An Investigation of Methods to Improve Area and Performance of Hardware Implementations of a Lattice Based Cryptosystem

Luke P. Beckwith

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Lingjia Liu, Chair
William J. Diehl, Co-chair
Paul K. Ampadu

October 14, 2020
Blacksburg, Virginia

Keywords: Post Quantum Cryptography, NewHope, FPGA, Cryptography, Pipelined Architecture, Trivium, Random Number Generation, Register Transfer Level Design, NIST

An Investigation of Methods to Improve Area and Performance of Hardware Implementations of a Lattice Based Cryptosystem

Luke P. Beckwith

(ABSTRACT)

With continuing research into quantum computing, current public key cryptographic algorithms such as RSA and ECC will become insecure. These algorithms are based on the difficulty of integer factorization or discrete logarithm problems, which are difficult to solve on classical computers but become easy with quantum computers. Because of this threat, government and industry are investigating new public key standards, based on mathematical assumptions that remain secure under quantum computing. This paper investigates methods of improving the area and performance of one of the proposed algorithms for key exchanges, "NewHope." We describe a pipelined FPGA implementation of NewHope512cpa which dramatically increases the throughput for a similar design area. Our pipelined encryption implementation achieves 652.2 Mbps and a 0.088 Mbps/LUT throughput-to-area (TPA) ratio, which are the best known results to date, and achieves an energy efficiency of 0.94 nJ/bit. This represents TPA and energy efficiency improvements of 10.05× and 8.58×, respectively, over a non-pipelined approach. Additionally, we investigate replacing the large SHAKE XOF (hash) function with a lightweight Trivium based PRNG, which reduces the area by 32% and improves energy efficiency by 30% for the pipelined encryption implementation, and which could be considered for future cipher specifications.

An Investigation of Methods to Improve Area and Performance of Hardware Implementations of a Lattice Based Cryptosystem

Luke P. Beckwith

(GENERAL AUDIENCE ABSTRACT)

Cryptography is prevalent in almost every aspect of our lives. It is used to protect communication, banking information, and online transactions. Current cryptographic protections are built specifically upon public key encryption, which allows two people who have never communicated before to setup a secure communication channel. However, due to the nature of current cryptographic algorithms, the development of quantum computers will make it possible to break the algorithms that secure our communications. Because of this threat, new algorithms based on principles that stand up to quantum computing are being investigated to find a suitable alternative to secure our systems. These algorithms will need to be efficient in order to keep up with the demands of the ever growing internet. This paper investigates four hardware implementations of a proposed quantum-secure algorithm to explore ways to make designs more efficient. The improvements are valuable for high throughput applications, such as a server which must handle a large number of connections at once.

Dedication

To my parents for their wisdom and advice.

To Evelyn for her love and support.

Acknowledgments

I would like to thank my advisor Dr. Diehl for his guidance and advice throughout my research. He has provided consistent and valuable insight and feedback throughout my studies. I would not have been able to complete this work with such promptness without his willingness to assist to advise my work over the summer and his focus on directing my efforts. Thank you to Dr. Lingjia Liu for his willingness to serve as the co-chair of my committee. Thank you to Dr. Paul Ampadu for serving as a committee member

Contents

List of Figures

List of Tables

List of Abbreviations

CBD Centered Binomial Distribution

CCA Chosen Ciphertext Attack

CPA Chosen Plaintext Attack

CPU Central Processing Unit

CVP Closest Vector Problem

DSP Digital Signal Processing

ECC Elliptic Curve Cryptography

FF Flip Flop

FFT Fast Fourier Transform

FHE Fully Homomorphic Encryption

FPGA Field Programmable Gate Array

GF Galois Field

GPU Graphics Processing Unit

HDL Hardware Description Language

IoT Internet of Things

ISE Instruction Set Extension

KEM Key Encapsulation Mechanism

LBC Lattice Based Cryptography

LUT Look Up Table

LWE Learning with Errors

MLWE Module Learning with Errors

MLWR Module Learning with Rounding

NIST National Institute of Standards and Technology

NSA National Security Agency

NTT Number Theoretic Transform

PKE Public Key Encryption

PQC Post Quantum Cryptography

PRNG Pseuduorandom Number Generator

RISC Reduced-instruction-set Computing

RLWE Ring Learning with Errors

RSA Rivest, Shamir, and Adelman

RTL Register transfer level

SHE Somewhat Homomorphic Encryption

SIDH Supersingular Isogeny Diffie Hellman

SVP Shortest Vector Problem

TPA Throughput Per Area

TRNG True Random Number Generation

XOF Extendable Output Function

Chapter 1

Introduction

Cryptographic protections will be required for all varieties of information technology from now into the far future, including large server-based applications and emerging edge computation and internet of things (IoT). Public key infrastructure, in which parties can communicate without a pre-established key, is a necessary component of cryptographic security. Public key ciphers generate a pair of related keys – one publicly available (pk), and one held privately (sk). pk is generated as a function of sk and other public system parameters. This calculation used to generate pk is written in such a way that it is computationally infeasible to recover sk from pk. A key exchange can be performed by having one party generate a public key pair, send the pk to the second party who then sends an encrypted payload that contains either the key itself or a value used to generate the key. The first party can then recover this value using sk. This value can then be used as a shared key for a more efficient secret key cipher.

Modern examples of public key ciphers include RSA (Rivest, Shamir, and Adelman), and ECC (Elliptic Curve Cryptography) [2]. These algorithms are based on the difficulty of integer factorization and discrete logarithms, respectively. In general, as computing power has increased the security of these algorithms is adjusted by increasing the key size. However, Shor's algorithm, introduced in 1995, can solve integer factorization and discrete logarithm problems in polynomial time, given a large enough quantum computer [3, 4]. This effectively means that RSA and ECC cannot be strengthened simply by employing larger keys – an

RSA or ECC private key of any size would be broken in a matter of days to hours. There have been many advances in quantum computing in recent years; Amazon and IBM have recently announced their own quantum computing services aimed at assisting researchers in development of quantum computing applications [5, 6]. The onset of quantum computing large enough to break current U.S. National Institute of Standards and Technology (NIST)-approved public key algorithms is estimated to occur within the near term [7].

Due to this threat, governments and industry, supported by academic research, have been investigating quantum secure public key infrastructure. One promising family of such algorithms is lattice-based cryptography (LBC). Proposed in [8], these algorithms are based on the hardness of solving shortest vector or closest vector problems in high-dimensional lattices. The U.S. National Security Agency (NSA) recently stated that they believe lattice-based systems hold the best promise for post-quantum secure algorithms [9]. This research focuses on one such algorithm called NewHope, which is based on Ring Learning With Errors (RLWE) [10]. This algorithm was investigated by Google for incorporation in browser technology in 2016 [11], and an updated version was submitted to the NIST Post Quantum Cryptography (PQC) Standardization Process [12]. Although NewHope was not selected by NIST as a final round candidate in 2020 [13], five of seven finalists are lattice-based schemes; therefore, architectural improvements to implementations of NewHope are relevant, whether or not this exact specification of NewHope is ultimately deployed in commercial applications.

In this research we investigate methods of improving the area and throughput of a register transfer level (RTL) FPGA implementation of NewHope512cpa v.1.1 [14], which is an asymmetric cryptosystem used for key establishment equivalent to 128-bit key strength and resistant to chosen plaintext attacks (CPA). The two primary motivating research questions are:

1. How can the performance of post quantum key exchange be improved for high through-

put applications?

2. Can we make performance and area improvements to LBC by selecting an alternate Pseudorandom Number Generator (PRNG)?

In total we realize four Field Programmable Gate Array (FPGA) implementations of NewHope512cpa:

1. A baseline implementation following the standard specification
2. A pipelined implementation following the standard specification
3. A Trivium-based implementation
4. A pipelined Trivium-based implementation

Our contributions in this work are as follows:

1. We develop functionally verified RTL implementations of the NewHope512cpa PQC key exchange mechanism (encryption and decryption) using the Verilog Hardware Description Language (HDL).
2. We develop a pipeline strategy to split the encryption stage into 8 layers and the decryption stage into 6 layers, which results in dramatically increased throughput with only small gain in area. The pipelined implementations achieve both the highest throughput and throughput per area (TPA) to date for encryption in this mode.
3. We investigate the area reduction that can be achieved by the replacement of the SHA-3 extendable output function (XOF) SHAKE, primarily used to generate pseudo-random numbers, with the lightweight Trivium pseudo-random number generator (PRNG). The Trivium-based implementations reduce the Look Up Tables (LUT) used in the design by 45.7% for the regular design and 32% for the pipelined.

4. We show how these design decisions improve the energy efficiency of encryption by 8.58× and decryption by 3.74×.

This paper is organized as follows: Chapter 2 provides relevant mathematical background; introductions to NewHope, the Trivium PRNG, and other relevant LBC candidates; and a review of previous work. Chapter 3 discusses the details of our pipelined implementation as well as integration of the Trivium PRNG, including justification. Chapter 4 presents the results of our four implementations, and provides comparisons to other relevant hardware implementations. Potential future research areas are discussed in Chapter 5, and we conclude this work in Chapter 6. The code created for this research can be found at: https://github.com/LBeckwith98/Newhope_Crypto.

Chapter 2

Background

In the effort to develop quantum secure algorithms, several different families of algorithms have emerged: code-based, multivariate, isogeny-based, hash-based, and lattice-based ciphers. In the third round of the NIST PQC standardization process, all finalist are lattice-based except for a single code-based public key cipher (Classic McEliece) and a multivariate based digital signature scheme (Rainbow) [13]. NewHope is based on the RLWE problem, a type of LBC. We will provide a brief description of the NIST PQC competition, the Trivum PRNG, and the PQC Public Key Encryption (PKE) families.

2.1 Mathematical Background

We will first provide an overview of relevant mathematical concepts such as algebraic structures and NP-hardness. The former is important for understanding the description of NewHope, and the latter for understanding how design decisions are made for cryptographic algorithms.

2.1.1 Algebraic Structures

The cryptosystem discussed in this paper uses elements from what is known as an algebraic ring. There are many different algebraic structures such as groups, rings, and fields. A group

is one of the most basic structures and consists of a set of elements with a operator under which the set is closed. A ring is a set, which is a group on addition, and also has a second operator called multiplication. A field is a set which is a group under both addition and multiplication. The specific definition of a ring is a set R that fulfills the follow requirements:

- There are two binary operators $+$ and $\cdot$
- $+$ is associative, commutative, has an identity, and has inverses
- $\cdot$ is associative and has an identity
- Multiplication is distributive with respect to addition

A ring of particular relevance to this discussion is the ring $R_q = \mathbb{Z}_q[x]/X^n + 1$. That is, the set of polynomials whose coefficients are in the range $[0, q-1]$ modulo $X^n + 1$. Since this is a ring, it is closed under polynomial addition and multiplication, but multiplicative inverses are not guaranteed. R_q is the algebraic ring that is used in the NewHope algorithm to implement the RLWE problem.

2.1.2 NP-Hardness

NP-hardness is a part of computational complexity theory that is used to classify algorithms. There are four classifications: P, NP, NP-Hard, and NP-Complete. P stands for polynomial-time solvable problems and NP for Non-deterministic Polynomial time problems. NP contains all problems which can be verified (but not necessarily solved) in polynomial time [15]. Informally, an NP-Hard problem is a problem that is at least as hard as the hardest problem in NP – though the problem itself does not actually have to be in NP. NP-Complete problems must be in NP, and are the hardest problems within NP. These

classifications help cryptographers determine which mathematical problems may be difficult enough on which to base a cryptosystem.

2.2 The Need for Post Quantum Cryptography

Public key cryptography originated in 1977 with the development of the RSA algorithm described in [16]. RSA bases its security on the difficulty of large integer factorization. That is, given a large integer $n = p * q$ where p, q are large primes, recover p and q. This may not seem challenging, but there is no efficient algorithm to do so, and thus is very difficult to solve on classical computers for large integers. Another more recent development is Elliptic Curve Cryptography (ECC) [17], which builds its security upon the intractability of performing discrete logarithms on elliptic curves, and is useful for key exchanges and key agreements. An example of the discrete logarithm problem over $\mathbb{Z}_p$ is: Let x be an integer. Given a,b $\in \mathbb{Z}_p$ where $b = a^x \ mod \ q$, determine the value of x.

