Optimized Kyber implementation on 16-bit MSP430 Microcontroller

DongHyun Shin, Youngbeom Kim, and Seog Chung Seo*
Department of Financial Information Security,
Kookmin University, South Korea
{dkldkl13, darania, scseo*}@kookmin.ac.kr

Abstract

Post-Quantum Cryptography (PQC) typically requires more memory and computational power compared to conventional public-key cryptography. In anticipation of future PQC migration, active optimization research is being conducted on various devices. However, no implementation attempts have yet been made in embedded environments with extreme memory and performance constraints. Notably, no efforts have been made to achieve quantum security in MSP430 environments, which are extensively used in Internet of Things (IoT) environment. Therefore, this paper presents, for the first time, a strategy for optimized implementation of Kyber in an MSP430 environment. We suggest an efficient NTT-based polynomial multiplication strategy through optimization of Montgomery and Barrett arithmetic using implicit shifts, and NTT merging techniques redesigned for the MSP430 environment. Furthermore, by leveraging the characteristics of MSP430's hardware multiplier and efficient register scheduling, we push the performance of Kyber to its limits. As a result, compared to the C code of PQCLEAN [1], we achieved performance improvements of 113.2%, 85.2%, and 131.8% in NTT, basemul, and NTT^{-1}, respectively. Moreover, In comparison to the Kyber implementation of PQCLEAN [1], our work results in performance improvements of 12.0% (and 15.7%, 15.1% respectively), 14.8% (and 18.3%, 17.2% respectively), and 24.3% (and 27.4%, 24.4% respectively) for the Keygen, Encaps, and Decaps operations of Kyber at security levels 1, 3, and 5 respectively.

Keywords: Crystals-Kyber, 16-bit MSP430 Microcontroller, optimize implementation, lattice-based cryptography, NTT

1 Introduction

With the advent of Shor's algorithm [2] and the development of quantum computers, conventional public-key systems based on discrete logarithms or factorization are under threat. Therefore, efforts are being made worldwide to build public key systems that are secure in a quantum computing environment. Since the NIST PQC (Post-Quantum Cryptography) competition was held in 2016, various optimization studies have been conducted on proposed algorithms, contributing to the performance evaluation of the competition. The first criteria of NIST's performance evaluation are speed and memory usage in Software (SW), and area and power consumption in Hardware (HW). The devices designated for SW performance evaluation by NIST are ARM-Cortex-M4, CPU, AVX2, and Artix-7 for hardware. The second evaluation item is countermeasures against side-channel attacks. NIST required all submitters to implement constant-time, and evaluations were conducted on the submitted algorithms from

The 7th International Conference on Mobile Internet Security (MobiSec'23), December 19-21, 2023, Okinawa, Japan, Article No. 10
*Corresponding author: Department of Financial Information Security, Kookmin University, Seoul, 02707, Republic of Korea

Algorithm 1 Kyber.CPAPKE Keygen [7, 8]

Ensure: $pk = (\hat{\mathbf{b}}, \rho)$, $sk = \hat{\mathbf{s}}$
0: $seed \leftarrow \{0, \cdots, 255\}^{32}$
0: $\rho, \sigma \leftarrow$SHAKE256$(64, seed)$
0: $\hat{\mathbf{A}} \leftarrow$ GenMatrixA(ρ)
0: $\mathbf{s} \leftarrow$ SampleVec$(\sigma, 0)$
0: $\mathbf{e} \leftarrow$ SampleVec$(\sigma, 1)$
0: $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{A}} \circ$ NTT$(\mathbf{s})$ + NTT$(\mathbf{e})$
0: **return** $pk = (\hat{\mathbf{b}}, \rho)$, $sk = \hat{s}$ =0

a side-channel analysis perspective. Lastly, the applicability of PQC is considered. The ease of migration to various commercial protocols (such as TLS/SSL, DNSSEC, IPSEC, etc.) [3, 4] is an important issue for future cryptographic systems. In the early stages of the NIST PQC competition, there was high interest in the speed of PQC. However, as algorithms with better performance than the existing elliptic curve-based cryptographic systems have emerged through various optimization studies, current research is primarily conducted on optimization of PQC's memory usage. The main platform for optimization research is the ARM-Cortex M4, which is a performance evaluation target device of NIST [5]. There exists research that benchmarks the algorithms submitted to the PQC competition in the ARM-Cortex M4 environment, and the latest research is continuously being updated in the respective library.

