficients in the NTT domain, $\overline{a}_i$, by using a_i, the coefficients in the polynomial domain with a degree of n. Thus, this computation is occasionally referred to as n-point NTT.

The twiddle factor of NTT operation, a primitive n-th root of unity constant, needs to satisfy the conditions $\omega^n \equiv 1 \pmod{q}$, $\omega^i \neq 1 \pmod{q}$ $\forall i < n$ and $q \equiv 1 \pmod{n}$. In inverse NTT (INTT) operation, $\omega^{-1} \in \mathbb{Z}_q$ is used instead of ω. Moreover, the coefficients after the last step of INTT operation need to be multiplied by n^{-1} in $\mathbb{Z}_q$.

There are different algorithms for NTT operations namely, iterative NTT algorithm, Cooley-Tukey (CT) butterfly based NTT, and the unified forward NTT algorithm.

The iterative NTT algorithm operates on subsets of pairs of coefficients and iteratively combines them. This method can be used to process the subsets in sequential indices. Alg. 2 shows the iterative NTT algorithm. NTT operation consists of 4 parts where first index values are calculated. Then, the corresponding memory addresses are read. Later, the butterfly operations are performed. Lastly, the coefficients are written into the memory.

Moreover, Alg. 3 shows the NTT algorithm based on CT butterfly. This algorithm takes the input polynomial in natural order and it produces the output polynomial in the bit-reversed order. It also takes the twiddle factor in the bit-reversed order. In order to perform polynomial multiplication operation by using this algorithm, GS butterfly needs to be used on the output polynomial that is in bit-reversed order. As a result, two separate butterfly configurations needs to be implemented which increases the occupied hardware of the architecture.

The last method for NTT operation is called the unified forward NTT algorithm as shown in Alg. 6. This method eliminates the need for zero-padding and preprocessing operation.

In regular NTT algorithms, the coefficients of the input polynomials need to be padded with 0 to perform NTT-based polynomial multiplication operation for any random n values. At the last step of the multiplication, the resulting polynomial needs to be reduced to the degree of $n-1$.

Algorithm 2 Iterative NTT Algorithm (Longa & Naehrig, 2016)

Require: $\boldsymbol{A}(x) \in \boldsymbol{Z}_q/(x^n+1)$
Require: primitive n-th root of unity $\omega \in \boldsymbol{Z}_q, n = 2^l$
Ensure: $\overline{\boldsymbol{A}}(x) = \boldsymbol{NTT}(A) \in \boldsymbol{Z}_q/(x^n+1$

```
for i = 1; i = l; i++ do
    m = 2^{l-i}
    for (j = 0; j < (2^{i-1} - 1); j++) do
        for k = 1; k = m - 1; k++ do
            i_e = 2 · j · m + k
            i_o = 2 · j · m + k + m
            i_w = 2^{i-1} · k
            U ← A[i_e]
            V ← A[i_o]
            W ← ω^{i_w} (mod q)
            E ← (U + V) (mod q)
            O ← (U − V) · W (mod q)
            A[i_e] ← E
            A[i_o] ← O
        end for
    end for
end for
return Ā(x)
```

A technique called *negative wrapped convolution* can be used to eliminate the padding and reduction operation. $\phi_m(x)$ must be in the form of x^n+1 and the input polynomial needs to be multiplied with the powers of ψ which called pre-processing operation.

The constant ψ which is a primitive $2n$-th root of unity should satisfy the conditions $\psi^{2n} \equiv 1 \pmod q$, $\psi^i \neq 1 \pmod q$ $\forall i < 2n$ and $q \equiv 1 \pmod{2n}$. CT butterfly design can be utilized to merge the pre-processing operation with NTT operation. Roy *et al.* introduced *merged* forward NTT operation (Roy et al., 2014) to perform this technique.

Furthermore, INTT operation can be performed in two ways; INTT based GS butterfly algorithm and the unified inverse INTT algorithm. INTT based GS butterfly algorithm, shown in Alg. 4, takes the input polynomial in bit-reversed order and it produces the output polynomial in natural order. It also takes the twiddle factor in the bit-reversed order.

To eliminate the reduction operation at the end of Alg. 4, the output polynomial needs to be multiplied with the powers of ψ^{-1} which called post-processing operation. A related method is proposed by Pöppelmann *et al.* to merge post-processing operations with INTT for *merged* inverse NTT operation (Pöppelmann, Oder & Güneysu, 2015). Alg. 7 shows the unified inverse NTT algorithm to implement this

Algorithm 3 NTT Algorithm based on CT butterfly (Longa & Naehrig, 2016)

Require: $\boldsymbol{a}(x) \in \boldsymbol{Z}_q^n$, in natural order
Require: $q \equiv \pmod{2n}, n = 2^l$
Require: a precomputed table $\psi_{rev} \in \boldsymbol{Z}_q^n$ powers of ψ in bit-reversed order
Ensure: $\boldsymbol{a} \leftarrow \boldsymbol{NTT}(a)$, in bit-reversed order

```
1: t = n
2: for m = 1; m < n; m = 2m do
3:     t = t/2
4:     for (i = 0; i < m; i++) do
5:         j1 = 2 · i · t
6:         j2 = j1 + t − 1
7:         S = ψ_rev[m + i]
8:         for (j = j1; j ≤ j2; j++) do
9:             U = a[j]
10:            V = a[j + t] · S
11:            a[j] = U + V (mod q)
12:            a[j + t] = U − V (mod q)
13:        end for
14:    end for
15: end for
16: return a
```

method.

It is possible to combine these technique by using two different butterfly structures. Thus, the computational complexity is decreased by eliminating the pre-processing and post-processing operations. This approach is utilized in this work. NTT-based polynomial multiplication operation is performed as shown in Eqn. 2.6. n-point merged NTT and INTT operations are represented by $\texttt{NTT}_n$ and $\texttt{INTT}_n$.

$$\boldsymbol{c} = \texttt{INTT}_n((\texttt{NTT}_n(\boldsymbol{a}) \odot \texttt{NTT}_n(\boldsymbol{b}))) \tag{2.6}$$

It is possible to utilize both CT and GS butterfly formations in a unified design with the minimum amount of hardware (e.g., area and latency) as it will be discussed in the following chapters.

NTT-based polynomial multiplication operation over $\boldsymbol{R}_q$ can be performed with $q \equiv 1 \pmod{n}$ instead of $q \equiv 1 \pmod{2n}$ (Lyubashevsky & Seiler, 2019). This method eliminates the pre-processing and post-processing operations as proposed by Lyubashevsky *et al.*.

The Kyber scheme is utilized this method. Thus, it is optimized by reducing one bit from coefficients for the integer computations (i.e., q reduced from 7681 to 3329). Therefore, the NTT/INTT operations are updated (Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler & Stehlé, 2018). Kyber (v1) and Kyber (v2) are referred to as the Kyber scheme with old and new parameters. Kyber (v2)

Algorithm 4 INTT Algorithm based on GS butterfly (Longa & Naehrig, 2016)

Require: $\boldsymbol{a}(x) \in \boldsymbol{Z}_q^n$, in bit-reversed order
Require: $q \equiv \pmod{2n}, n = 2^l$
Require: a precomputed table $\psi_{rev}^{-1} \in \boldsymbol{Z}_q^n$ powers of ψ^{-1} in bit-reversed order
Ensure: $\boldsymbol{a} \leftarrow \boldsymbol{INTT}(a)$, in natural order

1: $t = 1$
2: **for** $m = n;\ m > 1;\ m = m/2$ **do**
3: $\quad j_1 = 0$
4: $\quad h = m/2$
5: $\quad$ **for** $(i = 0;\ i < h;\ i{+}{+})$ **do**
6: $\quad\quad j_2 = j_1 + t - 1$
7: $\quad\quad S = \psi_{rev}^{-1}[h + i]$
8: $\quad\quad$ **for** $(j = j_1;\ j \leq j_2;\ j{+}{+})$ **do**
9: $\quad\quad\quad U = a[j]$
10: $\quad\quad\quad V = a[j + t]$
11: $\quad\quad\quad a[j] = U + V \pmod{q}$
12: $\quad\quad\quad a[j + t] = (U - V) \cdot S \pmod{q}$
13: $\quad\quad$ **end for**
14: $\quad\quad j_1 = j_1 + 2t$
15: $\quad$ **end for**
16: $\quad t = 2t$
17: **end for**
18: **for** $j = 0;\ j < n;\ j{+}{+}$ **do**
19: $\quad a[j] = a[j] \cdot n^{-1} \pmod{q}$
20: **end for**
21: **return** a

scheme differs from Kyber (v1) in the NTT-based multiplication operation. Kyber (v1) uses 256 degree-0 polynomials for the NTT opeation. It needs 256 modular multiplication operation for the coefficient-wise multiplication operation.

On the other hand, Kyber (v2) utilizes NTT operation with 128 degree-1 polynomials. Algorithm 5 shows the coefficient-wise multiplication operation in the NTT domain with 128 polynomial multiplications over $\mathbb{Z}_q[x]/(x^2 - \omega^i)$ where i depends on the position of coefficients.

Algorithm 6 and Algorithm 7 show the unified merged NTT and merged INTT algorithms for NTT-friendly schemes. The bit-reversel operation on b-bit integer a is represented by $\texttt{br}(a, b)$. NTT/INTT operation is chosen by fd where 2 is used for Kyber (v2) and 1 for other schemes.