These public key algorithms can provide security and confidence in communication by allowing users to share secret keys over insecure networks and by creation of digital signatures. Key sharing can be performed by using these algorithms to define a Key Encapsulation Mechanism (KEM). Symmetric key algorithms, which allow data to be efficiently and securely encrypted, require both parties to share the same secret value. Getting the shared value to the other party is not a trivial task; without PKE it may require physical transportation of the value to be secure since the key cannot be sent over a public network without risk of compromise. However, using a KEM, this value can be shared over public networks with confidence that no observer or attacker could determine what the secret value is. The other service these algorithms can provide is a digital signature. Due to the global nature of internet communication, a user must be careful to verify that they are truly communicating

with whom they think they are. Digital signatures allow a message receiver to verify the authenticity of the sender. The above services serve as the foundation of much of our current security systems.

These algorithms have various security levels that can be chosen based on the needs of the situation. This is usually done by increasing the key size. According to [18], the current security strength requirement for PKE is 112 bits, and by 2030 all systems should be running at least 128-bit secure algorithms. For RSA this means equivalent key sizes of 2048 and 3072 bits respectively; for ECC these become 224 and 256 bits respectively. One may notice that the key size required to achieve 128-bit security is much larger than 128 bits. That is because there are algorithms that exist which can reduce the difficulty of the underlying security problem.

While integer factorization and discrete logarithm problems are difficult, there are algorithms that are more efficient than the naive, or "brute-force" approach. If one could easily factor a very large number composed of two primes, they could trivially break RSA. Currently the best tools for solving these problems are algorithms such as the general number field sieve (GNFS) [19] and the quadratic number field sieve (QNFS), though the best algorithm depends on details such as the size of number being factored [20]. For discrete logarithms, Index Calculus is currently the best method [21].These algorithms contribute to the difference between bit strength and key size.

The above attacks are all based on classical computing which fundamentally operates at the bit-level. All these operations are ultimately composed of bits that can have one of two binary states, 1 or 0. Quantum computers do not operate using traditional binary bits, but rather quantum bits (referred to as qubits) and utilize their quantum mechanical properties such as entanglement, superposition, and interference [6]. The exact details of quantum computing are not necessary for our discussion, however it is important to understand how

this technology enables the usage of Shor's algorithm originally described in [22]. A brief description of Shor's algorithm is as follows: Let N be the composite number that is to be factored. First, pick a random number $a \in (1, N)$ and find the greatest common divisor of N and a. Check that $gcd(a, N) \neq 1$. If it does equal 1 you have already found a non trivial factor. Otherwise, find the value p such that $a^p = m * N + 1$ for some integer m. If p is odd or if $a^p \equiv -1 \ (mod \ N)$ then restart because the next step will fail. Otherwise, calculating $gcd(a^{p/2} + 1, N)$ and $gcd(a^{p/2} - 1, N)$ will yield non trivial factors of N. This is because of the following equality:

$$(a^{p/2} + 1)(a^{p/2} - 1) = a^p - 1 = (m * N + 1) - 1 = m * N \tag{2.1}$$

Now, this algorithm can be performed on a classical computer. However, the issue arises that finding the value p such that $a^p = m * N + 1$ is incredibly difficult and thus prevents this algorithm from being useful. However, with clever usage of the quantum Fourier transform, a quantum computer can be used to find the period of repeating values of the equation $f(x) = a^x \ mod \ N$ which corresponds to this value p. This allows the algorithm to factor integers incredibly fast. If a quantum computer were created that could run this algorithm on large numbers, there is no feasible size of an RSA key which could remain secure. Since the intractability of discrete logarithms is approximately equivalent to the intractability of factorization of large integers, ECC would likewise be rendered vulnerable.

Quantum computers are being researched and developed by companies such as Intel, Google, and IBM. Estimates of when this technology will become useful are difficult. However, NIST has reported that some experts in the field believe that quantum processors will be large enough to break contemporary PKE within the next 20 years [7].

2.3 National Institute of Standards and Technology Competition

The NIST PQC standardization process, or "competition," motivated by the quantum threat previously discussed, was initiated by a call for submissions in December 2016, with a deadline of November 2017 for submissions. Since then, the first and second rounds have both been completed, and the third round is underway. Figure 2.1 shows an overview of the timeline of the NIST PQC competition thus far. In the first round, 82 entries consisting of PKEs, KEMs, and digital signature systems were received, 69 of which were accepted into the competition. After a period of research and evaluation, this number was reduced to 26 algorithms for the second round. In the third and final round, the number of candidates was reduced to 7 finalist, and 8 alternate algorithms. Upon the completion of the third round, NIST is expected to proceed with standardization using a candidate from among the 7 finalist, though the alternate candidates may be standardized in the future as well. NIST estimates the final round to take 12-18 months [23].

Figure 2.1: Post Quantum Cryptography Competition Timeline

This competition was motivated by the recent advances in quantum computing. While the threat is not imminent, NIST anticipates that the transition to quantum-secure algorithms will not be a simple process [24]. Thus there is a push to commence standardization of these algorithms as soon as possible, so that there will be sufficient time for migration to these new algorithms. Evidence of government intent to accelerate the transition to viable PQC solutions is the Defense Advance Research Projects Agency (DARPA) "Cryptography for

Hyper-scale Architectures in a Robust Internet Of Things (CHARIOT)," which is centered on PQC solutions for IoT [25]. The current cryptographic infrastructure required 20 years to establish, and aggressive estimates state that large scale quantum computers may be available within 20 years [7]. During this standardization process NIST has encouraged the cryptographic community to perform research on the security and efficiency of these algorithms. The standard will be chosen based on the criteria of security, cost (key/ciphertext sizes, etc) and performance, and any uniquely beneficial algorithm characteristics, with the most important criteria being security [13].

2.3.1 Algorithm Families

We will now provide an overview of the different families of cryptosystems in the NIST competition. While there are five families of PQC systems in the competition, the hash-based and multivariate schemes are limited to digital signatures and thus have less relevance to this discussion. We will discuss the three families present in the PKE/KEM component of the competition: lattice-based, code-based, and super singular isogeny. The lattice-based section will provide a more rigorous discussion of the mathematical background due to its relevance to the research presented in this paper.

Lattice-Based

The two fundamental lattice problems relevant to cryptography are the closest vector problem (CVP) and the shortest vector problem (SVP). A simple definition of SVP is as follows: given a lattice basis B, find the shortest nonzero vector in the lattice defined by B (note that there may be multiple shortest vectors with the same length). Figure 2.2 shows a simple visual of this problem. The dots represent the lattice points defined by the basis composed of

b_1 and b_2. Here s' represents a vector with the shortest possible length in this lattice. Given only the basis, there is not an efficient algorithm to find a shortest vector. While a solution may be obtained by "brute force" in this simple example, given a larger, multidimensional lattice this approach becomes infeasible.

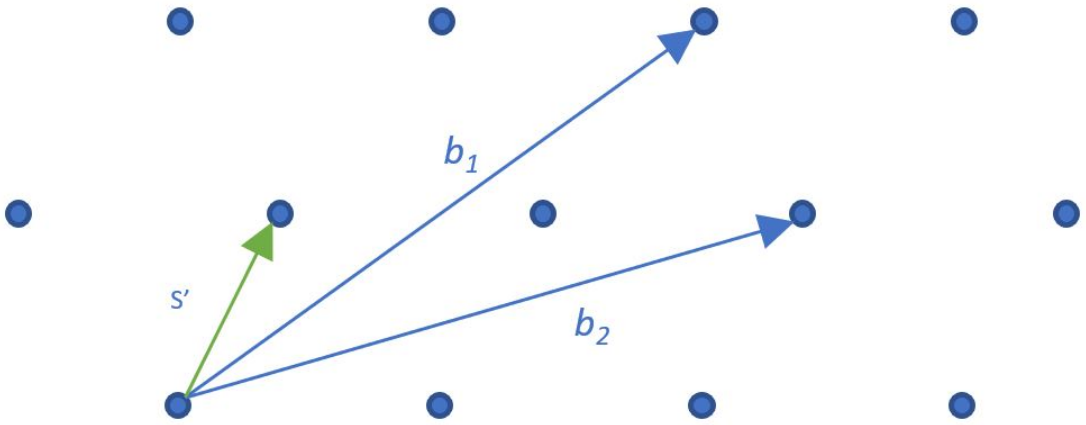

Figure 2.2: Shortest Vector Problem

A simple definition of CVP is: given a lattice B and a vector s', find the closest lattice vector to p_1. That is, find the vector p_1 in the lattice defined by B such that $\|s' - p_1\|$ is minimized. Figure 2.3 gives a visual example of the closest vector problem in two dimensions. As before, b_1 and b_2 define the basis and s' is the vector of interest. The goal is to find the closest point on the lattice to s'. That is, the point $p_i = c_1 * b_1 + c_2 * b_2$ such that the length of $s' - p_i$ is minimized. These lattice problems are known to be NP-hard and thus are a good foundation for building cryptosystems as previously discussed [14].

SVP and CVP are not used directly to create cryptosystems, but rather variants that rely on their security. NewHope is based on the RLWE problem, a variant of the Learning with Errors (LWE) problem which has been shown to be as hard as certain worst-case lattice problems [26]. The primary difference between RLWE and LWE is that RWLE is performed

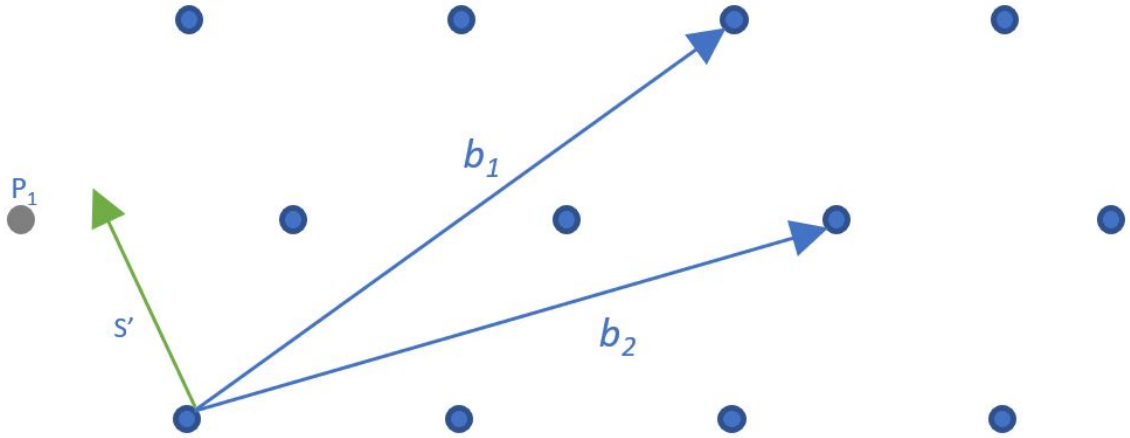

Figure 2.3: Closest Vector Problem

over an algebraic ring. In the case of NewHope, the ring elements are polynomials of degree n with coefficients in the range $[0, q-1]$ modulo the polynomial $X^n + 1$. For NewHope, n is either 512 or 1024 depending on the security level and $q = 12289$. We will refer to this ring as R_q. This additional structure provides several useful performance benefits, such as the ability to use the Number Theoretic Transform (NTT) (as discussed in appendix A.2) to reduce multiplication complexity from $O(n^2)$ to $O(n\ logn)$ as well as a reduction of the size of the public key [27].

The basic mathematical description of the RLWE problem is as follows: Let $s \in R_q$, be secret, $a \in R_q$ be chosen uniformly, and $e \in R_q$ be a small error vector. Given the pair (a, $b = a * s + e$), recover the secret value s. This problem is a form of the CVP: Let $b' = a * s$. Since e is a small error vector, b is close to b'. If it were possible to recover b' from b, it would be possible to recover the secret s with knowledge of the public parameters. However, as discussed previously the CVP cannot be efficiently solved, so b' cannot be efficiently found from b. Thus (a, b) can safely be released and used as a public key for encryption.

A basic mathematical explanation of a key exchange using RLWE is as follows: Suppose Alice

wants to receive a 256-bit secret value from Bob to use for a secret key cipher. First she selects a public parameter $a \in R_q$ and a secret value $s \in R_q$. She then generates $b \leftarrow a * s + e$ where $e \in R_q$ is a small error value. She sends (a, b) to Bob as the public key. Note that this is a case of the RLWE problem and therefore her secret s cannot be easily recovered from (a, b).

To send an encrypted value, Bob takes a 256-bit message and encodes it into a polynomial $v \in R_q$ such that it is resilient to noise (e.g., in NewHope with $n = 512$ Bob assigns $v_i = v_{i+256} = 0$ if bit i is 0 and $v_i = v_{i+256} = q/2$ if bit i is 1). In order ensure that v is secure he sets up an RLWE style problem, but with an extra piece of information that will allow Alice to be able to recover v. Bob samples three noise values $s', e', e'' \in R_q$ and calculates

$$\begin{aligned} u &\leftarrow a * s' + e' \\ v' &\leftarrow b * s' + e'' + v \end{aligned} \tag{2.2}$$

neither of which can be directly solved without knowledge of s'. Both of these polynomials are sent to Alice as the ciphertext.

