# Evaluation of modular multiplication methods used in lattice-based cryptography on FPGA/ASIC

Teodora-Alexandra Alexandrescu

Introduction to Scientific Working 2024/25

Supervisor: Aikata Aikata

Institute of Applied Information Processing and Communications
Graz University of Technology

**Abstract**

Lattice-based cryptography is the backbone for an entire class of systems offering post-quantum security. Many of these cryptosystems require rigorous mathematical approaches for laying the foundation of security, especially in the case of post-quantum computing or fully-homomorphic encryption schemes. A well-known NP-hard problem that is fundamental for both previously mentioned algorithms is Ring Learning with Errors, which uses polynomial arithmetic. In several cases, if these polynomials are small and a naive computational method is used, the performance costs will also be minimal. However, the higher the polynomial degree, the higher the need for optimization becomes.

To solve this problem, the concept of modular multiplication has been applied, where each coefficient in the polynomial result needs to be reduced modulo a quotient $q$ in order to reduce the coefficients. Thus, modular multiplication is the most fundamental resource in cryptographic schemes. This arithmetic operation, if performed efficiently, can bring a significant impact over the performance of lattice-based cryptographic implementations.

In this work, we address the problem of efficiency in terms of area and power consumption on digital platforms such as FPGA/ASIC and analyze three well-known modular multipliers for evaluation: Montgomery Reduction, Barrett's Reduction and Shift & Add. Furthermore, we seek to distinguish between two novel usecases of lattice-based cryptography, namely Post-Quantum Cryptography and Fully-Homomorphic Encryption, based on how they utilize the NP-Hard Ring Learning with Errors problem.

**Keywords:** Lattice-based Cryptography · Post-Quantum Cryptography· Fully-Homomorphic Encryption· Modular Multipliers · FPGA · ASIC

# 1 Introduction

Lattice-based constructions have gained a lot of attention in recent times due to their complexity, a characteristic that correlates with enhanced security. As the need for secure and lightweight lattice-based algorithms increases, their complexity is also an aspect that requires intensive study, especially in the context of platforms like FPGA or ASIC, where area and power are limited resources. Cryptosystems based on lattices derive their hardness from the Learning with Errors problem [Reg10], which is considered to be as hard as an NP-hard problem [MP13].

Fully-Homomorphic Encryption & Post-Quantum Cryptography are at the forefront of cryptographic research. While FHE enables computation on encrypted data, PQC addresses threats posed by quantum-counterparts. Both FHE and PQC schemes such as Genotype Imputation [GCB+22] or NTTRU [LS19] are lattice-based due to how they use the (Ring -) Learning with Errors properties: the ring structure and polynomial representation, usually employed during encryption.

At the heart of these lattice-based constructions lies polynomial multiplication, which is the most computationally intensive operation in Ring Learning with Errors cryptosystems, with its time-complexity spanning $O(n^2)$. To optimize this, Number Theoretic Transform (NTT) is applied to reduce the complexity to $O(nlogn)$. One crucial operation during NTT is modular reduction, which ensures that the polynomial coefficients lie within manageable bounds while still keeping the initial security properties. However, the higher the security degree, the more convoluted these operations become and hence resources such as occupied area or power consumption need to be preserved.

For this reason, we aim to provide an evaluation of modular multiplication methods targeted on digital platforms, in order to be aware of possible bottlenecks or caveats during the implementation of such schemes. In this study, we will mainly discuss the Montgomery Reduction, Barrett Reduction and the Shift & Add algorithm in the context of lattice-based schemes on FPGA/ASIC.

Montgomery Reduction [Mon85] is well-known for its efficiency in modular arithmetic due to replacing classical long division with bitwise operations, and by introducing the Montgomery form, which is a precomputation step that facilitates a faster, non-traditional division operation. On the other hand, Barrett's modular multiplication algorithm [Bar86] is more robust to changes in modulus due to the introduction of a precomputation step which highly depends on the modulus and its bit width. The inclusion of this step enables the algorithm to eliminate division in favor of less expensive multiplication, making it efficient.

A comparison between the Montgomery and Barrett modular multipliers has been carried out in [KP06] on a Xilinx Virtex 2 FPGA platform using VHDL and a general modulus (i.e. modulus without a specific structure, as seen in other studies). The goal was to compare area and timing results when computing $C = A \cdot B \mod M$, where $A$, $B$, $C$, $M$ are all $n$-bit integers with lengths up to 32 bits. For the Montgomery multipliers, four evaluation methods have been employed, and namely high-radix, separated/interleaved structure, trivial digit selection and quotient digit pipelining. As for the Barrett multiplier, an improved separated method has been applied: $A \cdot B$ is precomputed by using the inverse

of the modulus, $M^{-1}$. Intermediate area results for high-radix separated/interleaved Montgomery reduction with radix (number base) $r = 2^k$ show that: for the interleaved method, the area required in FPGA slices is higher at smaller $k$-values (where k is the word length or bits per digit), but as $k$ increases, the results for both the interleaved and separated methods converge to the same point, as depicted in Figure 1.