Algorithm 5 Algorithm of Multiplication in the NTT domain for Kyber (v2) (Lyubashevsky & Seiler, 2019)

Input: $\overline{\boldsymbol{a}}(x), \overline{\boldsymbol{b}}(x) \in \boldsymbol{R}_q$ in bit-reversed order
Input: $\omega \in \mathbb{Z}_q$
Output: $\overline{\boldsymbol{c}}(x) \in \boldsymbol{R}_q$ in bit-reversed order
1: **for** $(i = 0;\ i < 128;\ i++)$ **do**
2: $\quad a_0, a_1, b_0, b_1 \leftarrow \overline{\boldsymbol{a}}[2i], \overline{\boldsymbol{a}}[2i+1], \overline{\boldsymbol{b}}[2i], \overline{\boldsymbol{b}}[2i+1]$
3: $\quad \overline{\boldsymbol{c}}[2i] \leftarrow (a_0 \cdot b_1 + a_1 \cdot b_0) \bmod q$
4: $\quad \overline{\boldsymbol{c}}[2i+1] \leftarrow (a_1 \cdot b_1 \cdot \omega^{\mathtt{br}(i,7)+1} + a_0 \cdot b_0) \bmod q$
5: **end for**
6: **return** $\overline{\boldsymbol{c}}$

Algorithm 6 Unified Forward NTT Algorithm

Require: $\boldsymbol{a}(x) \in \boldsymbol{R}_q$, in natural order
Require: $n, q, \omega \in \mathbb{Z}_q$ (or $\psi \in \mathbb{Z}_q$)
Require: $fd \in \{1, 2\}$ (final degree)
Ensure: $\overline{\boldsymbol{a}}(x) \in \boldsymbol{R}_q$, in bit-reversed order
1: $k, l, v = 1, (n/2), \log_2(n/fd)$
2: **while** $(l \geq fd)$ **do**
3: $\quad$ **for** $(s = 0;\ s < n;\ s = j + l)$ **do**
4: $\quad\quad w, k = \omega^{\mathtt{br}(k,v)} \pmod q, k+1$
5: $\quad\quad$ **for** $(j = s;\ j < (s+l);\ j++)$ **do**
6: $\quad\quad\quad t = \boldsymbol{a}[j+l] \cdot w \pmod q$
7: $\quad\quad\quad \boldsymbol{a}[j+l] = \boldsymbol{a}[j] - t \pmod q$
8: $\quad\quad\quad \boldsymbol{a}[j] = \boldsymbol{a}[j] + t \pmod q$
9: $\quad\quad$ **end for**
10: $\quad$ **end for**
11: $\quad l = l/2$
12: **end while**
13: **return** $\boldsymbol{a}$

Algorithm 7 Unified Inverse NTT Algorithm

Require: $\overline{\boldsymbol{a}}(x) \in \boldsymbol{R}_q$, in bit-reversed order
Require: $n, q, \omega^{-1} \in \mathbb{Z}_q$ (or $\psi^{-1} \in \mathbb{Z}_q$)
Require: $fd \in \{1, 2\}$ (final degree)
Ensure: $\boldsymbol{a}(x) \in \boldsymbol{R}_q$, in natural order

$k, l, v = 0, fd, \log_2(n/fd)$
while $(l \geq (n/2))$ **do**
    for $(s = 0;\ s < n;\ s = j + l)$ **do**
        $w, k = \omega^{\mathtt{br}(k,v)+1} \pmod{q}, k+1$
        for $(j = s;\ j < (s+l);\ j++)$ **do**
            $\overline{\boldsymbol{a}}[j+l] = \overline{\boldsymbol{a}}[j] + \overline{\boldsymbol{a}}[j+l] \pmod{q}$
            $\overline{\boldsymbol{a}}[j] = \overline{\boldsymbol{a}}[j] - \overline{\boldsymbol{a}}[j+l] \pmod{q}$
            $\overline{\boldsymbol{a}}[j+l] = \overline{\boldsymbol{a}}[j+l] \cdot w \pmod{q}$
        end for
    end for
    $l = 2 \cdot l$
end while
for $(i = 0;\ i < n;\ i++)$ **do**
    $\overline{\boldsymbol{a}}[i] = \overline{\boldsymbol{a}}[i] \cdot n^{-1} \pmod{q}$
end for
return $\overline{\boldsymbol{a}}$

3. OUR WORK

In this chapter, we start explaining the polynomial multiplier in details by starting from the word-level Montgomery modular multiplier that utilizes incomplete arithmetic. In hierarchical order, we introduce our core unit, unified butterfly unit, which implements both the CT and GS butterfly operations. Lastly, we introduce the configurable memory control unit with the overall design.

3.1 Word-level Montgomery Modular Multiplier Unit

In this section, we present our modular multiplier unit in details. In our design, the modular multiplier unit utilizes two units: (i) a 32-bit integer multiplier (Fig. 3.1) and (ii) a word-level Montgomery modular reduction unit (Fig. 3.2) that provides configurability for scheme parameters on NTT-friendly primes.

The word-level Montgomery technique proposed in (Mert et al., 2019) and the incomplete arithmetic technique utilized in (Yanik et al., 2002) are used in the run-time configurable word-level Montgomery modular reduction unit. This unit can operate on a broad range of scheme parameters thanks to RTC. It uses fixed word size for Montgomery multiplication. Thus, compared to the other designs in the literature (Mert et al., 2020), it occupies less hardware with low complexity. It utilizes 8 DSP blocks and depending on the modulus size, it uses up to five clock cycles.

Fig. 3.1 (Derya, Mert, Öztürk & Savaş, 2022) shows the integer multiplication operation on the coefficients of the input polynomials. This unit is capable of performing up to 32 bits, i.e., it computes $d = a \cdot b$, where $a, b < 2^{32}$ and $d < 2^{64}$. Integer multiplication operation starts with dividing the input coefficients into two parts. The results that contain upper and lower 16-bits of the inputs are multiplied on four DSP blocks of FPGA. After registering the results, a 32-bit carry save adder is used to

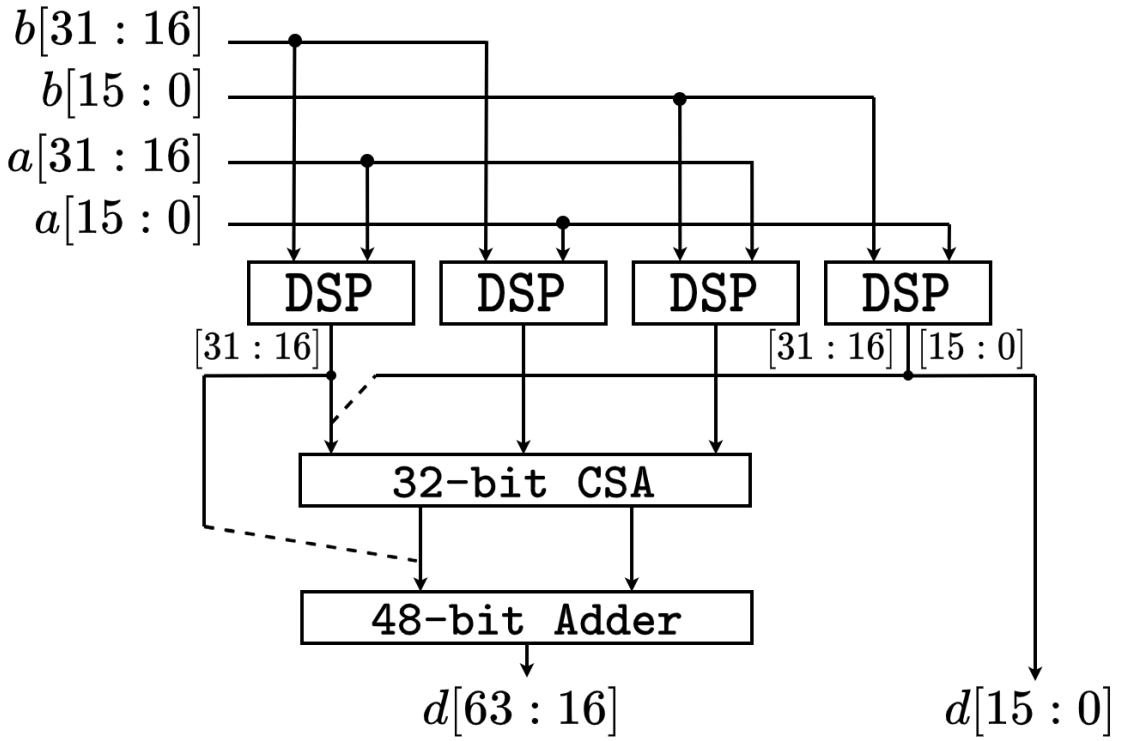

Figure 3.1 32-bit Integer Multiplier Unit

add them. Finally, the integer multiplier unit generates the multiplication result by concatenating the carry-save adder result with the upper and lower 16-bit parts of DSP multiplication result. This unit registers the multiplication result at the end. Thus, it has two clock cycle latency.

On the reduction side, there are various methods such as Barrett reduction and Montgomery reduction. Various methods offers different optimizations.

Alg. 8 shows the Barrett reduction operation. Barrett reduction uses approximate values on the integers. It also performs initial multiplication of $2k$-bits by k-bits. This results in more expensive operation compared to Montgomery reduction. To eliminate approximation and the expensive multiplication operation, we opt to use Montgomery reduction operation.

Eqn. 3.1 shows the regular Montgomery reduction algorithm. It takes $d = a \cdot b$ as input and calculates

$$c = d \cdot R^{-1} \pmod{q} \tag{3.1}$$

for $k = \lceil \log_2(q) \rceil$, where $R = 2^k$ and $q' = q^{-1} \pmod{R}$. The operation in Eqn. 3.4 is divided into w-sized parts as proposed in (Mert et al., 2019). This technique reformulates R as $R' = 2^w$ for w-sized operations. Therefore, if the word size is

Algorithm 8 Barrett Reduction (Barrett, 1987)

Input: $p, b \geq 3$
Input: $k = \lfloor \log_b \rfloor p$
Input: $0 \leq z < b^{2k}$
Input: $\mu = \lfloor b^{2k}/p \rfloor$
Output: $z \pmod p$
1: $\overline{q} \leftarrow \lfloor \lfloor z/b^{k-1} \rfloor \cdot \mu / b^{k+1} \rfloor$
2: $r \leftarrow z \pmod{b^{k+1}} - \overline{q} \cdot p \pmod{b^{k+1}}$
3: **if** $r < 0$ **then**
4: $\quad r \leftarrow r + b^{k+1}$
5: **end if**
6: **while** $r \geq p$ **do**
7: $\quad r \leftarrow r - p$
8: **end while**
9: **return** r

used as $w \leq \log_2(2n)$, $q' = q^{-1} \pmod{2^w}$ becomes -1. As a result, $d \cdot q' \pmod R$ operation can be performed with 2's complement. Lastly, the reduction operation reduces $2k$-bit input d to k-bit output $d \cdot R^{-1} \pmod q$ by repeating the w-sized reduction operation $L = \lceil \frac{k}{w} \rceil$ times.