Alice receives the two polynomials, but does not know s' and thus cannot recover v from v' directly. However, since she knows s, she is able to remove the influence of s' as follows

$$\begin{aligned} & v' - u * s \\ &= (v + e'' + b * s') - (a * s' + e') * s \\ &= (v + e'' + (a * s + e) * s') - a * s' * s - e' * s \\ &= v + e'' + (a * s * s' - a * s' * s) + (e * s' - e' * s) \\ &= v + e'' + (e * s' - e' * s) \end{aligned} \tag{2.3}$$

which leaves v plus a small amount of random noise. Due the resilience of the encoding of

the message, this noise can be removed during decoding and the original message can be recovered.

Code-Based

The third round code-based contender in the NIST PQC competition is Classic McEliece as defined in [28]. Classic McEliece is built using binary Goppa codes, a type of error correcting code, and provides a high confidence in its security, since it was first introduced in 1978 and has not had any major exploited vulnerabilities thus far. It also is easily configured to different security levels based on the parameter set; the authors presented 30 implementations each with different parameters sets and different security levels.

However, the high security confidence comes at the cost of performance which is much worse than other families. The fastest FPGA implementation reported in the NIST submission of Classic McEliece reported 202,787 cycles for key pair generation, 2,720 cycles for encryption, and 10,023 cycles for decryption with a maximum frequency of 28.6 MHz [28]. The key generation in particular is substantially slower than other types of PQC. The key and ciphertext sizes are much also larger than other system; the public key is 261,120 bytes, the private key is 6,452 bytes, and the ciphertext is 128 bytes.

On a very high level, McEliece uses a similar encryption concept as LWE; it uses noise to conceal the sensitive information in a way that only the receiver can undo. A public parameter, G', is released as the public key. Then if a message M is to be encrypted, the user calculates $M \cdot G' + e$ where e is a small amount of noise. The primary difference is that G' is generated by taking the generator matrix G for a binary Goppa code and multiplying it by a permutation matrix and a random invertible matrix [29]. This allows the sender to encode the message in a way that is recoverable for the receiver with G, but very difficult

for an attacker without this secret information.

Super Singular Isogeny

There was a single Super Singular Isogeny submission that made it to the second round of the NIST competition called SIKE, defined in [30]. It is based on Supersingular Isogeny Diffie-Hellman (SIDH) which follows a similar protocol to Diffie-Hellman using special elliptic curves. SIDH, like elliptic curve cryptography, utilizes a set of operations that form a group over the elements of an elliptic curve.

Unlike the previously discussed code-based systems, SIDH algorithms have reasonable performance and very small key sizes. The highest performance implementation running on an Artix-7 FPGA was able to perform key generation in 4 milliseconds, encapsulation in 7.01 milliseconds, and decapsulation in 7.42 milliseconds [30]. For a 32 byte payload in the compressed version, the public key is only 335 bytes, the secret key 602 bytes, and the ciphertext 410 bytes [30].

2.3.2 Round 3 Lattice Based Candidates

As previously mentioned, LBC is the leading contender for a PQC solution. There are 3 KEM or public-key encryption finalists in the NIST competition which are LBC: Kyber [31], NTRU [32], and Saber [33].

The NTRU submission is based on a much earlier specification published in 1998 [34], which builds its security upon SVP. Like NewHope, the NIST submission of NTRU uses SHA-3's hash and the XOF functions. While the NIST submission does not specify how to perform polynomial arithmetic, the 1998 submission uses the NTT to perform polynomial multiplication, so there is potential for results of our research to be applied to NTRU as

well.

The most recent Kyber specification includes two variants: the original proposal which uses SHA-3 (like NewHope), and a "90s" variant which uses AES-CTR and SHA-2 instead. The authors noticed that several implementations had replaced SHA-3 with older standards to increase performance, and decided it would be safer to release a standard rather than having numerous different implementations [31]. Kyber is based on the Module-LWE (MLWE) problem rather than the RLWE problem. RLWE provides more structure than plain LWE which allows for performance benefits. However, there is also concern that this structure may lead to more efficient attacks. MLWE splits the difference by providing more structure than LWE but less than RLWE [31]. The details of the algorithm differ slightly from NewHope, but the general structure reflects the same strategy as described previously in the RLWE section. Kyber also uses the NTT for polynomial multiplication and Centered Binomial Distribution (CBD) sampling like NewHope; therefore much of the work described in this paper could be applied to an FPGA implementation of Kyber as well.

Saber, defined in [33], is based on the Module Learning with Rounding (MLWR) problem rather than the LWE problem. MLWR is similar to MLWE, except that a rounding function is used to create the approximate equality rather than addition of random noise. The basic algorithm structure of the algorithm is also very similar to that used in NewHope, with the primary difference being the rounding and the lack of NTT use. Saber utilizes a modulus with a power of 2 which allows for trivial reduction, but prevents the use of the NTT. However, Saber authors are able to maintain performance due to the circumstances of their multiplications; all multiplications occur between a random element and a small element. Thus they are able to implement a much simpler and more efficient shift-add multiplier. While the NTT is not used, Saber still utilizes the SHA-3 algorithm family, and its similar structure means that our pipelining research could have applications to this algorithm.

2.4 Pseduorandom Number Generators

Random numbers play a vital role in cryptographic applications. Without a truly unpredictable method of generating secret values, such as keys, the key space may be reduced which gives an advantage to an attacker. There are two types of random number generators used in cryptography: true random number generators (TRNG) and pseudo random number generators (PRNG). TRNGs use a non-deterministic source for entropy, often a physical quantity such as noise on an electrical circuit or the timing of a user action [35]. A PRNG is a deterministic function that takes a random seed as input and generates a stream of random/unpredictable values. In general, TRNGs are much slower than PRNGs and thus are used only to generate a small amount of randomness that is then used as the seed of a PRNG.

2.4.1 National Institute of Standards and Technology Statistical Tests

In order to ensure that a PRNG or TRNG is secure enough for cryptographic applications, NIST developed a suite of fifteen statistical tests to verify the randomness of the output of a PRNG/TRNG [35]. For cryptographic applications, a PRNG should have the qualities of uniformity and independence. Uniformity means that the frequency of 1's and 0's in a bit stream is approximately equal, independence means that subsequences of the bit stream do not reveal information about each other [36]. These tests cannot be used to verify that a PRNG is completely secure, however they can identify poor sources of randomness [36]. If a PRNG passes these statistical tests, it is likely useful for cryptographic applications.

2.4.2 Trivium Pseudorandom Number Generator

While algorithms can be intentionally designed for PRNG applications, they can also be constructed using existing cryptographic algorithms such as block ciphers, stream ciphers, or hash functions [36]. We investigated the use of a Trivium-based PRNG to replace the large SHAKE function used for pseudorandomness, motivated by the lightweight Trivium PRNG used in [37]. The core of the PRNG, the Trivium cipher, is specified in [38] with a more in depth design decision discussion in [39]. Trivium is a stream cipher built based on the well established principles used to create block ciphers, designed with a focus on efficient and flexible hardware implementations. The description of the algorithm can be seen in Algorithm 1; each element of s is in GF(2), so $+$ represents XOR and $\cdot$ represents AND. The algorithm is intentionally designed to ensure that no part of the state is used for at least 64 iterations of the cipher. This allows the algorithm to be highly parallelized for high speed encryption when more resources are available [38].

Our FPGA implementation of the Trivium PRNG can be instantiated with an arbitrary number of cores, each of which consumes a 128-bit seed and produces 64-bits per clock cycle. We use an instance with 2 cores, which consumes a 256-bit seed and produces 128-bits of pseudorandom data per cycle.

2.5 NewHope

NewHope was one of the 17 2nd round NIST PQC candidates [40]. There have been several previous variants of this algorithm, such as the submission to USENIX in 2017 [10] and the modified version published later that year called NewHope-simple [41]. We will focus on the most recent specification, [14], version 1.1 of the packet submitted to the NIST

Algorithm 1 Trivium Pseudocode [38]

for $i = 1$ to N **do**
  $t_1 \leftarrow s_{66} + s_{93}$
  $t_2 \leftarrow s_{162} + s_{177}$
  $t_3 \leftarrow s_{243} + s_{288}$
  $z_i \leftarrow t_1 + t_2 + t_3$
  $t_1 \leftarrow t_1 + s_{91} \cdot s_{92} + s_{171}$
  $t_2 \leftarrow t_2 + s_{175} \cdot s_{176} + s_{264}$
  $t_3 \leftarrow t_3 + s_{286} \cdot s_{287} + s_{69}$
  $(s_1, ..., s_{93}) \leftarrow (t_3, s_1, .., s_{92})$
  $(s_{94}, ..., s_{177}) \leftarrow (t_1, s_{94}, .., s_{176})$
  $(s_{178}, s_{279}, ..., s_{288}) \leftarrow (t_3, s_{178}, .., s_{287})$
end for

competition. This version will be referred to simply as NewHope. NewHope is designed to function as a KEM, which means that it is meant to be used purely for the establishment of a symmetric secret key. The simplest design is the Chosen Plaintext Attack secure (CPA) implementation. Under this threat model, the attacker has access to the encryption module and is able generate ciphertexts for a chosen plaintext. The other model is Chosen Ciphertext Attack (CCA) secure. In this threat model, the attacker can decrypt a chosen plaintext in an attempt to gain information about the secret. Since this type of attack directly interacts with the secret, it is more complex to secure against. The authors provide four modes of operation: 512-bit CPA secure (roughly equivalent to 128-bit key strength), 512-bit CCA secure, 1024-bit CPA secure (roughly equivalent to 256-bit key strength), and 1024-bit CCA secure [14]. The greater bit level improves the security strength by increasing the degree of the polynomials within the algorithm, however they all produce a 256-bit plaintext. This also results in roughly a 2× performance decrease.

A brief description of the key generation, encryption, and decryption algorithms from [14] is shown in Algorithms 2, 3, and 4 and described below. All single letter variables (e.g., a, s, e, etc.) represent polynomials in the ring R_q. Further, variables with a circumflex are

in the NTT domain to facilitate coefficient-wise multiplication. The NTT is the equivalent of the Fast Fourier Transform (FFT), but for operations on fields of integers. Polynomial multiplication can be efficiently performed as follows: $a * b = NTT^{-1}(NTT(a) \circ NTT(b))$ where $\circ$ represents point-wise multiplication. Traditionally, in-place NTT algorithms require bit-reversal, but in NewHope the forward transformation is always performed on random noise so this step can be omitted as an optimization [14]. Another optimization taken is to generate a directly into the NTT-domain. This is acceptable because a is a uniform polynomial, and the NTT maps uniform polynomials to uniform polynomials [14].

In the first party's use of key generation (see Alg. 2), the random seed is expanded into a public and noise seeds. The function H is defined as the SHAKE XOF standardized by NIST in [42]. $\hat{a}$ is generated from the public seed, and $\hat{s}$ and $\hat{e}$ are generated from the noise seed using the sampler, which samples points in a CBD using the ψ function. Parameter $\hat{b}$ is computed, and encoded with the public seed into the 928-byte public key pk, which is forwarded to a second party.

The compressing, decompressing, polynomial encoding/decoding functions are all used to reduce the size of the ciphertext and keys when transmitting. Polynomial encoding takes advantage of the small modulus. Since $q = 12289$ and $log_2(q) = 13.58$, the top two bits of 16-bit coefficients can be removed without losing any data. The polynomial encoder takes the 512 16-bit coefficients (requiring 1024 bytes to store) and removes the top 2 bits of each value, lowering the area needed to store the polynomial to 896 bytes. While the polynomial encoding does not result in any data loss, the compression process is more aggressive and thus does lose some information in return for a smaller output. This process is described in Alg. 5 and accomplishes the compression by performing coefficient-wise modulus switching from q to 8 [14]. That is, each 16-bit coefficient is transformed to a 3-bit value and compacted into a byte array of $3 * 512/8 = 192$ bytes. This data loss is acceptable when applied the

portion of the ciphertext created using the message because the lower bits represent noise that contribute little to the decryption process [14].