Currently, NIST has adopted four types of PQC algorithms as standardization targets. As Kyber [6] is the only KEM algorithm based on a lattice, additional KEM algorithms are being selected during Round 4. To achieve quantum security, a migration process from existing public-key algorithms to standardization-target PQC is necessary. An important point during migration is that all devices and services must undergo migration. If any side continues to use the existing public key system, PQC communication is not possible between them, and they could be vulnerable to downgrade attacks in the future. Therefore, all devices using public key encryption need to migrate to PQC. Although various PQCs have been proven to be safe in quantum environments, they have the disadvantage of longer key lengths or more computational load compared to conventional public keys. Therefore, PQC optimization research is necessary in the Micro Control Unit (MCU), which has resource and performance constraints compared to CPUs. Currently, despite being the only NIST PQC PKE/KEM standardization target algorithm, there are no implementation results for Kyber in the MSP430 environment.

In this paper, we present the first implementation of NTT and Keccak, which account for the largest proportion of Kyber's operations in the MSP430 environment, propose optimized implementation methods, and compare performance. As a result, compared to the C code of PQCLEAN [1], we achieved performance improvements of 113.2%, 85.2%, and 131.8% in NTT, basemul, and NTT^{-1}, respectively. Moreover, In comparison to the Kyber implementation of PQCLEAN [1], our work results in performance improvements of 22.3% (and 24.1%, 22.3% respectively), 25.2% (and 27.4% 25.2% respectively), and 31.9% (and 34.1%, 31.9% respectively) for the Keygen, Encaps, and Decaps operations of Kyber at security levels 1, 3, and 5 respectively.

Algorithm 2 Kyber.CPAPKE Enc [7, 8]

Require: $pk = (\hat{\mathbf{b}}, \rho)$, message μ in R_q, seed $coin \in \{0, \cdots, 255\}^{32}$
Ensure: Ciphertext $(\mathbf{u}', h)$
0: $\hat{\mathbf{A}} \leftarrow \text{GenMatrixA}(\rho)$
0: $\mathbf{s}' \leftarrow \text{SampleVec}(coin, 0)$
0: $\mathbf{e}' \leftarrow \text{SampleVec}(coin, 1)$
0: $\mathbf{e}'' \leftarrow \text{SampleVec}(coin, 2)$
0: $\hat{\mathbf{t}} \leftarrow \text{NTT}(\mathbf{s}')$
0: $\mathbf{u} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \hat{\mathbf{t}}) + \mathbf{e}'$
0: $\hat{\mathbf{v}} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{v}}^T \circ \hat{\mathbf{t}}) + \mathbf{e}'' + \mu$
0: **return** $(\mathbf{u}' = \text{Compress}(\mathbf{u}), h = \text{Compress}(\mathbf{v}'))$ =0

Algorithm 3 Kyber.CPAPKE Dec [7, 8]

Require: Ciphertext $c = (\mathbf{u}', h)$, secret key $sk = \hat{\mathbf{s}}$
Ensure: Message $\mu \in R_q$
0: $\mathbf{u} \leftarrow \text{Decompress}(\mathbf{u}')$
0: $\mathbf{v}' \leftarrow \text{Decompress}(h)$
0: **return** $\mu = \mathbf{v}' - \text{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u}))$ =0

2 Background of Kyber

2.1 Crystals-Kyber

Kyber is a Module-LWE-based Post-Quantum Cryptographic (PQC) algorithm for Public Key Encryption (PKE) and Key Encapsulation Mechanism (KEM). Kyber constructs a KEM using the Fujisaki-Okamoto transformation of the PKE scheme that achieves IND-CPA security and attains IND-CCA2 security. Kyber has a total of 3 parameters, denoted as 1, 3, and 5 according to NIST security levels [Table 1]. Here, n represents the degree of polynomials, k represents the degree in the public square matrix, and q is the coefficient range of the polynomials. $\eta 1$ and $\eta 2$ are used as ranges for introducing random error values during encryption. (d_u, d_v) is a value utilized in the Compress and Decompress processes. δ represents the decryption failure probability for the parameter set. [Algorithm 1, 2, 3] provide pseudocode for Kyber's PKE scheme.