Fig. 3.2 shows the word-level Montgomery reduction unit and it can perform reduction operation on different parameters with run-time configurability. A technique called *negative wrapped convolution* is used for the NTT-friendly primes that satisfy Eqns. 3.2-3.3. The reduction operation is performed with smaller words that has w as the word size, as presented in (Mert et al., 2019).

$$q \equiv 1 \pmod{2n} \tag{3.2}$$

$$q = q_H \cdot 2^w + 1 \text{ where } w = \log_2(2n) \tag{3.3}$$

$$c = \frac{d + q \cdot (d \cdot q' \pmod R)}{R} \tag{3.4}$$

The iteration count for w-sized operation depends on the word size (w). The modular reduction operation is affected by the word size (w) in performance-wise (i.e., larger w is used to achieve fewer number of iterations). This work aims to support a broad range of parameters and it performs polynomial multiplication operation for n values between 256 and 1024. It is possible use different word sizes for various schemes that use different n values. This approach needs to be performed with

Algorithm 9 Word-Level Montgomery Reduction Algorithm

Input: $d = a \cdot b$ ($2K$-bit integer)
Input: $w = 8$ (word size)
Input: $L = \lceil \frac{K+2}{w} \rceil$ (iteration count)
Input: q (K-bit integer, $q = q_H \cdot 2^w + 1$)
Output: $c = d \cdot R^{-1} \pmod{q}$, $R = 2^{w \cdot L} \pmod{q}$
1: $T = d$
2: **for** ($i = 0$; $i < L$; $i++$) **do**
3: $T1_H = T \gg w$
4: $T1_L = T \pmod{2^w}$
5: $T2 = -T1_L \pmod{2^w}$
6: $C_{in} = T2[w-1] \vee T1_L[w-1]$
7: $T = T1_H + (q_H \cdot T2[w-1:0]) + C_{in}$
8: **end for**
9: $c = T$
10: **return** c

similar computations.

The design in (Mert et al., 2020) utilizes each n value with different w values as a simple solution to this design requirement. Therefore, it needs more logic area to support configurability while increasing the hardware complexity. To eliminates these drawbacks, this work uses a fixed word size. The smallest scheme parameters, supported by this work, are $n = 256$ with $q = 13 \cdot 2^8 + 1$ for Kyber (v2) scheme. Therefore, w is fixed as 8. Algorithm 9 shows the word-level Montgomery reduction algorithm.

The resources of FPGA board can perform multiply-and-accumulate operation $x \cdot y + z + cin$. One for each 2's complement unit, OR gate, and DSP block can be used to implement each w-sized reduction step as indicated by the steps 5-7 of Algorithm 9. Thus, this approach is used in this work to implement one w-sized reduction step. L times w-sized reduction steps is required to complete the reduction operation. The iteration count also dictates the number of DSP blocks in this work.

As shown in the work (Mert et al., 2019), an extra subtraction operation after the reduction operation is needed to obtain a result in the interval $[0, q)$. It is possible to get rid of the extra subtraction operation at the end by using a method called incomplete arithmetic in the Montgomery reduction operation.

The regular modular arithmetic requires the integers to be in the range $[0, q)$. If a value becomes larger than or equal to the modulus q, it needs to be reduced to stay in the interval $[0, q)$. Incomplete arithmetic technique (Yanik et al., 2002) is used to eliminate the reduction operation at the end. This technique gives an opportunity to work on the integers with word-level operations. Thus, the bit-level operations

that slow down the operation are not required anymore. As a result, the integers are in the interval $[0, 2^{k+1})$ instead of $[0, q)$ to work with the modular arithmetic units.

This design requires the inputs and output of the Montgomery reduction operation (a, b, and c) to be in the interval $[0, 2^{k+1})$. This ensures that Montgomery reduction unit to work flawlessly with no extra subtraction at the end of the operation. Additionally, R needs to be at least 2^{k+2} to assure the correctness of the operation. These values are replaced into Eqn. 3.4 to show that the output should be in the interval $[0, 2^{k+1})$. Eqns. 3.5-3.6 come up with this result. Algorithm 9 show that the word-level Montgomery reduction requires R to be $2^{w \cdot L}$ where L is $\lceil \frac{k+2}{w} \rceil$.

$$c < \frac{d + q \cdot (d \cdot q' \pmod R)}{R} = \frac{2^{2k+2} + 2^k \cdot 2^{k+2}}{2^{k+2}} \tag{3.5}$$

$$c < \frac{2^{2k+2} + 2^{2k+2}}{2^{k+2}} = \frac{2^{2k+3}}{2^{k+2}} = 2^{k+1} \tag{3.6}$$

The size of coefficient modulus (k) is the only parameter that sets the iteration count (L) since the word size ($w = 8$) is fixed. This design can perform operation on integers that at most $k = 3 \cdot 8 - 2 = 22$ bits where the iteration count is 3. In this case, it cannot support Dilithium and qTESLA-q-I schemes. In order to work with integers up to $k = 4 \cdot 8 - 2 = 30$ bits, L is determined as 4. This also determines the number of DSP blocks in the modular reduction unit as 4. There is no need to use extra DSP blocks if the parameter set requires the iteration count less than 4 (i.e., $k = 16$). The correct output is given by the output multiplexer.

Fig. 3.2 (Derya et al., 2022) shows the run-time configurable word-level Montgomery reduction unit. To improve the critical path, DSP blocks are registered. In total, this unit uses four DSP blocks and it has three clock cycles of latency.

The result or one of the inputs of Algorithm 9 needs to be multiplied by R to get rid of the extra multiple of R^{-1} from the result. This work utilizes the second approach. The twiddle constants ω/ω^{-1} and ψ/ψ^{-1} are multiplied by $R{=}2^{w \cdot L}$ before the start of the operation. GitHub repository of this work includes a Python script that shows the steps of the word-level Montgomery reduction algorithm with sample inputs. It can be used to understand better the algorithm.

An efficient modular reduction units are implemented on hardware mainly in three ways: (*i*) shift and add method, (*ii*) Montgomery reduction and (*iii*) Barrett reduction. The work in (Liu, Seo, Roy, Großschädl, Kim & Verbauwhede, 2015) uses

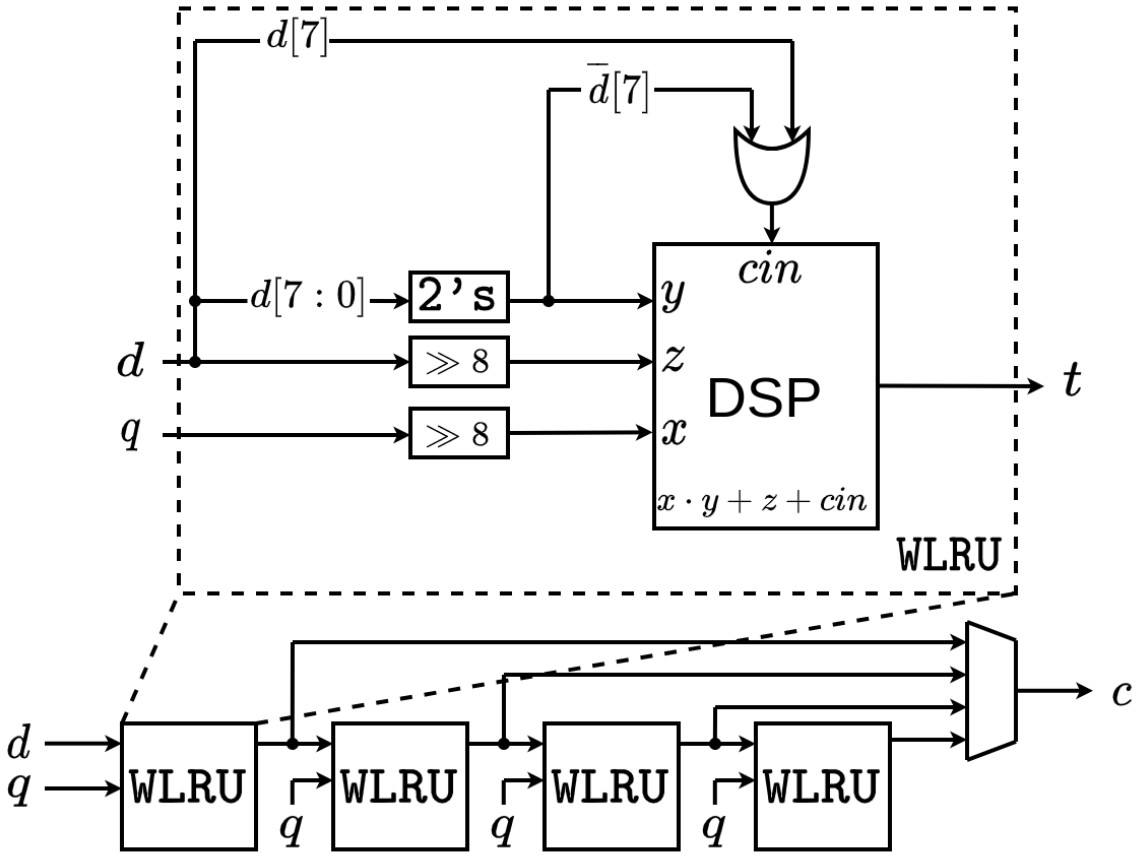

Figure 3.2 Reconfigurable Word-level Montgomery Reduction Unit

a modulus q in special form that is suitable for the shift and add method. On the other hand, the Montgomery and Barrett reduction methods can be used efficiently on random q values. It is also possible to use pre-computed reduced values in a table. The works in (Mert, Ozturk & Savas, 2020; Sinha Roy, Turan, Jarvinen, Vercauteren & Verbauwhede, 2019) uses a table-based modular reduction to perform the operation step by step by sliding a window on the table. This method is mainly used for very large q values.

The work in (Yaman et al., 2021) uses a fixed modulus $q = 3329$ from Kyber (v2) scheme. Its modular reduction unit uses a similar approach from Zhang *et al.* (Zhang, Yang, Chen, Yin, Wei & Liu, 2020). There is no need to convert the coefficients from Montgomery domain. The modular reduction operation does not use any multiplier unit since it only operates on a fixed modulus. Therefore, it does not offer any configurability options.

Moreover, the modular reduction unit in (Xing & Li, 2021) utilizes a fixed $q = 3329$ value. Barrett reduction is utilized in this unit. However, it does not have any configurability options. Similarly, the design in (Xin, Han, Yin, Zhou, Yang, Cheng & Zeng, 2020) uses the Barrett reduction in the modular reduction unit. It operates on q values up to 16-bit. It offers a limited operating window compared to our design

while offering RTC.

A fixed word size ($w = 11$) is used in the design of (Mert et al., 2019). It supports the interval $22 < \lceil \log_2(q) \rceil \leq 33$ for k while utilizing 3 DSP units. Since w is fixed, it does not offer RTC for q values.