Algorithm 2 CPA PKE Key Generation

$seed \leftarrow random()$
$[pubseed, noiseseed] \leftarrow H(seed)$
$\hat{a} \leftarrow GenA(pubseed)$
$s \leftarrow Sample(noiseseed, 0)$
$e \leftarrow Sample(noiseseed, 1)$
$\hat{s} \leftarrow NTT(s)$
$\hat{e} \leftarrow NTT(e)$
$\hat{b} \leftarrow \hat{a} \circ \hat{s} + \hat{e}$
$pk = [EncodePoly(\hat{b}), pubseed]$
$sk = EncodePoly(\hat{s})$
return (pk, sk)

For encryption (see Alg. 3), the second party decodes pk to recover $\hat{b}$ and the public seed, which is used to regenerate $\hat{a}$. s', e', and e" are sampled from the CBD using a user-supplied randomly-generated *coin*. The plaintext μ is mapped onto an element of R_q as described previously and combined with the public key before being compressed and returned to the first party as the 1088-byte ciphertext c.

Algorithm 3 CPA PKE Encryption

Input: $pk \in \{0, ..., 255\}^{7*n/4+32}$, $\mu \in \{0, ..., 255\}^{32}$, $coin \in \{0, ..., 255\}^{32}$
$pubseed = pk[7 * n/4 : 7 * n/4 + 31]$
$\hat{b} \leftarrow DecodePoly(pk[0 : 7 * n/4 - 1])$
$\hat{a} \leftarrow GenA(pubseed)$
$s' \leftarrow Sample(coin, 0)$
$e' \leftarrow Sample(coin, 1)$
$e'' \leftarrow Sample(coin, 2)$
$\hat{t} \leftarrow NTT(s')$
$\hat{u} \leftarrow \hat{a} \circ \hat{t} + NTT(e')$
$v \leftarrow Encode(\mu)$
$v' \leftarrow NTT^{-1}(\hat{b} \circ \hat{t}) + e'' + v$
$h \leftarrow Compress(v')$
return $c = [EncodePoly(\hat{u}), h]$

Finally, during decryption, the first party decodes and decompresses the ciphertext, and uses their secret key sk to recover the plaintext μ as described in Alg. 4. This plaintext can then be used to generate a secret key for a symmetric cipher. These functions are the core of the algorithm and are used to define a KEM.

Algorithm 4 CPA PKE Decryption

Input: $c \in \{0, ..., 255\}^{7*n/4+3*n/8}$, $sk \in \{0, ..., 255\}^{7*n/4}$

1: $\hat{u} \leftarrow DecodePoly(c[0 : 7 * n/4 - 1])$
2: $h = c[7 * n/4 : 7 * n/4 + 3 * n/8 - 1]$
3: $\hat{s} \leftarrow DecodePoly(sk)$
4: $v' \leftarrow Decompress(h)$
5: $\mu \leftarrow Decode(v' - NTT^{-1}(\hat{u} \circ \hat{s}))$

return μ

Algorithm 5 Ciphertext compression

Input: $v' \in R_q$

1: $k \leftarrow 0$
2: $t \leftarrow \{0, ..., 255\}^8$
3: $h \leftarrow \{0, ..., 255\}^{3*n/8}$
4: **for** l from 0 to $n/8 - 1$ **do**
5: $\quad i \leftarrow (l << 3)$
6: $\quad$ **for** j from 0 to 7 **do**
7: $\quad\quad t[j] \leftarrow v'_{i+j} mod\ q$
8: $\quad\quad t[j] \leftarrow (((t[j] << 3) + q/2)/q)\&7$
9: $\quad$ **end for**
10: $\quad h[k + 0] \leftarrow t[0]|(t[1] << 3)|(t[2] << 6)$
11: $\quad h[k + 1] \leftarrow (t[2] >> 2)|(t[3] << 1)|(t[4] << 4)|(t[5] << 7)$
12: $\quad h[k + 2] \leftarrow (t[5] >> 1)|(t[6] << 2)|(t[7] << 5)$
13: $\quad k \leftarrow k + 3$
14: **end for**

Return: h

2.6 Related Work

2.6.1 Hardware Implementations of NewHope

There has been a significant amount of research published on the second round ciphers. Much of it has been focused on evaluating the efficiency and performance of these algorithms. There have been several full FPGA implementations of NewHope such as [43], [44], [45]. The first FPGA implementation of NewHope was proposed in [43], which implemented NewHope-Simple. However, the performance of this implementation was surpassed by [44] which was 4.6× better in terms of area-time product. The authors of [45] present a highly efficient implementation featuring a low-complexity NTT/NTT^{-1} which reduces the number of modular multiplications required. This modification allowed the authors to create a very efficient implementation of the NewHope NIST submission. The implementation that previously had the highest throughput was published in [46]. In this paper the authors performed a hardware study of all second round candidates. They implemented several algorithms, including NewHope, in full RTL and the remaining algorithms using a HW/SW codesign approach.

Additionally, there have been many interesting HW/SW investigations. In [47] the authors developed a cryptographic processor targeting lattice-based ciphers in the NIST competition. This work was implemented on a CMOS chip, creating a processor with accelerators for operations such as polynomial arithmetic, sampling, the NTT transformation, and SHA-3 based PRNGs.

Further, there have been several efforts to create instruction set extensions to accelerate LBC. In [48], the authors created extensions (ISE) for the VesRiscv, a RISC-V processor, targeting NewHope and Kyber. Similarly, the authors of [49] developed ISEs and tightly couple

coprocessors targeting NewHope, Kyber, and Saber for the CV32E40P RISC-V processor.

2.6.2 Software Implementations of NewHope

There has also been research into how to increase performance of PQC algorithms without development of custom hardware. In [50], the authors investigated the potential for a NewHope PKE to be accelerated using Graphical Processing Units (GPU) in order to protect privacy of users of facial recognition software. NewHope has many operations that are well suited for the parallel hardware a GPU provides. For example, each calculation of a NTT layer is independent and thus can be performed in parallel. All polynomial arithmetic operations are also independent. Their final implementation was 63× faster for encryption and 83× faster for decryption than their baseline implementation running entirely on the Central Processing Unit (CPU).

Finally, there has been research into optimizing NewHope and RLWE systems on smaller processors. In [51], the authors present a implementation target at the ARMv8-A processor which is becoming popular for smart phone and tablet devices. The authors took advantage of the SIMD operations to vectors operations such as polynomial arithmetic and the NTT. The end result is a 3.6× speedup for encryption and 5.8 for decryption.

2.6.3 Fully Homomorphic Encryption

Another interesting application of RLWE crytography is the field of fully homomorphic encryption (FHE). While the concept of FHE dates back to 1978, the first major breakthrough was in 2009 when Craig Gentry proposed a method of creating a FHE by applying his bootstrapping method to a somewhat homomorphic encryption (SHE) system [52]. Gentry used a noise-based cryptosystem (like RLWE) which allowed some homomorphic capabilities.

The issue that Gentry solved is that each homomorphic operation increases the amount of noise in the ciphertext which eventually makes it indecipherable. Gentry solved this issue by refreshing the ciphertext by "recrypting" it. This operation involves homomorphically encrypting a ciphertext and then using an encrypted version of the decryption key to remove the original encryption. This process removes the build up of noise, so as long as the system is capable of homomorphically evaluating its own decryption circuit plus one NAND gate it is theoretically possible to perform any operation [52].

Gentry's original work in bootstrapping a PKE was applied to a LBC based cipher. He selected LBC because they have relatively simple decryption circuits and provide additive and multiplicative homomorphisms [53]. While this original work was not efficient enough to be useful, there has been progress towards more efficient schemes. Currently, three of the most popular schemes are BGV which is based on the RLWE problem [54], FV which is also based on the RLWE problem [55], and GSW which is based on the LWE problem [56].

These FHE systems use algorithms from the same cryptographic family as NewHope and thus research into accelerating them on FPGAs has followed similar paths. A large part of the research has been focused on creating polynomial multiplication co-processors, such as [57] and [58] for BFV, [59] for a NTRU based system, and [60] for a RLWE SHE scheme called YASHE. Since these RLWE systems all use similar protocols, there may be application of our research to these FHE systems. FHE is an "up-and-coming" area of cryptograpgic research ripe for wide-scale transition, as evidenced by the DARPA Data Protection in Virtual Environments (DPRIVE) effort to develop FHE-capable computational accelerators [61].

Chapter 3

Methodology

We begin discussion of our high throughput implementation with an overview of the encryption and decryption pipelining algorithms before discussing several of the unique elements of the design. Our design is an RTL implementation written in the Verilog and VHDL targeting the Xilinx Artix-7 FPGA. RTL designs describe the design at the level of synchronous digital circuits, which can perform basic logical and arithmetic operations between values stored in registers. Additionally, our design takes advantage of the Block Random Access Memory (BRAM) units and Digital Signal Processing (DSP) slices present in the FPGA [62, 63]. The former is a piece of hardware which provides an efficient method to store a large number of bits. The latter is a specialized unit that provides a physical multiplier along with the capability of pre and post arithmetic/logical operators. The primary benefits of RTL/FPGA design is the ability to perform pipelined and parallel operations. Pipelined means that intermediate values of a calculation are stored their own designated registers, so rather than waiting for an entire operation to complete before starting the next, a new value can be loaded in each clock cycle. Parallel means that multiple operations can be performed at the same time to increase efficiency.

3.1 Encryption Pipelining

Fig. 3.1 shows the high level structure of the design. There are four inputs to encryption: the 256-bit random coin for the sampler, the 256-bit pubseed used to generate $\hat{a}$, the encoded public key polynomial, and the 256-bit message μ which is to be encrypted. The design consists of 8 stages that require 10 "steps" to complete. The additional 2 steps are due to the delay of the stacked NTT implementation which will be discussed further in the sub-components section. The slowest step is the stacked NTT which completes within 2,305 cycles. In these stages, all operations are performed on polynomial objects with 512 coefficients in the range $[0, q-1]$. The multiplication symbols $\times$ refer to a module performing point-wise multiplication on the two inputs using Montgomery multiplication. The details of Montgomery multiplication are discussed in appendix A.1.

Because they are running at the same time, each stage must be resource and data independent, i.e., modules and variables stored in BRAM cannot be shared by two different stages. The data independence is handled using buffer modules built on BRAM which ensure that one stage does not overwrite values the next is using. Resource independence is handled by duplicating modules for different stages. This comes at a relatively small cost due to the small area of individual components, with the exception of the PRNG. The PRNG requires too many resources to duplicate, and thus all processes requiring data from it must be handled in a single stage. In order to minimize the latency of this stage, the 8,192 pseudorandom bits required for each binomial sampler are buffered at the start of each step. In the baseline implementation the pseudorandomness is produced using a modified version of the open source SHA-3 implementation developed by OpenCores [64]. Once the required randomness is buffered, control of the PRNG is handed over to GenA which generates the public a parameter as defined in [14]. This stage also encodes the 256-bit message into a polynomial object and decodes the encoded public key into a polynomial object.

Algorithm 6 Bit Reversal Algorithm

Input: $s \in R_q$ **Output:** $z \in R_q$

1: **for** $i \in [0, n-1]$ **do**
2: $\quad j = \sum_{k=0}^{log_2 n-1}(((i >> k)\&1) << (log_2(n) - 1 - k))$
3: $\quad$ **if** $i < j$ **then**
4: $\quad\quad z[i] = s[j]$
5: $\quad\quad z[j] = s[i]$
6: $\quad$ **end if**
7: **end for**

Stage 1 performs the pre-processing step for s' and e' before the NTT transformation by performing point-wise Montgomery multiplication with precomputed γ values. It also adds the encoded message to the error vector e''. Stage 2 performs the NTT transformation on s' and e'. Stage 3 multiplies $\hat{a}$ and $\hat{b}$ by $\hat{s'}$. Note that in stage 4 the NTT^{-1} transformation does require bit reversal since the inputs are not random noise. However, this is handled by the polynomial multiplication module that calculates $\hat{b} \circ \hat{s'}$. The bit reversal is a coefficient-wise permutation of the polynomial as shown algorithm 6. This is implemented in the polynomial multiplier by reversing the bit order of the output address.

Stage 5 performs the post-processing step for NTT^{-1}. Stage 6 completes the calculation of v' by adding the public-key component. The two output polynomials, v' and $\hat{u}$ are compressed and encoded respectively to form the ciphertext. This implementation is able to output a ciphertext every step (every 2,305 cycles) with a latency of 10 steps (23,050 cycles).

3.2 Decryption Pipelining

The decryption process is simpler than encryption and thus leads to a design with fewer stages and lower latency (Fig. 3.2). The throughput is also limited by the NTT, so the time required for a step to complete is the same as encryption. The decryption pipeline is composed of

6 stages that take 7 steps to complete. Stage 0 handles the decoding of polynomials and decompressing of the second component of the ciphertext. Stage 1 multiplies $\hat{u}$ by the secret key and performs bit reversal to prepare for the NTT transform. Stage 2 and 3 perform the NTT and the post-processing step. Stage 4 performs polynomial subtraction, and the final stage performs the decoding process defined in the NewHope specification which yields the plaintext μ. This implementation is able to output a plaintext every step (every 2,305 cycles) with a latency of 7 steps (16,135 cycles).

