A Minimalistic Perspective on Hardware Designs for Modern-day Public-Key Cryptosystems

Siddhartha Chowdhury

A Minimalistic Perspective on Hardware Designs for Modern-day Public-Key Cryptosystems

Thesis submitted to the
Indian Institute of Technology Kharagpur
For award of the degree

of

Master of Science (by Research)

by

Siddhartha Chowdhury

Under the guidance of

Prof. Debdeep Mukhopadhyay

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
November 2022

CERTIFICATE

This is to certify that the thesis entitled "A Minimalistic Perspective on Hardware Designs for Modern-day Public-Key Cryptosystems", submitted by Siddhartha Chowdhury (19CS71P02) to Indian Institute of Technology Kharagpur, is a record of bona fide research work under our supervision and we consider it worthy of consideration for the award of the degree of Master of Science (by Research) of the Institute. To the best of our knowledge, the results embodied in this thesis have not been submitted to any other University or Institute for the award of any other Degree or Diploma.

Debdeep Mukhopadhyay
Professor,
IIT Kharagpur,
Kharagpur, West Bengal,
India– 721 302.

Date :

DECLARATION

I certify that

a. The work contained in the thesis is original and has been done by myself under the general supervision of my supervisors.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed the guidelines provided by the Institute in writing the thesis.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

e. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

f. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Siddhartha Chowdhury
Department of CSE,
IIT Kharagpur.
Date:

**“Imagination is more important than knowledge.
Knowledge is limited. Imagination encircles the world.”**
- Albert Einstein

Dedicated to my family and friends

Acknowledgements

On the verge of completing my M.S., I would like to thank all those whose presence in my life has kept me motivated and focused every single day over the span of the last three years. First and foremost, I would like to thank my supervisor, Prof. Debdeep Mukhopadhyay, for his constant support and encouragement. His persistence to do quality work has helped immensely in shaping this thesis. He has been ever encouraging and supportive whenever I explored new research directions or pursued new research ideas, continually inspiring and motivating me to face fresh new challenges each day.

I am grateful to Prof. Bhargab B. Bhattacharya, Prof. Aritra Hazra and Prof. Anindya Sundar Dhar for serving on my *Departmental Academic Committee* and for their valuable advice, suggestions and encouragement.

I had been extremely fortunate to be blessed with such loving and caring parents, my uncle and aunt. They have been my greatest source of inspiration and constant pillars of support. I would also like to thank my partner in crime Maithili for her immense support and encouragement over a long and arduous journey.

In my daily work, I have been blessed with a friendly and cheerful group of fellow students, researchers, lab-mates and roommates who all have been strong pillars of support, both academically, emotionally and psychologically. I owe it to them for having given me the encouragement and strength to slowly pave my way through my journey at IIT Kharagpur. A special vote of thanks to Debapriya da, Sarani di, Manaar da, Urbi di, Harishima di, Rajat da, Arnab da, Sikhar da, Pranesh da, Prabhat da, Sayandeep da, Brojo da, Anirban, Durba, Kuheli, Suvadeep da, Chandan, Shuvodip da, Shubham, Shreya, Bhuvnesh, Anupam, Soumi, Sayani, Animesh, Tishya, Shubhi, Debadrita, Ayushi, Nimish, Soumik, Abhishek and Smita for providing a wonderful lab environment. I will also like to thank Akashdeep, Bishakh, Soumyajit da, Soumyadyuti and Suman for making my stay at IIT Kharagpur enjoyable and fascinating. They have been a constant source of inspiration and motivation during my stay at IIT Kharagpur. But most of all I will like to thank Joydeep da for handling any kind of paperwork on my behalf and always keeping the lab environment lively.

I would also like to thank Sayandeep and Sunandan for inspiring and motivating me to pursue my dream of higher education. Finally, a very special note of thanks to our Honda Dio Scooter, which not only did help me save valuable time during my M.S. but also did provide me with countless memories of numerous long rides.

Siddhartha Chowdhury

Abstract

The modern-day IoT ecosystem has grown so large, that it has engulfed most of the crucial aspects of our life. Thus attention towards the security features of these devices has become imperative. To fulfil the demands of these resource-constrained applications, continuous efforts are needed to improve the design and implementation of the hardware root of trust which comprises complex cryptographic primitives, however, ensuring that the area and energy are minimally consumed. Strategies for porting the complex arithmetic operations of state-of-the-art public-key algorithms like ECC onto a lightweight hardware platform to accommodate other peripheral components on a single chip have always been a challenge. But now with the advancement of quantum computers, public key cryptography algorithms such as ECC and RSA are under imminent threat. Fortunately, significant research effort has already been undertaken to develop new alternative ”quantum-safe” (also called ”post-quantum cryptography” aka. PQC) PKC schemes and a standardization process is ongoing due to National Institute of Standards and Technology (NIST).

In this thesis, we have proposed hardware accelerators for pre-quantum and post-quantum public key cryptographic algorithms. Our works are unified based on their design methodologies, which are programability, scalability and targeted towards resource-constrained frameworks. As the first contribution to our thesis, we have introduced a lightweight hardware elliptic curve scalar multiplier based on NIST-approved Koblitz curve K-163. In this work, we developed several techniques to propose a minimal instruction set, centered around ADDN (Add and Branch if less than zero) based OISC (One-Instruction-Set-Computing) coupled with a lightweight comba characteristic-2 finite field multiplier to ensure the aggressive utilization of the FPGA resources.

Subsequently, as the second work for this thesis, we have developed a programmable and scalable architecture of a post-quantum lattice-based key exchange mechanism Kyber. We primarily address the three design goals while designing this architecture: lightweight implementation, reasonable latency of execution, and agility of the implementations, so that they ease the overall time-to-market. Even though lattice-based schemes are an advantageous choice for hardware implementations, we choose Kyber because of its smaller prime value as compared to other lattice-based key encapsulation mechanisms in the NIST post-quantum standardization process.

Finally, as the third contribution of our work, we have developed a unified architecture of a lattice-based key encapsulation mechanism Kyber and a lattice-based signature algorithm Dilithium, both of which are now considered standards for post-quantum key encapsulation schemes and digital signature schemes respectively. A high degree of mathematical and structural similarity between these two schemes has helped us in achieving a lightweight shared architecture. Our micro-code-based design methodology, where updating the memory can tune the design to operate on different security parameter choices we were able to bring forth the desired agility. From an application point of view, we have accelerated the sub-atomic operations of TLS 1.3 using the same set of instructions.

Keywords: Elliptic Curve Cryptography, Field Programmable Gate Array, Post-Quantum, Kyber, Dilithium, TLS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years due to the ubiquitous presence of embedded devices in our everyday life, with devices ranging from smart cards, mobile phones to modern cars, particular attention is imperative for the security features of these devices. To fulfil the demands of these resource-constrained applications, continuous efforts are needed to improve the design and implementation of the hardware root of trust, which comprises complex cryptographic primitives, ensuring that the area and energy are minimally consumed. More specifically, efforts have been made to develop compact implementations for public-key cryptographic algorithms, a principal ingredient in assuring integrity and non-repudiability in security subsystems. Strategies for porting the complex arithmetic operations of the state-of-the-art public-key algorithms onto a lightweight hardware platform to accommodate peripheral components on a single chip have always been a challenge. Field Programmable Gate Array (FPGA) is well suited to this purpose due to attributes like in-house reconfigurability, shorter design cycles, etc. Additionally, modern FPGAs are coupled with multiple high-performance modules like digital signal processing (DSP) blocks and block RAMs (BRAMs). Developing a lightweight implementation of public-key cryptographic algorithms with a balanced consumption of these different modules along with standard FPGA resources (slices and LUTs) is a challenging design problem. Therefore, a judicious design choice is needed to develop single-chip FPGA SOCs housing a crypto accelerator, which utilizes the various FPGA primitives uniformly, like LUTs, BRAMs, DSPs, etc. This becomes even more imposing when the developed lightweight architecture needs to exhibit a competitive area-time product value, a standard performance metric used for evaluating any design.

Elliptic Curve Cryptography (ECC) has always been a dominant choice for lightweight and fast implementation of public key cryptography as compared to RSA. But now with the advancement of quantum computers, public key cryptography algorithms such as ECC and RSA are under imminent threat. Fortunately, significant research effort has already been undertaken to develop new alternative "quantum-safe" (also called "post-quantum cryptography" aka. PQC) PKC schemes and a standardization process is on-going due to National Institute of Standards and Technology (NIST). Among different alternatives, the schemes based on worst-case lattice problems have received particular attention, mainly due to their simplicity and ease of implementation. The finalist portfolio of NIST indeed contains 5 candidates based on lattice problems.

We have broadly categorised our work based on FPGA-based hardware implementation of pre-quantum and post-quantum public key cryptographic algorithms. Our works are unified based on their design methodologies, which are programmability, scalability and targeted towards resource-constrained framework. In the first phase of our work, we have implemented a single instruction-based processor to execute an elliptic curve scalar multiplication targeted for resource-constrained IoT-based applications. In the second phase, first, we have implemented a post-quantum lattice-based key exchange in a way such that, the hardware can be appended with other lattice-based schemes. In the final work, we have appended the design with a lattice-based digital signature scheme to implement post-quantum TLS on an FPGA-based framework.

1.1 Motivation and Objective

Lightweight hardware implementation of public key cryptosystems has received great momentum with several efficient techniques for hardware design space exploration. Secure IoT landscape often requires such resource-constrained cryptographic implementations on FPGA-based platforms. FPGA-based hardware designs often provide the much-needed programmability and a unique combination of efficient yet compute resource-constrained designs. Moreover, as we are now moving from a pre-quantum to a post-quantum era, similar design techniques are also needed to be developed for post-quantum public key cryptography schemes in spite of several structural and mathematical differences. For instance one of the marked differences between pre-quantum

and certain post-quantum public key schemes is the introduction of error which at times makes these post-quantum schemes non-deterministic. The work done in our thesis is motivated by the much-needed reconfigurability and programmability provided by an FPGA-based framework and to develop a minimalistic design approach for pre-quantum and post-quantum cryptosystems. We propose several techniques for achieving resource-constrained implementations in terms of area-time product for such crypto algorithms.
The contributing chapters of our thesis are motivated by the following objectives:

- **Resource Constrained Elliptic Curve Cryptography**: Operations in elliptic curve cryptography are computationally expensive and are often found to be difficult to implement for resource-constrained environments. Efficient execution of protocols such as elliptic curve digital signature algorithm (ECDSA) or elliptic curve Diffie-Hellman key exchange requires efficient implementation of elliptic curve scalar multiplication. However, implementing computationally heavy operations with a low hardware footprint requires significant area-time trade-off. This motivates our first work where we have implemented a lightweight architecture for an elliptic curve scalar multiplier with reasonable latency of execution. The architecture uses FPGA resources like BRAMs, DSPs effectively to have a minimal footprint of FPGA LUTs, thus leaving room for other peripheral designs to be hosted in a single FPGA.

- **Resource Constrained Architecture of Kyber, a Quantum-Safe Key Encapsulation Scheme**: Present day public key cryptosystems such as ECC or RSA, will be broken by modern-day quantum computers. However, a class of quantum-secure lattice-based public key algorithms have emerged that thwarts the threat of a quantum analysis. But designing such post-quantum cryptosystems for a resource-constrained environment is extremely challenging and incurs significant area-time trade-off. In order to meet the requirement of designing such a programmable but area-efficient architecture of a post-quantum public key algorithm, we have chosen Kyber which is a finalist in the National Institute of Standards and Technology (NIST) competition to standardize post-quantum public key algorithms. Kyber is a lattice-based key encapsulation mechanism based on module learning with error (LWE) problem.

- **Resource Constrained Post-Quantum TLS**: However, if we consider a public key infrastructure, key encapsulation mechanisms are often accompanied by digital signature algorithms for an authenticated key exchange scenario. So as a second work in phase two we have updated the hardware design of Kyber with a lattice-based digital signature algorithm, Dilithium, which is also a finalist for the NIST post-quantum standardization. We have chosen Kyber and Dilithium owing to their high degree of mathematical and structural similarities. To bring forth the much-needed agility, we choose a micro-code-based design style, where updating the memory can tune the design to operate on different parameter choices. An immediate advantage of this design choice is that we can support different levels of security on the same hardware. We have extended our proposed architecture by implementing sub-atomic operations of TLS 1.3 using the same set of instructions.

1.2 Our Contribution

This section summarizes the major contribution of our thesis.

- **Lightweight Elliptic Curve Scalar Multiplier using Single Instruction Architecture**: In this work we introduce a lightweight hardware elliptic curve scalar multiplier based on NIST-approved Koblitz curve K-163. Koblitz Curves offer excellent optimization opportunities for characteristic-2 Elliptic Curve Cryptosystems (ECC). However, porting such choices onto a lightweight and cost-effective FPGA platform was a major challenge. The underlying characteristic-2 algebra is not aligned with the on-chip components like DSP multipliers, which if not utilized leads to large LUT counts of the designs. In this work, we developed several techniques to propose a minimal instruction set, centered around ADDN (Add and Branch if less than zero) based OISC (One-Instruction-Set-Computing) coupled with a lightweight comba characteristic-2 finite field multiplier to ensure the aggressive utilization of the FPGA resources. The architecture uses FPGA resources like BRAMs, DSPs effectively to have a minimal requirement for FPGA LUTs, thus leaving room for other peripheral designs to be hosted in a single FPGA.

- **Resource Constrained Architecture for Post-Quantum Key Encapsulation Mechanism Kyber**: In this work we have developed a programmable and scalable architecture of a post-quantum lattice-based key exchange mechanism Kyber. We primarily address the following three design goals while designing this architecture: lightweight implementation, reasonable latency of execution, and agility of the implementations, so that they ease the overall time-to-market. Even though lattice-based schemes are an advantageous choice for hardware implementations, we choose Kyber because of its smaller prime value as compared to other lattice-based key encapsulation mechanisms in the NIST post-quantum standardization process. To bring forth the much-needed agility, we choose a micro-code-based design style, where updating the memory can tune the design to operate on different parameter choices. An immediate advantage of this design choice is that we can support different levels of security on the same hardware.

- **Resource Constrained Architecture for Post-Quantum TLS**: In this work, we have developed a unified architecture of a lattice-based key encapsulation mechanism Kyber and a lattice-based signature algorithm Dilithium, both of which are now considered standards for post-quantum key encapsulation scheme and digital signature schemes respectively. We have practically amalgamated the hardware components of Dilithium with the existing hardware we have developed as the second work for our thesis. We exploit the structural and mathematical similarities between these two schemes to develop a lightweight shared architecture. We develop an agile design by following a micro-code-based design methodology, where updating the memory can tune the design to operate on different security parameter choices. Finally, we demonstrate the utility of the design in a real-life use case by accelerating a TLS .13 protocol.

1.3 Work Done and Thesis Organization

The thesis consists of six chapters. In this section, we present the organization of the thesis, along with brief descriptions for each chapter.

Chapter 1: Introduction

This chapter provides a brief introduction followed by motivation, objective, scope and main contributions of the thesis.

Chapter 2: Background and Literature Survey

This chapter presents a comprehensive survey of the underlying mathematics of Koblitz curve and the post-quantum cryptographic schemes Kyber and Dilithium. It also covers the necessary background required for this thesis.

Chapter 3: Lightweight Implementation of Elliptic Curve Scalar Multiplier

In this chapter, we present a detailed description of the proposed single instruction processor architecture for lightweight implementation of elliptic curve scalar multiplier. In this work, we have proposed single instruction processor architecture for lightweight implementation of elliptic curve scalar multiplier. Our design strategy involves a minimalistic single-instruction computing framework to design an ECC processor on FPGA with minimalistic consumption of different FPGA primitives (DSPs, BRAMs, LUTs, registers, slices), however, with competitive area-time performance. We focus on implementing a characteristic-2 Koblitz NIST curve in this current work. We develop a programmable architecture to perform scalar multiplication using two different algorithms. Based on τNAF proposed by Solinas in [50] to reduce the number of non-zero terms in the scalar, we have explored the scalar multiplication formula proposed in [50]. On the other hand, while using the Montgomery ladder technique proposed in [33], owing to x-coordinate only operations, we were able to compute using less number of field multiplications. In this work, we target to explore these trade-offs using a Turing complete `URISC` (Ultimate Reduced Instruction Set Computer) instruction, ADDN (Add the operands and branch if the answer is less than zero), to design a lightweight ECC scalar multiplier with few necessary modifications.

Chapter 4: Resource Constrained Architecture for Kyber

This chapter focuses on the lightweight and programmable architecture of lattice-based key encapsulation scheme Kyber. The complete hardware architecture consists of the datapath elements of Kyber along with the instruction and data memory. The instructions stored within the instruction memory control the sequence of operations needed to execute Kyber. The datapath integrates the logic and functional modules necessary to execute Kyber. We have also observed that we can obtain a significant speed-up with respect to the canonical software implementation of Kyber as certain modules can be realized in hardware with practically negligible overhead. The Encode and Decode modules of Kyber are rewiring operations in hardware which consume significant latency in software. Implementing the compress/decompress module of Kyber in hardware involves a few shifts and multiplication with constant in hardware. The multiple outputs produced by these modules are fed to a multiplexer unit connected to each individual module, and the appropriate value is extracted based on the value of the select line. Also, polynomial multiplication is one of the most computationally intensive components in Kyber. In order to develop our lightweight and fast multiplier architecture, we exploit the structure of the field prime. Kyber is based on a prime of the form $q = k \cdot 2^m + 1$, where $k = 13$ and $m = 8$ ($k < 2^m$). We exploit this specific structure of prime to perform K^2-RED [12] reduction for Kyber. The reduction algorithm does not require any multiplications in hardware and is achieved by a few shifters and adders. We have chosen a lightweight Xilinx board NEXYS 4 DDR which houses an Artix-7 FPGA and a soft-core microprocessor Microblaze to implement the key encapsulation mechanism.

Chapter 5: Resource Constrained Architecture for Post Quantum TLS

This chapter focuses on appending the design shown in chapter 4 with a lattice-based digital signature scheme Dilithium. In this work, we append the programmable architecture of Kyber with a post-quantum digital signature scheme Dilithium. Moreover, we focus on a widely used real-life security protocol, TLS 1.3 and attempt to realize a post-quantum version using our design. More precisely, we have replaced the key encapsulation and digital signature scheme with Kyber and Dilithium on an FPGA-based platform. A unique feature of the proposed architecture is a unified NTT-based polynomial multiplier coupled with a specially integrated Keccak core that is commonly available for Kyber and Dilithium execution. The Keccak core is capable of computing both SHAKE and SHA3 and is primarily used for both pseudo-random number generation and hash computation. Sharing these two compute-intensive modules responsible for core processing tasks requires a combined and highly optimized co-processor architecture. Additionally, a few auxiliary blocks are further optimized to accelerate both Kyber and Dilithium. The complete hardware architecture encapsulates a combined datapath of Kyber and Dilithium, along with the instruction and data memory. The instructions stored within the instruction memory control the sequence of operations needed to execute either Kyber or Dilithium as required. The datapath integrates the logic and functional modules necessary to execute Kyber or Dilithium. The architecture consists of a unified NTT multiplier and a Keccak core as part of the shared architecture of Dilithium and Kyber. Sharing these two compute-intensive blocks has helped us achieve a low area-time product for the whole design.

Chapter 6: Conclusion and Future Work

This chapter provides a summary of the important aspects of the thesis and proposes possible future research directions.

Chapter 2

Preliminaries

In this chapter, we cover the necessary background details of our work with a special focus on ECC and lattice-based cryptography. We start with the notion of elliptic curve cryptography with notable attention towards a specific kind of curve known as the Koblitz Curve. Koblitz curves are defined over characteristic-2 fields and the underlying field arithmetic is different as compared to the prime field curves. We start with a discussion of the various mathematical primitives and will extend up to the relevant elliptic curve scalar multiplication techniques. This chapter also talks about the basics of lattice-based cryptography with special attention towards the key encapsulation mechanism Kyber and the digital signature scheme Dilithium. We conclude this chapter with a discussion on TLS 1.3 with our focus towards amalgamating Kyber and Dilithium for the development of a post-quantum version of TLS.

2.1 Cryptography

Cryptography is the study of techniques which enables secure communication between two parties in the presence of adversarial third parties. It plays a crucial role in constructing and analyzing protocols that prevents any malicious entity to read the private messages. These protocols are designed around certain well known computational hardness assumptions, which makes them hard to break in actual practice by any adversary.

In practice, cryptographic protocols should adhere to the following security goals:

- *Confidentiality:* Preventing access to the data from any unauthorized party
- *Integrity:* Preventing any alteration of data by unauthorized party
- *Origin Authentication:* Confirming authentication of the source of the data
- *Entity Authentication:* Confirming authentication of the parties involved in communication
- *Non-repudiation:* Preventing a party from denying his previous commitments

2.1.1 Symmetric and Asymmetric Key Cryptography

Symmetric and asymmetric key cryptography are the two pillars of modern-day cryptosystems. In symmetric key cryptography, the communicating parties agree upon a single *key* before exchanging information. The *key* must be kept secret and authentic. The communicating parties agree upon a standard symmetric key encryption/decryption algorithm to achieve message confidentiality or verify the authenticity of the transferred data. The drawback of symmetric key cryptosystem lies in the distribution of the secret key. This is where asymmetric key cryptosystem or public key cryptosystem comes to the rescue.

Key distribution has always been a very crucial aspect of symmetric key cryptosystem. For the successful execution of symmetric key cryptography, the key must be shared between two parties via a secure channel. The secure channel can either be a trusted third party or a trusted courier. But trusted third parties are not practical for all scenarios. Moreover, there are no efficient secure key exchange mechanisms based on symmetric key cryptography.

Asymmetric key or public key cryptography addresses the above-mentioned issues. In public key cryptographic algorithms, instead of a single key, two separate keys are used, namely a public key and a private key. For all standard public key algorithms, it is computationally infeasible to compute the private key from the knowledge of the public key. The security of the public key algorithms relies on a computationally hard problem. Among all the public key algorithms in the literature, RSA [47] is one of the most popularly used schemes. RSA was originally proposed in 1977 by Rivest, Shamir and Adleman and is based on the integer factorization problem. On the other hand, elliptic curve cryptography (ECC) was proposed by Neal Koblitz and Victor Miller in 1985 and is based on elliptic curve discrete logarithm problem [27]. But compared to RSA, ECC offers more security per bit as compared to RSA. This makes ECC more suitable for lightweight implementations due to the limited usage of memory bandwidth and power consumption as compared to RSA.

But now with the finalization of the post-quantum candidates by the National Institute of Standards (NIST), new lattice-based key encapsulation and digital signature schemes have also emerged. In fact, generic key encapsulation and digital signature schemes are the two pillars of public key infrastructure. Amalgamating these two schemes, we obtain a secure scheme of exchanging keys between two parties. The lattice-based schemes are based on worst-case lattice problems such as learning with error. In the following sections, we will elaborate on the basics of ECC and the lattice-based schemes.

2.2 Elliptic Curve Cryptography

An elliptic curve E over a field K is defined by the following equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

, where $a_1, a_2, a_3, a_4, a_6 \in K$. This equation is also known as *Weierstrass equation.* The elements of the field are the rational points on the elliptic curve, together with a special point $\mathcal{O}$ (called the "point at infinity"). A major building block of all elliptic curve cryptosystems is the scalar point multiplication, an operation of the form $k.P$ where k is a positive integer and P is a point on the elliptic curve. Computing $k \cdot P$ means adding the point P exactly $k - 1$ times to itself, which results in another point Q on the elliptic curve. The inverse operation, i.e., to recover k when the points P and $Q = k.P$ are given, is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP). This makes Elliptic Curve Cryptography a promising branch of public-key cryptography which offers similar security to other "traditional" DLP- based schemes in use today, with smaller key sizes and memory requirements, e.g., 160 bits instead of 1024 bits.

2.2.1 Koblitz Curves

Koblitz curves [26], also known as anomalous binary curves, are elliptic curves defined over $\mathbb{F}_2$ by

$$E_a : y^2 + xy = x^3 + ax^2 + 1 \tag{2.1}$$

where $a \in (0, 1)$.