Table 1: Kyber Parameter sets

parameter	level	n	k	q	η_1	η_2	(d_u,d_v)	δ
Kyber512	1	256	2	3329	3	2	(10, 4)	2^-139
Kyber768	3	256	3	3329	2	2	(10, 4)	2^-164
Kyber1024	5	256	4	3329	2	2	(11, 5)	2^-174

2.2 NTT-based Polynomial Multiplication

Since a naive polynomial multiplication requires $\mathcal{O}(n^2)$ complexity where n is the degree of a polynomial, Kyber makes use of the number-theoretic transform (NTT)-based polynomial multiplication providing $\mathcal{O}(n \log n)$ complexity. For Kyber's prime $q = 3329$ and $n = 256$, the

base field $\mathbb{Z}_q$ contains not primitive 512-th roots but primitive 256-th roots of unity. Thus, Kyber uses incomplete negacyclic NTT method. In other words, a polynomial over R_q is expressed by a vector of 128 polynomials of degree 1 in NTT domain. NTT-based polynomial multiplication consists of three steps: NTT conversion, point-wise multiplication, and inverse NTT conversion. In other words, the multiplication of two polynomials $f = \sum_{i=0}^{255} f_i X^i$ and $g = \sum_{i=0}^{255} g_i X^i$ in R_q is conducted as three steps.

- *Converting each polynomial into NTT domain:*
  $\hat{f} \leftarrow \mathsf{NTT}(f)$ and $\hat{g} \leftarrow \mathsf{NTT}(g)$, where $\mathsf{NTT}(f) = (\hat{f}_0 + \hat{f}_1 X, \ldots, \hat{f_{254}} + \hat{f_{255}} X)$,
  with $\hat{f_{2i}} = \sum_{j=0}^{127} f_{2j} \zeta^{(2i+1)j}$ and $\hat{f_{2i+1}} = \sum_{j=0}^{127} f_{2j+1} \zeta^{(2i+1)j}$.
  ζ is the 256-th root of unity, and $\zeta^{(2i+1)j}$ values are precomputed.
- *Conducting point-wise multiplication:*
  $\hat{h} = \hat{f} \circ \hat{g}$ where $\hat{h_{2i}} + \hat{h_{2i+1}} X = (\hat{f_{2i}} + \hat{f_{2i+1}} X)(\hat{g_{2i}} + \hat{g_{2i+1}} X) \mod (X^2 - \zeta^{2i+1})$,
  for $i = \{0, \ldots, 127\}$.
- *Applying inverse NTT conversion:*
  $f \cdot g = \mathsf{NTT}^{-1}(\hat{h})$

For NTT conversion and inverse NTT conversion, typically Cooley–Tukey (CT) method [9] and Gentleman-Sande (GS) method [10] are used, respectively.

2.3 16-bit MSP430 Microcontroller

MSP430 is a low-power MCU family developed by Texas Instruments. This MCU is designed for low-power application programs. The MSP430 series delivers high performance with minimal power consumption, facilitating efficient embedded system design. The MSP430 series finds applications in a wide range of fields, including compact household appliances, medical devices, industrial control systems, and sensor networks, offering efficient low-power operation and a diverse set of functionalities.

Table 2: Main Instruction Set of MSP430

Instruction	Operand	Definition
`adc(.b)`	dst	dst + carry ->dst
`add(.b)`	src, dst	src + dst → dst
`sub(.b)`	src, dst	dst + .not.src + 1 ->dst
`subc(.b)`	src, dst	dst + .not.src + carry ->dst
`mov(.b)`	src, dst	src ->dst
`rla(.b)`	dst	Rotate left arithmetically
`rlc(.b)`	dst	Rotate left through carry
`xor(.b)`	src, dst	src .xor. dst ->dst

2.3.1 Characteristics of MSP430

MSP430 is a 16-bit MCU, featuring a total of 12 general-purpose registers, designated R4 to R15. Each of these registers is 16 bits in size. A notable characteristic of the MSP430 is that most models incorporate a hardware multiplier. This hardware multiplier consists of a 16×16-bit multiplication unit, and in some cases, a 32×32-bit-bit unit is also available. In the context

of the Kyber algorithm, where polynomial coefficients lie in the range [0, q), a 16-bit data type is employed for implementation. As a result, we exclusively utilize the 16×16-bit hardware multiplier for our purposes.