3.3 Sub-components

For the baseline implementation, all modules match the functionality as defined in the NewHope specification. This section will discuss design details used to increase the throughput.

3.3.1 Stacked Number Theoretic Transform

The NTT is an important component of both the encryption and decryption algorithm. While it improves the performance of polynomial multiplication, it is complex to implement, and requires many cycles to complete nevertheless. Our implementation consists of a single dual port BRAM, a look up table for the precomputed ω values (i.e., "twiddle factors"), an address generator, and a single pipelined butterfly unit. During operation, the dual port BRAM is fully utilized; either 2 reads or 2 writes are performed on every cycle. For $n = 512$ there are 9 stages to the NTT, each requiring 256 calculations which use 2 coefficients. Thus it takes $9 * 2 * 256 = 4608$ cycles to complete. This is over $2\times$ the time that any other module requires to complete, however the module only requires approximately 350 LUTs,

200 Flip Flops (FF), and 3 DSPs. The low LUT usage is in part due to high utilization of the DSPs. DSPs contain multiple stages that give the ability to perform additions before the multiplication, and a logic or arithmetic operation afterwards [63]. By carefully designing the Montgomery reduction step described in Alg. 7, the entire pipelined calculation can fit in 2 DSPs. Despite the efficiency of the design, the cycles required to complete the transformation would have been a substantial bottleneck in the design. Other implementations have improved the NTT performance by splitting the input BRAM into multiple sections to allow higher data throughput, and/or by modifying the internal ω involved in the calculation to remove the need for the pre and post processing step (e.g.[45]). However due to the low area of our implementation, we can develop a "stacked" implementation consisting of multiple NTT modules. The stacked module consists of 2 NTTs, 4 BRAMs, and control logic to switch the BRAM access between input, output, NTT_0 and NTT_1. The structure of this module can be seed in figure 3.3. Due to the small area of the individual NTT modules, this allows a higher throughput with an acceptable area increase. The new throughput is 2,305 cycles with a latency of 4,610 cycles.

Algorithm 7 Montgomery Reduction

Input: $a \in \{0, ..., 2^{32} - 1\}$
$u = a * 12287$
$u = u \ \& \ (1 << 18) - 1$
$a = a + u * 12289$
return $(a >> 18)$

3.3.2 Polynomial Arithmetic

Another core component of NewHope is polynomial arithmetic. Addition and subtraction are straightforward operations performed on two values read from BRAM with the output being reduced using a single subtraction or addition by q when necessary. Multiplication is

slightly more complicated because it requires more robust reduction. As in the NewHope specification, this is performed using Montgomery multiplication and reduction which can be done efficiently using DSPs as previously stated. There is an optimization performed for the NTT pre and post processing steps: since the γ and γ^{-1} values are precomputed, they are computed in the Montgomery domain. This simplifies the hardware needed for the stages performing the processing for the NTT. Due to the additional operations needed for reduction in these calculations, all arithmetic operations are pipelined. Each calculation can be completed in 1,024 cycles (512 cycles to read input values, 512 cycles to write the results).

3.3.3 Data Buffers

In a pipelined design, it is necessary to ensure that writes from one stage do not interfere with reads from the next. It is also common to have output from one stage that is not needed until several stages later. In this design, these issues are handled by a buffer module, seen in figure 3.4. This module consists of a single BRAM unit partitioned into 2+ sections each consisting of 16-bit 512 address locations. This can be conveniently accessed by having the least significant 9 bits of the address access the elements of a polynomial, and the remaining most significant bits are controlled by the buffer module to determine which polynomial is being accessed. For a buffer with no delay, there are 2 partitions which switch between input or output. For a buffer with n stage delay, there are $n + 2$ partitions. The poly selector controls the $log_2(n + 2)$ most significant bits of the address and rotates between the polynomials like a circular buffer. For example, a buffer with a 3 stage delay would have an 11-bit address line where the bottom 9 bits are controlled externally and the most significant 2 bits are controlled by the poly selector module which cycles one polynomial each time shift is asserted.

Table 3.1: Comparison of SHAKE and Trivium

Module	E/bit (nJ/bit) @ 100 MHz	Max Freq. (MHz)	Bits/ Cycle	LUTs
Trivium PRNG	0.0166	159	128.00	813
SHAKE	0.4320	237	18.75	3452

3.4 Trivium Pseudorandom Number Generator Integration

In the baseline design, SHAKE is used as the source of randomness as described in the NewHope specification. This module requires a substantial portion of the implementation area (consuming over 3452 LUTs and 2772 FF, i.e., 50% of the LUTs of the design) while also having lackluster throughput (18.75 bits/cycle). A lightweight PRNG could increase throughput per area if it could be used without compromising security. We provide a brief justification for use of a Trivium based PRNG as a replacement for SHAKE. There are two issues that must be considered: 1) Is the Trivium based PRNG a good source for generating pseudo-randomness? and 2) Does replacing SHAKE with it clearly undermine any security claims made by the authors of NewHope?

We used an instance of the Trivium based PRNG with two cores that each consume a 128-bit seed and produce 64 bits per cycle. The quality of the randomness was verified using the NIST standard statistical tests as defined in [35]. 10 million bits of output were generated using a Vivado simulation with a PRNG instance with 2 cores. This mode consumes a 256-bit seed and thus matches the instance that is used in our original design. Additionally, it has been used before in [43] and [37]. The fact that it passes NIST tests and has been used in previous applications, supports its use in this work.

There are two applications of the SHAKE in the encryption process which we will address. First, the generation of the public parameter $\hat{a}$. It is argued in [65] that a strong PRNG is not required for secure generation of $\hat{a}$, but merely that $\hat{a}$ be chosen somewhat randomly. The authors of NewHope acknowledged this argument, but chose a more conservative route. However, thus far there does not appear to be an issue with this approach. For binomial sampling, no strong PRNG requirement is mentioned in the specification, and at least one other implementation [43] has used a similar Trivium based PRNG for the sampling of error vectors. We thus believe there is not a security concern associated with replacement in the encryption process. There is one other application of note for SHAKE in the encapsulation and decapsulation process. It is used to hash the shared value before use as a key. The NewHope specification states that the assumption for this step is that SHAKE is a "pseudo-random function" [14]. This operation could also be performed using Trivium, however this is also not a core part of the PKE algorithm. Depending on the application, the users may choose a key establishment method that is specific to their requirements.

Table 3.1 shows the area and performance results of the two PRNGs when synthesized. The Trivium based PRNG uses only 23% of the LUTs that SHAKE requires. Simply changing to the Trivium based PRNG results in a 32% reduction in the number of LUTs used for the pipelined encryption module. Additionally, it provides more pseudo-randomness per cycle than SHAKE (128 bits/cycle as opposed to 18.75 bits/cycle) which lowers the likelihood of the PRNG becoming the bottleneck if the throughput is increased. The smaller area and higher throughput of the Trivium based PRNG also leads to it requiring significantly less energy per bit of pseudorandomness produced.

The integration of the new PRNG is straightforward. The only modification is the removal of the *nonce* and counters appended to the seed. These were used for SHAKE because it is a XOF function, so the counter allowed a stream of output to be generated using a single

seed. For the Trivium PRNG, the internal state is updated with each output so the *nonce* and counters are not necessary.

3.5 Energy efficiency

We also investigated the relative energy efficiency between our designs, which is a useful metric for comparing cryptographic implementations, e.g., [66]. All elements of the FPGA (LUTs, FF, BRAM, DSPs) use power, and therefore are captured in this metric. Considering the lifespan of custom hardware, energy efficiency is also likely to be a primary concern of a designer. The cost of running a module for its lifespan is likely greater than the initial cost of creating it. Energy efficiency (nJ/bit) is calculated as Power(mJ/s)/Throughput(Mb/s), where power is measured during simulations of the synthesized design at 100 Mhz. The Throughput is also calculated assuming a clock of 100 MHz.