The work in (Mert et al., 2020) utilizes the word-level Montgomery reduction method while offering CTC for n, k, and w. These parameters are fixed at the design time. Therefore, the number of DSP blocks is fixed. As a result, once the design is compiled, it does not offer run-time configurability for scheme parameters. At the run-time, it only performs operations on a single parameter. Thus, it uses less hardware resources. However, it offers less configurability options compared to our design.

Moreover, the work in (Wang et al., 2020) uses a modulus with a constant bit-size while employing the regular Montgomery reduction method. While offering CTC and less hardware complexity, it does not offer RTC for scheme parameters.

The work in (Mert et al., 2020) offers RTC for modular multiplication unit. It employs the word-level Montgomery reduction method while the word size w is chosen with respect to Eqns. 3.2-3.3. It is possible to change the word size (w) at the run-time with the implemented additional control hardware. As a result, it uses more area resources than our design while offering comparable level of configurability. It is not possible to change the number of DSP blocks. Therefore, once the design is compiled, it can only work with a certain set of coefficient size.

Furthermore, the design in (Banerjee et al., 2019) utilizes the Barrett reduction algorithm while offering RTC for different q values with the control signals. The usage of area resources for this unit is high since it employs various reduction unit for each q value.

In the literature, there are different modular multiplier unit concepts available for different circumstances. As shown in our design, the occupied hardware resources increases for a reconfigurable concept. This paves a way to leverage its potentials for configurability while showing a negligible increase in the area resources.

3.2 Unified Butterfly Unit

The butterfly operations for NTT/INTT can be performed by using CT and GS butterfly configurations on the proposed unified butterfly unit (BU) respectively. It is possible to configure this unit at the run-time between CT and GS butterfly configurations. Fig. 3.3 (Derya et al., 2022) shows the proposed butterfly unit which consists one unit of modular adder, modular subtractor, and word-level Montgomery modular multiplier.

The output values of the proposed unit need to be configured with respect to the operation, NTT or INTT. *even* and *odd* output values can be configured by the control input *ct* iat the run-time. *even* output is calculated as $a+b\cdot w \pmod q$ and $a+b \pmod q$ for NTT/INTT operations respectively. *odd* output is computed as $a-b\cdot w \pmod q$ and $(a-b)\cdot w \pmod q$ for NTT/INTT operations. Furthermore, w input value is used for twiddle factors that are needed for the operations. It becomes a power of ω/ω^{-1} for Kyber (v2) schemes and ψ/ψ^{-1} for the other schemes in NTT/INTT operations respectively.

Based on the scheme parameters, the word-level Montgomery modular multiplier unit can be configured to have clock cycle latency between 2 and 5. The iteration count, L, dictates the latency. As a result, the output values, *even* and *odd*, have to be calculated at the same time. To make it possible, different delay paths utilizing extra registers (Fig. 3.3 depicts `2cc`, `3cc`, `4cc`, `5cc`) are used. The delay path is selected by the control signal i which is chosen by the iteration count L of the word-level Montgomery reduction unit.

Eqn. 2.6 shows the coefficient-wise multiplication for the polynomial multiplication operation. The proposed butterfly unit can be utilized to perform it. In GS configuration, *odd* output is calculated as $(a-b)\cdot w \pmod q$. It possible to set a and w inputs as the input operands of $a\cdot w \pmod q$ operation and b input needs to be set as zero. In this design, this approach is used to perform coefficient-wise multiplication.

Algorithm 5 shows the coefficient-wise multiplication algorithm for Kyber (v2) scheme which differs from the other schemes. The modular arithmetic operations of this algorithm can be performed by modular adder and modular subtractor units that use *add* and *sub* as outputs respectively. In Kyber (v2) scheme, two degree-1 polynomials in the NTT domain need to be multiplied in coefficient-wise. This step of the algorithm is performed in GS configuration as explained earlier. Intermediate values, $a_0\cdot b_0$, $a_0\cdot b_1$ $a_1\cdot b_0$ and $a_1\cdot b_1$, are generated for the next steps. At the

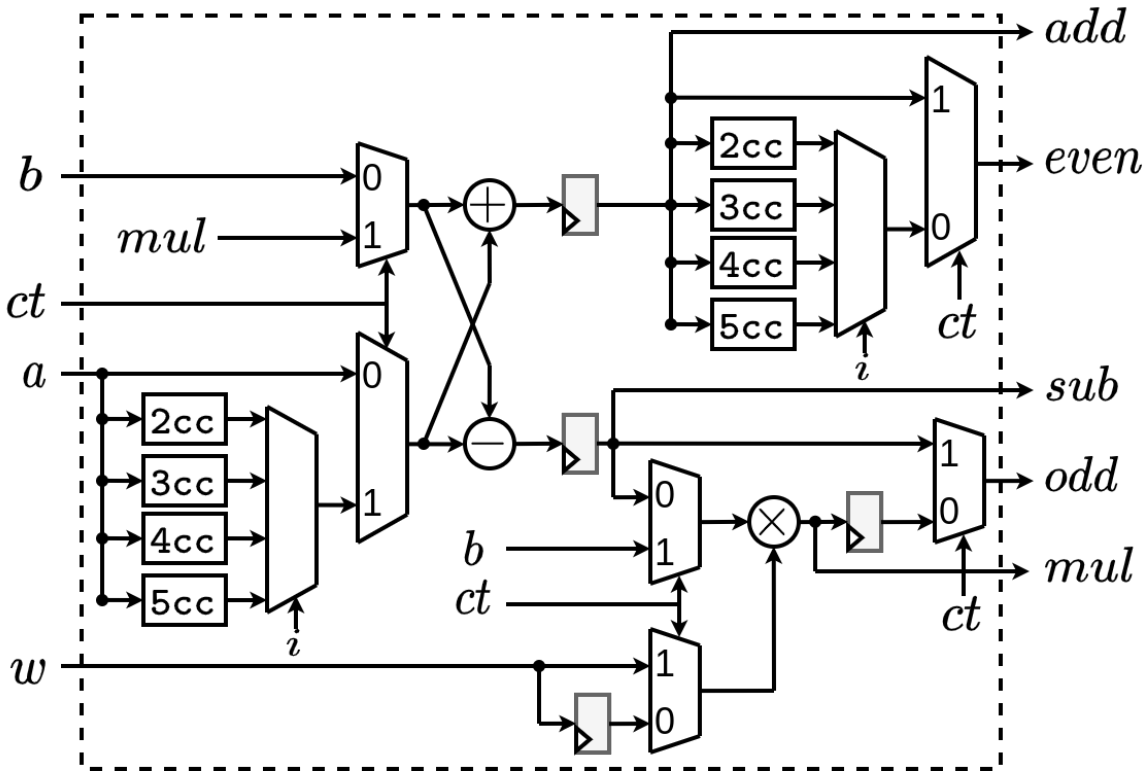

Figure 3.3 Unified Butterfly Unit

Step 3 of Alg. 5, the modular adder of the butterfly unit is used to perform modular addition. To perform Step 4, the butterfly unit is used in CT configuration to get $a+b\cdot w \pmod q$ output. To utilize it, a is used as $a_1 \cdot b_1$. w is used for the power of the twiddle factor and b is used for $a_0 \cdot b_0$ as input. In this work, the NTT domain multiplication of the Kyber (v2) scheme is utilized by using this method.

In the literature, the butterfly operations are mainly utilized in three methods: (i) only the GS configuration (Mert et al., 2020; Mert et al., 2019,2), (ii) only the CT configuration (Roy et al., 2014) and (iii) unified GS and CT configurations (Banerjee et al., 2019; Wang et al., 2020; Xin et al., 2020; Xing & Li, 2021; Yaman et al., 2021). Using only CT or GS configuration requires pre-processing and post-processing procedures for the NTT-based polynomial multiplication operation. As a result, this bring about additional $2n$ modular multiplication operations for polynomial degree of n.

Section 2 explains that pre-processign and post-processing operations can be eliminated by utilizing *merged* NTT and INTT operations with a unified GS-CT butterfly unit. Subsequently, unified GS-CT configuration needs extra control circuits to utilize both configurations. Therefore, this configuration results in higher hardware complexity and area usage with respect to the configurations with only GS or CT.

The work in (Yaman et al., 2021) uses a unified butterfly unit to implement the

butterfly operations. It can be configured to perform operations of NTT, INTT, and coefficient-wise multiplication. Nonetheless, it can only perform operations using Kyber (v2) scheme parameters. After INTT operation, the coefficients need to be multiplied by $n^{-1} \pmod{q}$. These multiplications are eliminated by using a method from Zhang *et al.* (Zhang et al., 2020).

Furthermore, the work in (Xing & Li, 2021) uses the same method to implement the butterfly operations. These works utilize optimizations for Kyber (v2) scheme. Still, they lack the support for any other scheme since their arithmetic units are utilized for the fixed data length.

The architecture in (Mert et al., 2019) utilizes the butterfly unit with only the GS configuration. It uses same amount of modular arithmetic units as our design. Due the control hardware that our design has, it can offer more configurability opportunities. Moreover, the work in (Mert et al., 2020) uses only GS configuration for the butterfly operations. The data length of the design is set at the compile-time. Thus, it does not offer RTC for scheme parameters as our design offers.

Furthermore, the butterfly unit in (Mert et al., 2020) has seven clock cycle of latency for even and odd outputs while employing only GS configuration. Therefore, the latency of the unit does not depend on the scheme parameters. Whereas, our design utilizes different delay paths with fewer clock cycles. Lastly, the work in (Roy et al., 2014) utilizes only CT configuration for the butterfly unit. Thus, it has less hardware complexity with respect to our design. As a result, it does not have the configurability capabilities that our design has.

The butterfly unit in (Wang et al., 2020) utilizes the unified CT and GS configuration method. The data length of the architecture is fixed at the compile time. Thus, it does not offer support for any other parameter at the run time. Moreover, the butterfly unit in (Banerjee et al., 2019) uses the unified butterfly operations method while employing one extra modular adder and modular subtractor compared to our design.

Furthermore, the work in (Xin et al., 2020) utilizes the same approach with one extra modular operator units. In our work, we employ the same method with less hardware sources. In the literature, different methods are utilized to perform polynomial multiplication operations. We observe that the as the design becomes more configurable, it also has more hardware complexity.