Due to the definition of the curve over $\mathbb{F}_2$, it encompasses a special property of Frobenius endomorphism. According to Frobenius map, since Koblitz curves are defined over $\mathbb{F}_{2^{\mathrm{m}}}$: if $P = (x, y)$ is a point on E_a, then so is the point (x^2, y^2). The mapping can be represented as $\tau(x, y) := (x^2, y^2)$. With this form of endomorphism, Solinas proposed τ-adic non-adjacent form known as τNAF [50] in order to reduce the number of non-zero terms in the scalar. However, the length of τNAF is almost twice the length of binary expansion of the integer scalar, which negates the advantage offered by Frobenius endomorphism. In order to overcome the drawback, Meier and Staffelbach proposed a solution in [38].

They established the fact that $kP = \gamma P$, where k and $\gamma \in \mathbb{Z}[\tau]$, such that $\gamma \equiv k \mod (\tau^{\mathrm{m}} - 1)$):

$$\gamma P = kP + \lambda(\tau^m - 1)P = kP + \lambda\mathcal{O} = kP \tag{2.2}$$

Where λ is some integer. Brumley and Järvinen in [14] based on this reduction mechanism, devised an algorithm commonly known as lazy reduction, which was the first hardware-friendly scheme proposed for the reduction of τNAF. In the case of Koblitz curves, division by τ only involves shifts and additions. The approach involves repeated division by τ for m times. The length of the τNAF may be at most $m + 4$ as stated in [14]. Double lazy reduction further improved the lazy reduction [1] scheme. In double lazy reduction scheme, division by τ^2 is performed $(m-1)/2$ times followed by a division by τ in the final step. However, due to the complexity in the algorithm, we have observed that latency for performing double lazy reduction is more than lazy reduction using our single instruction architecture framework.

2.2.2 Scalar multiplication

We have explored the proposed design's two widespread scalar multiplication techniques used for Kobliz Curve. Solinas in [50] proposed a scalar multiplication algorithm that replaces point doubling with Frobenius map operation, which can be performed with few shifts and additions in binary field arithmetic. In our proposed scheme, we have coupled the scalar multiplication methodology in [50] with a lazy reduction algorithm to obtain a reduction in scalar at the same time.

A different approach to compute kP was introduced by Montgomery in [39]. The same was optimized and proposed in [33] along with the usage of projective coordinates for point addition and doubling. The approach uses a x-coordinate only formulation which reduces the total number of field multiplication, thus reducing the overall latency of the design. We have a programmable framework to implement both methodologies using the same datapath with two different instruction sets.

2.2.3 Coordinate system and field inversion

In the proposed design while implementing the algorithm proposed by Solinas in [50] we have used the mixed affine addition formula where one point is taken as affine coordinates and one in LD projective coordinates. The formula for $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$ are as follows:

$$\begin{aligned}
&A = Y_1 + y_2 {Z_1}^2 && B = X_1 + x_2 Z_1 && C = BZ_1 \\
&Z_3 = C^2 && D = x_2 Z_3 && \\
&X_3 = A^2 + CA + B^2(C + a{Z_1}^2) && Y_3 = (D + X_3)(AC + Z_3) + (y_2 + z_2){Z_3}^2 &&
\end{aligned}$$

, where a is a curve constant.

But while computing the point multiplication proposed in [33], we have used standard projective coordinates and used the point addition and doubling formula as given in [33]:

$$\begin{aligned}
&x(2P_{\mathrm{i}}) = {X_{\mathrm{i}}}^4 + {Z_{\mathrm{i}}}^4 && Z_3 = (X_1 Z_2 + X_2 Z_1)^2 \\
&z(2P_{\mathrm{i}}) = {X_{\mathrm{i}}}^4 {Z_{\mathrm{i}}}^4 && X_3 = xZ_3 + (X_1 Z_2)(X_2 Z_1)
\end{aligned}$$

, where $i \in \{1, 2\}$. These require only 5 field multiplications and 5 squarings in each step compared to 9 field multiplications in mixed affine addition formula.

For field inversion we have implemented ItohλTsujii inversion [24] instead of Fermat's Little theorem. Using the addition chain for $\mathbb{GF}(2^{163})$ field we could avoid 153 field multiplications which helped us reduce a significant amount of clock cycles. While implementing the method proposed by Solinas in [50], two finite field inversions are required. On the contrary, a single finite field inversion is required when the algorithm proposed in [33] is implemented owing to the x-coordinate only formulation.

2.2.4 Single Instruction Processor Architecture

The first single instruction processor coined as URISC (Ultimate Reduced Instruction Set Computer) was proposed in [37]. Applying the idea of URISC, an elliptic curve scalar multiplier was proposed in [48]. In that paper, modified SBN (Subtract the operands and branch if the answer is less than zero) instruction had been used to implement scalar multiplication for the NIST P-256 curve using less than 100 slices on FPGA. But as scalar multiplication was implemented for prime curve in [48], DSP blocks could directly be used for arithmetic operations which is a challenge in the case of characteristic 2 curve.

In our proposed design we have chosen ADDN (Add the operands and branch if the answer is less than zero) for implementing ECC scalar multiplication on K-163 curve. The operation of ADDN instruction is shown in Algo. 1.

Algorithm 1: ADDN Operation

```
Input  : A,B,C
Output: ADDN A,B,C
1 $D.Mem[A] \leftarrow D.Mem[A] + D.Mem[B]$;
2 if $D.Mem[A] < 0$ then
3     jump to C;
4 else
5     jump to next instruction;
```

The next section will present the detailed idea of implementing point multiplication using our framework. The operation of ADDN instruction is shown in Algo. 1.

2.3 Lattice Based Cryptography

Lattice-based cryptography is the generic term for constructions of cryptographic primitives that involve lattices, either in the construction itself or in the security proof. Lattice-based constructions are currently important candidates for post-quantum cryptography. Unlike more widely used and known public-key schemes such as the RSA, Diffie-Hellman or elliptic-curve cryptosystems—which could, theoretically, be defeated using Shor's algorithm on a quantum computer—some lattice-based constructions appear to be resistant to attack by both classical and quantum computers. Furthermore, many lattice-based constructions are considered to be secure under the assumption that certain well-studied computational lattice problems cannot be solved efficiently.

In this section, we first give some basic definitions needed to understand why lattice-based cryptography is post-quantum resistant. Then, we introduce the learning with error (LWE) problem, which is the hard problem most commonly used to build cryptographic schemes, as well as variants of this problem that aim at making lattice-based schemes more efficient. Lastly, we introduce Kyber and Dilithium, the KEM and Digital Signature scheme with which a large part of this thesis' work has been developed.

Definition 1. *(Lattice [43]). An n-dimensional lattice is a discrete additive subgroup of $\mathbb{R}_n$. Equivalently, let $B = b_1, ..., b_n \subset \mathbb{R}_m$ consist of n linearly independent vectors; the lattice Λ generated by the basis B is*

$$\Lambda = \mathcal{L}(B) = Bz = \sum_{i \in [n]} z_i \cdot b_i \; : \; z \in Z_n$$

The parameter m is the dimension of the lattice, while the parameter n is the rank of the lattice. If $n = m$, the lattice $\mathcal{L}$ is called a full rank lattice. Additionally, a lattice $\mathcal{L}$ has an arbitrary number of basis that can generate it. The hardness of solving lattice problems typically depends on the properties of the given basis. If the basis vectors are (approximately) orthogonal and short enough, lattice problems become easier to solve. For this reason, the complexity of solving a lattice problem is usually reduced to the complexity of the algorithm used to find a good basis. Therefore, quantifying the security of lattice-based cryptography is directly related to the complexity analysis of the best known lattice reduction algorithms (such as Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm [40]). The quantum resistance of lattice-based cryptography comes from the fact that we do not know any quantum algorithm that can run in polynomial time complexity, so the problem remains hard for quantum computers.

2.3.1 Learning with errors and variants

Regev established the learning with errors (LWE) [45] problem, which has since grown to be a crucial component of lattice-based cryptography. In its decisional formulation, LWE states that, given a randomly generated $a \in \mathbb{Z}_q^n$, it is hard to distinguish with non-negligible advantage between n uniformly random samples drawn from $\mathbb{Z}_q^n \times \mathbb{Z}_q$ and the same number of samples drawn as:

$$(a, b = \langle a, s \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$$

where $s \in \mathbb{Z}_q^n$ is the secret element and $e \in \mathbb{Z}_q$ is a freshly generated error term which is introduced to make the problem hard to solve without information about the secret.

Designing LWE schemes involve deciding how the public matrix A, the secret s and the error term e are generated, the statistical distributions from which each are sampled, choosing the value of the parameters such as n and q, among other decisions. The parameter choice has to account for the best known attacks to guarantee the security of the scheme. An example of a scheme based on LWE is FrodoKEM [5], an alternate candidate in NIST PQC. For its medium level of security, Frodo uses $n = 976$ and $q = 216$. With such parameter choice, the arithmetic operations can fit within 16-bit operands thus benefiting from certain processor architectures, however, the matrix-vector multiplication remains the bottleneck of any implementation due to the large dimensions.

Aiming at improving the efficiency of the core operation in lattice-based schemes, the algebraic version of LWE, namely R-LWE [35], requires the samples to be drawn as polynomials belonging to the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ as:

$$(a, t = a * s + e) \in R_q \times R_q$$

where $s \in R_q$ is the secret element and $e \in R_q$ is the freshly generated error term, equivalently to LWE. Another way to look at R-LWE with respect to LWE is to think of the rows of the public matrix A as rotations of the polynomial of the ring scheme.

The design choices involved in R-LWE schemes are the same as in LWE schemes except that the parameter n in this case determines the degree of the polynomial. An example of a scheme based on R-LWE is NewHope [6], which uses $n = 1024$ and $q = 12289$, i.e., a 14-bit prime modulus, for its higher security version. This parameters allow efficient implementation of the multiplication utilizing the Number Theoretic Transform (NTT) [2], a variant of the Fast Fourier Transform (FFT) that requires exclusively integer arithmetic.

The R-LWE variant allows us to trade-off a matrix-vector multiplication for a polynomial convolution of somewhat similar complexity. However, R-LWE involved ideal lattices, a particular type of lattices with a stronger algebraic structure than standard lattices used in LWE. While no attacks are known yet that exploit this algebraic structure, module learning with errors (M-LWE) [29] has been proposed to interpolate between LWE and R-LWE. M-LWE samples are drawn as vectors of l polynomials belonging to the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ as:

$$(a, b = \langle a, s \rangle + e) \in R_q^n \times R_q$$

where $s \in R_q^l$ is the secret element and $e \in R_q^l$ is the freshly generated error term. M-LWE schemes can be looked at as a LWE scheme with a small matrix of dimension $l \times l$ composed by ring polynomials. The dimension of the matrix is orders of magnitude lower than in actual LWE schemes and the polynomial multiplications, which also have a lower degree than in R-LWE, can be potentially parallelized. An example of a M-LWE scheme is Kyber [13], a finalist candidate in NIST PQC, which uses $l = 3$, $n = 256$ and $q = 3329$, i.e., a 12-bit prime modulus, for its medium security level. As in the case of NewHope, the parameters choices in Kyber enable efficient polynomial multiplication by means of the NTT.

2.3.2 Common Operations in Lattice-Based Cryptography

In this section, we discuss few of the operations involved in lattice-based cryptography, which define the building blocks for its implementation. Among these, we review a bit more in detail the polynomial multiplication, which has been subject of study for a large part of this thesis.

Sampling

Sampler is one of the basic building block for lattice-based cryptography. Lattice-based schemes set certain requirements on the statistical distributions that the elements must follow to achieve provably security, low enough failure probability or simply correctness. Such requirements imply sampling either from uniform distributions or from small distributions that in practice are discrete Gaussian or binomial distributions.

However, sampling from a discrete Gaussian distribution is a challenging task in which is involved the loss of accuracy due to the limited precision, the time-memory efficiency of the sampling and the side-channel security of the implementation. Gaussian sampling has been an active research area over the past years. The most popular methods for implementing discrete Gaussian sampling efficiently are rejection sampling [41], Ziggurat sampling [36], Bernouilli sampling [23], Knuth-Yao sampling [25] and CDT-based sampling [42]. Since Gaussian sampling is not a subject of study in this thesis, we do not give further details. Instead, we refer the interested reader to the given references.

Binomial distributions in lattice-based cryptography were firstly used as a replacement for Gaussian distributions due to the difficulty of implementing Gaussian sampling both in an efficient and secure manner [6]. Sampling from a centered binomial distribution with parameter μ denoted as β_μ can be implemented very efficiently by generating samples as $\sum_0^{\mu-1}(a_i - a_{i+1})$ where all a_i are independent uniformly distributed random bits.

Polynomial multiplication

In all lattice-based schemes, being ring or module, polynomial multiplications become the core operation. From an implementation point of view, this means that polynomial multiplication is the most crucial operation to be accelerated in order to improve the efficiency of the schemes. For this reason, a large part of the work developed during this thesis focuses on optimizing polynomial multiplication on different platforms and for different design criteria.

An important consideration about polynomial multiplication within the context of lattice-based cryptography is that the multiplication happens in a ring defined by $\mathbb{Z}_q[x]/f(x)$, where $f(x)$ is a polynomial. This means that every multiplication is followed by a ring reduction operation. A typical choice is $f(x) = x^n + 1$. In practice, this choice makes the polynomial equivalent to a negatively wrapped convolution, also referred to as negacyclic convolution. The rule of thumb for the ring reduction by $x^n + 1$ is to equalize $x^n + 1 = 0$ and, thus, substitute $x^n = -1$ in the resulting product and simplify the expression.

There are two possibilities to perform polynomial multiplication. First, it can be computed directly in the coefficient form using the schoolbook algorithm yielding a time complexity $O(n^2)$. Alternatively, the polynomials can be converted to the point-value domain by evaluating them in certain points. Then, the polynomial multiplication is performed in the point-value domain where the time complexity is only linear. Lastly, the result is converted back to the coefficient form using an interpolation function which is the inverse of the evaluation. Although the complexity of the multiplication in the point-value domain is only linear, the complexity of the whole operation is determined by the algorithms used to switch domains forward and back. The most popular algorithms for this are Karatsuba, Toom-Cook and the NTT (Number Theoretic Transform). In this thesis, we will be mostly working with NTT-based polynomial multiplication.

The Number Theoretic Transform

The Fast Fourier Transform (FFT) is an efficient method to evaluate a polynomial in n distinct points with a time complexity of only $O(n\,logn)$. The points where the polynomial is evaluated are denoted as ω_n^k for $k = 0, 1, \dots, n-1$, where ω_n is the n-th primitive root of unity. Using the FFT one can convert the polynomials from to the point-value domain where the complexity of the multiplication is linear and perform the inverse transformation after, which has the same complexity as the forward transformation, thus effectively reducing the complexity of the overall multiplication to $O(n\,logn)$.

However, the roots of unity in the Fast Fourier Transform (FFT) are complex number that require expensive floating point operations. Instead, the NTT [2] is a number theoretic particularization of the FFT for prime fields $\mathbb{F}_q$. If the prime that defines the field satisfies $q \equiv 1\,mod\,n$, then all the roots of unity for computing the NTT are contained in the field, i.e., for the field defined as $\mathbb{Z}_q$ all the roots of unity ω_n are integer numbers and only modular arithmetic is required.

A polynomial $a(x) = \sum_{i=0}^{n-1} a_i x_i$ can be transformed into the NTT domain as $\tilde{A} = NTT(A)$ where $A = (a_0, a_1, \dots, a_{n-1})$ is the vector containing the coefficients of $a(x)$ and similarly $\tilde{A} = (\tilde{a}_0, \tilde{a}_1, \dots, \tilde{a}_{n-1})$ The forward NTT transform is defined as

$$\tilde{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \,(mod\, q)\, for\, i = 0, 1, \dots, n-1.$$

Since the NTT is an isomorphism there exists the inverse transform $INTT = NTT^{-1}$ such that $A = INTT(NTT(A))$ which is defined as

$$a_i = n^{-1} \cdot \sum_{j=0}^{n-1} \tilde{a}_j \omega^{ij} \,(mod\, q)\, for\, i = 0, 1, \dots, n-1.$$

When using the NTT to perform a polynomial multiplication, one has to take into account that the result of multiplying two polynomials with n coefficients each is a polynomial with $2n1$ coefficients. In practice, this is implemented by evaluating the polynomial on $2n$ points. However, when the NTT is used within the context of lattice-based cryptography, the polynomial multiplication is followed by a ring reduction that can be implicitly performed by the Fourier transform if the polynomial is evaluated on n points only instead of the required $2n$ points. Moreover, in this case and if $q \equiv 1\, mod\, 2n$ the convolution is negatively wrapped which coincides with the ring reduction operation in R-LWE cryptosystems.

2.3.3 Kyber

Kyber is an IND-CCA (Indistinguishability under chosen ciphertext attack) secure KEM, whose security is based on the Module-Learning with errors (LWE) problem. There are three major algorithms covered in Kyber; key-generation, encryption, and decryption. The main parameters of Kyber are the degree n (256) of the polynomial ring, a prime q (3329) that defines the underlying ring structure, positive integers η_1 and η_2 used for a binomial distribution, and an integer k such that $k \cdot n$ is the dimension of the corresponding LWE problem. The different security levels Kyber512, Kyber768, and Kyber1024 are achieved by varying k $(2, 3, 4)$, η_1 $(3, 2, 2)$ and η_2 $(2, 2, 2)$. The Kyber

KEM is developed in two stages. The first stage is to create an INDCPA-secure public key encryption method that is resistant to passive attackers (Indistinguishable under Chosen Plaintext Attack). The second stage is to make Kyber resistant to an active attacker by using a slightly modified version of the Fujisaki-Okamoto (FO) [20] transform and obtaining an IND-CCA2-secure key encapsulation method (Indistinguishable under Adaptive Chosen Ciphertext Attack). The IND-CPA and IND-CCA algorithms are described in detail in [13].

The first part of Kyber consists of the following routines:

IND-CPA-Keygen: The CPA Key Generation of Kyber produces a pair of public key and a secret key. The secret key s is a vector of k polynomials in R_q, each coefficient being deterministically sampled from the CBD distribution. The vector is transformed and stored in the NTT domain, becoming $\hat{s}$.

The public key is the pair (t, ρ) where $t = As + e$ is a vector of k polynomials and ρ is the public seed from which the matrix A is created. The public information A is a $k \times k$ matrix of polynomials in R_q. Since A is only used in the NTT form, the designers of Kyber have chosen to save k^2 transformations and sample it directly in the NTT domain from a uniformly random distribution. As a consequence, the NTT explicitly appears in the scheme to clarify when a transformation is needed. t is passed through the compress function without affecting the correctness of the scheme. The function finally returns $sk := \hat{s}$ and $pk := (Compress(t), \rho)$.

IND-CPA Encryption: The key encryption algorithm takes as input a public key, the message m to be encrypted and a random coin. The message to be encrypted has a fixed size of 256 bits and it can be represented as a polynomial in R_q with coefficients in $\{0, 1\}$ (i.e., each bit is represented by a coefficient). After t is decompressed, the public information is generated from a uniform distribution using the seed ρ. It is to be noted that $\hat{A}$ will be the same as the one in the key generation phase because it is generated from the same seed. The ephemeral secret key $r \in R_q^k$ together with the errors $e_1 \in R_q^k$ and $e_2 \in R_q$ is deterministically generated from CBD using the input seed μ. An error tolerance is created by mapping the coefficients of the message m from 0 to 0 and from 1 to $\lceil q/2 \rfloor$, where $\lceil x \rfloor$ represents the closest integer to x, ties being rounded up. In this context, this operation is identified with the LWE error correction. The output ciphertext is of form (u, v) where u and v are $u := NTT^{-1}(\hat{A}^T \cdot \hat{s}) + e_1$ and $v := NTT^{-1}(NTT(t)^T \cdot \hat{r}) + e_2 + m$. The function returns $(Compress(u), Compress(v))$.

IND-CPA-Key Decryption: In the first step the *Decompress*() function restores the initial size of the ciphertext values u and v. The message is recovered from the equation $m = v - s^T \cdot u$. Each coefficient of m is decrypted to 0 if its value is closer to 0 or to 1 if its value is closer to $\lceil 2/q \rfloor \cdot m$.

For simplicity, in the second part, we assume Alice as the one that encrypts/encapsulates a message and Bob as the one that decrypts/decapsulates it. The second part of Kyber consists of the following routines:

IND-CCA Key Generation: The public key pk and the pre-secret key sk' are obtained by calling *CPA.KeyGen*(). The output of *CCA.KeyGen*() will be the public key pk and the secret key sk, which is the concatenation of the following values: sk', the public key pk, the hash of the public key $H(pk)$ and a random 256-bit value z. The hash of the public key will be used in the decapsulation step and it is appended to the secret key so that it is not recomputed every time.

IND-CCA Key Encapsulation: The *CCA.KeyEncaps*() function ensures that Alice and Bob can deterministically generate the same ciphertext, given the message m and some shared randomness μ. The first step is to generate the random message m and then derive the seed μ, together with a temporary key $\bar{K}$ by hashing the hash of the message to be encrypted and the hash of the public key. The ciphertext c is obtained by applying $CPA.KeyEncrypt(pk, m, \mu)$, ensuring that Alice will deterministically derive the values e_1, e_2 and r from the seed μ. The shared key K that will be used for symmetric encryption between Alice and Bob is obtained by hashing the temporary key $\bar{K}$ together with the hash of the ciphertext c.

IND-CCA Key Decapsulation: In the CCA decapsulation mechanism, Bob decrypts the ciphertext into the message m' using the CPA decryption function. Having m' and knowing the hash of his public key, Bob can derive the temporary key $\bar{K}'$ and the shared seed μ'. He will now re-encrypt the message m' using the CPA.KeyEncrypt() function and the seed μ'. If the ciphertext that Bob computed matches the ciphertext that he received from Alice, he will accept the encapsulation and he will compute the shared key K as the hash of the temporary key $\bar{K}'$ and the hash of the ciphertext c. Otherwise, he will compute a random key K using the random value z.

2.3.4 Dilithium

Dilithium is a signature scheme based on Lyubashevsky's Fiat-Shamir with aborts framework and is based on hard problems in module lattices. The signature scheme is composed of three algorithms: Key-generation, Signing, and Verification. The main parameters of Dilithium are the degree of the polynomial ring n (256), a prime q (8380417) and dimension of the public matrix A, k and l. Dilithium specifies three sets of parameters: signified by the pair of integers $(k, l) = (4, 4), (6, 5)$ and $(8, 7)$, which indicate three security levels of Dilithium.

Important building blocks used by these algorithms are explained below:

Key Generation: The key generation proceeds by choosing a random 256-bit seed ρ and expanding into a matrix $A \in R_q^{k \times l}$ by an extendable output function. The secret keys s_1, s_2 are generated by expanding a random seed ρ' and have uniformly random coefficients between $-\eta$ and η. The value $t = As_1 + s_2$ is computed. The secret key is ρ, s_1, s_2, t, while the public key is ρ, t_1 with t_1 can be defined as $t = t_1 \cdot 2^d + t_0$ for some small t_0.

Signature Generation: The signing procedure starts by splitting the entire $t = As_1 + s_2$ into t_1 and t_0 such that $t_1 \cdot 2^d + t_0 = t$. The next step of the signing algorithm has the signer sample y with coefficients in $S_{\gamma_1 - 1}$ and then computes $w = Ay$. Then the signer writes $w = 2\gamma_2 \cdot w_1 + w_0$, with w_0 between $-\gamma_2$ and γ_2(inclusively), and computes $c = H(\rho, t_1, w_1, \mu) \in B_{60}$. After obtaining c, the signer computes $z = y + cs$. If some coefficient of z is at least $\gamma_1 - \beta$, then the signing procedure restarts. The process also restarts if the magnitude of some coefficient of $r_0 = LowBits_q(w - cs_2, 2\gamma_2)$ is at least $\gamma_2 - \beta$. This part of the protocol is necessary for security- it makes sure that nothing about the secret key s_1, s_2 is leaked. The last check makes sure that $r_1 = w_1$ and this is necessary for correctness. If all the checks pass and a restart is not necessary, then it can be shown that $HighBits_q(Az - ct, 2\gamma_2) = w_1$. Since we want to compress the public key, the verifier only knows t_1. Hence, the signer needs to provide a "hint" h which will allow the verifier to compute $HighBits_q(Az - ct, 2\gamma_2)$. So the output of the signing algorithm is $\sigma = (z, h, c)$.