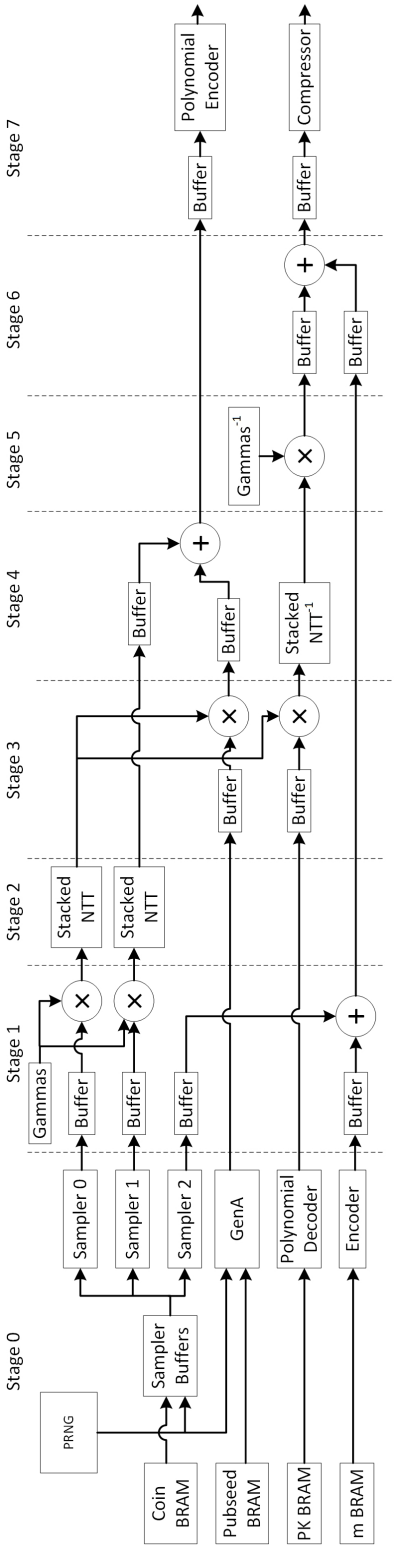

Figure 3.1: Pipeline Structure of the Encryption Algorithm

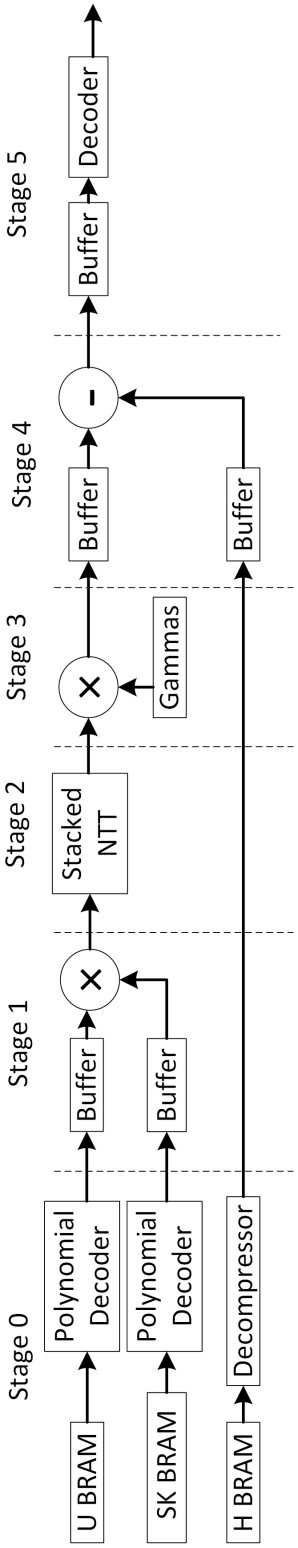

Figure 3.2: Pipeline Structure of the Decryption Algorithm

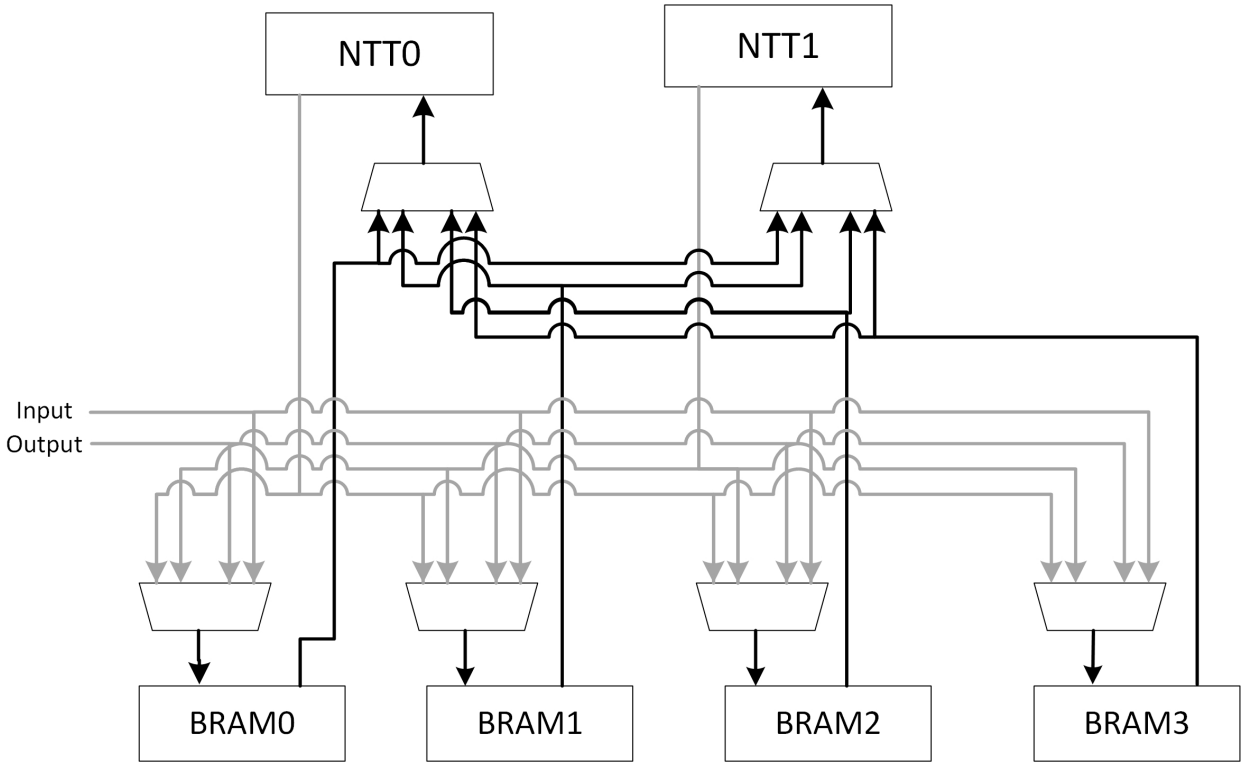

Figure 3.3: Stacked Number Theoretic Transform structure

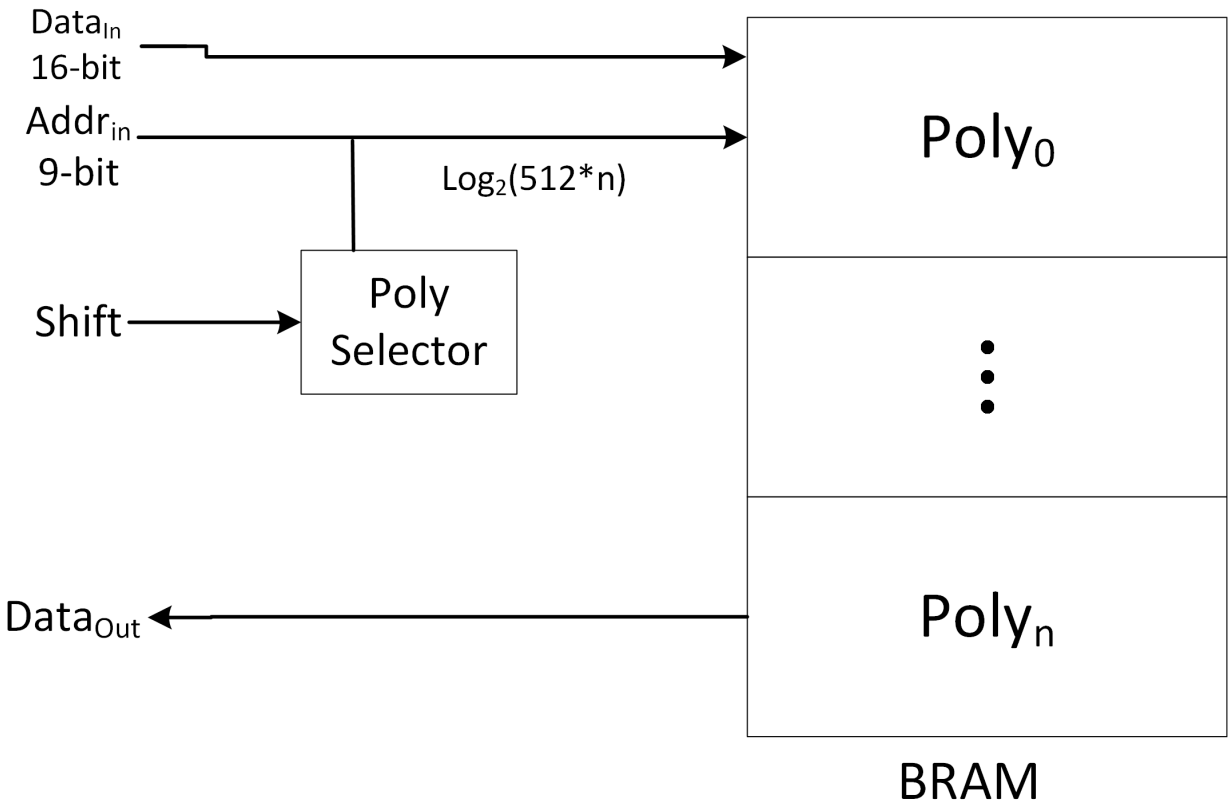

Figure 3.4: Data buffer structure

Chapter 4

Results and Analysis

FPGA implementations are developed with RTL in mixed Verilog and VHDL, using the Xilinx Vivado Design Suite. We present the area, performance, and power results of four implementations of 512-bit NewHope in CPA secure mode: a baseline strictly following the specification guidelines, a pipelined version also following the specification, and regular and pipelined implementations where encryption uses Trivium in lieu of SHAKE. Results are shown in Tables 4.1, 4.2, and 4.3, and include previous implementations for comparison. Some previous works split their design into two sections: one for encryption and one for decryption. This is specified by the labels above the data. The TPA calculation uses the number of LUTs for the specific component when provided; otherwise it uses the total number.

4.1 Pipelining Impact

Pipelining of the encryption module increases the throughput by 15.3× while maintaining a comparable maximum clock frequency. To our knowledge, the resulting encryption implementation has a higher throughput than any other published FPGA implementation of NewHope. The decryption throughput is also improved by 7×. This comes with a moderate increase in area. The growth in encryption resources was: 51.6% LUTs, 48% FFs, 250% DSPs, 675% BRAM. The increase in BRAMs is due to the need to buffer a large number

of polynomials; the increase in DSPs was due to the increase in NTT modules and polynomial multipliers. The growth in decryption resources was: 56% LUTs, 51% FFs, 66% DSPs, 400% BRAM. While area increases are significant, they are outweighed by the 15× improvement in throughput for encryption and the 7× improvement for decryption. This is confirmed by TPA ratios (based on Mbps/LUT), which are dramatically higher for the pipelined implementations.

Scheme	Cycles (k)	Cycle Latency (k)	Freq (MHz)	Time (μs)	Ciphertext Size (bits)	Throughput (bits/μs)	TPA (Kbits/(LUT×s)
NewHope512cpa **(this work)**	Enc/Dec 23.3/11.5	-	Enc/Dec 114/123	Enc/Dec 203.5/93	8704	Enc/Dec 42.7/2.7	Enc/Dec 8.77/3.06
NewHope512cpa Pipeline **(this work)**	Enc/Dec 2.3/2.3	Enc/Dec 23/13.8	Enc/Dec 172/169	Enc/Dec 13.3/13.6	8704	Enc/Dec 652.2/18.82	Enc/Dec 88.2/13.4
NewHope512cpa with Trivium **(this work)**	Enc/Dec 23.311.5	-	Enc/Dec 112/123	Enc/Dec 208/93	8704	Enc/Dec 41.8/2.7	Enc/Dec 15.81/3.06
NewHope512cpa Pipelined with Trivium **(this work)**	Enc/Dec 2.3/2.3	Enc/Dec 23/13.8	Enc/Dec 153/169	Enc/Dec 15.1/13.6	8704	Enc/Dec 580.7/18.82	Enc/Dec 115.3/13.4
NewHope512cpa [46]	Key/Enc/Dec 1946/3061/1487	-	225	Key/Enc/Dec 8.6/13.6/6.6	8704	Enc/Dec 639.8/38.7	Enc/Dec 71/4.3
NewHope512cpa [45]	Key/Enc/Dec 4.2/6.6/2.5	-	200	Key/Enc/Dec 21/33/12.5	8704	Enc/Dec 263.8/20.5	Enc/Dec 38.9/3.02
NewHope1024cpa [45]	Key/Enc/Dec 8/12.5/4.8	-	200	Key/Enc/Dec 40/62.5/24	17408	Enc/Dec 278.5/10.7	Enc/Dec 41.1/1.57
NewHope-simple 1024-bit [43]	Enc/Dec 179/55	-	Enc/Dec 117/125	Enc/Dec 1532/442	17408	Enc/Dec 11.35/0.58	Enc/Dec 2.52/.11
NewHope (2016) 1024-bit [44]	Client/Server 10.3/2.8	-	Enc/Dec 131/133	Enc/Dec 78.6/21/1	17408	Enc/Dec 110.7/12.1	Enc/Dec 5.9/.58
NTRUEncrypt SVES (256-bit Security) [67]	Enc/Dec 3743/4186	-	297	Enc/Dec 12.6/14.09	16496	Enc/Dec 1308/18.16	Enc/Dec 36.94/.512

Table 4.1: Implementation Performance Results and Comparison

4.2 PRNG Substitution Impact

We also include implementations using the Trivium based PRNG with and without pipelining. Our non-pipelined implementation featuring the Trivium PRNG is smaller than any

Scheme	LUT	FF	DSP	BRAM	Device
NewHope512cpa **(this work)**	Enc/Dec 4875/896	Enc/Dec 3505/588	Enc/Dec 15/9	Enc/Dec 4/2	XC7A100T
NewHope512cpa Pipeline **(this work)**	Enc/Dec 7397/1406	Enc/Dec 5188/888	Enc/Dec 39/15	Enc/Dec 27/8	XC7A100T
NewHope512cpa with Trivium **(this work)**	Enc/Dec 2643/896	Enc/Dec 2161/588	Enc/Dec 15/9	Enc/Dec 3.5/2	XC7A100T
NewHope512cpa Pipelined with Trivium **(this work)**	Enc/Dec 5044/1406	Enc/Dec 3532/888	Enc/Dec 39/15	Enc/Dec 35/8	XC7A100T
NewHope512cpa [46]	9009	8786	4	12	XC7Z020
NewHope512cpa [45]	6780	4026	2	7	XC7Z020
NewHope1024cpa [45]	6781	4127	2	8	XC7Z020
NewHope-simple 1024-bit [43]	Enc/Dec 4498/5142	Enc/Dec 4635/4452	Enc/Dec 2/2	Enc/Dec 4/4	XC7A35T
NewHope (2016) 1024-bit [44]	Client/Server 18756/20826	Client/Server 9412/9975	Client/Server 8/8	Client/Server 14/14	XC7Z020
NTRUEncrypt SVES (256-bit Security) [67]	(Slices) 35,435	-	-	2	XCVU440

Table 4.2: Implementation Area Results and Comparison

implementation seen in literature. While its TPA is overshadowed by the pipelined version, a lower area implementation could still have applications in area constrained devices that do not require ability to rapidly handle a large number of key exchanges. The pipelined design featuring Trivium is noticeably smaller than the traditional pipelined implementation; it decreases by 32% LUTs and 32% FFs. While the throughput does decrease due to the limit of Trivium's critical path, it still results in a 30% increase in TPA. Should the Trivium PRNG be proven as a secure replacement for SHAKE, a valuable reduction in area would result.

Table 4.3: Energy efficiency Results. Calculated as (nJ/bit) @ 100 MHz for implementations in this work. “Pipl” is pipelined; “Trvm” is Trivium.

Design	Enc (nJ/bit)	Dec (nJ/bit)
NewHope512cpa	11.62	59.5
NewHope512cpa (pipl)	1.35	15.9
NewHope512cpa (Trvm)	5.01	59.5
NewHope512cpa (Trvm-pipl)	.94	15.9

4.3 Energy Efficiency Results

Our implementations also had a meaningful impact on energy per bit (nJ/bit) required to perform encryption/decryption as seen by the results in Table 4.3. At 100 MHz, pipelining improved energy efficiency by 8.58× for encryption and 3.74× for decryption when compared to the baseline. Additionally, the replacement of SHAKE with Trivium improved energy efficiency by 56% for non-pipelined, and 30% for the pipelined implementations.