3.3 Configurable Memory Control and Overall Design

Algorithms 6-7 show the NTT/INTT algorithms which performs the calculations on the loop structures. Each operation can be parallelized by unrolling the each loop. Therefore, it is possible to adjust the throughput by changing the number of the butterfly units. In this design, it is possible to set the number of BUs at the compile time. The number of BUs cannot exceed $n/2$ and it has to be power-of-two. After taking the number of BUs, the hardware with the required logical blocks is generated by the proposed work. Fig. 3.4 (Derya et al., 2022) shows that each butterfly unit takes two BRAMs (BRAM 0 and BRAM 1) to store the input coefficients. One BRAM unit (BRAM TW) is utilized to store the pre-computed values of the twiddle factors (primitive $2n$-th root of unity).

Our work utilizes a compile-time control unit for generating the needed control signals for the architecture. Depending on the scheme parameters and the number of BUs, it produces necessary read and write address values for BRAMs. The ring size, n, determines the generated address values for BRAMs. The depth of input coefficients BRAM is $512/\texttt{BU}$. Twiddle BRAM uses a depth of $\sum_{i=0}^{i=9} 2^{\min(512/\texttt{BU},i)}$ where BU is the number of BUs. Each BRAM address has 32-bit data length.

NTT/INTT algorithms uses $\log_2(n)$ stages for PQC schemes, except Kyber (v2) where they have $\log_2(n)-1$ stages. In each stage, $n/2$ butterfly operations need to be performed. After each stage, single NTT operation can be divided into smaller NTT operations by utilizing *divide-and-conquer* method. The first stage starts with n-point NTT operation. After the first stage, two $n/2$-point NTT operations can be performed on the coefficients. This method proposes an efficient way to schedule the address scheme which can support various ring sizes. This approach is utilized recursively after each stage. Therefore the size of NTT operation becomes in half.

An additional register is used for *odd* output of the butterfly unit. The NTT algorithm performs writing into the same BRAM block in consecutive clock cycles. This extra register is used to write consecutive outputs. Furthermore, contrary to the NTT operation, smaller INTT operation are merged into larger ones for the INTT operation. The address scheduling of the INTT operation is in the reverse order of NTT operation.

For the first $\log_2(n) - \log_2(\texttt{BU}) - 1$ NTT stages, two kinds of address values are generated for BRAMs. In each consecutive clock cycle, the control unit select the different type of the adress value. One of the address values starts with 0 and

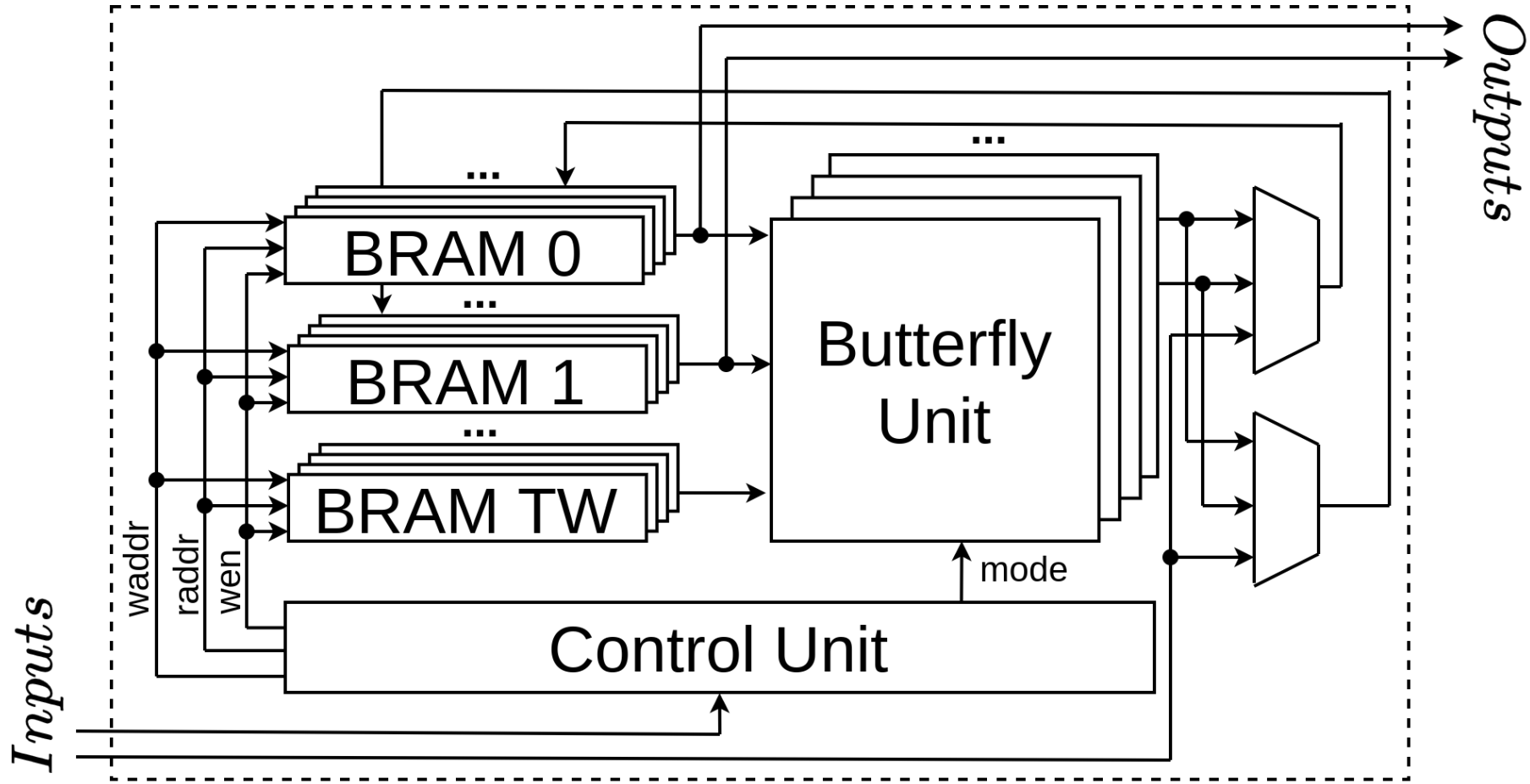

Figure 3.4 Overall Design

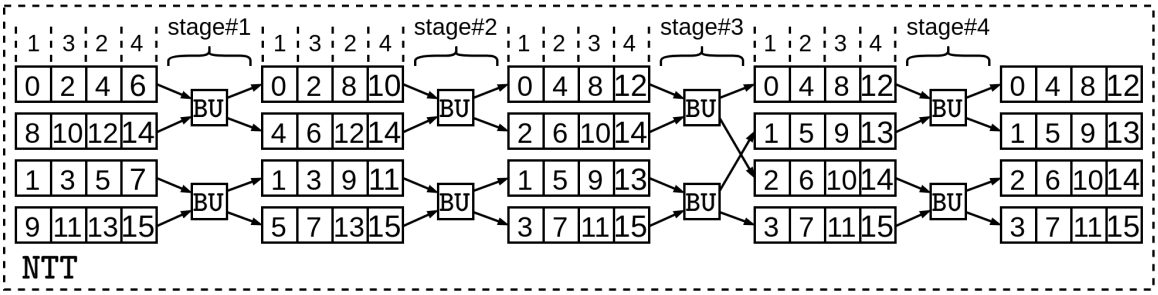

Figure 3.5 NTT Memory Access Pattern for $n = 16$ with two Butterfly Units

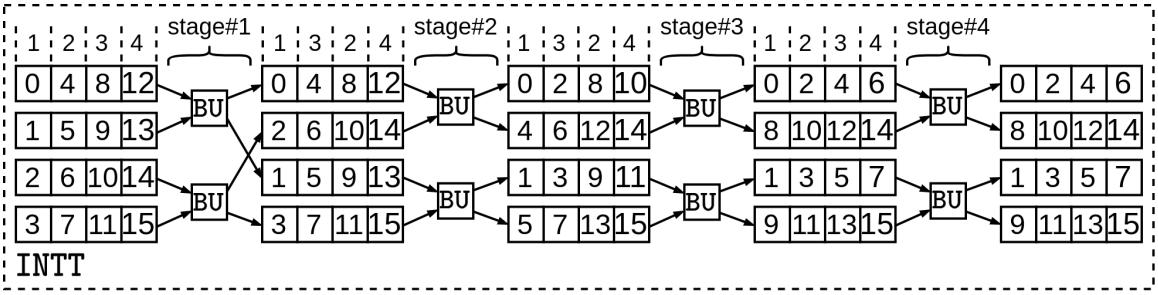

Figure 3.6 INTT Memory Access Pattern for $n = 16$ with two Butterfly Units

increments in every two clock cycles. Similarly, the other one increments in the same way. However, it starts with $n/(2^{\log_2(\texttt{BU})+s+2})$ where s is the number of the current NTT stage. As a result, the difference between them stays fixed. Until the latter type of address becomes $n/(2 \cdot \texttt{BU}) - 1$, the control unit generates these two types of address values. Moreover, the other NTT stages use only one type address value that starts with 0 and increments in every clock cycle to $n/(2 \cdot \texttt{BU}) - 1$.

Fig. 3.5 (Derya et al., 2022) shows the memory address scheduling for NTT operation of a desing with n as 16 and two BUs. Before the NTT operation, BRAMs are loaded with the input coefficients in a suitable way to start the operation. Fig. 3.5 shows that the operation starts with the coefficients pairs of 0^{th}-8^{th} and 1^{st}-9^{th}. For the first two stages, there is a data collusion while writing the coefficients into BRAMs. The coefficients from the same BU need to be written into the same BRAM. This problem is solved by reading the coefficients in an alternating order from BRAMs i.e. 0,2,1,3 instead of 0,1,2,3. The additional register of *odd* output is used during these stages.

The data collusion disappears after the first two stages and the memory read schedule increments in every clock cycle, i.e., 0,1,2,3. The control signals are used to store the coefficients into different BRAMs than they used to be before the NTT operation. This data reordering ensures that the coefficients are stored in the correct BRAM for NTT operation.

Fig. 3.6 (Derya et al., 2022) shows the memory address scheduling for INTT operation of a design with n as 16 and two BUs. Coefficient-wise multiplication is performed before INTT operation. As it can be seen from the Fig. 3.5, the last

stage of NTT operation orders the coefficients in BRAMs with coefficient pairs that are suitable to INTT operation. As a result, there is no need for data reordering before INTT operation.