Signature Verification: The verification works by using the signature and the public key to reconstruct the w_1 and then check that $c = H(\rho, t_1, w_1, \mu)$ and that all the coefficients of z_1 are less than $\gamma_1 - \beta$, and that the number of 1's in h is no greater than ω. The number of ones in h is determined by how many values of ct' cause a carry to occur. Since ct' is not too large, there is a significantly larger probability that a carry does not occur.

2.3.4.1 Transport Layer Security

TLS [46] is the current standard protocol for establishing secure communication on the Internet. The current version is TLS 1.3, which this thesis focuses on. TLS consists of three basic steps: connection establishment, TLS handshake and the encryption of application data using symmetric cryptography. While the first and last steps are not threatened by quantum computers, as long as the symmetric encryption keys used in the last step are long enough, the second step consists of authentication and key exchange that both rely on public-key cryptography.

The three main steps of TLS are illustrated in Figure 2.3.1. In the first step, the client contacts the server with the *Client_Hello* message consisting of specific parameter including the protocol version, a random nonce, a session ID, and a list of cipher suites and compression methods. To reduce network traffic, the client also sends its key material (*Client_Key_Share*) for the key establishment in the assumption that the corresponding algorithm is supported by the server. In the second step, the server replies with the *Server_Hello* that is similar to the *Client_Hello* and chooses the best possible protocol version. Additionally, the server sends its key material for key establishment (*Server_Key_Share*), the server certificate chain (*Server_Certificate*) for authentication, a confirmation message for the encryption of subsequent messages (*Change_Cipher_Spec*), and the indication of its readiness for secure communication (*Finished*). The server reply is signed by its private key. In the last step, the client also transmits a confirmation for encryption of subsequent messages (*Change_Cipher_Spec*) and its readiness to communicate securely (*Finished*).

2.4 Summary

In this chapter, we have discussed some aspects of cryptography and provided a brief description of elliptic curve cryptography. We have also discussed the various operations required to execute elliptic curve scalar multiplication based on Koblitz curve. Lastly, we give an overview of lattice-based cryptography and a brief overview of Kyber and Dilithium. In the next chapter, we will focus on the hardware implementation

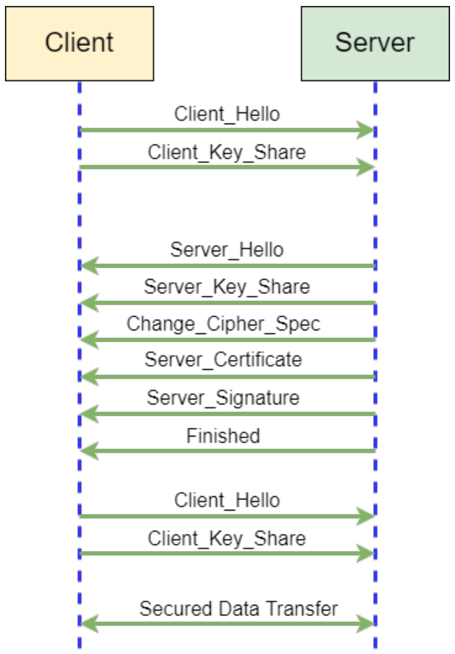

Figure 2.3.1: Illustration of TLS 1.3

of the elliptic curve scalar multiplier based on the Koblitz curve. We propose a single instruction architecture based on the ADDN instruction to implement the complete elliptic curve scalar multiplication.

Chapter 3

A Minimalistic Perspective on Koblitz Curve Scalar Multiplication for FPGA Platforms

In this chapter, we will introduce our contribution to the lightweight single instruction architecture of Elliptic Curve Scalar multiplier on FPGA based platform.

3.1 Introduction

In this work, our design strategy involves a minimalistic single instruction computing framework to design an ECC processor on FPGA with minimalistic consumption of different FPGA primitives (DSPs, BRAMs, LUTs, registers, slices), however with competitive area-time performance. We focus on implementing a characteristic-2 Koblitz NIST curve in our current work. We develop a programmable architecture to perform scalar multiplication using two different scalar multiplication algorithms. Based on τNAF proposed by Solinas in [50] to reduce the number of non-zero terms in the scalar, we have explored the scalar multiplication formula proposed in [50]. On the other hand, while using the Montgomery ladder technique proposed in [33], owing to x-coordinate only operations, we can compute using less number of field multiplications. In this work, we target to explore these trade-offs using a Turing complete `URISC` instruction, ADDN (Add the operands and branch if the answer is less than zero) to design a lightweight ECC scalar multiplier with few necessary modifications.

The idea of designing an ECC processor using a single instruction-based approach was first proposed in [48], where the authors had used SBN (Subtract the operands and branch if negative) instruction to design an ECC scalar multiplier for NIST prime field curve. But such prime field arithmetic can be mapped to on-chip DSPs of FPGAs due to their conventional arithmetic and thus can drastically reduce LUT consumptions. However, characteristic-2 algebra on which Koblitz curve is defined cannot be directly implemented on DSPs, thus requiring dedicated techniques as proposed in this work. We have also implemented ItohλTsujii[24] for a fast inversion operation without any additional hardware. The experimental results show the proposed scalar multiplier exhibits a competitive area-time metric and consumes least LUT resources among reported literature, making the design ideal for even using cost-effective Artix-7 FPGA platforms for developing crypto-SOCs. It may be emphasized that on such an FPGA 96.79% LUTs, 96.3% BRAMs, 98.75% DSPs are free for realizing other peripheral circuits required in a SOC, thus showing the minimalistic perspective of the design approach.

The remaining sections are arranged as follows: Section 3.2 demonstrates the design challenges and components. Section 3.3 presents the complete working of the proposed architecture. Section 3.4 presents the results and comparison of the proposed design followed by a conclusion in Section 3.4.1.

3.2 ADDN Instruction and Elliptic Curve Scalar Multiplication

In this section, we will have a detailed discussion regarding the challenges a designer may face along with the components required while implementing the proposed single instruction architecture.

3.2.1 Single Instruction Processor Architecture

Figure 3.2.1 depicts the general architecture of the proposed single instruction processor.

The three main components of the generic processor as shown in Figure 3.2.1 are Instruction Memory, Data Memory, and Arithmetic and Logic Unit (ALU). The architecture mentioned above has been modified based on our requirements and implemented

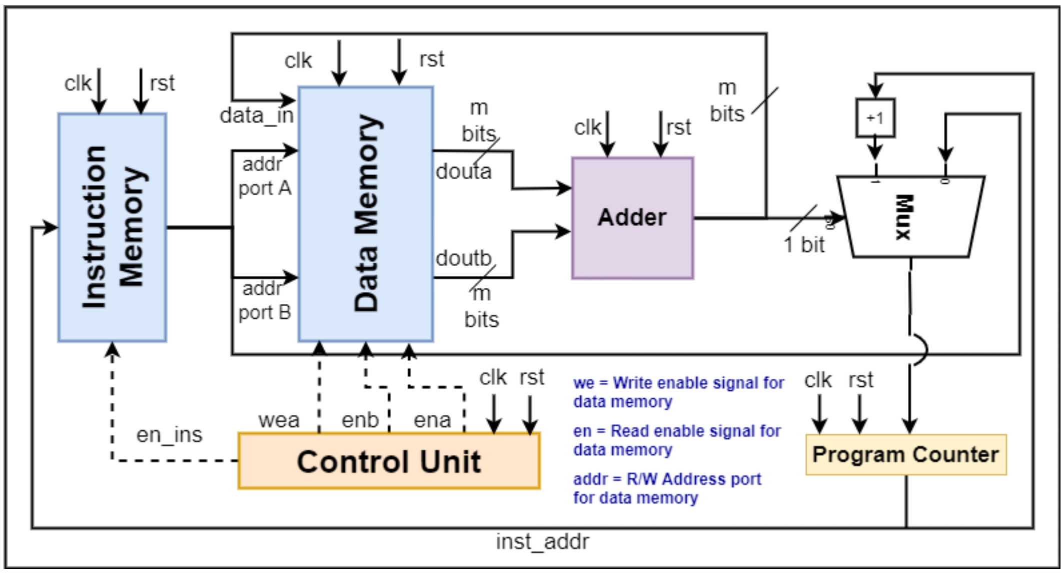

Figure 3.2.1: Architecture of ADDN processor .

on Artix-7 FPGAs.

3.2.2 Challenges for Finite Field Hardware Design

While inspecting the hardware design strategy for a finite field architecture we can visualize certain challenges:

- **Finite Field Addition/Subtraction**: Characteristic 2 field addition/subtraction is equivalent to bitwise XOR. Hard-IPs like DSP cannot perform XOR or AND as they are meant to operate on integer values. Thus LUT overhead will increase if we perform the same operation using FPGA logic units.

- **Right Shift Operation**: Right shift operation using ADDN instruction is impractical to realize as even by repeated subtraction, in worst case scenario we would require exponential time (2^{163} operations) for $\mathbb{GF}(2^{163})$.

- **Field Multiplication and Squaring**: Field multiplication and squaring are similarly impractical to realize using ADDN instruction. In the worst-case scenario, we may need to run a loop of ADDN instruction for 2^{163} times.

3.2.3 Optimization of ADDN Instruction

The traditional ADDN instruction takes in 3 parameters as described in Algo.1. However, our modified ADDN instruction takes in 4 parameters as described in Algo. 2.

Algorithm 2: Addition using Modified ADDN

```
   Input  : A,B,C,D
   Output: D.Mem[C]=D.Mem[A]+D.Mem[B]
1  $D.Mem[C] \leftarrow 0$;
2  ADDN C,A,B,2; //2$\rightarrow$ Next instruction
```

This modification in the ADDN instruction allows us to directly write the addition's result in a separate memory address. As mentioned in [48] we have also used two variants of the modified ADDN instruction: ADDN_{nw} and ADDN_{w}. ADDN_{nw} does not allow the data to be written back into the data memory while ADDN_{w} switches on the data write-back mode of the data memory. We have used 2 data memories configured as true dual-port block RAM in the proposed architecture. Along with these modifications, we have connected the Adder module with the second output port of the first data memory and the first output port of the second data memory as in Figure 3.2.2. The output of the Adder module travels to all the data input ports of the data

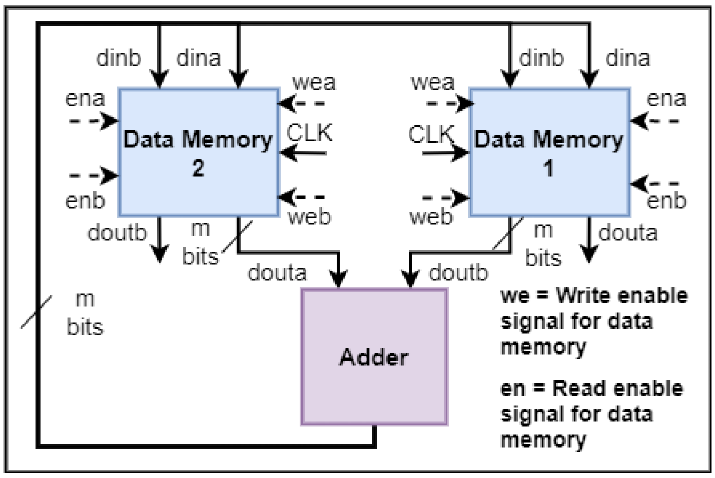

Figure 3.2.2: Modified structure of ADDN processor

memories. This arrangement allows the concurrent reading and writing of data from the same data memory, thus allowing the pipe-lining of several instructions that could not have been possible using a single data memory. Figure 3.2.3 demonstrates the pipe-lining as below:

1. **Clock Cycle 1:** Data 1 and Data 2 read from Data Memory 1 (doutb port) and Data Memory 2 (douta port) and gives input to adder.

2. **Clock Cycle 2:** Data 3 and Data 4 read from Data Memory 1 (doutb port) and Data Memory 2 (douta port) and gives input to adder. The adder performs addition operation on Data 1 and Data 2.

3. **Clock Cycle 3:** Data 5 and Data 6 read from Data Memory 1 (doutb port) and Data Memory 2 (douta port) and gives input to adder. The output of the adder is written to dina port of Data Memory 1.

If we note the operations in clock cycle 3, the data is read and written into Data Memory 1 via different ports. This contribution has significantly helped in the reduction of latency of the architecture. This concurrent operation will not be possible with a single data memory in traditional ADDN architecture.

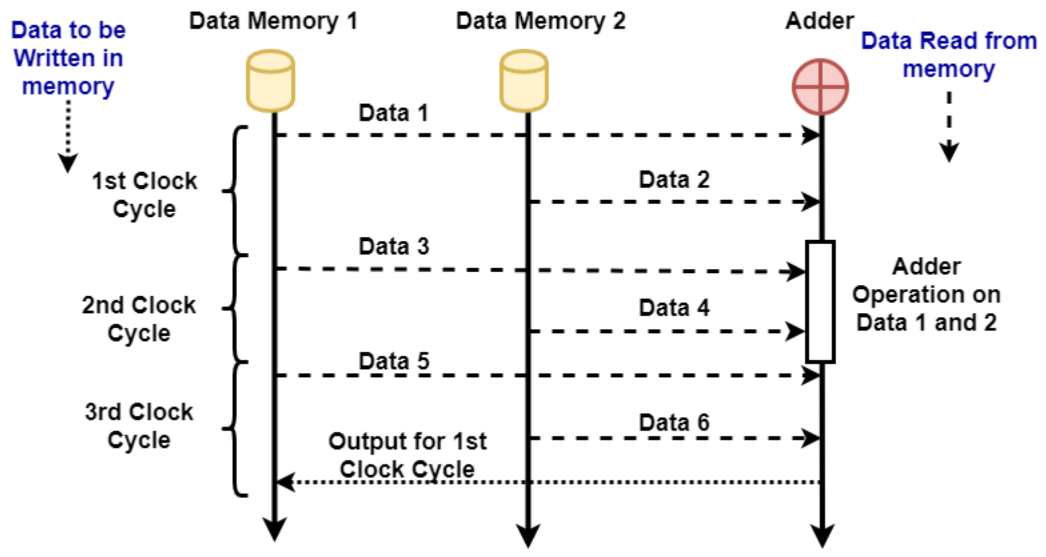

Figure 3.2.3: Modified ADDN processor Operation

3.2.4 Finite Field Addition and AND operation

In the proposed design, we are mostly confined to characteristic 2 arithmetic. In characteristic 2 field, addition is identical to XOR operation. AND operation is required while implementing lazy reduction[14]. In contrast to integer addition and subtraction, which can be carried out using DSP cores in an FPGA, the same cannot be done in the case of characteristic 2 field. To realize XOR and AND operation using DSP adder, we need to perform the following as depicted in Figure 3.2.4. The rewiring operation in Figure 3.2.4 expands the integer by inserting zeros in alternate bit positions. The sum part of the addition results in XOR of the two numbers, and the carry part results in AND of the two numbers. Using a flag (**XORCon**) we can retrieve the desired AND/XOR output into the data memory. Hence, performing these operations using DSP cores in an FPGA helps in the reduction of LUTs in the architecture.

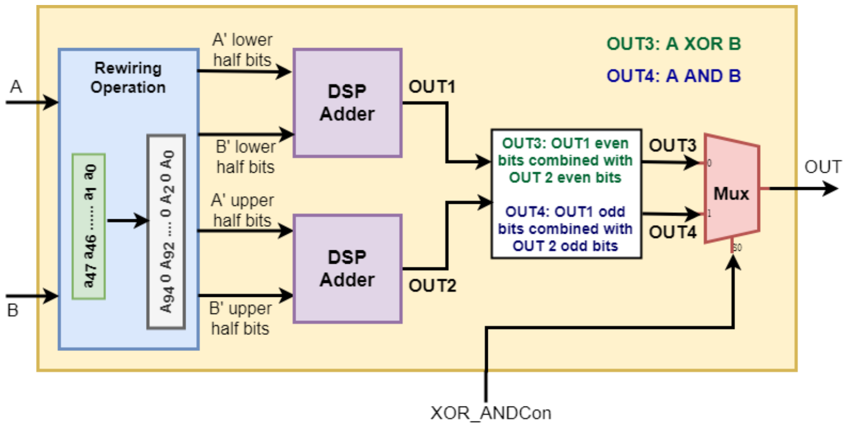

Figure 3.2.4: XOR and AND operation using ADDN

3.2.5 Finite Field Multiplication

Finite field multiplication using only ADDN instruction is impractical for a real-life scenario as, we may need to run a loop for 2^{163} times for elements $\in \mathbb{GF}(2^{163})$.

To solve this problem, we have designed a lightweight field multiplier with **rstMul** flag to reset the multiplier core. We have used the Comba multiplication algorithm as mentioned in [32] with word size 48 bits by pipe-lining certain operations. Figure 3.2.5 and 3.2.5 shows the architecture of the lightweight field multiplier used in the proposed design.

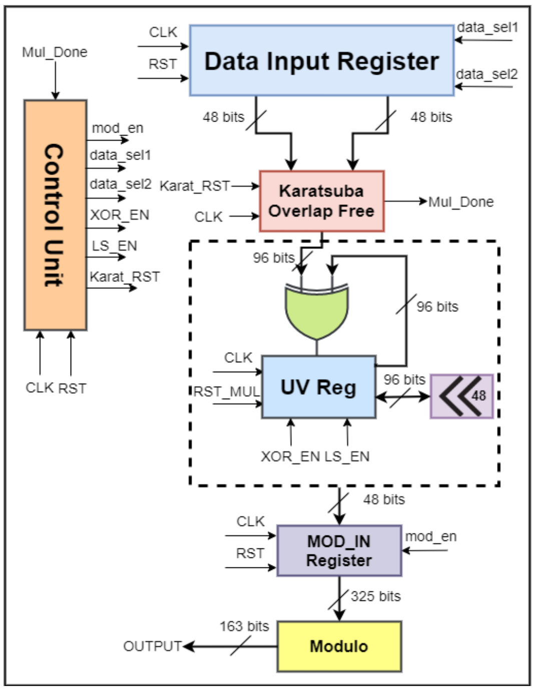

Figure 3.2.5: Low Area Architecture

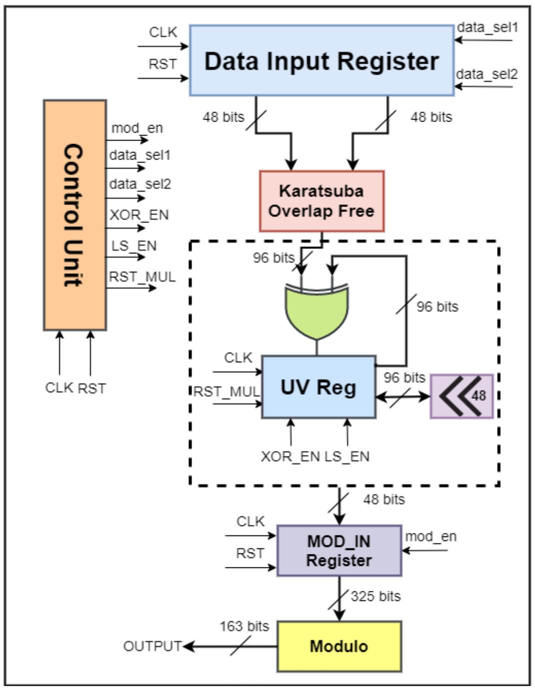

Figure 3.2.6: High Area Architecture

The core 48 bit field multiplier can be implemented in two different ways as depicted in Figure 3.2.5 and 3.2.6. The 48 bit overlap-free combinational Karatsuba multiplier[19] in Figure 3.2.6 produces output in single clock cycle while the sequential one in Figure 3.2.5 requires 5 clock cycles to produce an output. A similar hardware implementation of overlap-free Karatsuba was done in[49]. Thus overall, for a 163-bit field multiplication, the multiplier in Figure 3.2.5 requires 112 clock cycles while Figure 3.2.6 requires 32 clock cycles. Both the multipliers are followed by a similar combinatorial modulo operation block. The one in Figure 3.2.5 consumes a much lesser area compared to Figure 3.2.6 owing to the difference in the architecture of the Karatsuba multiplier block. The XOR operation is performed using a combinatorial XOR block which can be implemented similarly as mentioned in Sec. 3.2.4 but considering the latency of the field multiplier, we have opted for the combinatorial XOR block. The operations are pipelined using the control path designed within the field multiplier. We have implemented a separate combinatorial squarer block for field squaring that requires 9 clock cycles, including data loading for a single squaring. Owing to the total number of squaring operations required due to Frobenius map, point addition and doubling and Itoh-Tsuji inversion, a separate squarer block reduces the execution time of the complete scalar multiplication operation.

3.2.6 Right Shifter on ADDN Processor

Right shift operation is required several times while implementing scalar multiplication operation on the Koblitz curve. Shifting key bits or division by 2, both can be performed using right shift operation. Executing these operations by repeated subtraction using ADDN is not feasible. So with a zero LUT overhead, we can implement a right shifter module in the design enabled by a flag bit. The flag-bit **rsCon** is used to write the right-shifted output into the data memory.

3.2.7 Instruction Memory and Control Path

The instruction memory is configured as single-port ROM can hold up to 2^{10} instructions. The instructions are 60 bits wide. A single instruction consists of the 22 flag bits, four 7 bits operand address corresponding to the four ports of the data memory, and a 10-bit jump address. The flag bits consists of:

- Read-Write Operations: It controls the read/write operations for all the input ports of the two data memories
- ALU Operation: It controls which of the ALU block provide output to the data memory

The instruction format of the modified design is as shown in Figure 3.2.7. Instead of having a hardwired control path, we have micro-coded the control signals within the processor instructions, which helps us save a few resources.

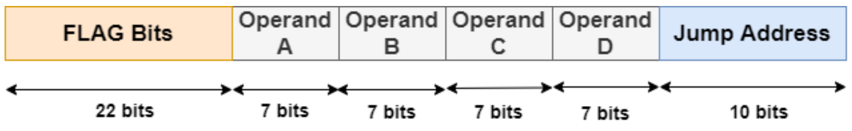

Figure 3.2.7: Instruction structure of the ADDN processor

3.3 ECC Co-Processor using ADDN

In this section, we will present a detailed description of our proposed ADDN ECC Processor. The DSPs in the architecture are configured for 48 bits operands. The instruction format of the whole design is as shown in Figure 3.2.7. The flag bits in Figure 3.2.7 controls the operation of the dedicated ALU blocks, and the read/write operations of the data memories. The multiplier core is initiated by **rstMul** flag bit. The key is stored in the data memory in an inverted manner. The right-shifter helps in traversing through the key, which is controlled via **rsCon** flag bit. The MUX in the program counter sections serves the purpose of selecting the next line address or the jump address based on the MSB of the output ALU blocks. The output of the ALU blocks are written into data memory by selecting the correct output via the select signal of the DIN Select module and the write control bits of the data memory. The complete ADDN processor is shown in Figure 3.3.1. The output port of the DIN Select module is connected to all the data input ports of the two data memories used. This allows the parallelization of data read and write cycle. The instruction memory is put to a halt while field multiplication is performed. This helps in avoiding no operation instructions, thus saving instruction memory. We can implement two different methodologies of scalar multiplication using architecture specified in Figure 3.3.1 with only two different instruction sets.