We also investigated the power usage at several different frequencies. Specifically, we calculated the power usage for each implementation at 75, 100, and 125 MHz. These results verify that our design maintains these power benefits across many running frequencies. Figure 4.1 shows a plot of these results. This figure shows that the power usage of our pipelined and trivium based designs are consistently more efficient.

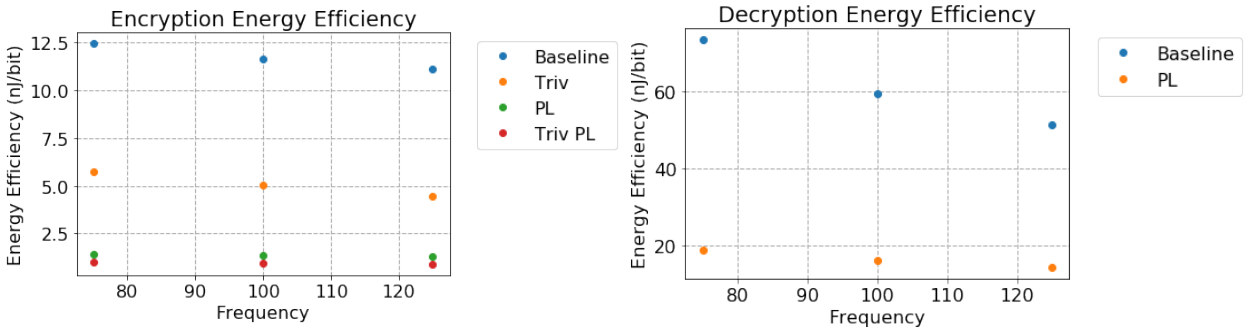

Figure 4.1: Energy Efficiency of Implementation at Various Frequencies

4.4 Comparison to Previous Work

Tables 4.1 and 4.2 include five different FPGA implementations of various NewHope specifications for comparison. We will directly compare our design for both encryption and encapsulation since they use the same hardware and differ by a very small number of cycles. We additionally provide a comparison with the most up-to-date NTRU implementation.

The highest performance implementation previously seen in literature is from [46]. This implementation is able to perform very efficient encryption and decryption, however our regular pipelined implementation has higher throughput for encryption (652 bits/μs vs 640 bits/μs) while consuming fewer LUT/FFs.

For completeness, we will also discuss the remaining implementations. The implementations from [45] feature a highly efficient NTT and NTT^{-1} design that allow the design to achieve impressive throughput. While their design has substantially lower latency, our pipeline technique allows our design to achieve higher throughput (1 ciphertext/23 μs). In an application handling a large number of connections, our higher throughput is beneficial. The implementation of NewHope-Simple from [43] is one of the earlier implementations. While it has a higher bit-strength ($n = 1024$) which naturally reduces the performance, the transformation from $n = 512$ to $n = 1024$ roughly doubles the number of calculations required and thus only doubles the latency. This can be observed in the two implementations provided in [45]. Even if our design's performance were halved, it would still provide over a 30× increase in throughput for a similar area. Similarly, [44] is another early implementation of NewHope. Using the same argument as above, out design provides over a 2× increase in performance with a much smaller area.

Finally, while many other PQC systems do not have as much research into hardware implementations as NewHope, [67] provides a efficient implementation of NTRUEncrypt which is

similar to the NTRU NIST submission. While its performance initially seems much better, when scaled by the area it can be seen that our implementation has better TPA performance.

Chapter 5

Future Work

In this section we will discuss future research directions relevant to the research presented in this work, along with any connections to the work we performed.

5.1 Application to other Post Quantum Cryptographic Systems

The pipelining and PRNG techniques we applied to NewHope have the potential to be applied to other quantum secure algorithms as well. There are many LWE cryptosystems which have slight differences with NewHope, such as different polynomials sizes and modulo values, but have a similar structure that could benefit from our research. One good candidate is the previously mentioned Kyber cryptosystem. While its security is based on MLWE not RLWE, it has a very similar structure. The public A parameter is generated using an XOF function, the error vectors are generated from the centered binomial distribution, and the encryption and decryption algorithms are very similar [31].

A brief description of the encryption and decryption is provided for reference. For encryption, the variables r, e_1, e_2 are sampled from the CBD, t is the public key, μ is the encoded message, and A is the public parameter. Then the encryption calculation is: $c_1 = NTT^{-1}(A^T \circ NTT(r)) + e_1$ and $c_2 = NTT^{-1}(t \circ NTT(r)) + e_2 + \mu$. These two values are then compressed

and encoded into the cipher text. The decryption is then performed similarly to NewHope, where c_1 is used in combination with the secret key to recover the message value from the noisy c_2 value.

The structure of Kyber is nearly the same as NewHope. Thus our pipelining technique could be applied to Kyber as well, and should the Trivium based PRNG be sufficiently secure it could be used for the generation of A and error vectors.

5.2 Polynomial Multiplication

Another interesting line of research is polynomial multiplication. Many of the RLWE algorithms take advantage of the NTT to reduce the complexity of polynomial multiplication. However, the NTT is itself a complex and large module. There has been research investigating how the schoolbook multiplier can be improved for LBC applications. In [68], the authors present a schoolbook polynomial multiplier that has been optimized for LBC. They exploit some of the unique circumstances of LBC (i.e., the fact that multiplications are performed against symmetric noise vectors) and are able to achieve optimizations such as fitting two parallel multiplications into a single DSP slice. The best implementation presented by the authors is able to perform a multiplication in 34K cycles, with a maximum frequency of 333 MHz, while only consuming 317 LUTs, 198 FFs, and a single DSP. This is slower than the polynomial multiplication performed in our implementation which take 10,240 cycles with a max frequency of 172 MHz (for NTT, point-wise multiplication, and NTT^{-1}). However, our design consumes 582 LUTs, 426 FFs, and 12 DSPs over the three modules to perform the same operation. There there might be ways to utilize the schoolbook multiplier in our pipelined design to increase the TPA by instantiating multiple multipliers as we did with our stacked NTT.

5.3 Implementation of Other Security Levels

Another valuable line of research would be to modify our design for other security levels. With no modifications, changing the security level to $n = 1024$ would lead to roughly half the throughput. This is because each operation accesses every polynomial coefficient. For example, for all polynomial arithmetic operations the current performances is 1,024 cycles since a cycle to read and a cycle to write is required for each coefficient of the polynomial. For $n = 1024$, the number of coefficients is doubled, so the new cycle count would be 2,048. A similar explanation applies to the compression functions. The number of cycles required by the NTT would be slightly more than doubled. Recall than the NTT consists of $log_2 n$ layers each performing n operations. Thus when n is doubled the cycle increase is $\frac{2nlog_2(2n)}{nlog_2(n)} = 2.22$ There would also be a slight area increase, but it would be minor since the main modification required is simply performing the same operations twice as many times. However, the pipeline design could be updated to increase the performance at the cost of increased area. For example, the stacked NTT could be expand to four NTT modules and six BRAM modules to keep the throughput at 2,306 cycles. Similar stacked implementations could be made to arithmetic/encoding/sampling modules as necessary.

The design could also be expanded to include the CCA secure algorithms. These algorithms use the CPA PKE encryption and decryption functions, so no modification of the current implementation would be required. However, the encapsulation function requires more utilization of the PRNG, and the decapsulation function uses the encryption function in addition to the decryption function in order to check the input. It would be valuable to see how our design impacts these factors.

5.4 Application to Fully Homomorphic Encryption

This work also has potential applications in the field of FHE. As previously discussed, many of the most modern FHE cryptosystems are based on RLWE or similar problems. Thus, the methods presented here could be used to improve the performance and area of FHE implementations. This is the current focus of DARPA DPRIVE [61], which is specifically targeting the creation of hardware accelerators for lattice-based FHE implementations in order to improve the performance. If enough performance improvements could be made to FHE systems to make them viable for large-scale use, it would greatly improve the security of systems such as the cloud.

Chapter 6

Conclusions

Post quantum cryptography will soon become a vital part of our digital security protocols. This work has presented hardware implementations of the NewHope quantum secure cryptosystem. The four implementations display how the performance, area, and energy efficiency of the design can be improved using pipelining, and by replacing the SHAKE XOF function with the lightweight Trivium PRNG.

The pipelining implementations dramatically improved the throughput for a comparable area. Our pipelined encryption implementation improves throughput by 15.2× with area growth of only 6.8× in terms of BRAMs and 52% in terms of LUTs, while improving energy efficiency by 8.58×. This is valuable for server-like applications where many connections must be handled at once, leading to a focus on throughput and energy usage. To our knowledge, we have produced the best known throughput and throughput-per-area ratios using algorithmic-level pipelining.

The modification of SHAKE XOF to the Trivium PRNG had a significant impact on the area of the design as well as reducing energy usage. This change improved energy efficiency by an additional 32% when compared to the regular pipelined version. Additionally, to the best of your knowledge the non-pipelined Trivium implementation achieves the smallest area to date. The change to a lightweight PRNG is valuable both in terms of reduction of power used to create a ciphertext and in reduction of area for resource constrained devices.

These implementations can support both lightweight devices which require low area and server-like applications that prioritize throughput and energy efficiency. Our innovations show how current and future PQC implementations can be improved.

Bibliography

[1] "A dft and fft tutorial," Feb. 2019. http://www.alwayslearn.com/DFT%20and%20FFT%20Tutorial/DFTandFFT_FFT_Butterfly_8_Input.html.

[2] N. I. of Standards and Technology, *FIPS PUB 186-4: Digital Signature Standard (DSS)*. NIST, Jul. 2013.

[3] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[4] Shor and P. W., "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, p. 1484–1509, Oct. 1997.

[5] AWS, "Quantum computing is now available on aws through amazon braket," 2020.

[6] IBM, "What is quantum computing?." https://www.ibm.com/quantum-computing.

[7] L. Chen, S. Jordan, Y. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone, *NIST 8105: Report on Post-Quantum Cryptography*. NIST, Apr. 2016.

[8] M. Ajtai, "Generating hard instances of lattice problems," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 3, 1996.

[9] "U.s. national security agency (nsa) weighs in on post quantum cryptography (pqc)," Jul. 2020.

[10] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange—a

new hope," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 327–343, USENIX Association, Aug. 2016.

[11] M. Braithwaite, "Experimenting with post-quantum cryptography," Jul. 2016.

[12] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, D. Stebila, M. R. Albrecht, E. Orsini, V. Osheter, K. G. Paterson, G. Peer, and N. P. Smart, "Newhope algorithm specifications and supporting documentation v1.02," Apr. 2020.

[13] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, *NISTIR 8309: Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST, Jul 2020.

[14] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, D. Stebila, M. R. Albrecht, E. Orsini, V. Osheter, K. G. Paterson, G. Peer, and N. P. Smart, "Newhope algorithm specifications and supporting documentation v1.1," Apr. 2020.

[15] "The classes p and np," 2016. https://web.archive.org/web/20160919023326/http://www.cs.uky.edu/~lewis/cs-heuristic/text/class/p-np.html.

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

[17] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[18] E. Barker, *NIST Special Publication 800-57 Part 1, Revision 5, Recommendation for Key Management*. NIST, May. 2020.

[19] G. Pandey, "A guide to general number field sieve for integer factorization," *Investigations in Mathematical Sciences*, vol. 4, pp. 83–98, 09 2014.

[20] C. Pomerance and P. Erdös, "A tale of two sieves," 1998.

[21] J. H. Silverman and J. Suzuki, "Elliptic curve discrete logarithms and the index calculus," 1998. https://link.springer.com/content/pdf/10.1007/3-540-49649-1_10.pdf.

[22] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," 1995.

[23] NIST, "Pqc standardization process: Third round candidate announcement," Jul. 2020. https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement.

[24] NIST, "Call for proposals," 2017. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals.

[25] "Sbir opportunity: Cryptography for hyper-scale architectures in a robust internet of things (chariot)," Aug. 2020. https://beta.sam.gov/opp/f5b34b42148b4fc5b02d3e418575a886/view.

[26] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *In STOC*, pp. 84–93, ACM Press, 2005.

[27] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings." Cryptology ePrint Archive, Report 2012/230, 2012. https://eprint.iacr.org/2012/230.

[28] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang, "Classic mceliece: conservative code-based cryptography," Mar. 2019. https://classic.mceliece.org/nist/mceliece-20190331.pdf.

[29] M. Marcus, T. Lange, and P. Schwabe, "White paper on mceliece with binary goppa codes," Feb. 2019. https://www.hyperelliptic.org/tanja/students/m_marcus/whitepaper.pdf.