The 16×16-bit hardware multiplier in the MSP430 is equipped with dedicated registers: MPYS, MACS, OP2, RESLO, and RESHI. These registers are exclusively reserved for the operations of the multiplier, making them independent from the general-purpose registers. While it is possible to use the instructions employed for general-purpose registers, it comes with a higher cost. Therefore, it is more efficient to utilize these registers solely for multiplication operations. By moving values into the MPYS and OP2 registers, the multiplier begins the operation and stores the result in the RESLO and RESHI registers. Additionally, using the MACS register instead of the MPYS register accumulates the newly computed value with the existing value stored in the RESLO and RESHI registers, resulting in an accumulation of the product. We have leveraged these architectural features and the hardware multiplier efficiently to optimize the Kyber algorithm. [Table 2] presents commonly used instructions in the MSP430. Appending .b to the instructions operates on the lower byte.

3 Proposed Kyber Optimization on 16-bit MSP430 Microcontroller

In this section, we present the several implementation strategies for optimizing the performance of Kyber. Our proposed implementations cover major operation NTT-based polynomial multiplication, which mainly take up the overall running-time of Kyber. Thus, they can contribute to significant speed-up.

3.1 Faster Implementation of Reduction Algorithms in Kyber Operations

3.1.1 Faster Implementation of Barrett reduction

Barrett reduction [11] involves selecting a Barrett constant in the form of a power of 2 and efficiently reducing within the Q range through shift operations. In the case of PQCLEAN's Barrett reduction, the Barrett constant is chosen as 2^{26}, resulting in a reduction to the range [-$\frac{q-1}{2}$, $\frac{q-1}{2}$]. Ultimately, during the packing process, when coefficients are negative, Q is added to align the range to (q, q).

When changing the Barrett constant to 2^{16}, the subtraction range becomes $(-q, q)$. Similar to the case of choosing 2^{26} as the Barrett constant, by adding Q only to negative coefficients during the packing process, the poly_reduce function can be used. As a result, we have implemented the Barrett reduction with the Barrett constant changed to 2^{16} to provide advantages in a 16-bit environment. By selecting the Barrett constant as 2^{16} in a 16-bit environment, shift operations can be omitted. Instead of shifting the multiplication result, utilizing the register corresponding to the upper 16 bits (RESHI) directly avoids the need for shift operations. [Algorithm 5] presents the Faster Barrett reduction code written in assembly language. When choosing the Barrett constant as 2^{16}, the precomputed value of v is calculated as ((1<<16) + q/2) / q. To multiply v and a using the hardware multiplier, values need to be moved into the MPYS and OP2 registers. The 32-bit result of multiplication is stored in the RESLO and RESHI registers. After multiplication, when multiplying the upper word of the multiplication result with q, the upper word value is directly used from the RESHI register without shift op-

erations. Therefore, to leverage the advantages of a 16-bit environment, we have implemented the Barrett reduction by changing the Barrett constant to 2^{16}.