3.4 Result and Comparison

In this section, we will analyze the performance of the proposed ADDN processor for the ECC Scalar Multiplication. The design goal is to develop a design with minimal footprint on the slices and properly utilize the BRAMs and DSPs judiciously to ensure that the FPGA can also house other circuits needed in an application. Table 3.1 shows the statistics of resource utilization of the proposed design with respect to the available resources in Artix-7 (XC7A100TCS) FPGA board. Table 3.2 reflects 4 proposed designs where the data paths of all the designs are the same except for 2 different multiplier cores; one combinational and another sequential showing two design data-points each for Solinas and Montgomery techniques. Combining the results of Tables 3.1 and 3.2 we can see that, the most competitive result in terms of Area-Time metric (**0.269**) is

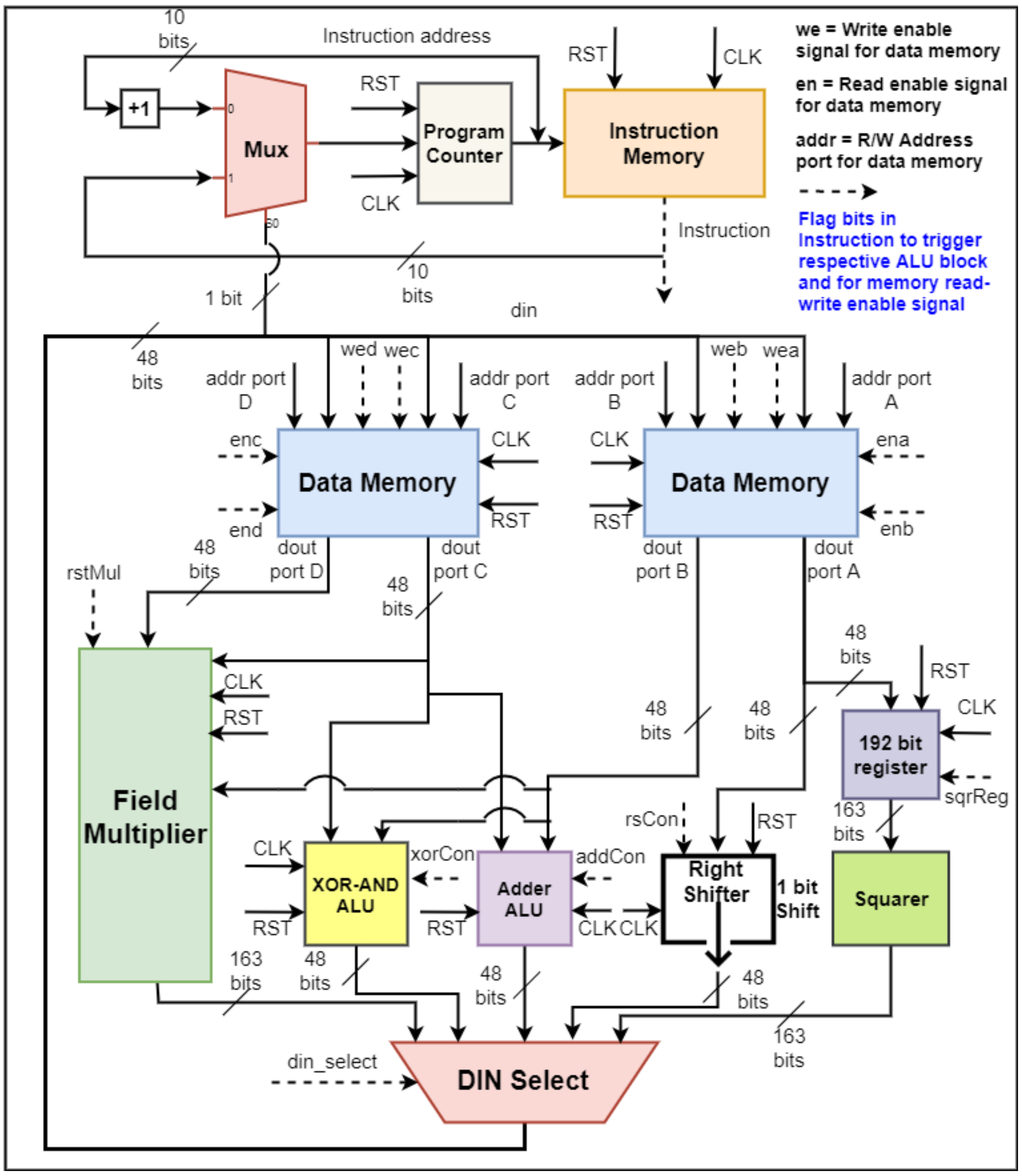

Figure 3.3.1: Architecture of ADDN processor

obtained for the design with the multiplier core depicted in Figure 3.2.6 and the scalar multiplication implemented using Montgomery ladder. Thus the $x-$coordinate only formulation with no additional operations to reduce the scalar has proved to be more advantageous with respect to the method in [50]. The design consumes only 657 slices, leaving 96.79% LUTs, 96.3% BRAMs, 98.75% DSPs free, while requiring 0.410 ms to execute the complete scalar multiplication.

Platform	Slice	LUT	Register	DSP	BRAM
Artix-7 (XC7A100TCS)	4.15% (657)[1]	3.21% (2033)	0.98% (1237)	1.25% (3)	3.7% (5)
	2.64% (419)[2]	1.92% (1220)	1.15% (1458)		

[1]48 bit combinational Karatsuba Multiplier which produces output in 1 clock cycle
[2]48 bit sequential Karatsuba Multiplier which produces output in 5 clock cycles

Table 3.1: Resource utilization details of the ECC processor

Works	Platform	LUTs	Area (Slices)	Max Freq (MHz)	Latency (μ-sec)	$\frac{AT}{10^6}$
[8]	ALTERA STRATIX II	18964 ALM	-	-	9.84	-
	ALTERA STRATIX II	23084 ALM	-	-	9.15	-
[30]	Virtex-4	-	7427	-	29.28	0.217
[32]	Spartan-3 XC3S400	3850	1232 (8 BRAMs)	93.084	456	0.561
	Virtex-4 XC4VFX12	3815	1219 (8 BRAMs)	155.376	273	0.332
	Virtex-E XCV2000E	4320	1641 (18 BRAMs)	43.983	964	1.58
[34]	Xilinx XCV2000E	7362 (g = 14)	-	-	135	0.993
	Xilinx XCV2000E	10017 (g = 41)	-	-	75	0.751
[22]	Spartan 3 XC3S200	2220	1180 (4 BRAMs)	70.5	2700	3.18
	Spartan 3 XC3S50	847	417 (4 BRAMs)	75	17517	7.304
	Spartan 3 XC3S50	1056	543 (4 BRAMs)	76	5360	2.91
[31]	Virtex-5 XC5LX110T	7073	2199 (5 BRAMs)	223.46	68	0.149
		8609	2708 (5 BRAMs)	222.67	55	0.148
Proposed[1] (Montgomery)	**Artix-7 (XC7A100TCS)**	**2033**	**657 (5 BRAMs, 3 DSPs)**	**109.5**	**410**	**0.269**
Proposed[2] (Montgomery)	**Artix-7 (XC7A100TCS)**	**1220**	**419 (5 BRAMs, 3 DSPs)**	**120.25**	**919**	**0.385**
Proposed[1] (Solinas)	**Artix-7 (XC7A100TCS)**	**2033**	**657 (5 BRAMs, 3 DSPs)**	**109.5**	**663**	**0.435**
Proposed[2] (Solinas)	**Artix-7 (XC7A100TCS)**	**1220**	**419 (5 BRAMs, 3 DSPs)**	**120.25**	**1049**	**0.439**

[1]48 bit combinational Karatsuba Multiplier which produces output in 1 clock cycle
[2]48 bit sequential Karatsuba Multiplier which produces output in 5 clock cycles

Table 3.2: Comparison of ECC ADDN Processor with Existing Designs

Most of the existing designs implement an ECC scalar multiplication using a high-performance architecture where the latency of the design is crucial. The designs in Table 3.2 are mostly implemented in Virtex-4 or older Xilinx platforms and none of them utilizes DSPs for finite field arithmetic. It would be unfair to directly compare the statistics of our design to these as they are on different families. However to demonstrate the minimalistic perspective of the design we choose a modern low-end Artix-7 board. A high-performance ECC scalar multiplier on K-163 with a digit-level finite field multiplier over Gaussian normal basis is presented in [8]. The target of the design is to achieve low latency at the cost of area wherein it employs four parallel field multipliers for speed-up. The designs in [30] and [31] supports all 5 Koblitz curves recommended by NIST. The design of [30] had tried to reduce latency by the use of the Karatsuba-Ofman field multiplier. However, the slice consumption of the architecture is 10 times more than even our high area architecture. In [31] several parallelization techniques have been applied in the field multiplier section. Additionally, an improved τNAF point multiplication algorithm is proposed to improve the efficiency of elliptic curve point multiplication. Despite having an impressive area-time metric, the architecture consumes more resources than ours. Scalable architecture with an improved point addition algorithm and use of comba multiplication algorithm is described in [32]. They have also targeted a low latency, high-performance design with high LUT and BRAM consumptions. An architecture for $\mathbb{GF}(2^{163})$ is shown in [34], where a digit-serial multiplier with improved latency is used. The design has a fairly high resource consumption which is 10 times more than ours. A low area overhead scalable design is shown in [22] using a micro coding technique. The control path of the design is coded within the instruction memory. This way, they have achieved a fairly low area consumption of the overall architecture. However, the design latency is 10 times more than the present proposed work.

3.4.1 Conclusion

In this work, we develop an ADDN-instruction based ECC processor for Koblitz curves. Several design choices for judicious usage of FPGA hard IPs have been proposed for reducing the area overhead while ensuring a competitive area-time product. The design is reconfigurable to support two different methodologies for scalar multiplication. We furnish experimental results to show that the design consumes minimal resource on cost-effective FPGAs, leaving ample room for hosting other important SOC circuitry.

But now as it is a well-accepted fact that quantum computers solve such hard problems (mainly attributed to Shor's algorithm), and, therefore, become imminent threats to existing PKC. Fortunately, significant research has already been undertaken to develop new alternative "quantum-safe" (also called "post-quantum cryptography" aka. PQC) PKC schemes and a standardization process is ongoing due to the National Institute of Standards and Technology (NIST). Among different alternatives, the schemes based on worst-case lattice problems have received particular attention, mainly due to their simplicity and ease of implementation.

In our next work, we propose PQC implementation in the context of automotive security. Our primary aim is to address the three design goals presented in the last paragraph. As the first step to this, we judiciously choose the PQC candidates from the NIST portfolio, which are best-suited for automotive purposes. Clearly, lattice-based schemes are the obvious choice for both key-encapsulation mechanism (KEM) and signatures, given their simple constructions and ease of implementation. Even among the lattice-based schemes, we choose Kyber owing to its smaller value of the prime. We propose a highly programmable, instruction-based architecture of Kyber using which implements all the security levels of Kyber on the same framework.

Chapter 4

Resource Constrained and Programmable Architecture for KYBER Key Exchange Mechanism

4.1 Introduction

The focus of contemporary public-key cryptography (PKC) is on a select group of well-known hard issues, including integer factorization, discrete logarithm, elliptic curve discrete logarithm, etc. It is now widely acknowledged that quantum computers are capable of solving such challenging issues, primarily due to Shor's algorithm, and that they pose an immediate danger to PKC as it currently exists. Fortunately, the National Institute of Standards and Technology has already made major research effort to produce new alternative "quantum-safe" (also known as "post-quantum cryptography," or PQC) PKC schemes (NIST). The post-quantum standardisation process proceeded through three rounds of examination before producing four standard candidates, of which one is a key exchange mechanism and the other three are digital signature methods. Among different alternatives, the schemes based on worst-case lattice problems have received particular attention, mainly due to their simplicity and ease of implementation. The NIST has chosen Kyber as the lone applicant in the category of key exchange mechanisms that is based on module learning with error hardness assumption.

There have been numerous studies focusing on the performance of PQC protocols that can be achieved in hardware-software co-design. These studies aim to evaluate multiple candidates with comparable computing jobs on the same platform and produce somewhat accurate rankings between them. The more time-consuming portions of the protocol, primarily the NTT (Number Theoretic Transform) and hash functions, are offloaded to dedicated hardware accelerators, while other portions are implemented using software in processors like the ARM Cortex series. It is feasible to replace the hardwired ARM core with well-known RISC-V soft cores, which may be implemented with programmable logic in FPGA. This allows the entire design to be implemented in hardware, mostly avoiding slow data transmission across the hardware/software boundary.

However, from an application perspective such as automotive security, efficient hardware implementation of these PQC algorithms is required for hardware security modules (HSMs). The E-safety vehicular intrusion protection applications (EVITA) project [17] has highlighted the importance of protecting communication between ECUs in vehicular networks. To at least guarantee the authenticity and integrity of the data origin, appropriate measures are needed. As a result, the necessity for protocols like Kyber for protecting in-vehicle communication is currently being looked into. The majority of the current automotive PQC proposals, however, are software implementations [53]. The sole article on hardware implementation [51] that suggests using PQC based on already existing designs, but it doesn't discuss how PQC implementations should be tailored for an automobile setting. Due to the complexity of the PQC algorithms and the real-time requirements of an automotive system due to the safety critical needs, developing efficient hardware solutions are imperative. The designs, on the other hand, must be compact and less energy-intensive. In addition to these competing demands, the hardware should be flexible to accommodate the many PQC algorithm options and their adoption with the shortest time to market. The need for lightweight implementations is a crucial restriction in the automobile context. However, considering that automotive data frequently handles crucial control tasks, speed cannot be completely sacrificed in order to conserve resources. In addition to these two opposing design objectives, implementation agility is crucial since it reduces the time-to-market.

In this chapter, we show a PQC implementation where we primarily address the three design goals presented in the last paragraph. Lattice-based schemes are advantageous choices, given their simple constructions and ease of implementation. Even among the lattice-based schemes, we choose Kyber because of its lower value of prime which makes it suitable for a lightweight implementation. During the design, several critical optimizations were made to make the design programmable and scalable to come up with a lightweight design that can be appended with other compatible lattice-based schemes. To bring forth the much-needed agility, we choose a micro-code-based design style, where updating the memory can tune the design to operate on different parameter choices. An immediate advantage of this design choice is that we can support different levels of security on the same hardware. We prototype the entire design on a NEXYS 4 DDR FPGA platform along with a Microblaze soft-core processor and show that our design is competitive with the state-of-the-art designs.

4.1.1 Objectives of Our Work

In this work our major objectives are as follows:

- We have designed an efficient FPGA-based hardware architecture for Kyber
- We have implemented a programmable Number Theoretic Transform (NTT) multiplier, where the butterfly unit can be programmed to perform forward NTT, inverse NTT and point-wise multiplication.
- We have implemented a common Keccak core with a wrapper to implement the several hashing algorithms required.
- Our programmable architecture can accommodate all the security levels of Kyber.
- Our design is competitive in terms of Area-Time product based on the available literature.
- We plan to incorporate other lattice-based post-quantum schemes into our design banking on our agile design methodology.

4.2 Proposed Architecture and Design Decisions

A unique feature of the proposed architecture is the programmability it offers, which accommodates all the security levels of Kyber within the same framework. We have a high-speed NTT core along with a Keccak core, that is capable of computing all the variants of SHAKE and SHA3, and is primarily used for both pseudo-random number generation and hash computation. In this section, we shall elaborate design details and strategies of the architecture.

4.2.1 High Level Overview of the Architecture

The complete hardware architecture of the proposed design is presented in Figure 4.2.1. The diagram shows the datapath of Kyber along with the instruction and data memory. The instructions stored within the instruction memory control the sequence of operations needed to execute Kyber. The datapath integrates the logic and functional modules necessary to execute Kyber. We have also observed that we can obtain a significant speed-up with respect to the canonical software implementation of Kyber as certain modules can be realized in hardware with practically negligible overhead. The Encode and Decode modules of Kyber are rewiring operations in hardware which consume significant latency in software. Implementing the compress/decompress module of Kyber in hardware involves a few shifts and multiplication with constant in hardware. The multiple outputs produced by these modules are fed to a multiplexer unit connected to each individual module, the appropriate value is extracted based on the value of the select line. The detailed description of the NTT multiplier and the Keccak blocks are described in the sections below.

4.2.2 Design of NTT Multiplier

Polynomial multiplication is one of the most computationally intensive components in Kyber. In order to develop our lightweight and fast multiplier architecture, we exploit the structure of the field prime. For Kyber prime $q = 3329$, the base field $\mathbb{Z}_q$, contains the primitive 256'th root of unity, which renders it difficult to apply negacyclic convolution while performing number theoretic transform. The polynomial size in case of Kyber is $n = 256$ and the minimum criteria to apply negacyclic convolution is that

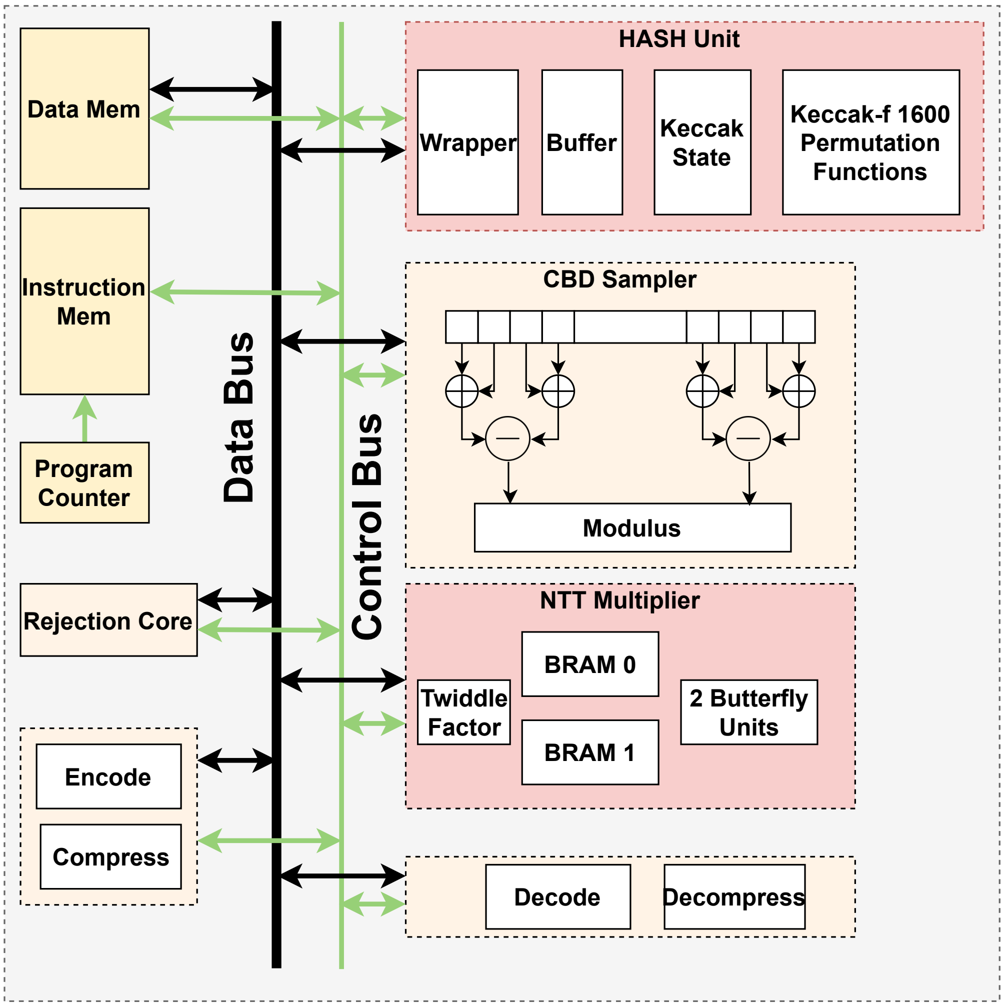

Figure 4.2.1: Hardware architecture of the crypto co-processor

$2n$'th root of unity must exist for the base field $\mathbb{Z}_q$. Thus, we perform incomplete NTT for Kyber. If $2n$'th root of unity would have existed for the base field, we could have performed $log_2 n$ levels number theoretic transform, but for Kyber we can only perform $log_2 n - 1$ levels of NTT. Incomplete-NTT produces $n/2$ (128) polynomials of degree 1. Point-wise multiplication in case of Kyber involves the multiplication of 128 degree-1 polynomials. In the rest of the section, we will elaborate on how we have implemented the polynomial multiplication architecture of Kyber.

Strcuture of the NTT-based Multiplier. We compute the number theoretic transform using the 256'th root of unity of Kyber prime. While executing pointwise multiplication with seven levels of NTT we followed the Karatsuba multiplication technique which is also adopted by [52]. Point-wise multiplication or basecase multiplication (referred in Kyber specification [7]) is: $h_{2i} + h_{2i+1}X = (f_{2i} + f_{2i+1}X) \cdot (g_{2i} + g_{2i+1}X) \ mod\,(X^2 - \zeta^{2br(i)+1})$. In a straightforward way, we need to perform 5 multiplications. But as we can see from Equation 4.1, the number of multiplication decreases to 4 from 5 due to Karatsuba-based approach.

$$\begin{aligned} h_{2i} &= f_{2i}g_{2i} + f_{2i+1}g_{2i+1} \cdot \zeta^{2br(i)+1} \\ h_{2i+1} &= (f_{2i} + f_{2i+1})(g_{2i} + g_{2i+1}) - (f_{2i}g_{2i} + f_{2i+1}g_{2i+1}) \end{aligned} \tag{4.1}$$

Reduction Architecture Based on the Structure of Prime. Kyber is based on a prime of the form $q = k \cdot 2^m + 1$, where $k = 13$ and $m = 8$ ($k < 2^m$). The previous hardware architectures of Kyber [12] exploit this form of prime to use the K^{2}RED reduction algorithm. We exploit this specific structure of prime to perform K^{2}RED reduction for Kyber as shown in Algorithm 3. The K^{2}RED algorithm takes an integer C as input and outputs an integer D such that $D \equiv k^2 \cdot C \ mod\, q$.

Algorithm 3: K^{2}RED Reduction Algorithm for Kyber

```
  Input  : A binary number $C = (c_{23}, ..., c_0)_2$, $k = 13$, $m = 8$, $q = 3329 = k \cdot 2^m + 1$
  Output: $C'' = k^2 \cdot C \ mod\, q$
1 $C_l = (c_7, \dots, c_0)_2$
2 $C_h = (c_{23}, \dots, c_8)_2$
3 $C' = k \cdot C_l - C_h$
4 $C'_l = (c'_7, \dots, c'_0)_2$
5 $C'_h = (c'_{15}, \dots, c'_8)_2$
6 $C'' = k \cdot C'_l - C'_h$
```

The K^{2}RED algorithm [12] is outlined in Algorithm 3 that implements the above-mentioned reduction. Note that, we can eliminate the extra k^2 factor by inserting an additional k^{-2} in the multiplication phase. We decompose the value of k into ($k = 16-3$) for Kyber to optimise the modulo operation. The value C' can now be expressed as $C' = ((C_l \ll 4) - (3 \cdot C_l + C_h))$.These mathematical expressions can be realized using pure combinatorial logic. The modular reduction unit in our NTT multiplier architecture consists of these two combinatorial units only, which aids us in achieving a reduced latency. The small increase in resource consumption due to the combinatorial block is thus compensated with a reduced latency causing a reduction in area-time product.

Apart from using the prime, we have implemented both the Cooley-Tukey and the Gentleman-Sande algorithm to implement the NTT multiplier. For forward NTT, we used Cooley-Tukey and for reverse NTT, we used Gentleman-Sande algorithms to avoid the bit-reversal step. This technique is quite commonly used in some of the previous works. The Cooley-Tukey algorithm uses the Decimation in Time approach while Gentleman-Sande uses the Decimation in Frequency approach.