In the first stage, the memory read schedule increments in every clock cycle, i.e., 0,1,2,3. On the next two stages, data collision occurs as there are coefficients that needs to be written into same BRAM block in the same clock cycle. To overcome this problem, the memory read schedule is performed in alternating order i.e. 0,2,1,3 instead of 0,1,2,3. As used in NTT operation, the extra register for *odd* output is used during these stages.

For the last stage, there is no data collision. Thus, the memory read schedule is performed in incremented order i.e. 0,1,2,3. To sum up, INTT memory access pattern is the reverse of NTT memory access pattern.

The address scheduling for the coefficient-wise multiplication operation is directed by the control unit. The input coefficients are stored in address values between 0 and 512/`BU` in BRAM blocks. The butterfly units take coefficients from BRAMs with respect to the address signals produced by the control unit. The butterfly unit is used in GS configuration to perform coefficient-wise multiplication operation. The inputs are set as the first operand, the second operand, and zero for a, w and b, respectively.

Fig. 3.7 shows the address schedule of Kyber (v2) coefficient-wise multiplication. Kyber (v2) scheme utilizes different algorithm for the coefficient-wise multiplication operation and it uses different address scheduling. Algorithm 5 shows that there needs to be modular multiplication and addition operations in Steps 3-4. The necessary modular multiplication operations are performed and the results are stored in BRAMs. On the next step, Step 3-4 of Algorithm 5 are performed using the butterfly units.

In the literature, there are two methodologies used to utilize NTT-based polynomial multipliers: (*i*) an independent hardware accelerator (Mert et al., 2020; Mert et al., 2019,2; Yaman et al., 2021) and (*ii*) a sub-block unit in a cryptographic hardware (Banerjee et al., 2019; Roy et al., 2014; Wang et al., 2020; Xin et al., 2020; Xing & Li, 2021). Independent hardware accelerators are implemented on FPGAs to utilize their high-performance blocks.

Compared to sub-block units, they show superior performance since sub-block units are utilized on low-constrained devices. There needs to be a host processor for the sub-block NTT multiplier to operate. They focus on low power rather than high performance. Thus, they are implemented on the low-constrained devices.

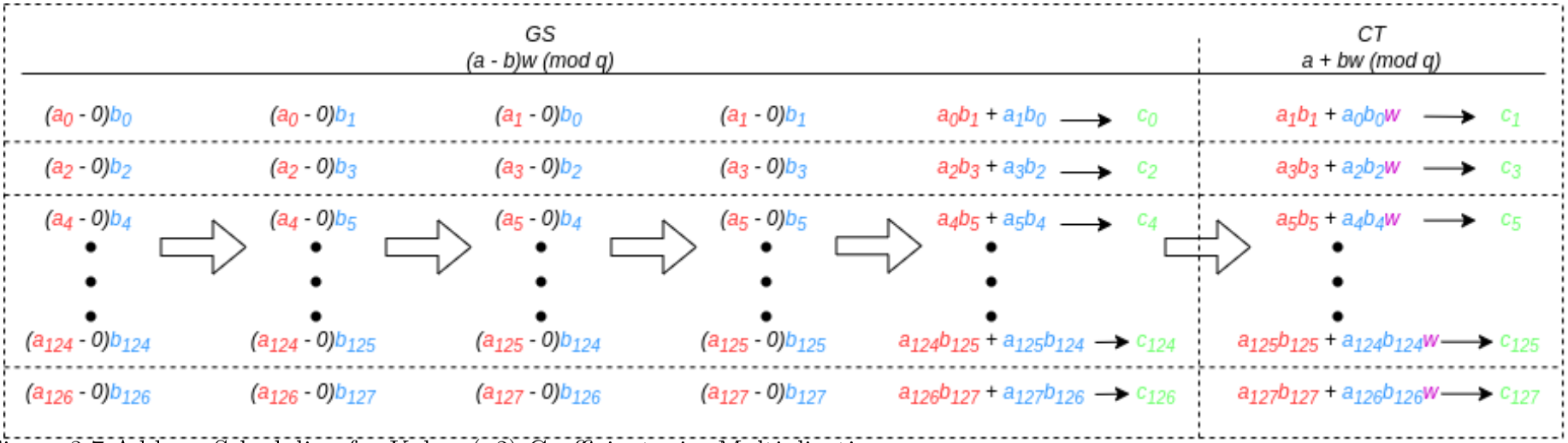

Figure 3.7 Address Scheduling for Kyber (v2) Coefficient-wise Multiplication

The architecture in (Yaman et al., 2021) proposes a hardware accelerator for Kyber (v2) scheme with NTT-based polynomial multiplication. The overall design concept of the design is similar with our work. It utilizes BRAM blocks for storing input coefficients and it uses recursive NTT method with a control unit to perform NTT operations. The control unit uses a similar address scheduling as our work. However, it can only perform operations on Kyber (v2) scheme. Therefore, it does not have any support for other schemes. Algorithm 5 is used with pipeline optimizations for the coefficient-wise multiplication operation. Compared to the design in (Yaman et al., 2021), our work offers more configurability options for other schemes by utilizing a negligible amount of more resources while having similar performance results. Additionally, our work offers CTC to adjust the throughput rate.

Furthermore, the design in (Mert et al., 2019) uses a similar approach to implement the NTT-based polynomial multiplication. It utilizes a fixed number of BRAM block and butterfly units. As a result, it does not have CTC capabilities on the design. It does not have any support for Kyber (v2) scheme while employing a similar address scheduling method to perform the operations. We use somewhat more hardware resources while offering more configurability opportunities.

The work in (Mert et al., 2020) proposes a hardware accelerator with a fixed number of butterfly units and BRAM blocks with a similar overall design. Thus, it is not possible to adjust the throughput rate with no CTC capabilities. On the other hand, the control unit in (Mert et al., 2020) can be used for different scheme parameters. It generates control signals with respect to the parameter set. It utilizes large amount of multiplexers while employing run-time support for the scheme parameters. The control unit in our work uses the control unit with more efficiency by using arithmetic operations on small multiplexers to generate control signals. Our work has a similar performance results on selected parameter sets while having more configurability capabilities with support for Kyber (v2) scheme.

Furthermore, the design in (Mert et al., 2020) offers CTC for design parameters that sets the number of building block at the compile-time. Thus, it is possible to adjust the parallelism to set the throughput rate. While using a similar address pattern with our design, the control unit in (Mert et al., 2020) is utilized for a unique set of scheme parameter. Thus, it does not offer RTC for scheme parameters. Compared to the design in (Mert et al., 2020), we utilize more hardware sources to offer more configurability capabilities.

The design in (Xing & Li, 2021) utilizes NTT framework to implement operations for Kyber (v2) scheme. NTT-based multiplier is used as a sub-block that uses two butterfly units with 2 RAM blocks. This design does not have RTC for scheme

parameters and CTC for design parameters. By using more hardware source, we offer more configurability capabilities compared to the design in (Xing & Li, 2021).

Moreover, the work in (Banerjee et al., 2019) implements a crypto-processor for PQC with configurability options. While performing NTT operations, it uses out-of-place NTT structure that requires the coefficients of the input and output polynomials to be stored on different memory units. Compared to the work in (Banerjee et al., 2019), we use same memory units for storing the input and output polynomials at the same time.

A hardware for PQC is introduced in the work (Wang et al., 2020) that uses NTT-based polynomial multiplication approach. The memory access pattern is optimized to have two butterfly operations simultaneously. Two units of memory blocks are used to store input polynomials, and two memory blocks are utilized to store twiddle factors. The control unit is compiled once for a unique set of scheme parameters.

Moreover, the work in (Roy et al., 2014) employs a similar approach for the control unit by using different number of memory blocks. Compared to those designs, our unit has RTC for the control unit. Therefore, it is possible to use it on different scheme parameters. Overall, it is possible to say that as the configurability of the design increases, the hardware complexity of the overall design increases.

4. RESULTS AND COMPARISON

In this chapter, firstly, we give discussion about the prior works in the literature. Then, we present our implementation results and make comparison with respect to the works in the literature.

4.1 Prior Works

Table 4.1 Prior Works

Design	**Platform**	n	$\lceil \log_2(q) \rceil$	**LUT/FF/DSP/BRAM (mm^2 for ASIC)**	**Freq. (MHz)**	**Latency (Clock Cycles)**		
						NTT	**INTT**	**P.M.**
(Banerjee et al., 2019)†	40 nm CMOS	256	13	0.28	72	1289	–	–
		512	14			2826		
		1024	14			6155		
(Mert et al., 2020)	Virtex-7	256	23	888 / – / 7 / 5	125	1096	–	–
				5K / – / 56 / 12		200		
		512	14	537 / – / 3 / 5.5		2340		
				2.5K / – / 24 / 12		324		
		1024	14	575 / – / 3 / 11		5160		
				17.1K / – / 96 / 48		200		
(Fritzmann et al., 2020)*†	Zynq-7000	256	12	2.9K / 170 / 9 / 0	45	1935	1930	–
		512	14			8169	8684	
		1024	14			18537	20171	
(Fritzmann & Sepúlveda, 2019)†	UMC 65 nm	256	13	0.329	25	2056	–	–
		512	14			4616		
		1024	14			10248		
(Yaman et al., 2021)*	Spartan-6	256	12	985 / 444 / 1 / 5	138	904	904	3359
	Artix-7			948 / 352 / 1 / 2.5	190			
(Fritzmann et al., 2021)*†	Artix-7	256	–	2.4K / 1.9K / 7 / 4.5	153	3584	–	–
		512				8192		
		1024				20480		
(Mert et al., 2020)	Virtex-7	1024	28	1K / 1K / 7 / 2	125	5290	–	–
				16K / 14K / 56 / 24		490		
(Greconici et al., 2020)*	Cortex-M3	256	12	– / – / – / –	16	10819	12994	–
		256	23			19347	21006	
		1024	14			77001	93128	
(Mert et al., 2020)†	Virtex-7	256	16	39.6K / – / 224 / 96	150	104	–	288
		512				153		468
		1024				249		815

P.M.: Polynomial Multiplication; ^{a}: 1 BU; ^{b}: 8 BUs; ^{c}: 32 BUs; *: supports Kyber (v2); †: supports multiple n and q at run-time.

In the literature, PQC schemes are implemented efficiently on both hardware and

software platforms. Table 4.1 shows the prior works in the literature. These works support a wide range of scheme parameters with different configurability capabilities.