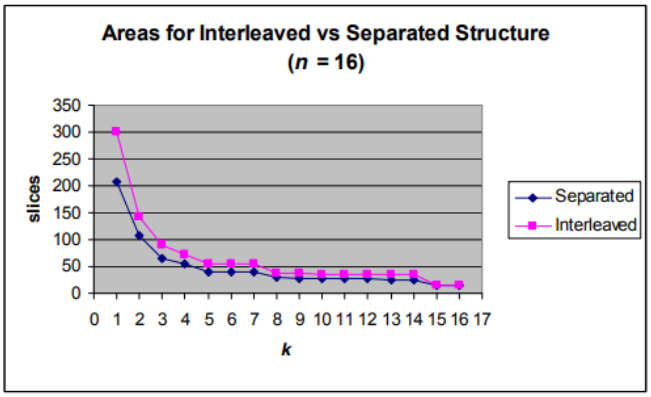


Figure 1: Area of Interleaved & Separated Structures at Different Radices [KP06]

The final results show that for the current setup with a general modulus, the choice between Montgomery and Barrett highly depends on the word length. However, for word lengths up to 16 bits, Barrett's reduction with the precomputed inverse of the modulus seems to be the appropriate choice.

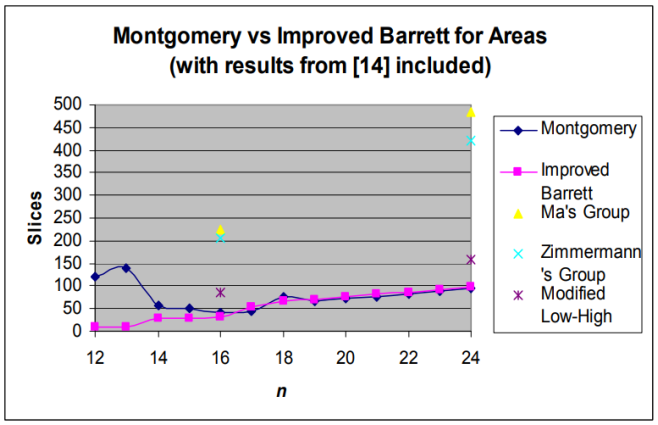


Figure 2: Comparison of areas of modular multipliers at different wordlengths [KP06]

Shift & Add is a standard multiplication method used in digital design. Its significance stems from the ability to reduce different algorithms (or particular parts of them) to simple operations like bitshifts and additions. In [SWA12] it is proven that modular reduction speed is increased by 80% for a modified Barrett reduction algorithm when

choosing Mersenne/ quasi-Mersenne numbers as moduli. By exploiting their properties, the overall long-integer multiplication speed is increased when using simple shift and addition operations in the reduction step.

A design space study of modular multipliers for Fully-Homomorphic Encryption (FHE) implemented on ASIC is presented in [SNG+23]. The study explores both bit-parallel and bit-serial designs at the system and block levels. The results indicate that bit-parallel multipliers are more area- and power-efficient than bit-serial designs at the block level. Finally, the study concludes that Montgomery modular multiplication with a constrained modulus [1] offers the most efficient design in terms of area and power consumption, at both the system and block levels. This ensures optimal polynomial reduction for FHE on ASIC platforms.

At a larger scale, modular reduction is an important operation within NTT, which is widely used in lattice-based schemes. This transformation is usually employed on butterfly cores, which are mathematical structures that allow for efficient modular polynomial operations by reducing the complexity of multiplying polynomials. By optimizing the modular reduction operation, we obtain more efficient polynomial multiplication. In [BAM21], several improvements for modular reduction in NTT for the post-quantum scheme Kyber [BDK+18] are presented: pipelining, the KRED/K2RED functions [LN16] and parallelization. The KRED optimization particularly contributes to the parallelization process due to the application of vector integer instructions in their implementation. Due to parallelism, pipelining is also improved. The results in [BAM21] state that the NTT core in this design achieves over a 44% performance improvement, measured by the $A \times T$ metric (area $\times$ time), on an Artix-7 FPGA.