Algorithm 4: Algorithm for Number Theoretic Transform

```
Input  : $p, N, q; twiddle\_factor\_array[N]$
Output: $\hat{p}$
1 $twiddle\_count = 1$
2 for $s = 2^{N-1}$ to 1 by $s/2$ do
3     for $start = 0$ to $N - 1$ by $j+s$ do
4         $zeta = twiddle\_factor\_array[++twiddle\_count]$
5         for $j = start$ to $start + s$ do
6             $t = zeta \cdot p[j + s] \bmod q$
7             $p[j + s] = p[j] - t \bmod q$
8             $p[j] = p[j] + t \bmod q$
```

NTT computation. We opted for Cooley-Tukey for forward NTT (*Decimation in Time*) and we use Gentleman-Sande method for reverse NTT (*Decimation in Frequency*) avoids bit-reversal, in polynomial multiplications. A similar architecture was also proposed in [52], though the use of our fast reduction technique has helped us achieve results at much lower latency. Figure 4.2.2 shows the basic structure of a butterfly unit which is capable of processing two coefficients at a time.

The proposed NTT multiplier houses 2 such butterfly computation units that are capable of processing four polynomial coefficients after each iteration. Additionally, the NTT multiplier block has two separate BRAM units capable of holding four coefficients in a single memory cell separated by an index of s, where $s \in \{128, 64, 32, 16, 8, 4, 2\}$ feeding the two butterfly units. The corresponding schedule of the multiplier execution is illustrated in Figure 4.2.3, where a_i are the polynomial coefficients. Both forward and reverse NTT require 224 clock cycles.

The butterfly unit in Figure 4.2.2 can operate in 3 separate modes: Forward NTT,

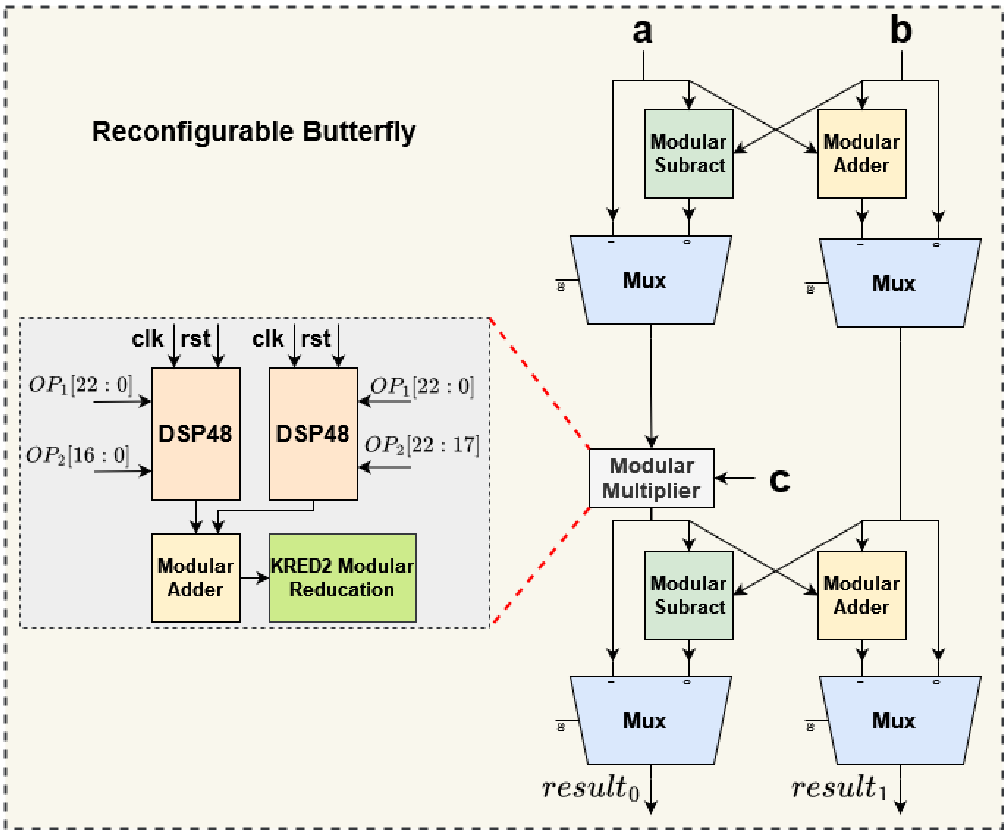

Figure 4.2.2: Re-configurable butterfly unit

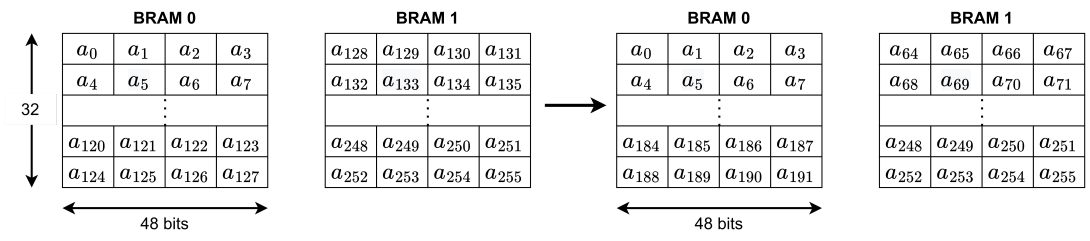
BRAM 0
BRAM 1
BRAM 0
BRAM 1
32
48 bits
48 bits
a_0 a_1 a_2 a_3
a_4 a_5 a_6 a_7
a_{120} a_{121} a_{122} a_{123}
a_{124} a_{125} a_{126} a_{127}
a_{128} a_{129} a_{130} a_{131}
a_{132} a_{133} a_{134} a_{135}
a_{248} a_{249} a_{250} a_{251}
a_{252} a_{253} a_{254} a_{255}
a_0 a_1 a_2 a_3
a_4 a_5 a_6 a_7
a_{184} a_{185} a_{186} a_{187}
a_{188} a_{189} a_{190} a_{191}
a_{64} a_{65} a_{66} a_{67}
a_{68} a_{69} a_{70} a_{71}
a_{248} a_{249} a_{250} a_{251}
a_{252} a_{253} a_{254} a_{255}

Figure 4.2.3: NTT multiplier scheduling for Kyber in the NTT RAM

Inverse NTT, and point-wise multiplication based on the values of the select lines of the MUXes. Depending on the operating mode, the input *c* in Figure 4.2.2 can be switched between twiddle factors or a polynomial coefficient. These operations are controlled by a group of polynomial multiplier instructions as described in Section 5.2.1. The control signals micro-coded in these instructions bring out agility both in terms of different security levels and between the two schemes. Additionally, with our fast reduction technique as described above, the NTT multiplier helped us bring down the area-time product of our design.

4.2.3 Programmable Keccak Core

The Keccak core [11] in this architecture is utilized for cryptographic hash computation and pseudo-random number generation. Kyber requires four modes of the Keccak core - namely SHA3-256, SHA3-512, SHAKE-128, and SHAKE-256. All of these modes are implemented using the Keccak sponge structure internally equipped with separate wrappers and individual buffers that are multiplexed based on the micro-coded control signals. The requisite wrapper is selected based on the micro-coded control signals generated by the group of Keccak instructions as mentioned in Section 5.2.1. The basic architecture of the Keccak core implemented in our design is shown in Figure 4.2.4 and 4.2.5. They share the same Keccak-f[1600] function but with different rate r and different message suffix appended during padding for domain separation. The constitution of r bits out of 1600-bit input is decided by the wrapper module. Parts of register data, appended suffix and part of padding data are selected through multiplexer under the control of the underlying FSM and the Instruction memory. This also defines the agility of our architecture as we are using a single Keccak core for both schemes across all the security levels. Keccak core works in a way such that the data would be fetched from the buffer until enough data has been collected to begin the next hashing process. The control signals are fed into module which controls the absorb, and squeeze mode. Absorb indicates the input corresponding to rate part should be XORed with r bits of the current 1600-bit Keccak state as one 24-round Keccak-f function can not fully absorb the input message bits. A $finish$ signal indicates to form the 1600 bits input to Keccak core for the next hash cycle to occur. The Keccak buffer directly feeds to the NTT block and the Rejection core to save time for the polynomial multiplication block. In a way, this also brings down the resource consumption of the entire architecture aiding the reduction of the area-time product of the design. The final Keccak sponge implementation requires 24 clock cycles, and the final resource overhead for the complete hash unit is 10879 LUTs (Artix-7 FPGA). Despite the heavy resource overhead of the Keccak unit, the multiplexed architecture is necessary to support the required security level across Kyber.

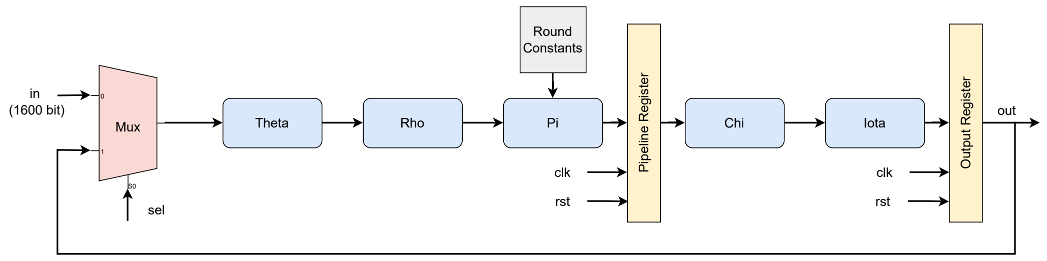

Figure 4.2.4: Proposed architecture of transformation round

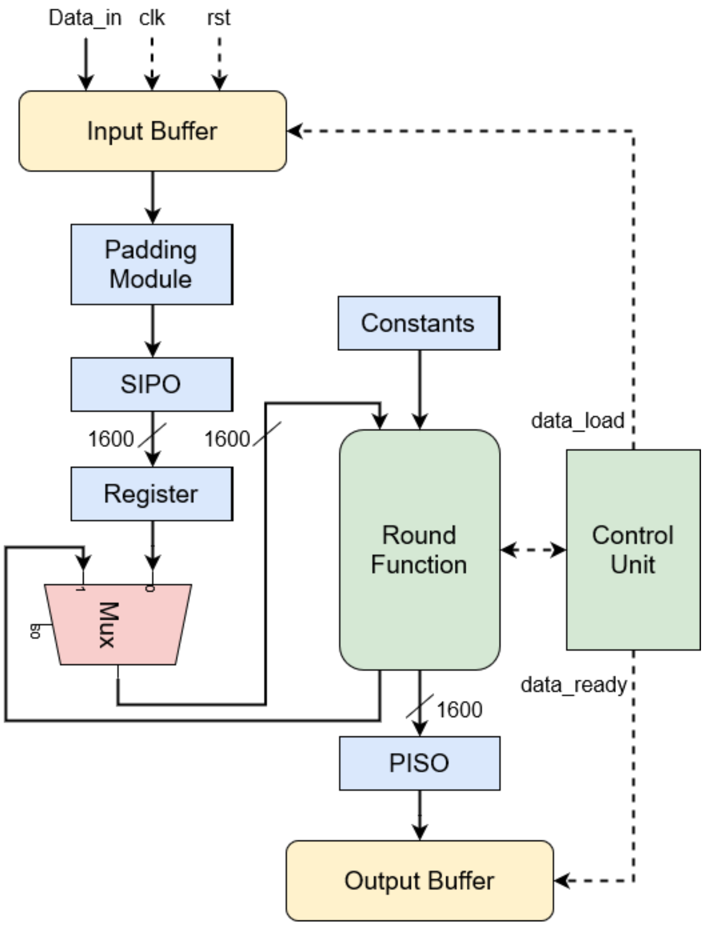

Figure 4.2.5: Proposed architecture of the KECCAK hash function

4.2.4 Auxiliary building blocks

Apart from the abovementioned functional modules, several other auxiliary hardware modules are present in this design which we discuss briefly. The data memory stores the polynomial coefficients, and two polynomial coefficients are available from the data memory simultaneously during computation. Most of the modules developed involve set of combinatorial circuits as arithmetic operations with certain constants based on different security levels. The canonical implementations of the auxiliary modules of Kyber are well-described in [7]. The polynomial arithmetic module with adder and subtractor is designed to operate in non-blocking fashion such NTT module can operate concurrently.

4.2.4.1 Compress/Decompress

The decompress unit performs division by power-of-two and rounding operation which is trivial to implement in hardware. On the other hand, the compress operation requires division by q and rounding. Some works in the literature use Barrett reduction and division algorithms to perform compress operation. The compress operation basically involves the mathematical operation $Compress_q(x, d) = \lceil 2^d/q \cdot x \rfloor mod^+ 2^d$. The value of d are as follows: $1, 4, 5, 10, 11$. We have implemented this by rounding off the value $2^d/q$ to the nearest integer and multiplied it with a power of 2. Then after a few bit shifts and addition operations, we obtain the proper output. The reverse operation helps in obtaining the output of the decompress operation. This module is implemented in a very optimized manner with the required precision such that the output matches with the canonical implementation of the Compress/Decompress module. The compress module is implemented in hardware as shown in Figure 4.2.6.

4.2.4.2 Sampling

Kyber uses a binomial sampler that samples integers from a centered binomial distribution. We integrate two separate samplers $\mathcal{S}_1$ and $\mathcal{S}_2$ with $\eta = 2$ and $\eta = 3$, respectively. We implement 16 concurrent adder cores to optimize sampler $\mathcal{S}_1$. Sixteen samples are obtained from $\mathcal{S}_1$ and stored in a buffer. In contrast, $\mathcal{S}_2$ uses an input buffer for concatenating with the input in the next cycles. Three 64-bit words are read to obtain 32 two-word samples from $\mathcal{S}_2$. The group of instructions as described in Section 5.2.1 helps us in selecting between the two sampling cores based on the different security levels.

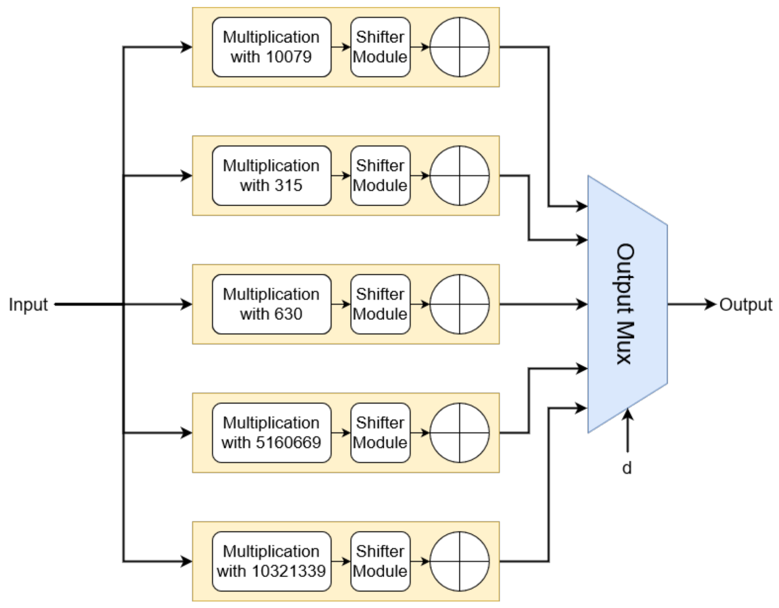

Figure 4.2.6: Compress Module in Kyber

4.2.4.3 Encode and Decode Unit

In case of Kyber byte arrays and polynomial vectors are needed to serialize to byte arrays. When encoding is applied to a vector polynomial, it is applied to individual polynomials and output byte array is concatenated. In our hardware architecture, we concatenate 1, 4, 5, 10, 11 bits long coefficients that is generated from the compress module. The same goes for the decode unit.

4.2.5 Prototype for Automotive Context

We have chosen a lightweight Xilinx board NEXYS 4 DDR which houses an Artix-7 FPGA and a soft-core microprocessor Microblaze which are recently being promoted for automotive contexts[1]. In order to achieve a stand-alone design from an automotive perspective, we have applied the reset signal to our hardware design using the softcore Microblaze microprocessor available on our target platform. Figure 4.2.7 illustrates the handshake between the Microblaze and our crypto co-processor. Our prototype can be extended to implement other post-quantum algorithms to enhance the performance of modern in-vehicle HSMs.

[1]https://www.xilinx.com/support/documentation/white_papers/wp501-microblaze.pdf

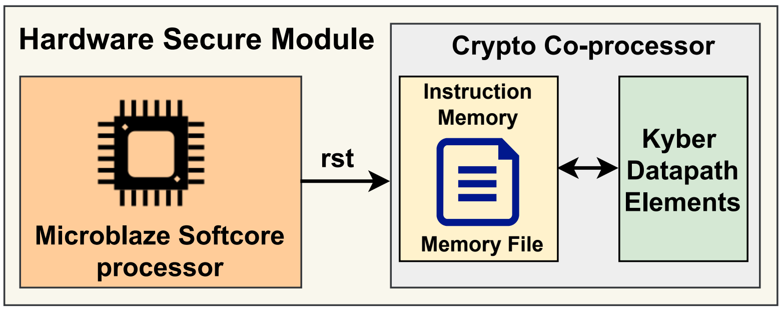

Figure 4.2.7: Overall architecture of the prototype Architecture

4.2.5.1 Programmable Architecture Development for Executing Kyber

In our work, we have opted for a bottom-up process for this architecture development. Such a process of development is common in custom designs as it involves the development of the basic datapath elements in an optimized manner. The top-level architecture in this design comprises polynomial arithmetic modules, control logic, and memory interconnected through the data path and control path. Hence, each sub-block has been developed separately and integrated to realise the full design.

Block memory has been used to implement the primary controller. The small area footprint and reconfigurability of BRAM are the main benefits of adopting a BRAM-based FSM as opposed to a strictly logic-based FSM. Pure logic-based FSMs perform better because they operate faster (BRAM operates slower than standard FPGA logic fabric). However, a lightweight programmable architecture cannot use such an FSM since it incurs a significant area overhead and makes the design fixed. As a result, we use a BRAM-based FSM as the main control element in our architecture. The control signals are associated with a set of limited instructions and functionalities. As the motivation of our design mostly involves the acceleration of Kyber, we have not included generic or atomic instructions in our instruction set which may unnecessarily increase the latency of the architecture. For example, in order to execute SHA3 in our architecture, we repeatedly execute the KECCAK instruction and padding instruction based on the length of the input. The intermediate states involve few load and store instructions via a limited set of intermediate registers. Now as there are specialized sets of hardware for modules like compress/decompress, Encode/Decode, etc. special set of start signals within the opcode of the instruction set activates these modules. The same goes for the polynomial multiplier module.

Algorithm	Components	LUTs	DSPs	BRAMs
Kyber	Compress/Decompress	258	-	-
	Encode	581	-	-
	Decode	268	-	-
	COPY	30	-	-
	CMOV	35	-	-
	Rejection Core	185	-	-
	Verification Core	98	-	-
	KECCAK	10879	-	-
	Data Memory	-	-	19
	NTT Multiplier	2899	4	1
	Controller	2618	-	5
Total		17845	4	25

Table 4.1: Total area utilization of the different components in the architecture

Works	Algorithm	LUTs	FF	DSP	BRAM	Max Freq (MHz)	Keygen		Encaps		Decaps		AT Product
							cycles	μs	cycles	μs	cycles	μs	
[9]	Kyber-512	15K	3K	11	14	25	75K	3000	132K	5280	142K	5680	164.4
	Kyber-768	15K	3K	11	14	25	112K	4480	178K	7120	191K	7640	221.4
	Kyber-1024	15K	3K	11	14	25	149K	5960	223K	8920	241K	9640	278.4
[52]	Kyber-512	7K	5K	2	3	161	4K	24.8	5K	31.05	7K	43.47	0.5
	Kyber-768	7K	5K	2	3	161	6K	37.2	8K	49.68	10K	62.11	0.77
	Kyber-1024	7K	5K	2	3	161	9K	55.9	11K	68.32	14K	89.95	1.07
[16]	Kyber-512	12K	10K	8	15	210	-	-	3K	14.28	4K	19.04	0.42
	Kyber-768	12K	10K	8	15	210	-	-	4K	19.04	5K	28.5	0.55
	Kyber-1024	12K	10K	8	15	210	-	-	5K	23	7K	33	0.75
[12]	Kyber-512	11K	10K	8	13	200	2K	10	2K	10	4K	20	0.34
	Kyber-768	12K	10K	12	14	200	3K	15	3K	15	5K	25	0.48
	Kyber-1024	13K	12K	16	16	185	3K	15	4K	20	6K	30	0.728
Our Design	Kyber-512	17K	12K	4	25	161	2.5K	15.85	3.4K	21.27	4.2K	26.32	0.85
	Kyber-768	17K	12K	4	25	161	3.5K	21.9	4.3K	27.11	5.1K	31.85	1.05
	Kyber-1024	17K	12K	4	25	161	4.4K	27.5	5.3K	33.11	6.4K	39.89	1.3

Table 4.2: Comparison of resource utilization and timing details of Kyber with existing architecture

With this flexibility comes the hardware-software communication cost. If the user has to send every instruction one by one to execute, then this communication time would take a toll on the total run-time. It also requires constant user interaction. To avoid this, we design a program controller with a small memory for storing instructions. The user just needs to send all the data and instructions in the beginning, and then hand over the control to the program controller. The program controller then ensures that all the instructions get executed correctly and returns a done signal in the end.

4.3 Results and Comparisons

In this section, we will analyze the performance of the proposed architecture with respect to state-of-the-art designs. The architecture consumes 17845 LUTs, 12531 FFs, 4 DSPs and 25 BRAMs achieving a Maximum frequency of 161 MHz. The resource utilization and latency of implemented Kyber are showcased in Table 4.2. All the designs in Table 4.2 are based on Artix-7 platform. We have presented the comparison of area-time product with respect to the state-of-the-art design of Kyber by adding up their LUT consumption and multiplying with their achieved latency of implementation.

If we look into the existing designs of Kyber, [9] have implemented several lattice-based schemes on a RISC-V-based architecture with the target to achieve a low-power design. But contrary to their hardware/software co-design approach, we have outperformed them based on their achieved latency. On the other hand, [52] has showcased a compact design of Kyber achieving a low area consumption with reasonable latency. Designers of [12] have by far implemented the most high-speed design of Kyber based on their efficient modular reduction technique and optimized NTT multiplier.

4.4 Conclusion

In this work, we optimize the datapaths of Kyber such that other compatible post-quantum algorithms can be appended with this design. This leads to a comprehensive block which suits nicely hardware security modules for automotive applications. Furthermore, our design offers an agile choice to the designer by adopting a micro-code-based architecture which can seamlessly program the block to operate at different security levels. We demonstrate through several experiments on the prototype FPGA cum Microblaze solution. In the next chapter, we show that this same framework is amalgamated with Dilithium and we have built up a basic structure of post-quantum TLS. We show that by combining this Kyber and Dilithium based on their similarities, we are able to achieve the best area-time product compared to the standard design in the literature.

Chapter 5

PQTLS: A Unified Agile Co-Processor for Post-Quantum TLS

5.1 Introduction

Modern public-key cryptography (PKC) is centered on a few well-known hard problems, such as integer factorization, discrete logarithm, elliptic curve discrete logarithm, etc. It is a well-accepted fact by now that quantum computers solve such hard problems (mainly attributed to Shor's algorithm), and, therefore, become imminent threats to existing PKC. Fortunately, significant research effort has already been undertaken to develop new alternative "quantum-safe" (also called "post-quantum cryptography" aka. PQC) PKC schemes, and a standardization process is ongoing due to National Institute of Standards and Technology (NIST). Among different alternatives, the schemes based on worst-case lattice problems have received particular attention, mainly due to their simplicity and ease of implementation. The finalist portfolio of NIST indeed contains 5 candidates based on lattice problems.

While post-quantum schemes are already in place, incorporating them in commodity products remains an open problem, because several adjustments have to be made with respect to resource utilization, bandwidth, energy consumption, etc. – both in the schemes, and the respective products. Nevertheless, it has been deemed important [53], and efficient implementations of the key-encapsulation and digital signature schemes – the two most important components of PKC have already started coming up. Automotive security is one of the most prominent areas, where PQC support is needed [44].