An ASIC implementation of a custom crypto-processor that supports different scheme parameters for Round 2 candidates of NIST's competition is presented in the work (Banerjee et al., 2019). The architectures in (Mert et al., 2020,2) utilizes CTC for design parameters to accelerate NTT operation. While the work (Mert et al., 2020) offers RTC for scheme parameters, the architecture (Mert et al., 2020) only supports a fixed set of scheme parameters.

The design in (Fritzmann et al., 2020) introduces implementations for lattice-based PQC schemes on ASIC and FPGA platforms that supports different scheme parameters. Furthermore, a power optimized NTT accelerator is implemented on ASIC platform for different PQC schemes in the work (Fritzmann & Sepúlveda, 2019).

The architecture in (Yaman et al., 2021) introduces optimizations for a polynomial multiplier on FPGA platform for the Kyber (v2) scheme with no support for RTC and CTC.

The architecture in (Fritzmann et al., 2021) utilizes a NTT multiplier and presents set extensions on RISC-V instructions for different PQC schemes. Moreover, Kyber (v2), Dilithium, and NewHope schemes are implemented on Cortex-M3 platform in the work (Greconici et al., 2020).

Furthermore, the design in (Mert et al., 2020) utilizes RTC for scheme parameters while employing NTT-based accelerator. Lastly, the work in (Seiler, 2018) introduces optimizations for AVX2 implementation on the NTT operations of Kyber (v1) and NewHope schemes. Also, the work in (Alkim et al., 2020) proposes the ISA extension on RISC-V for Kyber (v2) and NewHope schemes.

4.2 Implementation Results and Comparison

Table 4.2 Implementation Results

Design	Platform	n	$\lceil \log_2(q) \rceil$	LUT/FF/DSP/BRAM (mm^2 for ASIC)	Freq. (MHz)	Latency (Clock Cycles)		
						NTT	INTT	P.M.
Ours*†	Virtex-7	256^a	12	2128 / 1144 / 8 / 3	174	922	1184	3812
		256^a	13			1052	1314	3680
	Artix-7	256^a	23	2119 / 1058 / 8 / 3	117	1052	1318	3688
		512^a	14			2334	2854	8072
	32 nm	1024^a	14	0.053	462	5152	6182	17506
		1024^a	29			5162	6195	17552
	Virtex-7	256^b	12	11K / 5422 / 64 / 12	186	138	176	572
		256^b	13			156	197	550
	Artix-7	256^b	23	11K / 5182 / 64 / 12	140	156	198	552
		512^b	14			318	391	1100
	32 nm	1024^b	14	0.353	416	672	812	2296
		1024^b	29			682	819	2320
	Virtex-7	256^c	12	61K / 17K / 256 / 48	167	84	101	306
		256^c	13			95	112	319
	Artix-7	256^c	23	63K / 18K / 256 / 48	126	103	121	345
		512^c	14			126	141	428
	32 nm	1024^c	14	2.205	416	192	233	658
		1024^c	29			202	244	690

P.M.: Polynomial Multiplication; ^{a}: 1 BU; ^{b}: 8 BUs; ^{c}: 32 BUs; *: supports Kyber (v2); †: supports multiple n and q at run-time.

Table 4.2 shows the implementation results of our work with comparisons to the other works in the literature. The implementation results are obtained from Vivado 2019.1 tool that synthesized Verilog codes on Xilinx Virtex-7 FPGA (xc7vx690tffg1761-2) and Artix-7 FPGA (xc7a200t-2fbg676c) with default settings. Our hardware implementation utilizes 2128 LUTs, 8 DSPs, 3 BRAMs units for area optimized implementation approach that consists of only one butterfly unit. The clock frequency of the area-optimized design is 174 MHz on Virtex-7 FPGA. Additionally, an ASIC implementation of our design on a 32 nm standard cell library is presented in Table 4.2 excluding on-chip memory with different number of butterfly units.

Table 4.2 shows that the resource usage is increased as the number of butterfly units is increased for all three platforms. As a result, the latency in clock cycles is decreased. Moreover, the number of clock cycles is heavily depended on the ring size, n. As the q bit-size is increased from 12 to 23 in the same ring size, 256, the number of clock cycles in latency is more or less similar. As the ring size is increased to 512 and 1024, the number of clock cycles in latency is dramatically increased.

Table 4.3 shows the resource usage of each sub-block on Virtex-7 and Artix-7 FPGA

platforms. As it can be seen, the synthesis results on both platforms show similar resource utilization for each sub-block. The performance differs in the clock speed as Table 4.2 shows. Virtex-7 platform offers more clock speed than Artix-7 platform as shown by the synthesis results of our design.

Table 4.3 Resource Utilization of Sub-Blocks

BU	**Block**	**Virtex-7**	**Artix-7**
		LUTs/FFs/DSPs/BRAMs	
1	Overall	2128/1144/8/3	2119/1058/8/3
	⌊Mem. Con.	786/263/0/0	775/263/0/0
	⌊But. Unit	703/474/8/0	705/488/8/0
	⌊Mod. Mul.	239/186/8/0	241/100/8/0
8	Overall	10973/5422/64/12	10908/5182/64/12
	⌊Mem. Con.	1358/422/0/0	1358/422/0/0
	⌊But. Unit	7529/3400/64/0	7461/3160/64/0
	⌊Mod. Mul.	1926/1096/64/0	1918/856/64/0
32	Overall	61731/17846/256/48	63032/18182/256/48
	⌊Mem. Con.	7410/1457/0/0	7738/1466/0/0
	⌊But. Unit	46553/10728/256/0	47459/11096/256/0
	⌊Mod. Mul.	7680/1512/256/0	7690/1835/256/0

Fig. 4.1 shows the relation between area and latency for our hardware accelerator with a different number of BUs. It is easy to configure this work into different applications that needs different area-performance requirements since it is possible to change the throughput rate with only one parameter. For instance, a fast NTT multiplier is needed for a high performance application or multiple SHA3 blocks. To make a useful comparison, it is needed to compare the designs that target the same platform with identical scheme parameters. Thus, we present an estimate comparison in this section, not an ideal one.

Our ASIC implementation in one BU configuration shows superior performance with higher clock frequency and fewer clock cycles while employing low area usage compared the crypto-processor Sapphire (Banerjee et al., 2019). Moreover, the designs in (Mert et al., 2020,2) introduces introduces polynomial multipliers with compile-time support without run-time support. Our hardware with different number of BUs uses negligible amount of more resources while having comparable performance results compared the works in (Mert et al., 2020,2). Therefore, it is reasonable to say that both RTC and CTC options can be supported with no performance as our hypothesis claims.

The architecture in (Fritzmann et al., 2020) offers RTC with one butterfly unit on

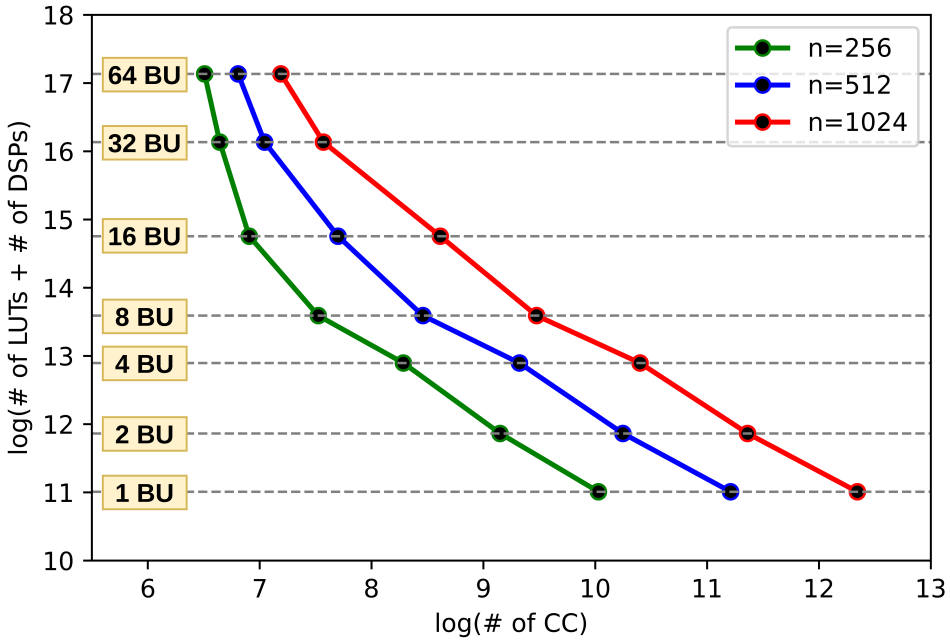

Figure 4.1 Area vs Latency for n={256, 512, 1024} with Different Number of Butterfly Units (BU)

Zynq-7000 platform. On the same parameter sets, our one BU design utilizes lower resources on Virtex-7 and Artix-7 platforms while having higher clock frequency with lower number of clock cycles for NTT and INTT operations.

The ASIC implementation in (Fritzmann & Sepúlveda, 2019) employs RTC for different scheme parameters with one butterfly unit. Our ASIC implementation with one BU employs lower area usage with higher clock frequency and lower number of clock cycles on same the parameter sets. It is good to bear in mind that 32-nm library offers more performance compared to UMC 65-nm library.

The work in (Yaman et al., 2021) shows a slightly better performance in Kyber (v2) scheme compared to our work. However, contrary to our design, it lacks the support for other PQC schemes. As a result, supporting multiple PQC schemes with CTC & RTC options shows an increase in the resource utilization.

The architecture in (Fritzmann et al., 2021) has one butterfly unit with RTC for different scheme parameters. Compared to our design in Artix-7 platform, it has more clock speed while having more number of clock cycles in NTT operation. On the other hand, our Virtex-7 result on the same parameter sets has more clock speed with fewer number of clock cycles.

The design in (Greconici et al., 2020) offers NTT and INTT operations for different scheme parameters on Cortex-M3 platform. Our implementation results on three platforms offer more clock speed with lower number of clock cycles. It should be noted that we have not implemented our design on micro controller as the work

in (Greconici et al., 2020).

Our 32 butterfly unit configuration shows a similar performance compared to the NTT-based polynomial multiplier in (Mert et al., 2020). Even though the work in (Mert et al., 2020) has RTC on scheme parameters, it does not offer CTC and support for Kyber (v2) scheme. The resource utilization of the work in (Mert et al., 2020) is lower than our design. However, it does not scale to other scheme parameters as our design does. Thus, it does not have the configurability capabilities of our work.