[30] R. Azarderakhsh, M. Campagna, C. Costello, L. D. Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, and D. Urbanik, "Supersingular isogeny key encapsulation," Apr. 2020. https://sike.org/files/SIDH-spec.pdf.

[31] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, J. M. S. Vadim Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber algorithm specifications and supporting documentation (version 2.0)," Mar. 2019.

[32] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, J. M. Schanck, P. Schwabe, W. Whyte, and Z. Zhang, "Ntru algorithm specifications and supporting documentation," Mar. 2019. https://ntru.org/f/ntru-20190330.pdf.

[33] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Mod-lwr based kem (round 2 submission)," 2019.

[34] J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *Algorithmic Number Theory* (J. P. Buhler, ed.), (Berlin, Heidelberg), pp. 267–288, Springer Berlin Heidelberg, 1998.

[35] N. I. of Standards and Technology, *NIST SP 800-22: A Statistical Test Suite for Random*

and Pseudorandom Number Generators for Cryptographic Applications. NIST, Apr. 2010.

[36] W. Stallings, *Cryptography and Network Security: Principles and Practice*. USA: Prentice Hall Press, 7th ed., 2017.

[37] K. Zamiri Azar, F. Farahmand, H. Mardani Kamali, S. Roshanisefat, H. Homayoun, W. Diehl, K. Gaj, and A. Sasan, "Coma: Communication and obfuscation management architecture," Sep. 2019.

[38] C. D. Canniere and B. Preneel, "Trivium specifications," *eSTREAM, ECRYPT Stream Cipher Project*, vol. 2006, 2006.

[39] C. De Cannière, "Trivium: A stream cipher construction inspired by block cipher design principles," in *Information Security* (S. K. Katsikas, J. López, M. Backes, S. Gritzalis, and B. Preneel, eds.), (Berlin, Heidelberg), pp. 171–186, Springer Berlin Heidelberg, 2006.

[40] G. Alagic, J. Alperin-Sheriff, D. C. D. Apon, Q. Dang, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and Y. Liu, *NIST 8105: Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. NIST, Jan. 2019.

[41] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Newhope without reconciliation." Cryptology ePrint Archive, Report 2016/1157, 2016.

[42] N. I. of Standards and Technology, *FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. NIST, Aug 2015.

[43] T. Oder and T. Güneysu, "Implementing the newhope-simple key exchange on low-cost

fpgas," in *Progress in Cryptology – LATINCRYPT 2017* (T. Lange and O. Dunkelman, eds.), (Cham), pp. 128–142, Springer International Publishing, 2019.

[44] P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang, "High performance post-quantum key exchange on fpgas." Cryptology ePrint Archive, Report 2017/690, 2017. https://eprint.iacr.org/2017/690.

[45] N. Zhang, B. Yang, C. Chen, S. Yin, S. Wei, and L. Liu, "Highly efficient architecture of newhope-nist on fpga using low-complexity ntt/intt," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, pp. 49–72, Mar. 2020.

[46] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the nist post-quantum cryptography standardization process using hardware and software/hardware co-design approaches." Cryptology ePrint Archive, Report 2020/795, 2020. https://eprint.iacr.org/2020/795.

[47] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, pp. 17–61, Aug. 2019.

[48] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, "Isa extensions for finite field arithmetic - accelerating kyber and newhope on risc-v." Cryptology ePrint Archive, Report 2020/049, 2020. https://eprint.iacr.org/2020/049.

[49] T. Fritzmann, G. Sigl, and J. Sepúlveda, "Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography." Cryptology ePrint Archive, Report 2020/446, 2020. https://eprint.iacr.org/2020/446.

[50] P. Duong-Ngoc, T. N. Tan, and H. Lee, "Efficient newhope cryptography based facial security system on a gpu," *IEEE Access*, vol. 8, pp. 108158–108168, 2020.

[51] S. Streit and F. De Santis, "Post-quantum key exchange on armv8-a: A new hope for neon made simple," *IEEE Transactions on Computers*, vol. 67, no. 11, pp. 1651–1662, 2018.

[52] F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand, "A guide to fully homomorphic encryption." Cryptology ePrint Archive, Report 2015/1192, 2015. https://eprint.iacr.org/2015/1192.

[53] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, (New York, NY, USA), p. 169–178, Association for Computing Machinery, 2009.

[54] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping." Cryptology ePrint Archive, Report 2011/277, 2011. https://eprint.iacr.org/2011/277.

[55] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[56] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based." Cryptology ePrint Archive, Report 2013/340, 2013. https://eprint.iacr.org/2013/340.

[57] S. S. Roy, F. Turan, K. Jarvinen, F. Vercauteren, and I. Verbauwhede, "Fpga-based high-performance parallel architecture for homomorphic computing on encrypted data." Cryptology ePrint Archive, Report 2019/160, 2019. https://eprint.iacr.org/2019/160.

[58] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 353–362, 2020.

[59] E. Öztürk, Y. Doröz, E. Savaş, and B. Sunar, "A custom accelerator for homomorphic encryption applications," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 3–16, 2017.

[60] T. Pöppelmann, M. Naehrig, A. Putnam, and A. Macias, "Accelerating homomorphic evaluation on reconfigurable hardware." Cryptology ePrint Archive, Report 2015/631, 2015. https://eprint.iacr.org/2015/631.

[61] "Building hardware to enable continuous data protections," Mar. 2020. https://www.darpa.mil/news-events/2020-03-02.

[62] "7 series fpgas memory resources: User guide," Jul. 2010. https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf.

[63] "7 series dsp48e1 slice: User guide," Mar. 2018. https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[64] OpenCores, "Sha3(keccak)," Nov. 2017. https://opencores.org/project.

[65] S. D. Galbraith, "Space-efficient variants of cryptosystems based on learning with errors," 2012.

[66] A. Caforio, F. Balli, and S. Banik, "Energy analysis of lightweight aead circuits." Cryptology ePrint Archive, Report 2020/607, 2020. https://eprint.iacr.org/2020/607.

[67] F. Farahmand, M. U. Sharif, K. Briggs, and K. Gaj, "A high-speed constant-time hardware implementation of ntruencrypt sves." Cryptology ePrint Archive, Report 2019/322, 2019. https://eprint.iacr.org/2019/322.

[68] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp. 2459–2463, 2019.

Appendices

Appendix A

Supplemental Material

A.1 Montgomery Multiplication and Reduction

In many cryptographic algorithms, modular multiplication is a common calculation. However, the reduction step is costly when implemented directly. Montgomery multiplication and reduction allow calculation of $a * b \ (mod\ n)$ to be performed efficiently and in constant time. This is accomplished by transforming the operands into the Montgomery domain. Note that due to the overhead of this transformation, if only a single operation is being calculated it will not improve performance. However, if multiple calculations are being performed, or if one operand is a constant which can be precomputed into the Montgomery domain, it will be more efficient.

The first step is to select a residue $R > n$ such that R is relatively prime to n, and has properties that make division and remainder operations simple (e.g., a power of 2 which allows these operations to be performed with shifts and masks). A value can then be transformed into the Montgomery domain by calculating $\bar{a} = aR \ (mod\ n)$. The inverse R^{-1} is also calculated for use in multiplication, and in order to transform values out of the Montgomery domain. Multiplication can then be calculated as follows:

$$\bar{c} = \bar{a} * \bar{b} * R^{-1} = aR * bR * R^{-1} = abR \tag{A.1}$$

Montgomery reduction, which calculates $\bar{c}R^{-1}\ mod\ n$ can then easily be performed as follows:

$$\begin{aligned} &m = (\bar{c}\ mod\ R)k\ mod\ R \\ &t = (\bar{c} + mn)/R \\ &if\ t \geq N\ return\ (t{-}N),\ else\ return\ t \end{aligned} \tag{A.2}$$

where k is an integer such that $RR^{-1} = kN + 1$. Recall that R is a power of two, so modulus can be performed with a bit mas,k and division with shifts. Using this, modular multiplication in the Montgomery domain can be efficiently calculated by

$$\bar{c} = mont\_red(\bar{a} * \bar{b}) \tag{A.3}$$

Operating in the Montgomery domain greatly increase performance of these operations, while also allowing constant time operation in FPGA designs.

A.2 Number Theoretic Transform

The NTT is a generalization of the Fast Fourier Transform (FFT) performed over the ring $\mathbb{Z}/q\mathbb{Z}$ rather than over $\mathbb{C}$. This allows for much simpler calculations while still providing the benefits of the Fourier transformation. The primary benefit in this application is that polynomial multiplication can be accomplished by point-wise multiplication in the NTT domain. The mathematical description of the NTT algorithm on a polynomial g with N coefficients in the range [0,q-1] is:

$$\begin{aligned} &NTT(g) = \textstyle\sum_{i=0}^{N-1} \hat{g}_i X^i, \\ &\hat{g}_i = \textstyle\sum_{j=0}^{N-1} \gamma^j g_j \omega^{ij}\ mod\ q \end{aligned} \tag{A.4}$$

where ω is an n-th primitive root of unity and $\gamma = \sqrt{\omega}\ mod\ q$ [14]. While concise, the pure mathematical description does not necessarily provide a clear view of how this operation is actually being performed. Figure A.1 shows a visual of how this sum of sums is performed. Each vertex represents a single butterfly calculation that uses two of the coefficients as inputs and produces a single coefficient as an output. The γ and ω can be precomputed to decrease the complexity of the calculation.

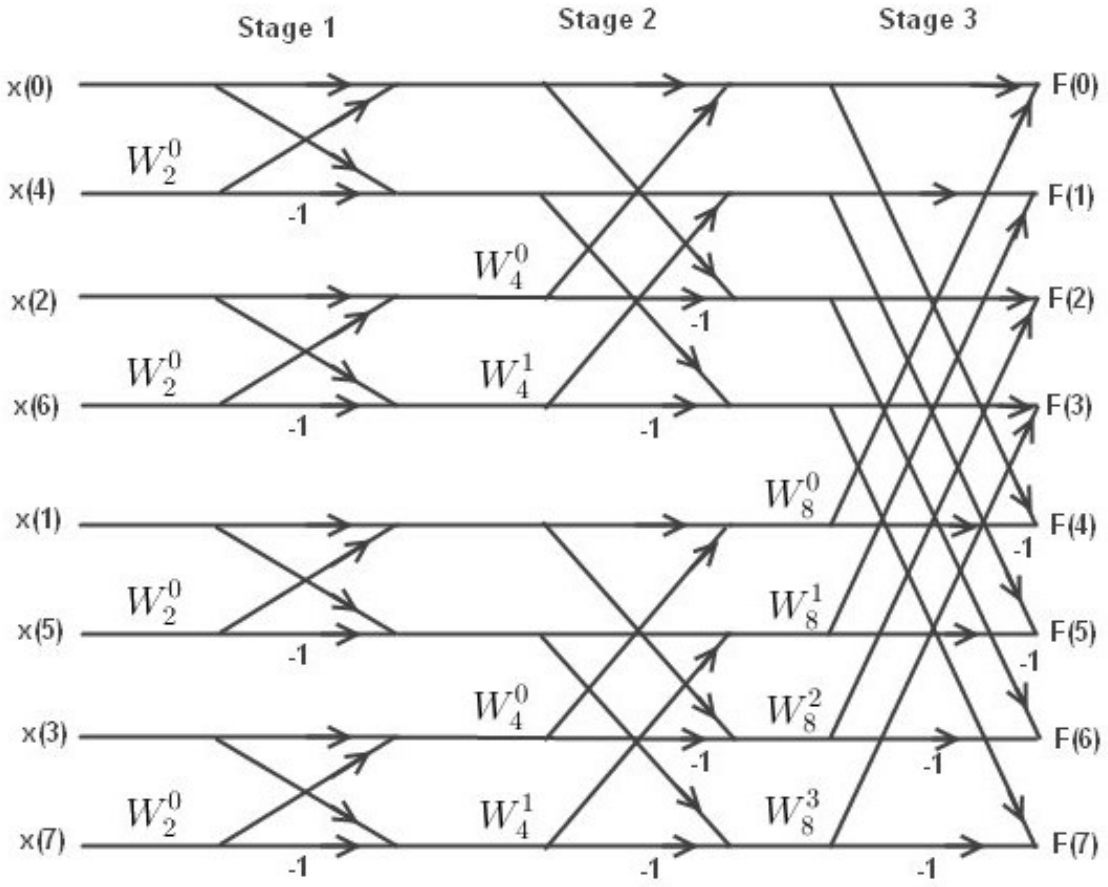

Figure A.1: Structure of FFT. Adapted from [1]