Especially for remote diagnostics and software over-the-air updates, public-key support (and therefore PQC) is essential. Recently, this has been acknowledged in [51] , where several PQC candidates (XMSS, BIKE, Dilithium, and Kyber) have been studied in a hardware/software co-design framework and recommended for automotive hardware security modules (HSMs). The need for having PQC support in hardware is evident, given that it rules out certain software attack surfaces. This need has also been acknowledged in the E-safety vehicular intrusion protection applications (EVITA) project [17] as a requirement in the security level "high". Communication between ECUs or connected devices must be secured. Appropriate mechanisms are required to ensure at least data origin authenticity and integrity. For some scenarios (such as piracy or privacy protection), also confidentiality might be required. Thus, a need for protocols such as TLS combined with the latest post-quantum algorithms, for securing in-vehicle communication is now under investigation. However, most of the existing proposals for automotive PQC are software implementations [53]. The only work [51] on hardware implementation proposes the use of PQC based on existing designs and does not shed light on how the PQC implementations need to be optimized for an automotive context. Due to the complexity of the PQC algorithms and the real-time requirements of an automotive system due to the safety critical needs, developing efficient hardware solutions are imperative. On the other hand the designs need to be lightweight and less resource hungry. Other than these conflicting requirements, the hardware should also be agile, to cater to the various choices of PQC algorithms and their adoption with the least time to market. An important constraint in the automotive context is that the implementations have to be lightweight. On the other hand, speed cannot be fully compromised to save resources given that automotive data often handles critical control tasks. Other than these two conflicting design goals, another important factor is the agility of the implementations, so that they ease the overall time-to-market.

In this chapter, we propose PQC implementation in the context of automotive security. Our primary aim is to address the three design goals presented in the last paragraph. As the first step to this, we judiciously choose the PQC candidates from the NIST portfolio which are best-suited for automotive purposes. Lattice-based schemes are advantageous choices for both key-encapsulation mechanism (KEM) and signatures given their simple constructions and ease of implementation. Even among the lattice-based schemes, we choose Kyber and Dilithium owing to their high degree of mathematical and structural similarities. Our next step is to amalgamate these two schemes which allow us to create lightweight hardware compared to the state-of-the-art. During the design, several critical optimizations were made to make the designs amenable to the merger, and we carefully optimize the area-time product (thanks to some novel observations on the polynomial multiplication unit) to come up with a lightweight design with practical speed. To bring forth the much-needed agility, we choose a micro-code-based design style, where updating the memory can tune the design to operate on different parameter choices. An immediate advantage of this design choice is that we can support different levels of security on the same hardware. We prototype the entire design on a NEXYS 4 DDR FPGA platform along with a Microblaze soft-core processor and show that our design outperforms the state-of-the-art in terms of area-time product.

It is worth mentioning that a concurrent work [3] has combined two other NIST candidates (namely, Saber and Dilithium). However, their main goal was to present a unified design approach for PQC schemes where different common blocks can be combined. The proposed design brings certain changes in the traditional Saber structure, such as the use of prime-lifting strategy, to make Saber support odd primes and amenable to combination with Dilithium. While we acknowledge this as a potential alternative strategy, we also emphasize that it is not the best-suited for automotive context given design time and effort would be much higher than that of a Kyber-Dilithium combination.

The rest of the chapter is organized as follows. A brief background and design rationale is presented in Section 5.2. Section 5.3 demonstrates the design challenges and components. Section 5.4 presents the results and comparison of the proposed design followed by a conclusion in Section 5.5.

5.1.1 Our Contribution

In this chapter our major contributions are as follows:

- We have implemented a combined design of Kyber and Dilithium banking of a similar structure of the NTT multiplier.

- We have also implemented a common KECCAK core with a wrapper to implement several hashing algorithms for the two schemes.
- We have replaced the key encapsulation and the signature algorithm of TLS 1.3 to create a version of post-quantum TLS which may be implemented for resource-constrained architecture.
- Our design is the best in terms of Area-Time product based on the available literature.

5.2 Design Rationale

The state-of-the-art Post-quantum cryptography (PQC) algorithms as per the NIST standardization portfolio, are mostly based on lattice problems. Lattice-based cryptographic algorithms have been a preferred choice for practical realizations due to certain implementation-friendly mathematical properties. Most of the public-key applications, such as authenticated key exchange (AKE) or transport layer security (TLS) (which are also common in automotive applications) require both KEM and signature schemes. In order to fulfil this need, we, therefore, choose a lattice-based KEM and a signature scheme from the NIST portfolio. This choice is crucial for maintaining a practical performance under the resource constraints imposed by automotive applications. As already pointed out in the introduction, [3] have chosen Saber and Dilithium and combined them within a single hardware core. The referred work is, however, not targeted towards automotive applications. Secondly, Saber is based on an MLWR (Module Learning with Rounding) whereas Dilithium is based on MLWE (Module Learning with Error), which are two different hard-problems. Moreover, Saber, due to its parameter choices and mathematical properties, does not inherently support a Number Theoretic Transform (NTT)-based multiplier. On contrary, Dilithium utilizes an NTT-based architecture. In order to combine these two, the authors of [3] have used prime lifting technique to forcefully make Saber support a similar NTT architecture as Dilithium. This effectively doubles the width of the Saber datapath (making it 24 bits, while the original proposal was of 13 bits), and also incur extra NTT operations which are computationally intensive.

Motivated by the aforementioned issues with a Saber-Dilithium combination, we have chosen Kyber and Dilithium for our purpose. These two provide certain useful mathematical properties which are amenable to an efficient combination. More precisely, Kyber and Dilithium share several compute-intensive components and a common prime structure leading to a unified polynomial multiplier (NTT-based) along with common Keccak-based hash functions. We note that combined hardware is a natural choice for reducing the area. However, it requires resources to be shared, which also thwarts the possibility of parallelization and may affect the overall throughput. Interestingly, in our context, such resource sharing does not affect the performance much as protocols, such as TLS or AKEs, usually do not compute the KEM and signatures together. In principle, only the key-generation part of KEM and signature may (optionally) run in parallel, but the rest of the components are always run at distinct steps of such protocols. This also motivates the choice of having combined hardware, as the overall performance does not suffer much due to the combination.

However, coming up with a proper combination is a non-trivial problem even though the two chosen designs ease the combination. In practice, there are several design choices and observations involved which can influence the performance. The goal of this work is, therefore, to come up with such design decisions which eventually lead to a practical area-time tradeoff.

5.2.1 Keeping it Agile

Apart from the area-time product, another crucial aspect of automotive designs are their agility[1]. In order to achieve agility in our proposed architecture, we have resorted to a microcode-based design framework so that we can update the instructions to implement different post-quantum protocols. The agility is not only persistent across different protocols but also across separate security levels of the same scheme. In our proposed architecture, we have a datapath with discrete components controlled by signals generated from the instructions in the instruction memory. We have designed custom instructions to execute both Kyber and Dilithium using the micro-coded control signals. The instructions are 106-bits wide: 40 bits for instruction code and 66 bits for operand addresses. We can group the instructions as follows:

Polynomial Arithmetic Instructions: This group of instructions controls the poly-

[1]https://semiengineering.com/the-battle-for-post-quantum-security-will-be-won-by-agility/

nomial arithmetic operations which include the NTT multiplier, polynomial addition, and subtraction blocks. The instructions are capable of configuring the parameters of the NTT multiplier which enables the polynomial multiplier block to perform forward NTT, inverse NTT and point-wise multiplications.

Keccak Instructions: This specific group of instructions manages the execution control of the cryptographic hash module, particularly - starting the Keccak permutation engine and executing the sponge and absorb functions. These instructions also control the Keccak wrapper module functions that generate the digest for SHA3 and SHAKE.

Read/Write Instructions: This group of instructions is responsible for reading and writing polynomial data to and from Data memory.

Kyber Auxiliary Functions: This group of instructions is responsible for controlling certain auxiliary components of Kyber such as Encode, Decode, etc as mentioned in [7].

Dilithium Auxiliary Functions: This group of instructions is responsible for controlling certain auxiliary components of Dilithium such as Power2Round, MakeHint, UseHint, etc. based on the standard implementation as mentioned in [18].

For different security levels of Dilithium and Kyber several parameters are being affected within the scheme as indicated in [18] and [7]. One of the major changes is the number of polynomial multiplications while executing different parts of the protocol. The major advantage of our micro-code-based design methodology lies in the fact that we can repeatedly use the same polynomial arithmetic modules by repeating the same instructions over and over with different data memory addresses.

For example, while implementing Kyber three different values of k $(2, 3, 4)$, signify 3 different security levels. The value of k, has multiple effects while implementing Kyber. One of the effects involves performing NTT on k different polynomials at a time for a $k \times 1$ polynomial matrix. So, when $k = 2$, the number of consecutive instructions required to perform NTT are 496 while for $k = 3$, it is 672. Even while executing the two different schemes Kyber and Dilithium, we are switching between two different primes. A dedicated multiplexer switches between the two primes based on the value of the select line which is triggered by a dedicated control signal micro-coded in the instruction set.

It is also worth mentioning that the architecture in [3] suffers from a lack of agility due to the choice of Saber as their KEM scheme. The authors have reported results only for a combination of medium and low security levels of Saber and Dilithium respectively. The lifted prime used in their hardware architecture might be suitable for LightSaber (low security) but FireSaber (high security) needs a larger prime, thus defying the narrative of a unified architecture.

5.3 Proposed Architecture and Design Decisions

A unique feature of the proposed architecture is a unified NTT-based polynomial multiplier coupled with a specially integrated Keccak core that is commonly available to both Kyber and Dilithium execution. The Keccak core is capable of computing both SHAKE and SHA3, and is primarily used for both pseudo-random number generation and hash computation. Sharing these two compute-intensive modules responsible for core processing tasks requires a combined and highly optimized co-processor architecture. Additionally, there are a few auxiliary blocks that are further optimized to accelerate both Kyber and Dilithium. In this section, we shall elaborate design details and strategies of the combined architecture.

5.3.1 High Level Overview of the Shared Architecture of Kyber and Dilithium

The complete hardware architecture of the proposed design is presented in Figure 5.3.1. The diagram shows the combined datapath of Kyber and Dilithium along with the instruction and data memory. The instructions stored within the instruction memory control the sequence of operations needed to execute either Kyber or Dilithium as required. The datapath integrates the logic and functional modules necessary to execute Kyber or Dilithium. Figure 5.3.1 depicts a unified NTT multiplier and a Keccak core as part of the shared architecture of Dilithium and Kyber. The sharing of these two compute-intensive blocks has helped us in achieving a low area-time product for the whole design. We also observed that we can obtain a significant speed-up with respect to the canonical software implementation of Kyber or Dilithium as certain modules can be realized in hardware with practically negligible overhead. The Encode and Decode modules of Kyber are rewiring operations in hardware which consume significant latency in software. Power2Round module of Dilithium basically performs bit-wise break up of an element in $\mathbb{Z}_q$ into higher-order and lower-order bits. An element $r = r_1 \cdot 2^d + r_0$ will be broken into r_0 and r_1, where $r_0 = r \, mod \, \pm 2^d$ and $r_1 = (r - r_0)/2^d$. Implementing this module in hardware involves a few shifts and bit extraction in hardware. The Decompose, MakeHint, and UseHint modules are interrelated and employ reasonably lightweight combinatorial blocks to realize addition, subtraction and multiplication with fixed constants based on the security levels implemented. The multiple outputs produced by these modules are fed to a multiplexer unit connected to each individual module, the appropriate value is extracted based on the value of the select line. The detailed description of the unified NTT multiplier and the Keccak blocks are described in the sections below.

5.3.2 Design of Unified NTT Multiplier

Polynomial multiplication is one of the most computationally intensive components in both Kyber and Dilithium. In order to develop our unified multiplier architecture, we exploit the structure of the field prime. Even though Kyber and Dilithium are based on NTT-friendly primes, their structural difference incurs a burden to construct a unified architecture. One of the major differences is that only 256'th root of unity exists for the Kyber prime; whereas for Dilithium prime, both 256'th and 512'th root of unities exist, due to which Kyber supports 7 levels of NTT while Dilithium supports 8 levels of NTT. Thus, we perform incomplete NTT for both Kyber and Dilithium. Incomplete NTT gives $n/2$ polynomials of degree 1. For incomplete-INTT, multiplication operation of two degree-1 polynomials is performed in the ring $\mathbb{Z}_q[x]/(x^2 - \omega^i)$ where ω is the n-th root of unity and i depends on the index of coefficients. These polynomials cannot be multiplied coefficient-wise. In the rest of the section, we will elaborate how we have combined the polynomial multiplication architecture of Kyber and Dilithium we have set up common ground.

Unification Based on Root of Unity. We computed the number theoretic transform using the 256'th root of unity for both Kyber and Dilithium. This eventually decreases the level of NTT for Dilithium from 8 to 7 but reduces the total number of multiplications in the process. But, we have full autonomy to perform even 8 levels of NTT using the same architecture. While executing pointwise multiplication with seven lev-

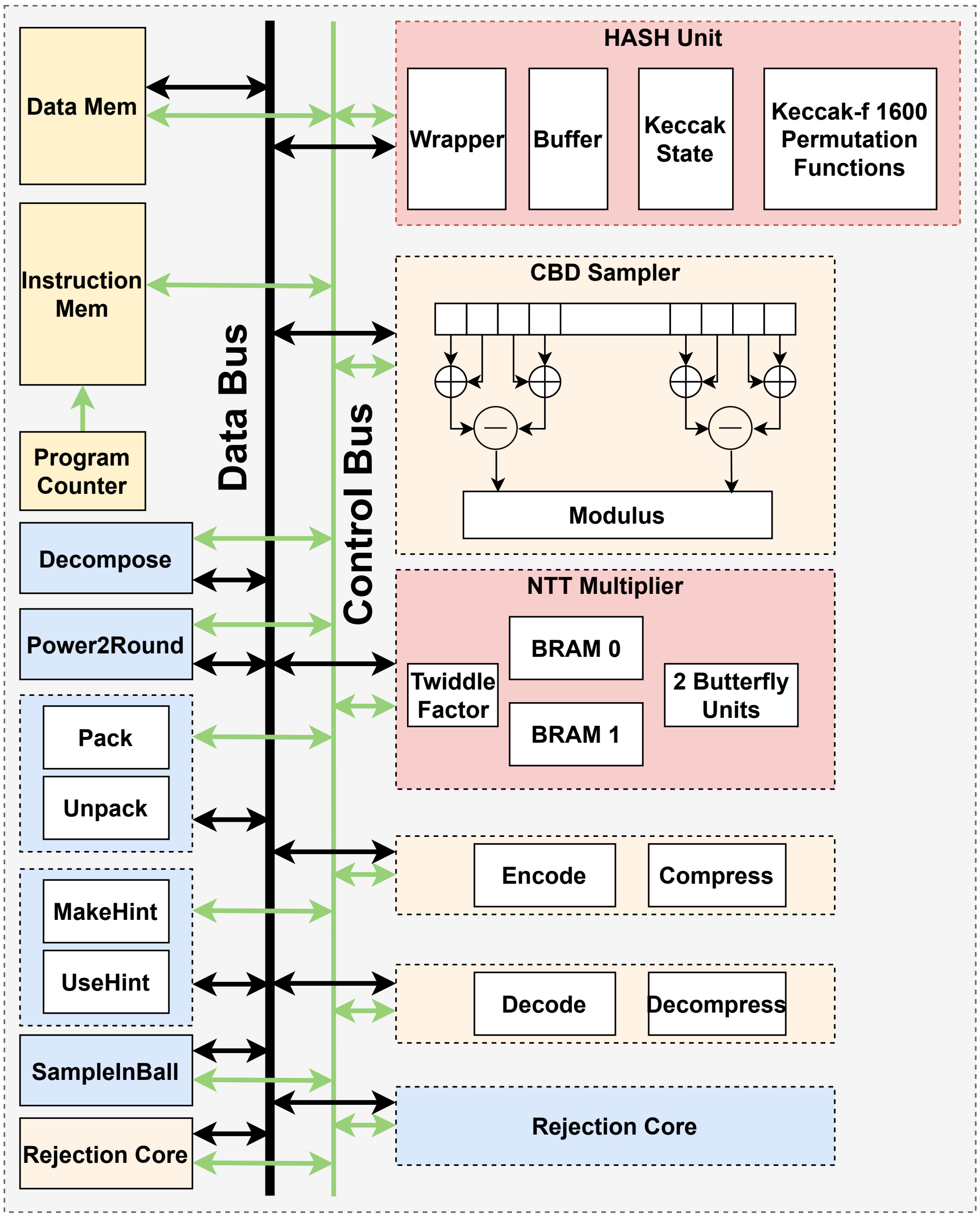

Figure 5.3.1: Hardware architecture of the crypto co-processor

els of NTT we followed the Karatsuba multiplication technique which is also adopted by [52]. Point-wise multiplication or basecase multiplication (referred in Kyber specification [7]) is: $h_{2i} + h_{2i+1}X = (f_{2i} + f_{2i+1}X) \cdot (g_{2i} + g_{2i+1}X) \, mod \, (X^2 - \zeta^{2br(i)+1})$. In a straight-forward way, we need to perform 5 multiplications. But as we can see from Equation 5.1, the number of multiplication decreases to 4 from 5 due to Karatsuba-based approach. When Dilithium is executed we had to perform 8-levels of NTT at few instances in order to preserve the mathematical structure of Dilithium but elsewhere we did stick to seven levels of NTT.

$$\begin{aligned} h_{2i} &= f_{2i}g_{2i} + f_{2i+1}g_{2i+1} \cdot \zeta^{2br(i)+1} \\ h_{2i+1} &= (f_{2i} + f_{2i+1})(g_{2i} + g_{2i+1}) - (f_{2i}g_{2i} + f_{2i+1}g_{2i+1}) \end{aligned} \tag{5.1}$$

Multiplication of two Dilithium polynomials with 512'th root of unity involves 3328 multiplications. In contrast, the above-mentioned Karatsuba-based method combined with 256'th root of unity involves 3200 multiplications. This eventually helps us in reducing the latency of executing the schemes.

Reduction Architecture Based on the Structure of Prime. In our proposed architecture we have used two different primes for Kyber and Dilithium in order to avert the disadvantage of computing an extra NTT computation as mentioned in Section 5.2. Kyber is based on a prime of the form $q = k \cdot 2^m + 1$, where $k = 13$ and $m = 8$ ($k < 2^m$). The previous hardware architectures of Kyber [12] exploit this form of prime to use the K^{2}RED reduction algorithm. In our proposed work we observed that the Dilithium prime also possesses the same structure as Kyber when $k = 1023$ and $m = 13$ and we have implemented as per Algorithm 6. We exploit this specific structure of prime to perform K^{2}RED reduction for both Dilithium and Kyber. The K^{2}RED algorithm takes an integer C as input and outputs an integer D such that $D \equiv k^2 \cdot C \bmod q$.

Algorithm 5: K^{2}RED Reduction Algorithm for Kyber

```
Input  : A binary number $C = (c_{23}, ..., c_0)_2$, $k = 13$, $m = 8$, $q = 3329 = k \cdot 2^m + 1$
Output: $C'' = k^2 \cdot C \bmod q$
1 $C_l = (c_7, \dots, c_0)_2$
2 $C_h = (c_{23}, \dots, c_8)_2$
3 $C' = k \cdot C_l - C_h$
4 $C'_l = (c'_7, \dots, c'_0)_2$
5 $C'_h = (c'_{15}, \dots, c'_8)_2$
6 $C'' = k \cdot C'_l - C'_h$
```

The K^{2}RED algorithm [12] is outlined in Algorithm 5 that implements the above-mentioned reduction. Note that, we can eliminate the extra k^2 factor by inserting an additional k^{-2} in the multiplication phase. We decompose the value of k into ($k = 16-3$)

Algorithm 6: K^{2}RED Reduction Algorithm for Dilithium

```
Input  : A binary number
         $C = (c_{45}, ..., c_0)_2$, $k = 1023$, $m = 13$, $q = 8380417 = k \cdot 2^m + 1$
Output: $C^{\varepsilon} = k^2 \cdot C \, mod \, q$
1 $C_l = (c_{12}, ..., c_0)_2$
2 $C_h = (c_{45}, ..., c_{13})_2$
3 $C^{'} = k \cdot C_l - C_h$
4 $C^{'}_l = (c^{'}_{12}, ..., c^{'}_0)_2$
5 $C^{'}_h = (c^{'}_{33}, ..., c^{'}_8)_2$
6 $C^{''} = k \cdot C^{'}_l - C^{'}_h$
```

for Kyber and into ($k = 1024 - 1$) for Dilithium to optimise the modulo operation. The value $C^{'}$ can now be expressed as $C^{'} = ((C_l \ll 4) - (3 \cdot C_l + C_h))$ for Kyber, and as $((C_l \ll 10) - (C_l + C_h))$ for Dilithium.These mathematical expressions can be realized using pure combinatorial logic. The modular reduction unit in our NTT multiplier architecture consists of these two combinatorial units followed by a multiplexer which produces output based on whether we are executing Dilithium or Kyber, which aids us in achieving a reduced latency. The small increase in resource consumption due to the combinatorial block is thus compensated with a reduced latency causing a reduction in area-time product.

Even though the Dilithium prime supports 512-th root of unity, we have deliberately implemented the NTT multiplier using 256-th root of unity, in order to compute only 7 levels of NTT during most of the computation (except when the matrix A was expanded). This as matter of fact decreases the total number of multiplications when combined with the Karatsuba technique to multiply two terms which we will elaborate on later in his subsection. Apart from using the prime, we have implemented both the Cooley-Tukey [15] and the Gentleman-Sande [21] algorithms to implement the NTT multiplier. For forward NTT, we used Cooley-Tukey and for reverse NTT, we used Gentleman-Sande algorithms to avoid the bit-reversal step. This technique is quite commonly used in some of the previous works. The Cooley-Tukey algorithm uses the Decimation in Time approach while Gentleman-Sande uses the Decimation in Frequency approach.

NTT computation. We opted for Cooley-Tukey for forward NTT (*Decimation in Time*) and we use Gentleman-Sande method for reverse NTT (*Decimation in Frequency*) avoids

Algorithm 7: Algorithm for Number Theoretic Transform

```
Input  : $p, N, q; twiddle\_factor\_array[N]$
Output: $\hat{p}$
1 $twiddle\_count = 1$
2 for $s = 2^{N-1}$ to 1 by $s/2$ do
3     for $start = 0$ to $N - 1$ by $j$+$s$ do
4         $zeta = twiddle\_factor\_array[++twiddle\_count]$
5         for $j = start$ to $start + s$ do
6             $t = zeta \cdot p[j + s] \bmod q$
7             $p[j + s] = p[j] - t \bmod q$
8             $p[j] = p[j] + t \bmod q$
```

bit-reversal, in polynomial multiplications. A similar architecture was also proposed in [52], though the use of our fast reduction technique has helped us achieve results at much lower latency. Figure 5.3.2 shows the basic structure of a butterfly unit which is capable of processing two coefficients at a time.

The proposed NTT multiplier houses 2 such butterfly computation units that are capable of processing four polynomial coefficients of Dilithium and eight polynomial coefficients of Kyber after each iteration. Additionally, the NTT multiplier block has two separate BRAM units capable of holding two coefficients of Dilithium and eight coefficients of Kyber in a single memory cell separated by an index of s, where $s \in \{128, 64, 32, 16, 8, 4, 2\}$ feeding the two butterfly units. Hence, for example, when Dilithium is executed, the two butterfly cores receive the j'th, $(j+s)$'th, $(j+1)$'th and $(j+s+1)$'th coefficients simultaneously, where $j \in [0, 255]$. The corresponding schedule of the multiplier execution is illustrated in Figure 5.3.3, where a_i are the polynomial coefficients. Both forward and reverse NTT require 448 clock cycles.