Algorithm 4 C code for barrett reduction

Require: 16-bit `a` , with `a` $\in [-2^{15}, 2^{15})$,
Ensure: 16-bit `a` , with `a` $\in (-\frac{q}{2}, \frac{q}{2})$

```
int16_t;
int16_t v = ((1 « 26) + q/2)/q);
t = ((int 32_t)v * a + (1 « 25)) » 26;
t *= q;
return a - t; =0
```

Algorithm 5 Assembly code for Faster barrett reduction

Require: 16-bit `a` , with `a` $\in [-2^{15}, 2^{15})$,
Ensure: 16-bit `a` , with `a` $\in (-q, q)$

```
mov a, &MPYS
mov #20, &OP2 {t = v * a, t is a 16-bit data type}
mov &RESHI, &MPYS
mov q, &OP2 {t = t * q}
sub &RESLO, a {a = a - t}
return a =0
```

3.1.2 Faster Implementation of Montgomery reduction

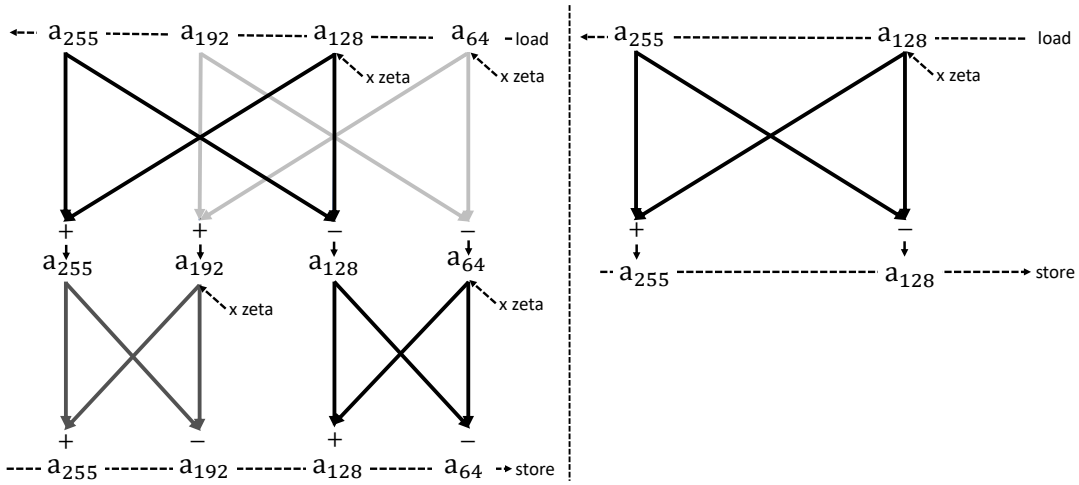
PQCLEAN's Montgomery reduction [12] employs a Montgomery constant selected as 2^{16} for subtraction within the range of $(-Q, Q)$. Similar to the previously mentioned Faster Barrett reduction, it omits the 16-bit shift operations. When the upper 16 bits or lower 16 bits of the 32-bit multiplication result are needed, the corresponding registers are used directly to reduce the number of instructions. [Algorithm 7] presents the Faster Montgomery reduction code written in assembly language.

Algorithm 6 C code for Montgomery reduction

Require: 32-bit `a` , with `a` $\in [-q \cdot 2^{15}, q \cdot 2^{15})$,
Ensure: 16-bit `t` , with `t` $\in (-q, q)$

```
int16_t;
t = (int16_t)a * QINV;
t = (a - (int32_t)t * q) » 16;
return t; =0
```

Figure 1: The picture on the left is 2-layer merging, and the picture on the right is a single-layer implementation.



Algorithm 7 Assembly code for Faster Montgomery reduction

Require: 32-bit $\mathtt{a} = \mathtt{a}_1||\mathtt{a}_0$, with $\mathtt{a} \in [-q \cdot 2^{15}, q \cdot 2^{15})$,
Ensure: 16-bit $\mathtt{a}_1$, with $\mathtt{a}_1 \in (-q, q)$

0: `mov` $\mathtt{a}_0$`, &MPYS`
0: `mov QINV, &OP2` {t = QINV ∗ a, t is a 16-bit data type}
0: `mov &RESLO, &MPYS`
0: `mov` q`, &OP2` {(int32_t)t = t ∗ q}
0: `sub &RESLO,` $\mathtt{a}_0$
0: `subc &RESHI,` $\mathtt{a}_1$ {t = a - (int32_t)t}
0: **return** $\mathbf{a}_1$ {return t(=$\mathtt{a}_1$)} =0