5. CONCLUSION

In this thesis, we propose a configurable NTT-based polynomial multiplier architecture with high level of optimizations that operates on NTT-friendly PQC schemes. We introduce an architecture that offers compile-time configurability (CTC) for latency and area (i.e., the number of unified butterfly units) and run-time configurability (RTC) for the scheme parameters (i.e., n and q). Additionally, we show the design approach differences between our work and the proposed work in the literature and specifically highlight the configurability capabilities of each work.

The proposed hardware targets the NTT-friendly lattice-based PQC schemes that have ring sizes between 256 and 1024 and coefficient bit-size up to 30 bits. It can perform modular operations including NTT, INTT, and NTT-based polynomial multiplication. Thus, the design is capable of adjusting the correlation between latency and area by modifying a single parameter. It is applicable to different platforms that target an accelerator for lattice-based PQC schemes.

We compare our implementation results that consists of different configurations of the design on both FPGA and ASIC platforms. Table 4.2 shows the comparison with respect to the state-of-art implementations in the literature. It is shown that our design shows similar efficiency in latency against other architectures. The proposed architecture supports run-time configurability for different scheme parameters while accelerating a broad range of lattice-based PQC schemes. To support unique design requirements on different platforms, the compile-time configurability support for the trade-off between area and latency can be utilized. As far as we know, there is no other architecture that supports both. This configurability options are utilized without an impact on the latency while employing a negligible rise on the resource utilization.

This research is supported in part by the by The Scientific and Technological Research Council of Turkey under Grant Number 118E725.

BIBLIOGRAPHY

Alkim, E., Barreto, P. S., Bindel, N., Longa, P., & Ricardini, J. E. (2019). The Lattice-Based Digital Signature Scheme qTESLA. *IACR Cryptology ePrint Archive*, *2019*, 85.

Alkim, E., Ducas, L., Pöppelmann, T., & Schwabe, P. (2016). Post-quantum key exchange—a new hope. In *25Th {USENIX} security symposium ({USENIX} security 16)*, (pp. 327–343).

Alkim, E., Evkan, H., Lahr, N., Niederhagen, R., & Petri, R. (2020). ISA Extensions for Finite Field Arithmetic - Accelerating Kyber and NewHope on RISC-V. Cryptology ePrint Archive, Report 2020/049. `https://eprint.iacr.org/2020/049`.

Banerjee, U., Ukyab, T. S., & Chandrakasan, A. P. (2019). Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. *IACR Trans. on CHES*, 17–61.

Barrett, P. (1987). Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In Odlyzko, A. M. (Ed.), *Advances in Cryptology — CRYPTO' 86*, (pp. 311–323)., Berlin, Heidelberg. Springer Berlin Heidelberg.

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., & Stehlé, D. (2018). CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *IEEE Euro S&P*, (pp. 353–367).

Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., & Stehlé, D. (2017). CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634. `https://eprint.iacr.org/2017/634`.

Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R., & Smith-Tone, D. (2016). Report on post-quantum cryptography.

Chu, E. & George, A. (1999). *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC press.

Derya, K., Mert, A. C., Öztürk, E., & Savaş, E. (2022). Coha-ntt: A configurable hardware accelerator for ntt-based polynomial multiplication. *Microprocessors and Microsystems*, *89*, 104451.

Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., & Stehlé, D. (2018). Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. on CHES*, 238–268.

D'Anvers, J.-P., Karmakar, A., Roy, S. S., & Vercauteren, F. (2018). Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. Cryptology ePrint Archive, Report 2018/230. `https://ia.cr/2018/230`.

Fouque, P.-A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., & Zhang, Z. (2018). Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*.

Fritzmann, T., Beirendonck, M. V., Roy, D. B., Karl, P., Schamberger, T., Verbauwhede, I., & Sigl, G. (2021). Masked accelerators and instruction set extensions for post-quantum cryptography. Cryptology ePrint Archive, Report

2021/479. https://ia.cr/2021/479.

Fritzmann, T. & Sepúlveda, J. (2019). Efficient and Flexible Low-Power NTT for Lattice-Based Cryptography. In *2019 IEEE Int. Symposium on HOST*, (pp. 141–150).

Fritzmann, T., Sharif, U., Müller-Gritschneder, D., Reinbrecht, C., Schlichtmann, U., & Sepulveda, J. (2019). Towards reliable and secure post-quantum co-processors based on risc-v. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, (pp. 1148–1153).

Fritzmann, T., Sigl, G., & Sepúlveda, J. (2020). Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 239–280.

Fritzmann, T., Sigl, G., & Sepúlveda, J. (2020). RISQ-V: Tightly Coupled RISC-V Accelerators for Post-Quantum Cryptography. *IACR Trans. on CHES*, *2020*(4), 239–280.

Greconici, D. O. C., Kannwischer, M. J., & Sprenkels, D. (2020). Compact dilithium implementations on cortex-m3 and cortex-m4. Cryptology ePrint Archive, Report 2020/1278. https://ia.cr/2020/1278.

Liu, Z., Seo, H., Roy, S. S., Großschädl, J., Kim, H., & Verbauwhede, I. (2015). Efficient ring-lwe encryption on 8-bit avr processors. In *International Workshop on Cryptographic Hardware and Embedded Systems*, (pp. 663–682). Springer.

Longa, P. & Naehrig, M. (2016). Speeding up the number theoretic transform for faster ideal lattice-based cryptography. Cryptology ePrint Archive, Paper 2016/504. https://eprint.iacr.org/2016/504.

Lyubashevsky, V. & Seiler, G. (2019). NTTRU: Truly Fast NTRU Using NTT. *IACR Trans. on CHES*, *2019*(3), 180–201.

Mavroeidis, V., Vishi, K., Zych, M. D., & Jøsang, A. (2018). The impact of quantum computing on present cryptography. *ArXiv*, *abs/1804.00200*.

McEliece, R. J. (1978). A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, *44*, 114–116.

Mert, A. C., Karabulut, E., Ozturk, E., Savas, E., & Aysu, A. (2020). An extensive study of flexible design methods for the number theoretic transform. *IEEE Transactions on Computers*, 1–1.

Mert, A. C., Karabulut, E., Öztürk, E., Savaş, E., Becchi, M., & Aysu, A. (2020). A Flexible and Scalable NTT Hardware : Applications from Homomorphically Encrypted Deep Learning to Post-Quantum Cryptography. In *2020 DATE*, (pp. 346–351).

Mert, A. C., Ozturk, E., & Savas, E. (2019). Design and implementation of a fast and scalable ntt-based polynomial multiplier architecture. Cryptology ePrint Archive, Report 2019/109.

Mert, A. C., Ozturk, E., & Savas, E. (2020). Low-latency asic algorithms of modular squaring of large integers for vdf evaluation. *IEEE Transactions on Computers*, 1–1.

Mert, A. C., Öztürk, E., & Savaş, E. (2020). Fpga implementation of a run-time configurable ntt-based polynomial multiplication hardware. *Microprocessors and Microsystems*, *78*, 103219.

Nejatollahi, H., Dutt, N., Ray, S., Regazzoni, F., Banerjee, I., & Cammarota, R. (2019). Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, *51*(6).

Nguyen, D. T., Dang, V. B., & Gaj, K. (2019). A high-level synthesis approach to the software/hardware codesign of ntt-based post-quantum cryptography algorithms. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, (pp. 371–374).

Nguyen, D. T., Dang, V. B., & Gaj, K. (2020). High-Level Synthesis in Implementing and Benchmarking Number Theoretic Transform in Lattice-Based Post-Quantum Cryptography Using Software/Hardware Codesign. In Rincón, F., Barba, J., So, H. K. H., Diniz, P., & Caba, J. (Eds.), *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, (pp. 247–257)., Cham. Springer International Publishing.

Pöppelmann, T. & Güneysu, T. (2014). Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *2014 IEEE Int. Symp. on Circuits and Systems*, (pp. 2796–2799).

Pöppelmann, T., Oder, T., & Güneysu, T. (2015). High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In *International Conference on Cryptology and Information Security in Latin America*, (pp. 346–365). Springer.

Regev, O. (2010). The learning with errors problem (invited survey). In *2010 IEEE 25th Annual Conference on Computational Complexity*, (pp. 191–204).

Riazi, M. S., Laine, K., Pelton, B., & Dai, W. (2020). Heax: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, (pp. 1295–1309)., New York, NY, USA. Association for Computing Machinery.

Roy, S. S., Vercauteren, F., Mentens, N., Chen, D. D., & Verbauwhede, I. (2014). Compact Ring-LWE Cryptoprocessor. In Batina, L. & Robshaw, M. (Eds.), *Cryptographic Hardware and Embedded Systems – CHES 2014*, (pp. 371–391)., Berlin, Heidelberg. Springer Berlin Heidelberg.

Seiler, G. (2018). Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Crypto. ePrint Arch., Report 2018/039.

Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, (pp. 124–134).

Sinha Roy, S., Turan, F., Jarvinen, K., Vercauteren, F., & Verbauwhede, I. (2019). Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, (pp. 387–398).

Tan, W., Case, B. M., Hu, G., Gao, S., & Lao, Y. (2020). An ultra-highly parallel polynomial multiplier for the bootstrapping algorithm in a fully homomorphic encryption scheme. *Journal of Signal Processing Systems*, 1–14.

Wang, W., Tian, S., Jungk, B., Bindel, N., Longa, P., & Szefer, J. (2020). Parameterized hardware accelerators for lattice-based cryptography and their application to the hw/sw co-design of qtesla. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2020*(3), 269–306.

Xin, G., Han, J., Yin, T., Zhou, Y., Yang, J., Cheng, X., & Zeng, X. (2020). VPQC: A Domain-Specific Vector Processor for Post-Quantum Cryptography Based on RISC-V Architecture. *IEEE Trans. on Circuits and Systems I: Regular Papers*, *67*(8), 2672–2684.

Xing, Y. & Li, S. (2021). A compact hardware implementation of cca-secure key exchange mechanism crystals-kyber on fpga. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2021*(2), 328–356.

Yaman, F., Mert, A. C., Öztürk, E., & Savaş, E. (2021). A hardware accelerator for polynomial multiplication operation of crystals-kyber pqc scheme. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, (pp. 1020–1025).

Yanik, T., Savas, E., & Koc, C. K. (2002). Incomplete reduction in modular arithmetic. *IEE Proceedings - Computers and Digital Techniques*, *149*(2), 46–52.

Zhang, N., Yang, B., Chen, C., Yin, S., Wei, S., & Liu, L. (2020). Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT. *IACR Trans. on CHES*, *2020*(2), 49–72.