The butterfly unit in Figure 5.3.2 can operate in 3 separate modes: Forward NTT, Inverse NTT, and point-wise multiplication based on the values of the select lines of the MUXes. Depending on the operating mode, the input c in Figure 5.3.2 can be switched between twiddle factors or a polynomial coefficient. These operations are controlled by a group of polynomial multiplier instructions as described in Section 5.2.1. The control signals micro-coded in these instructions bring out agility both in terms of different security levels and between the two schemes. Additionally, with our fast reduction technique as described above, the NTT multiplier helped us bring down the area-time

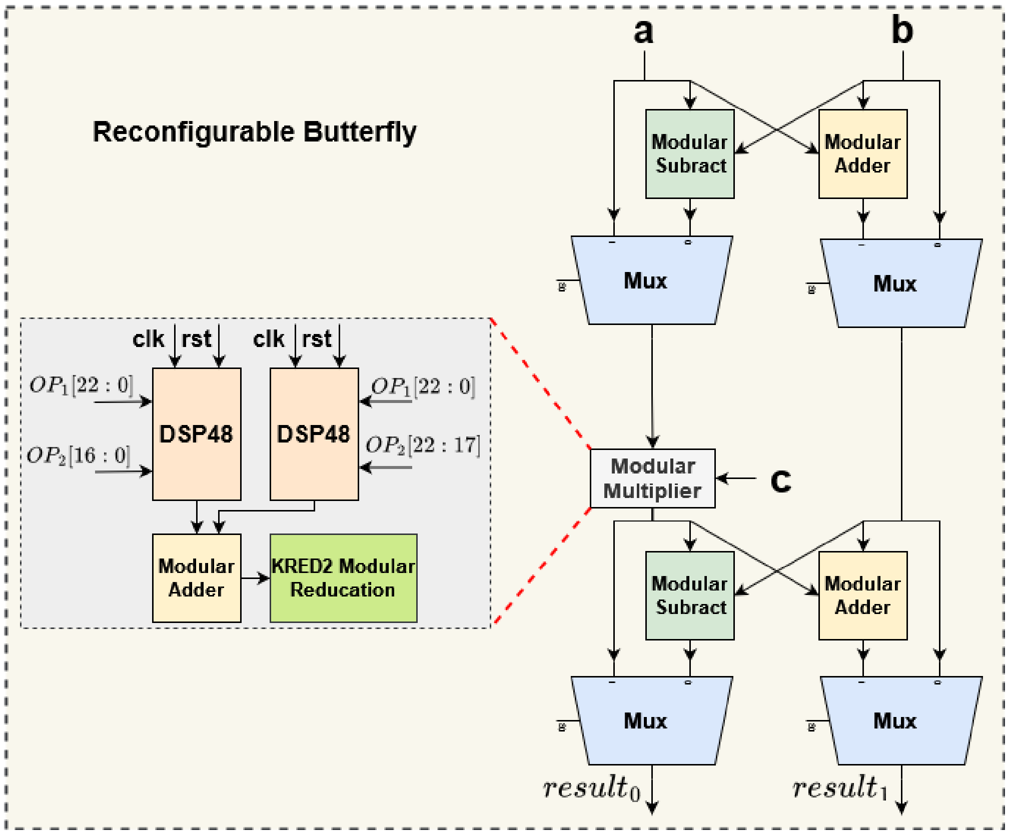

Figure 5.3.2: Re-configurable butterfly unit

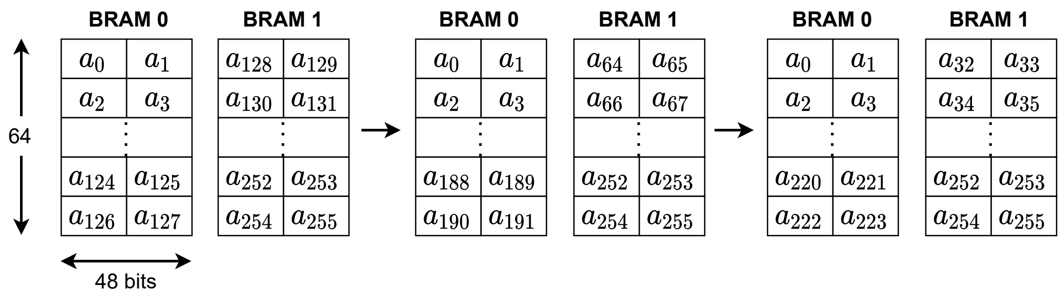
BRAM 0
BRAM 1
BRAM 0
BRAM 1
BRAM 0
BRAM 1
64
48 bits
a_0 a_1 a_2 a_3 a_{124} a_{125} a_{126} a_{127}
a_{128} a_{129} a_{130} a_{131} a_{252} a_{253} a_{254} a_{255}
a_0 a_1 a_2 a_3 a_{188} a_{189} a_{190} a_{191}
a_{64} a_{65} a_{66} a_{67} a_{252} a_{253} a_{254} a_{255}
a_0 a_1 a_2 a_3 a_{220} a_{221} a_{222} a_{223}
a_{32} a_{33} a_{34} a_{35} a_{252} a_{253} a_{254} a_{255}

Figure 5.3.3: NTT multiplier scheduling for Dilithium in the NTT RAM

product of our design.

5.3.3 Sampling

Kyber sampler. Kyber uses a binomial sampler that samples integers from a centered binomial distribution. We integrate two separate samplers $\mathcal{S}_1$ and $\mathcal{S}_2$ with $\eta = 2$ and $\eta = 3$, respectively. We implement 16 concurrent adder cores to optimize sampler $\mathcal{S}_1$. Sixteen samples are obtained from $\mathcal{S}_1$ and stored in a buffer. In contrast, $\mathcal{S}_2$ uses an input buffer for concatenating with the input in the next cycles. Three 64-bit words are read to obtain 32 two-word samples from $\mathcal{S}_2$. The group of instructions as described in Section 5.2.1 helps us in selecting between the two sampling cores based on the different security levels.

Dilithium sampler. Unlike Kyber, the Dilithium architecture uses three separate rejection cores. The KECCAK unit feeds the data directly into the rejection core and is implemented using a lookup approach. The instructions as specified in Section 5.2.1 control the selections of the rejection cores along with their respective operations. The public matrix-vector and secret key coefficient generation in Dilithium use uniform rejection sampling.

5.3.4 Programmable Keccak Core

The Keccak core [11] in this architecture is utilized for cryptographic hash computation and pseudo-random number generation by both Kyber and Dilithium. Kyber requires four modes of the Keccak core - namely SHA3-256, SHA3-512, SHAKE-128, and SHAKE-256. Whereas, Dilithium requires two modes - SHAKE-128 and SHAKE-256. All of these modes are implemented using the Keccak sponge structure internally equipped with separate wrappers and individual buffers that are multiplexed based on the micro-coded control signals. The requisite wrapper is selected based on the micro-coded control signals generated by the group of Keccak instructions as mentioned in Section 5.2.1. The basic architecture of the Keccak core implemented in our design is shown in Figure 5.3.4 and 5.3.5. They share the same Keccak-f[1600] function but with different rate r and different message suffix appended during padding for domain separation. The constitution of r bits out of 1600-bit input is decided by the wrapper module. Parts of register data, appended suffix and part of padding data are selected through multiplexer under the control of the underlying FSM and the Instruction memory. This also defines the agility of our architecture as we are using a single

Keccak core for both schemes across all the security levels. Keccak core works in a way such that the data would be fetched from the buffer until enough data has been collected to begin the next hashing process. The control signals are fed into module which controls the absorb, and squeeze mode. Absorb indicates the input corresponding to rate part should be XORed with r bits of the current 1600-bit Keccak state as one 24-round Keccak-f function can not fully absorb the input message bits. A $finish$ signal indicates to form the 1600 bits input to Keccak core for the next hash cycle to occur. The Keccak buffer directly feeds to the NTT block and the Rejection core to save time for the polynomial multiplication block. In a way, this also brings down the resource consumption of the entire architecture aiding the reduction of the area-time product of the design. The final Keccak sponge implementation requires 24 clock cycles, and the final resource overhead for the complete hash unit is 10879 LUTs (Artix-7 FPGA). Despite the heavy resource overhead of the Keccak unit, the multiplexed architecture is necessary to support the required security level across Kyber and Dilithium.

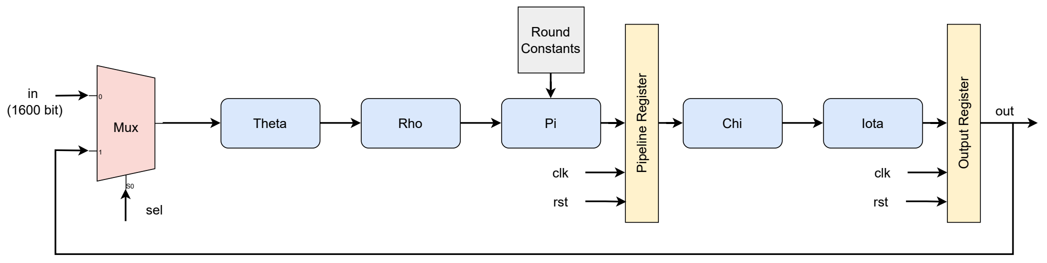

Figure 5.3.4: Proposed architecture of transformation round

5.3.5 Auxiliary building blocks

Apart from the abovementioned functional modules, several other auxiliary hardware modules are present in this design which we discuss briefly. The data memory stores the polynomial coefficients, and two polynomial coefficients are available from the data memory simultaneously during computation. Individual modules are developed for **Pack** and **Unpack** routines which involve rewiring operations followed by certain addition, subtraction and shift operations in hardware, which are much slower in software. Other auxiliary modules including **Power2Round**, **MakeHint**, **UseHint**, **Encoder**, **Decoder**, and **Decomposer** involves a set of combinatorial circuits as arithmetic operations with certain constants based on different security levels. The canonical

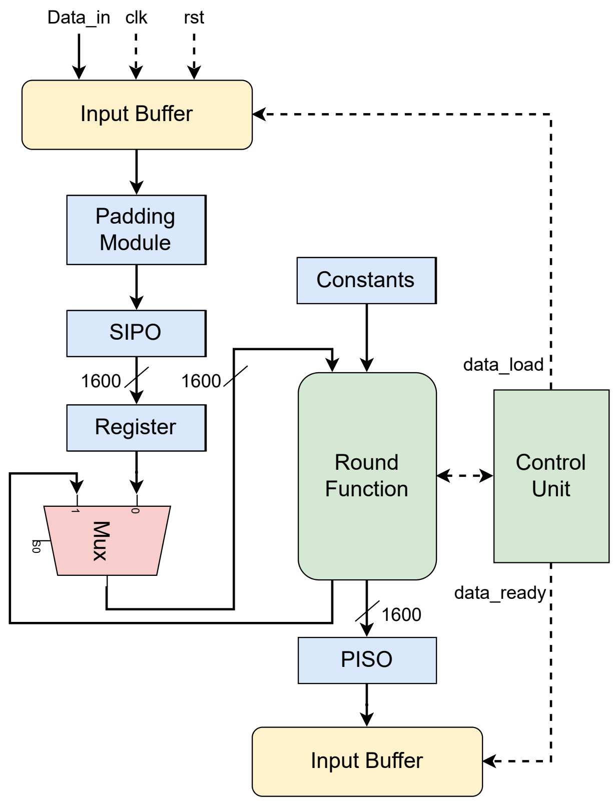

Figure 5.3.5: Proposed architecture of the KECCAK hash function

implementations of the auxiliary modules of Kyber and Dilithium are well-described in [7] and [18]. The polynomial arithmetic module with adder and subtractor is designed to operate in a non-blocking fashion such NTT module can operate concurrently.

5.3.5.1 Compress/Decompress

The decompress unit performs division by power-of-two and rounding operation which is trivial to implement in hardware. On the other hand, the compress operation requires division by q and rounding. Some works in the literature use Barrett reduction and division algorithms to perform compress operation. The compress operation basically involves the mathematical operation $Compress_q(x, d) = \lceil 2^d/q \cdot x \rfloor mod^+ 2^d$. The value of d are as follows: $1, 4, 5, 10, 11$. We have implemented this by rounding off the value $2^d/q$ to the nearest integer and multiplying it with a power of 2. Then after a few bit shifts and addition operations, we obtain the proper output. The reverse op-

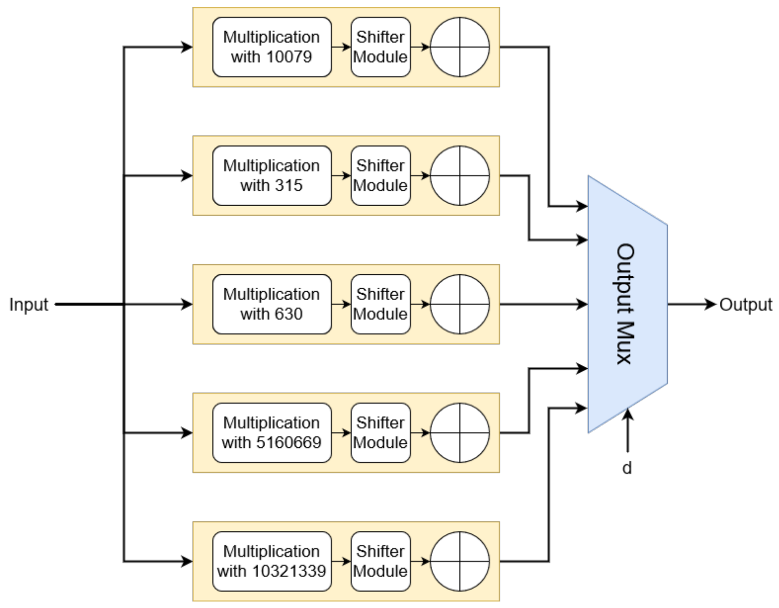

Figure 5.3.6: Compress Module in Kyber

eration helps in obtaining the output of the decompress operation. This module is implemented in a very optimized manner with the required precision such that the output matches the canonical implementation of the Compress/Decompress module. The compress module is implemented in hardware as shown in Figure 5.3.6.

5.3.5.2 Encode and Decode Unit

In case of Kyber, byte arrays and polynomial vectors are needed to serialize to byte arrays. When encoding is applied to a vector polynomial, it is applied to individual polynomials and the output byte array is concatenated. In our hardware architecture, we concatenate 1, 4, 5, 10, 11 bits long coefficients that are generated from the compress module. The same goes for the decode unit.

5.3.5.3 Power2Round, HighBits and LowBits Operations

Power2Round operation in Dilithium involves splitting the individual polynomial coefficients into upper 10 bits and lower 13 bits. The polynomial consisting of the upper 10 bits are packed into the public key and the polynomial consisting of the lower 10 bits is packed into the secret key. In our implementation as we have used positive coefficients, if the MSB is 1 then the coefficients are transformed into positive by adding q. This

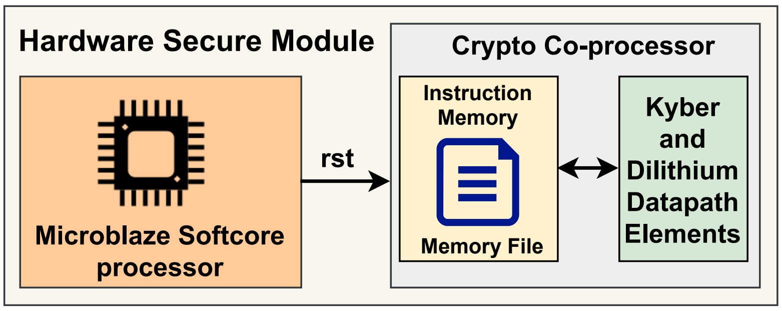

Figure 5.3.7: Overall architecture of the prototype HSM

operation is a simple rewiring operation with a reasonable number of LUTs involved. HighBits and LowBits are implemented as lookup operations in our design which also provides optimization while implementing the MakeHint and UseHint modules.

5.3.5.4 MakeHint and UseHint Operations

MakeHint operation involves the values of $w - cs_2$ and $w - cs_2 + ct_0$ being stored separately in such a way that both can be read at the same time and eventually the HighBits for both are looked up. If the HighBits differ a new offset is shifted in. In the same way for UseHint operation, HighBits for each coefficient are looked up and then the correct value is shifted which is eventually used to compute the value $\tilde{c}$. This is finally used for signature verification.

5.3.6 Prototype for Automotive Context

We have chosen a lightweight Xilinx board NEXYS 4 DDR which houses an Artix-7 FPGA and a soft-core microprocessor Microblaze which are recently being promoted for automotive contexts[1]. In order to achieve a stand-alone design from an automotive perspective, we have applied the reset signal to our hardware design using the softcore Microblaze microprocessor available on our target platform. Figure 5.3.7 illustrates the handshake between the Microblaze and our crypto co-processor. Our prototype can be extended to implement post-quantum TLS protocol on the Microblaze microprocessor and our proposed hardware shared architecture as a crypto-accelerator in order to enhance the performance of modern in-vehicle HSMs.

[1]https://www.xilinx.com/support/documentation/white_papers/wp501-microblaze.pdf

5.3.6.1 Programmable Architecture Development for Executing the TLS Protocol

In this work, we have opted for a bottom-up process for this architecture development. Such a process of development is common in custom designs as it involves the development of the basic datapath elements in an optimized manner. The top-level architecture in this design comprises polynomial arithmetic modules, control logic, and memory interconnected through the data path and control path. Hence, each sub-block has been developed separately and integrated to realise the full design.

Block memory has been used to implement the primary controller. The small area footprint and reconfigurability of BRAM are the main benefits of adopting a BRAM-based FSM as opposed to a strictly logic-based FSM. Pure logic-based FSMs perform better because they operate faster (BRAM operates slower than standard FPGA logic fabric). However, a lightweight programmable architecture cannot use such an FSM since it incurs a significant area overhead and makes the design fixed. As a result, we use a BRAM-based FSM as the main control element in our architecture. The control signals are associated with a set of limited instructions and functionalities. As the motivation of our design mostly involves the acceleration of Kyber and Dilithium, we have not included generic or atomic instructions in our instruction set which may unnecessarily increase the latency of the architecture. For example, in order to execute SHA3 in our architecture, we repeatedly execute the KECCAK instruction and padding instruction based on the length of the input. The intermediate states involve few load and store instructions via a limited set of intermediate registers. Now as there are specialized sets of hardware for modules like compress/decompress, Power2Round, etc. special set of start signals within the opcode of the instruction set activates these modules. The same goes for the polynomial multiplier module. The only unifying factor when the multiplier works for Dilithium and Kyber is the same length of the polynomial,i.e. 256. The prime modulo for both schemes is already programmed into the multiplier module.

Now in order to execute TLS, the server side needs to perform the key generation step for both Kyber and Dilithium in our case. With just two instructions passed from the Microblaze to the Artix-7 FPGA on the Nexys4-DDR board, key generation steps are executed for both Dilithium and Kyber which are already programmed onto the instruction memory of the design. Now the key generation for both the schemes can be performed in parallel, even though the schemes share a common polynomial multiplier and a KECCAK Unit. We have prepared the micro-coded instructions in a special manner for this acceleration. In Figure 5.3.8, we can see that based on the timeline certain steps of key generation are executed in parallel. While executing the key generation step of Dilithium itself we are able to execute the rejection sampling and the KECCAK operation parallelly. Similarly, while executing the key generation step for Kyber, the KECCAK module and the Binomial Sampler module can be operated parallelly. Also it is evident from the scheduling diagram, the KECCAK modules and the NTT modules are not operated parallelly while the key generation process runs for Kyber and Dilithium. We were able to follow this particular scheduling owing to the agility of our design. We did not have to alter the datapath of our design in order to accelerate this key generation process for both schemes. But the next steps, such as the encapsulation process, decapsulation process, signature generation step and signature verification steps are needed to be executed sequentially. The reason for this is that Key Encapsulation step and the Signature verification step should run on the client side in a sequential manner while the Key decapsulation step and the Signature generation step runs on the server side in a sequential manner. We have included a special decoder module on the hardware part which interprets to the start of these operations once the appropriate signal is received from the Microblaze softcore processor. Basically, the program counter receives instructions to start the execution of operation from a particular address in the instruction memory.

With this flexibility comes the hardware-software communication cost. If the user has to send every instruction one by one to execute, then this communication time would take a toll on the total run-time. It also requires constant user interaction. To avoid this, we design a program controller with a small memory for storing instructions. The user just needs to send all the data and instructions in the beginning, and then hand over the control to the program controller. The program controller then ensures that all the instructions get executed correctly and returns a done signal in the end.

5.3.6.2 Our Implementation of TLS

The TLS 1.3 protocol involves the following steps:

1. The TLS 1.3 handshake commences with the “Client Hello” message – with one

5.3.6.2. Our Implementation of TLS

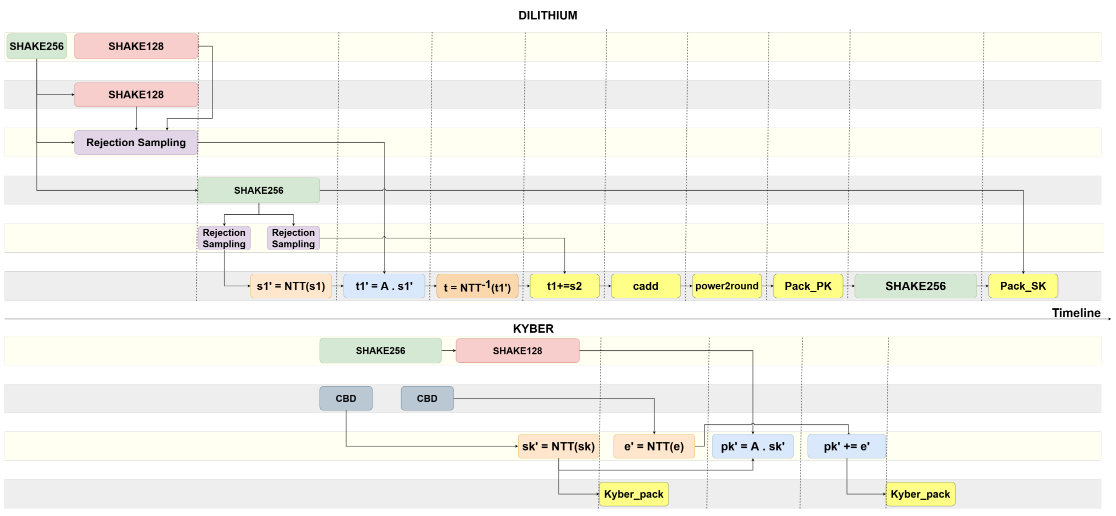

Figure 5.3.8: Parallel Scheduling of the Key generation of Kyber and Dilithium

significant change. The client sends the list of supported cypher suites and guesses which key agreement protocol the server is likely to select. The client also sends its key share for that particular key agreement protocol.

2. In reply to the "Client Hello" message, the server replies with the key agreement protocol that it has chosen. The "Server Hello" message also comprises of the server's key share, its certificate as well as the "Server Finished" message.

3. Now, the client checks the server certificate, generates keys as it has the key share of the server, and sends the "Client Finished" message. From here on, the encryption of the data begins.

In our implementation, we have proposed to replace the traditional algorithms used in TLS 1.3 with Kyber and Dilithium. With our agile methodology, the user can select the security level with which the TLS protocol is need to be implemented. We have excluded the integration of the TLS with the network layer and rather motivated our design to accelerate the Key Encapsulation and Signature generation operation. In Figure 5.3.9, we have reintroduced our version of TLS 1.3. In our design, the entire Key Encapsulation operation and Signature generation operation is implemented on the Artix-7 FPGA of the Nexys4 DDR board. We have programmed the soft-core processor Microblaze on the Nexys4-DDR board to trigger the respective key generation, key encapsulation/decapsulation, signature generation and verification operations whenever required. As described in Figure 5.3.9, Dilithium and Kyber key generation, Dilithium Signature generation and Kyber key decapsulation occurs on the server side, whereas the Dilithium Signature verification and Key Encapsulation happen on the client side. In this work, we have shown that we have executed all the operations on a single device which will not be the case in a real-world implementation.