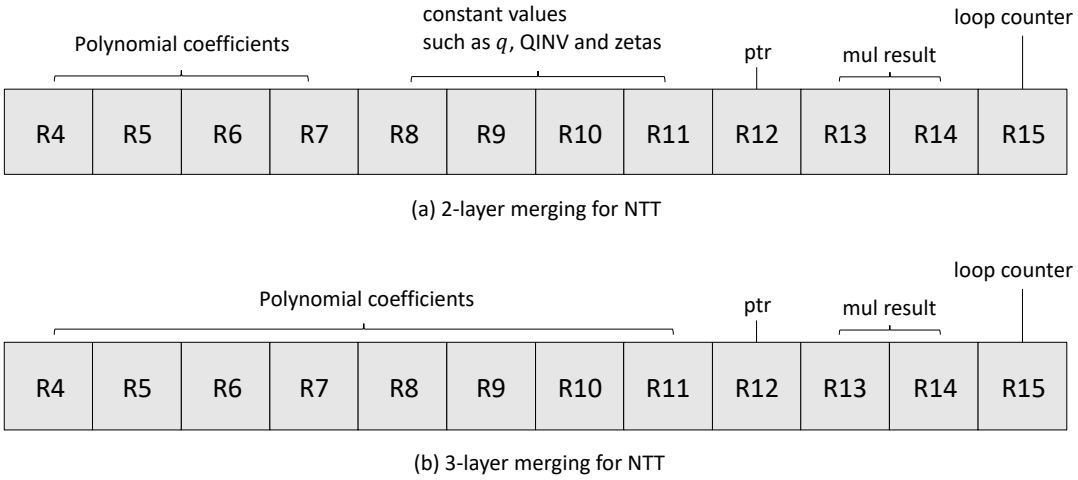
3.2 Optimized Merging Strategy in NTT

In Kyber, the NTT process is divided into a total of 7 layers of CT-butterfly operations, which further splits a single 255-degree polynomial into 128 linear polynomials. As Kyber employs $Q = 3329$, most implementations store polynomial coefficients in 16-bit data types. However, embedded environments like AVR and ARM have limited register sizes and quantities, making it impractical to store all coefficients in registers. Consequently, a common implementation approach involves loading two coefficients per register, performing CT-butterfly operations, and then storing the results. In other words, this approach calculates the NTT layer by layer, necessitating coefficient loading for all coefficients with each execution of a single layer.

In order to efficiently organize the NTT, the roles of the four primary registers should be fixed as follows: one register for array addressing, two registers for 32-bit multiplication results used for Montgomery reduction and CT-butterfly operations, and one register for loop control.

Similar to other environments, the MSP430 also provides only 12 available registers. Consequently, it's not possible to store all coefficients in registers during NTT implementation. Thus, the conventional approach involves loading coefficients for each layer. To minimize the instruc-

Figure 2: Register scheduling for NTT



tion count for load and store operations in NTT, we adapted and implemented the widely-used merging technique for the MSP430 environment. The merging technique optimizes load and store instructions by efficiently utilizing registers. [Figure 1] illustrates the overall structure of the m2-layer merged NTT and single-layer NTT. This technique was initially proposed by Abdulrahman et al. [5].

The aforementioned merging technique involves loading a coherent set of 2^n coefficients into registers, performing n-layer CT-butterfly calculations simultaneously, and then storing the results. In essence, it reduces the number of load and store instructions compared to computing a single layer of NTT separately. Furthermore, by applying 2-layer merging, it leaves 4 registers available. Consequently, constants such as KYBER_Q, QNIV, and zetas, which are repetitively used, can be stored in registers. Performing NTT calculations with these constants fixed in registers results in fewer clock cycles compared to using immediate instructions. However, with 3-layer merging, all 8 available coefficients are loaded, leaving no registers available. Therefore, as the NTT layers progress, since zeta values frequently change, employing 2-layer merging or calculating a single layer first is more efficient than 3-layer merging. Additionally, 4-layer merging requires 16 registers, making it inefficient for the MSP430 environment. The proposed register scheduling is illustrated in [Figure 2].