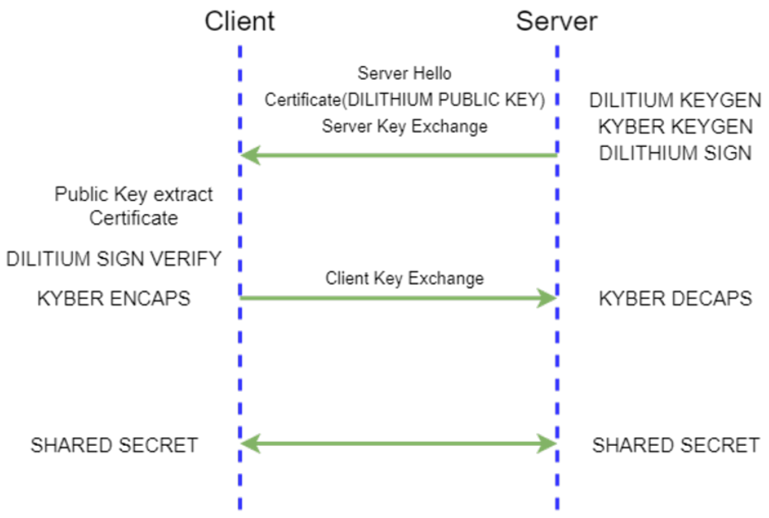

Figure 5.3.9: Our version of TLS 1.3 implemented in the proposed design

5.4 Results and Comparisons

In this section, we will analyze the performance of the proposed architecture with respect to state-of-the-art designs. The architecture consumes 22125 LUTs, 12531 FFs, 4 DSPs and 25 BRAMs achieving a Maximum frequency of 161 MHz. The resource utilization and latency of implemented Kyber and Dilithium are showcased in Table 5.3 and Table 5.2 respectively. All the designs in Table 5.3 and Table 5.2 are based on Artix-7 platform. In Figure 5.4.1 we have presented the comparison of area-time products with respect to the state-of-the-art design of Dilithium and state of the art design of Kyber by adding up their LUT consumption and multiplying with their achieved latency of implementation. We have significantly outperformed every known design of Kyber and Dilithium combined with respect to our achieved area-time product ($LUT \times latency \times 10^{-6}$). In Table 5.1, we have showcased the area utilization b the different

Algorithm	Components	LUTs	DSPs	BRAMs
Dilithium	Decompose	504	-	-
	MakeHint	80	-	-
	UseHint	708	-	-
	Powe2Round	75	-	-
	Pack	658	-	-
	Unpack	325	-	-
	Encode	354	-	-
	Decode	160	-	-
	SampleInBall	485	-	-
	Verify	16	-	-
	Rejection Core	718	-	-
Kyber	Compress/Decompress	258	-	-
	Encode	581	-	-
	Decode	268	-	-
	COPY	30	-	-
	CMOV	35	-	-
	Rejection Core	185	-	-
	Verification Core	98	-	-
Common	KECCAK	10879	-	-
	Data Memory	-	-	19
	NTT Multiplier	2899	4	1
	Controller	2618	-	5
Total		22125	4	25

Table 5.1: Total area utilization of the different components in the unified architecture

Works	Algorithm	LUTs	FF	DSP	BRAM	Max Freq (MHz)	Keygen		Sign		Verify		AT Product
							cycles	μs	cycles	μs	cycles	μs	
[28]	Dilithium II	27433	10681	45	15	163	18761	115	29057	178	19687	121	8.2
	Dilithium III	30900	11372	45	21	145	33102	228	45068	310	32050	221	16.4
	Dilithium V	44653	13814	45	31	140	50982	363	70376	503	52712	377	39.2
[10]	Dilithium V	53187	28318	16	29	116	14037	121	24358	210	14642	126	17.87
Our Design	Dilithium II	22125	12531	4	25	161	16112	100	32168	200	18178	113	6.9
	Dilithium III	22125	12531	4	25	161	32102	199.4	56294	350	29084	181	11.7
	Dilithium V	22125	12531	4	25	161	49102	305	80420	500	43627	270	17.03

Table 5.2: Comparison of resource utilization and timing details of Dilithium with existing architecture

components of Kyber and Dilithium. As it can be observed, KECCAK and the NTT multiplier core consume the maximum number of LUTs.

The authors of [4] have implemented the same combination of Kyber and Dilithium in their design. Their results are better owing to the usage of Zynq Ultrascale plus

Works	Algorithm	LUTs	FF	DSP	BRAM	Max Freq (MHz)	Keygen		Encaps		Decaps		AT Product
							cycles	μs	cycles	μs	cycles	μs	
[9]	Kyber-512	15K	3K	11	14	25	75K	3000	132K	5280	142K	5680	164.4
	Kyber-768	15K	3K	11	14	25	112K	4480	178K	7120	191K	7640	221.4
	Kyber-1024	15K	3K	11	14	25	149K	5960	223K	8920	241K	9640	278.4
[52]	Kyber-512	7K	5K	2	3	161	4K	24.8	5K	31.05	7K	43.47	0.5
	Kyber-768	7K	5K	2	3	161	6K	37.2	8K	49.68	10K	62.11	0.77
	Kyber-1024	7K	5K	2	3	161	9K	55.9	11K	68.32	14K	89.95	1.07
[16]	Kyber-512	12K	10K	8	15	210	-	-	3K	14.28	4K	19.04	0.42
	Kyber-768	12K	10K	8	15	210	-	-	4K	19.04	5K	28.5	0.55
	Kyber-1024	12K	10K	8	15	210	-	-	5K	23	7K	33	0.75
[12]	Kyber-512	11K	10K	8	13	200	2K	10	2K	10	4K	20	0.34
	Kyber-768	12K	10K	12	14	200	3K	15	3K	15	5K	25	0.48
	Kyber-1024	13K	12K	16	16	185	3K	15	4K	20	6K	30	0.728
Our Design	Kyber-512	22K	12K	4	25	161	2.5K	15.85	3.4K	21.27	4.2K	26.32	1.05
	Kyber-768	22K	12K	4	25	161	3.5K	21.9	4.3K	27.11	5.1K	31.85	1.304
	Kyber-1024	22K	12K	4	25	161	4.4K	27.5	5.3K	33.11	6.4K	39.89	1.61

Table 5.3: Comparison of resource utilization and timing details of Kyber with existing architecture

board. They have used a separate architecture for KECCAK and NTT multiplier. But they have not shown any application perspective in their design. On the other hand, the authors of [28] have implemented high parallelisms in the NTT architecture at the cost of high DSP usage. But our optimized scheduling of the Dilithium architecture has helped us achieve a better latency with respect to the results achieved by [28]. On the other hand, the authors of [10] were able to achieve a reduced latency by implementing multiple Keccak cores which in turn have increased their resource consumption. We have considered their implementation result which was based on the Artix-7 platform to showcase a fair comparison with our design. However, we outperform their design in terms of area-time product. Further, if we look into the existing designs of Kyber, [9] have implemented several lattice-based schemes on a RISC-V-based architecture with the target to achieve a low-power design. But on contrary to their hardware/software co-design approach, we have outperformed them based on their achieved latency. On the other hand, [52] has showcased a compact design of Kyber achieving a low area consumption with reasonable latency. But as we have balanced our design rationale between a high-speed design methodology and reasonable resource consumption, we were able to outperform this design with respect to our obtained latency to implement Kyber. Designers of [12] have by far implemented the most high-speed design of Kyber based on their efficient modular reduction technique and optimized NTT multiplier. Adding up the resource consumption of Dilithium and Kyber implementation of [28] and [12] respectively (represented as Artifact I in Figure 5.4.1), multiplied with the

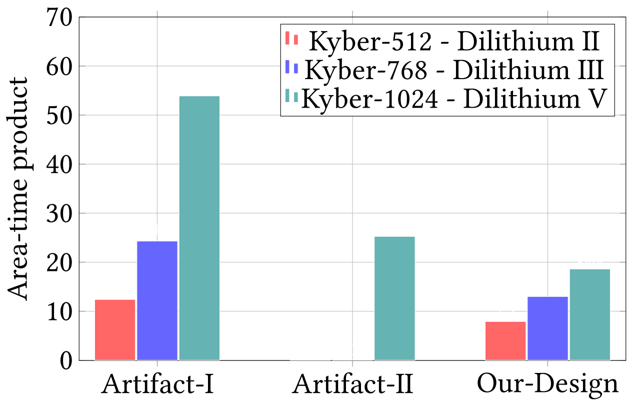

Figure 5.4.1: Comparison of area-time product with existing designs

latency of execution as we can see in Figure 5.4.1 that we have outperformed in term of *AT* product of our design. Even if we add up the resource consumption of [10] and [12] and multiply with the latency of execution (represented as Artifact II in Figure 5.4.1), we have a better result in terms of *AT* product as represented in Figure 5.4.1.

5.5 Conclusion

In this chapter, we identify for the first time the opportunity to effectively share the datapaths of two promising post-quantum algorithms, Kyber and Dilithium which can achieve key encapsulation and signatures. This leads to a comprehensive block which suits nicely hardware security modules for automotive applications. Furthermore, our design offers an agile choice to the designer by adopting a micro-code-based architecture which can seamlessly program the block to operate at different security levels. We demonstrate through several experiments on the prototype FPGA cum Microblaze solution that the design achieves best area-time product with respect to the state-of-the-art.

Chapter 6

Conclusion and Future Work

Security and protection of sensitive information is very crucial in modern world. In this context, cryptography and efficient implementation of complex cryptographic algorithms are of great importance. For secure communication between two parties, efficient implementation of key distribution protocols is a necessary requirement. Public key cryptographic protocols are the necessary ingredients for this key distribution and key management procedure. ECC is one of the most preferred public key algorithms as more security per key bit can be achieved as compared to RSA. Moreover, due to the recent advancement of quantum computers, quantum-safe schemes such as Kyber and Dilithium have also emerged recently. In this thesis we have proposed a lightweight hardware accelerator for ECC based on Koblitz curve. We have also proposed a resource-efficient hardware accelerator for the post-quantum key exchange mechanism Kyber and later we have amalgamated this design with another post-quantum digital signature scheme Dilithium. We have also proposed a post-quantum version of TLS 1.3.

The main highlights of this thesis are as follows:

1. We have developed an ADDN-instruction-based ECC processor for Koblitz curves. We have made several design choices for judicious usage of FPGA hard IPs that have been proposed for reducing the area overhead while ensuring a competitive area-time product. The design is reconfigurable to support two different methodologies for scalar multiplication. We furnish experimental results to show that the design consumes minimal resources on cost-effective FPGAs, leaving ample room for hosting other important SOC circuitry.

2. We have implemented an optimized design for Kyber. We have optimized the datapath elements of Kyber in such a way that other compatible post-quantum algorithms can be appended to this design. This leads to a comprehensive block which suits nicely hardware security modules for automotive applications. Furthermore, our design offers an agile choice to the designer by adopting a micro-code-based architecture which can seamlessly program the block to operate at different security levels. We demonstrate through several experiments on the prototype FPGA cum Microblaze solution.

3. We have identified for the first time the opportunity to effectively share the datapaths of two promising post-quantum algorithms, Kyber and Dilithium which can achieve key encapsulation and signatures. This leads to a comprehensive block which suits nicely hardware security modules for automotive applications. Furthermore, our design offers an agile choice to the designer by adopting a micro-code-based architecture which can seamlessly program the block to operate at different security levels. We demonstrate through several experiments on the prototype FPGA cum Microblaze solution that the design achieves best area-time product with respect to the state-of-the-art.

6.1 Future Work

The research presented in this thesis can be extended in various directions. Some of them are listed below:

- In this thesis we have presented a lightweight hardware accelerator for an elliptic curve scalar multiplier, which can be easily used as a component for a hardware root of trust and integrated to achieve any secure key exchange protocol on a SOC based platform which houses an ARM-based processor and an FPGA.
- This thesis provides an optimized implementation for post-quantum TLS 1.3 which can be further extended for real-time automotive HSM-based applications. Our design needs to be integrated with the required network layer to implement a full-fledged version of the post-quantum TLS 1.3 protocol where we can further test the feasibility of the proposed design.

References

[1] Jithra Adikari, Vassil S. Dimitrov, and Kimmo Järvinen. “A Fast Hardware Architecture for Integer to τNAF Conversion for Koblitz Curves”. In: *IEEE Transactions on Computers* 61 (2012), pp. 732–737.

[2] R. Agarwal and C. Burrus. “Fast Convolution using fermat number transforms with applications to digital filtering”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 22.2 (1974), pp. 87–97. DOI: 10.1109/TASSP.1974.1162555.

[3] Aikata Aikata et al. *A Unified Cryptoprocessor for Lattice-based Signature and Key-exchange.* Cryptology ePrint Archive, Paper 2021/1461. 2021.

[4] Aikata Aikata et al. *KaLi: A Crystal for Post-Quantum Security.* Cryptology ePrint Archive, Paper 2022/1086. 2022.

[5] Erdem Alkim et al. “FrodoKEM Learning With Errors Key Encapsulation Algorithm Specifications And Supporting Documentation”. In: 2019.

[6] Erdem Alkim et al. “Post-Quantum Key Exchange: A New Hope”. In: *Proceedings of the 25th USENIX Conference on Security Symposium.* SEC’16. Austin, TX, USA: USENIX Association, 2016, pp. 327–343. ISBN: 9781931971324.

[7] Roberto Avanzi and et. al. *CRYSTALS-Kyber.* Proposal to NIST PQC Standardization, Round3. 2021.

[8] Reza Azarderakhsh and Arash Reyhani-Masoleh. “High-Performance Implementation of Point Multiplication on Koblitz Curves”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 60.1 (2013), pp. 41–45. DOI: 10.1109/TCSII.2012.2234916.

[9] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. *Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols (Extended Version).* Cryptology ePrint Archive, Paper 2019/1140. 2019. DOI: `10.13154/tches.v2019.i4.17-61`.

[10] Luke Beckwith, Duc Tri Nguyen, and Kris Gaj. *High-Performance Hardware Implementation of CRYSTALS-Dilithium.* Cryptology ePrint Archive, Paper 2021/1451. 2021.

[11] Guido Bertoni et al. *Keccak.* Cryptology ePrint Archive, Paper 2015/389. 2015. DOI: `10.1007/978-3-642-38348-9_19`.

[12] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. *High-Speed NTT-based Polynomial Multiplication Accelerator for CRYSTALS-Kyber Post-Quantum Cryptography.* Cryptology ePrint Archive, Paper 2021/563. 2021.

[13] Joppe Bos et al. "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P).* 2018, pp. 353–367. DOI: `10.1109/EuroSP.2018.00032`.

[14] Billy Bob Brumley and Kimmo U. Jarvinen. "Conversion Algorithms and Implementations for Koblitz Curve Cryptography". In: *IEEE Transactions on Computers* 59.1 (2010), pp. 81–92. DOI: `10.1109/TC.2009.132`.

[15] James W. Cooley and John W. Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of Computation* 19 (1965), pp. 297–301.

[16] Viet Ba Dang et al. *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, Paper 2020/795. 2020.

[17] Dr.-Ing. and Robert Bosch Gmbh. "Securing Vehicular On-Board IT Systems : The EVITA Project". In: 2009.

[18] S. Bai *et al. CRYSTALS-Dilithium.* Proposal to NIST PQC Standardization, Round3. 2021.

[19] Haining Fan et al. "Overlap-free Karatsuba-Ofman polynomial multiplication algorithms". In: *IET Inf. Secur.* 4 (2010), pp. 8–14.

[20] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology.* CRYPTO '99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 537–554. ISBN: 3540663479.

[21] W. M. Gentleman and G. Sande. "Fast Fourier Transforms: For Fun and Profit". In: *Proceedings of the November 7-10, 1966, Fall Joint Computer Conference.* AFIPS '66 (Fall). San Francisco, California: Association for Computing Machinery, 1966, pp. 563–578. ISBN: 9781450378932. DOI: 10.1145/1464291.1464352. URL: https://doi.org/10.1145/1464291.1464352.

[22] Mohamed N. Hassan and Mohammed Benaissa. "Efficient Time-Area Scalable ECC Processor Using µ-Coding Technique". In: *WAIFI.* 2010.

[23] James Howe et al. "On Practical Discrete Gaussian Samplers for Lattice-Based Cryptography". In: *IEEE Transactions on Computers* 67.3 (2018), pp. 322–334. DOI: 10.1109/TC.2016.2642962.

[24] Toshiya Itoh and Shigeo Tsujii. "A Fast Algorithm for Computing Multiplicative Inverses in $\mathbb{GF}(2^m)$ Using Normal Bases". In: *Inf. Comput.* 78 (1988), pp. 171–177.

[25] Donald Ervin Knuth and Andrew Chi-Chih Yao. "The complexity of nonuniform random number generation". In: 1976.

[26] N. Koblitz. "CM-Curves with Good Cryptographic Properties". In: *CRYPTO '91.* 1992, pp. 279–287. ISBN: 978-3-540-46766-3.

[27] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of Computation* 48 (1987), pp. 203–209.

[28] Georg Land, Pascal Sasdrich, and Tim Güneysu. *A Hard Crystal - Implementing Dilithium on Reconfigurable Hardware.* Cryptology ePrint Archive, Paper 2021/355. 2021.

[29] Adeline Langlois and Damien Stehlé. "Worst-Case to Average-Case Reductions for Module Lattices". In: *Des. Codes Cryptography* 75.3 (June 2015), pp. 565–599. ISSN: 0925-1022. DOI: 10.1007/s10623-014-9938-4. URL: https://doi.org/10.1007/s10623-014-9938-4.

[30] K. C. Cinnati Loi, Sen An, and Seok-Bum Ko. "FPGA implementation of low latency scalable Elliptic Curve Cryptosystem processor in $\mathbb{GF}(2^m)$". In: *2014 IEEE International Symposium on Circuits and Systems (ISCAS).* 2014, pp. 822–825. DOI: 10.1109/ISCAS.2014.6865262.

[31] K. C. Cinnati Loi and Seok-Bum Ko. "Parallelization of scalable elliptic curve cryptosystem processors in $\mathbb{GF}(2^m)$". In: *Microprocess. Microsystems* 45 (2016), pp. 10–22.

[32] K.C. Cinnati Loi and Seok-Bum Ko. "High performance scalable elliptic curve cryptosystem processor for Koblitz curves". In: *Microprocessors and Microsystems* 37.4 (2013), pp. 394–406. ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2013.03.003. URL: https://www.sciencedirect.com/science/article/pii/S0141933113000483.

[33] Julio César López-Hernández and Ricardo Dahab. "Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation". In: *CHES.* 1999.

[34] J. Lutz and A. Hasan. "High performance FPGA based elliptic curve cryptographic co-processor". In: *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.* Vol. 2. 2004, 486–492 Vol.2. DOI: 10.1109/ITCC.2004.1286701.

[35] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *Advances in Cryptology – EUROCRYPT 2010.* Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23. ISBN: 978-3-642-13190-5.

[36] George Marsaglia and Wai Wan Tsang. "The Ziggurat Method for Generating Random Variables". In: *Journal of Statistical Software* 5.8 (2000), pp. 1–7. DOI: 10.18637/jss.v005.i08. URL: https://www.jstatsoft.org/index.php/jss/article/view/v005i08.

[37] Farhad Mavaddat and Behrooz Parhami. "URISC: The Ultimate Reduced Instruction Set Computer". In: *The International Journal of Electrical Engineering & Education* 25.4 (1988), pp. 327–334. DOI: 10.1177/002072098802500408. eprint: https://doi.org/10.1177/002072098802500408. URL: https://doi.org/10.1177/002072098802500408.

[38] Willi Meier and Othmar Staffelbach. "Efficient Multiplication on Certain Non-supersingular Elliptic Curves". In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. Berlin, Heidelberg: Springer-Verlag, 1992, pp. 333–344. ISBN: 3540573402.

[39] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48 (1987), pp. 243–264.

[40] Huguette NAPIAS. "A generalization of the LLL-algorithm over euclidean rings or orders". In: *Journal de Théorie des Nombres de Bordeaux* 8.2 (1996), pp. 387–396. ISSN: 12467405, 21188572. URL: `http://www.jstor.org/stable/43974220` (visited on 11/21/2022).

[41] J. Von Neumann. "Various techniques used in connection with random digits". In: *Applied Math Series, Notes by G. E. Forsythe, in National Bureau of Standards*. Vol. 12. 2018, pp. 36–38. DOI: `10.1109/EuroSP.2018.00032`.

[42] Chris Peikert. "An Efficient and Parallel Gaussian Sampler for Lattices". In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 80–97. ISBN: 978-3-642-14623-7.

[43] Chris Peikert. "Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem: Extended Abstract". In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: Association for Computing Machinery, 2009, pp. 333–342. ISBN: 9781605585062. DOI: `10.1145/1536414.1536461`. URL: `https://doi.org/10.1145/1536414.1536461`.

[44] Prasanna Ravi et al. "Authentication Protocol for Secure Automotive Systems: Benchmarking Post-Quantum Cryptography". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020, pp. 1–5. DOI: `10.1109/ISCAS45731.2020.9180847`.

[45] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: *J. ACM* 56.6 (Sept. 2009). ISSN: 0004-5411. DOI: `10.1145/1568318.1568324`. URL: `https://doi.org/10.1145/1568318.1568324`.

[46] Eric Rescorla. "The Transport Layer Security (TLS) Protocol Version 1.3". In: *RFC* 8446 (2018), pp. 1–160.

[47] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[48] Debapriya Basu Roy, Poulami Das, and Debdeep Mukhopadhyay. "ECC on Your Fingertips: A Single Instruction Approach for Lightweight ECC Design in GF(p)". In: *Selected Areas in Cryptography – SAC 2015*. Ed. by Orr Dunkelman and Liam Keliher. Cham: Springer International Publishing, 2016, pp. 161–177. ISBN: 978-3-319-31301-6.

[49] Debapriya Basu Roy and Debdeep Mukhopadhyay. "An Efficient High Speed Implementation of Flexible Characteristic-2 Multipliers on FPGAs". In: *VDAT*. 2012.

[50] Jerome A. Solinas. "Efficient Arithmetic on Koblitz Curves". In: *Towards a Quarter-Century of Public Key Cryptography: A Special Issue of DESIGNS, CODES AND CRYPTOGRAPHY An International Journal. Volume 19, No. 2/3 (2000)*. Ed. by Neal Koblitz. Boston, MA: Springer US, 2000, pp. 125–179. ISBN: 978-1-4757-6856-5. DOI: 10.1007/978-1-4757-6856-5_6. URL: https://doi.org/10.1007/978-1-4757-6856-5_6.

[51] Wen Wang and Marc Stöttinger. *Post-Quantum Secure Architectures for Automotive Hardware Secure Modules*. Cryptology ePrint Archive, Paper 2020/026. 2020.

[52] Yufei Xing and Shuguo Li. "A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (Feb. 2021), pp. 328–356. DOI: 10.46586/tches.v2021.i2.328-356. URL: https://tches.iacr.org/index.php/TCHES/article/view/8797.

[53] Daniel Zelle et al. "On Using TLS to Secure In-Vehicle Networks". In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ARES '17. Reggio Calabria, Italy: Association for Computing Machinery, 2017. ISBN:

9781450352574. DOI: 10.1145/3098954.3105824. URL: https://doi.org/10.1145/3098954.3105824.

Publications Related to Thesis

Conference:

- [C1] **Siddhartha Chowdhury**, Debapriya Basu Roy, Debdeep Mukhopadhyay, *"A Minimalistic Perspective on Koblitz Curve Scalar Multiplication for FPGA Platforms."*, VLSI-SoC, 2020.
- [C2] **Siddhartha Chowdhury**, Sayandeep Saha, Angshuman Karmakar, Debdeep Mukhopadhyay, Ingrid Verbauwhede *"PQTLS: A Unified Agile Co-Processor for Post-Quantum TLS."*, **[Submitted to TCHES 2023]**.

Other Publications

Conference:

- [C3] Akashdeep Saha, Sayandeep Saha, **Siddhartha Chowdhury**, Debdeep Mukhopadhyay, Bhargab B Bhattacharya, *"LoPher: SAT-Hardened Logic Embedding on Block Ciphers"*, 57th ACM/IEEE Design Automation Conference (DAC), 2020.