Kyber's NTT is composed of a total of 7 layers. Therefore, we can consider 2-2-2-1 merging, 2-2-3 merging, and 1-3-3 merging. Firstly, 2-2-3 merging introduces more immediate instructions compared to 2-2-2-1 merging, leading to additional clock cycles. However, due to fewer load and store operations for polynomial coefficients, the overall number of clock cycles is reduced. Both 2-2-3 merging and 1-3-3 merging involve the same number of load and store operations for polynomial coefficients. However, 1-3-3 merging incurs higher costs in terms of replacing constants compared to 2-2-3 merging.

Therefore, in the MSP430 environment, we have determined that the 2-2-3 merging technique yields optimal results. While NTT employs CT-butterfly operations, the Inverse NTT employs GS-butterfly operations. However, since the logic performed at each layer is similar to that of NTT, we have applied the 2-2-3 merging technique to NTT^{-1} as well.

3.3 Optimizing Point-wise Multiplication with Karatsuba Multiplication

In Kyber, two polynomials that have undergone NTT operations must perform point-wise multiplication. Since Kyber performs up to 7 layers, the point-wise multiplication process involves a total of 128 multiplications between first-order polynomials. PQCLEAN implements the multiplication of first-order polynomials in a schoolbook form. Excluding the process of multiplying ζ for mapping to ring elements, a schoolbook multiplication of first-order polynomials requires four multiplications. We aim to fully exploit the feature of MSP430 that supports hardware multiplication. However, to benefit from the hardware multiplier, values must be moved to the MPYS or OP2 registers of the multiplier. Furthermore, to use the result of the multiplication, it must be moved from the RESLO or RESHI registers to a general register. Thus, the multiplication operation incurs a greater load than the ADD or SUB commands. Therefore, as an optimization strategy, we choose to transform schoolbook-style multiplication into Karatsuba method [13], reducing the number of multiplications and increasing the number of additions and subtractions.

In point-wise multiplication, public matrix $(a_1X + a_0) \in \hat{A}$ and the secret vector $(s_1X + s_0) \in \mathbf{s}$ is calculated in the form as follows:

$$(a_1X + a_0) \cdot (s_1X + s_0)$$

When implemented with schoolbook multiplication, a total of 4 multiplications and 1 addition are required, as shown below.

$$a_1s_1X^2 + (a_1s_0 + a_0s_1)X + a_0s_0$$

We implemented the multiplication between first-order polynomials by transforming it into the well-known Karatsuba multiplication form. It can be calculated with three multiplications, two additions, and two subtractions, as shown below. Although the number of additions and subtractions has increased by three in total, the reduction of one 16-bit multiplication results in a net gain in the overall cycle.

$$a_1s_1X^2 + ((a_1 + a_0) \cdot (s_1 + s_0) - a_1s_1 - a_0s_0)X + a_0s_0$$

4 Evaluation

4.1 Benchmarking Setup

We choose the `IAR EW simulator` for performance measurement in the MSP430 environment. The target device is the `MSP430F67791`. We utilized the `IAR EW compiler`, with the optimization level set to `high(speed)`. Our performance measurement comparison is with the code from PQCLEAN. Regrettably, PQCLEAN [1] does not offer code for the MSP430 environment, so we compare it to code using pure C. The comparison code was also compiled with the `high(speed)` option.

4.2 Result of NTT-based Operation

[Table 3] showcases the performance improvement rates for the core operations of Kyber that we optimized in this paper. The comparison group is the PQCLEAN [1]. Firstly, through the optimization methods proposed in this paper, we achieve performance improvement of 46.0%

Table 3: Cycle counts for NTT, NTT^{-1}, and point-wise multiplication (basemul) of Kyber and montgomery reduction, and barrett reduction on 16-bit MSP430 environment (basemul measured by single point-wise multiplication). 1,000 cc is denoted by k.

Implementation	language	NTT	basemul	NTT^{-1}
Our work	asm	38k (+113.2%)	216 (+85.2%)	66k (+131.8%)
PQCLEAN [1]	C	81k (-)	400 (-)	153k (-)

Implementation	language	Montgomery	Barrett
Our work	asm	50 (+46.0%)	41 (+31.7%)
PQCLEAN [1]	C	73 (-)	54 (-)

and 31.7% for Montgomery and Barrett reductions respectively. This is attributed to the use of a hardware accelerator via efficient register scheduling and the modification of arithmetic operations to allow implicit shifts. Building upon this and the merging techniques, NTT and NTT^{-1} achieve a notable performance improvement rate of 113.2% and 131.8% respectively. Additionally, our basemul implementation, which utilizes the Karatsuba multiplication method, also achieve a performance enhancement of 85.2%.

Table 4: Cycle counts (cc) for Keygen, EnCaps, and Decaps of all security level of Kyber on 16-bit MSP430. 1,000 cc is denoted by k.

Implementation	variant	PQCLEAN [1]	Our work
Kyber512	KeyGen	3,288k	(+12.0%) 2,936k
	Encaps	4,419k	(+14.8%) 3,849k
	Decaps	4,316k	(+24.3%) 3,471k
Kyber768	KeyGen	5,546k	(+15.7%) 4,793k
	Encaps	7,289k	(+18.3%) 6,163k
	Decaps	7,138k	(+27.4%) 5,603k
Kyber1024	KeyGen	8,727k	(+15.1%) 7,585k
	Encaps	10,892k	(+17.2%) 9,297k
	Decaps	10,668k	(+24.4%) 8,575k

4.3 Result of Kyber.KEM

[Table 4] shows the performance of the entire process of Kyber KEM. In comparison to the Kyber implementation of PQCLEAN [1], our work results in performance improvements of 12.0% (and 15.7%, 15.1% respectively), 14.8% (and 18.3%, 17.2% respectively), and 24.3% (and 27.4%, 24.4% respectively) for the Keygen, Encaps, and Decaps operations of Kyber at security levels 1, 3, and 5 respectively.

5 Conclusion and Future Work

We present the first implementation of Kyber on a 16-bit MSP430 environment. Our research primarily focused on polynomial multiplication. Initially, we implemented efficient reduction by leveraging the architectural characteristics of the 16-bit environment. In addition, we redesigned the NTT Merging technique, which minimizes memory access, to fit the MSP430 architecture through efficient register scheduling. We accelerated the process by transforming the conventional Schoolbook-based Point-wise Multiplication to a Karatsuba-based approach and utilizing a hardware multiplier. In the future, we plan to conduct optimization research on Dilithium, a Digital Signature Algorithm (DSA) which is part of the NIST standardization target and belongs to the same family as Kyber, in the 16-bit MSP430 environment.

5.0.1 Acknowledge

This work was partly supported by the Institute of Information and communications Technology Planning and Evaluation (IITP) Grant by the Korean Government through Ministry of Science and ICT (MSIT) (A study on PQC optimization and security protocol migration to neutralize advanced quantum attacks in Beyond 5G-based next-generation IoT computing environments, 50%) under Grant 2022-00207416, and partly supported by the National Research Foundataion of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2022R1C1C1013368, 50%)

References

[1] Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. Improving software quality in cryptography standardization projects. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 19–30, 2022.

[2] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[3] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1461–1480, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Seungyeon Bae, Yousung Chang, Hyeongjin Park, Minseo Kim, and Youngjoo Shin. A performance evaluation of ipsec with post-quantum cryptography. In *International Conference on Information Security and Cryptology*, pages 249–266. Springer, 2022.

[5] Amin Abdulrahman, Vincent Hwang, Matthias J Kannwischer, and Amber Sprenkels. Faster kyber and dilithium on the cortex-m4. In *Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings*, pages 853–871. Springer, 2022.

[6] Pakize Sanal, Emrah Karagoz, Hwajeong Seo, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Kyber on arm64: Compact implementations of kyber on 64-bit arm cortex-a processors. In *International Conference on Security and Privacy in Communication Systems*, pages 424–440. Springer, 2021.

[7] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.

[8] Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray CC Cheung, Çetin Kaya Koç, and Donglong Chen. Improved plantard arithmetic for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):614–636, 2022.

[9] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[10] W Morven Gentleman and Gordon Sande. Fast fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578, 1966.

[11] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology—CRYPTO'86: Proceedings*, pages 311–323. Springer, 1986.

[12] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[13] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962.