# 2 Background

## 2.1 Notation

We use $\Lambda(a_1, a_2, ..., a_n)$ to represent a lattice and its basis vector. Gaussian distributions are denoted with $\rho$, whereas Gaussian error distributions are marked by $\chi$. Vector norms are represented by $|| \cdot ||$, and vector dot products by $\langle\ ,\ \rangle$. We use $\mathbb{Z}_q$ to represent the set of integers modulo a quotient $q \geq 2$. We represent the set of $d$-dimensional vectors in $\mathbb{Z}_q^d$ with $d \geq 1$. The ring of all polynomials over a finite field of moduli is represented by $\mathbb{Z}_q/\langle x^d + 1\rangle$, where $x^d + 1$ is the modulus polynomial (i.e. all elements of the ring are reduced by this modulus). Mersenne primes are denoted as $M_p$.

## 2.2 Brief description of Lattices

In group theory, a lattice is a discrete subgroup of the additive group, or a free abelian group of dimension $m$ which spans the vector space $\mathbb{R}^m$ and has a periodic structure. Each point in the lattice is separated by a minimum distance, a property that underpins the periodic nature of the lattice structure. Coordinate-wise addition or subtraction of two lattice-points produces a new lattice-point.

[1] it must follow specific properties, like M $\equiv 1 \mod 2N$, which restricts the number of possible moduli.

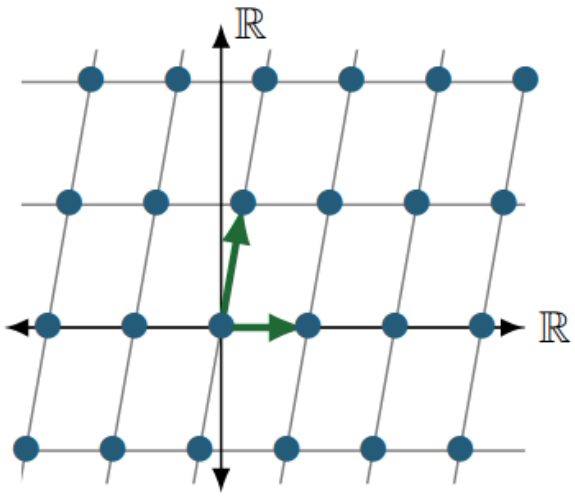


Figure 3: Lattice vectors in Euclidean plane [Eic]

Mathematically, given $n$ linearly independent vectors $a_1, a_2, ..., a_n \in \mathbb{R}^m$ as the basis, we define the lattice generated by them as follows [Reg]:

$$\Lambda(a_1, a_2, ..., a_n) = \left\{ \sum x_i a_i \mid x_i \in \mathbb{Z} \right\}$$

In simpler terms, a lattice can be described as a vector space limited to integer multiples of its basis vectors. A depiction of a bi-dimensional lattice can be observed in Figure 3.

The minimum distance of a lattice in a norm $|| \cdot ||$ is the length of the shortest nonzero lattice vector. If the norm is unspecified, the default value is the Euclidean norm.

The Gaussian distribution is an important metric in lattices, especially in lattice-based cryptography where computationally infeasible problems employ the distribution to generate random noise in order to enhance hardness. For any lattice vector $\mathbf{x}$ with a center $\mathbf{c}$ in the discreteness of the lattice, and a scaling factor $s$, we define the discrete Gaussian distribution as follows [MR07]:

$$\rho_{s,\mathbf{c}} = e^{-\pi \|(\mathbf{x}-\mathbf{c})/s\|^2}$$

In other words, when the discrete Gaussian distribution is applied to a lattice vector near a given center, we can generate a noisy version of the center vector, which adds controlled randomness to the cryptographic application. Worst-case lattice problems such as Learning with Errors use this Gaussian distribution to create noise for the purpose of hardness.

### 2.3 Learning with Errors

Modern-day experts base their encryption algorithms on hard mathematical problems such as the Learning with Errors problem to guarantee security within a certain standard.

After 2005, the Learning with Errors problem published by Oded Regev [Reg10] became a standard for cryptographic constructions due the strict security requirements it enforces. A study made by Micciancio & Peikert proves its hardness to resemble the norms of approximate lattice problems in the worst case [MP13]. The problem states that a secret vector $\mathbf{s}$ should be retrieved from a system of linear noisy equations, where the public vector $\mathbf{a}$ is drawn uniformly at random and the error term $e$ is drawn from a Gaussian distribution. While it may be possible to recover the secret $\mathbf{s}$ by performing Gaussian decomposition over the system of equations, the addition of the noise term makes it computationally infeasible to retrieve the secret for appropriate parameters.

**Definition.** Given a lattice $\Lambda \subseteq \mathbb{R}$ with a vector basis $a_1, a_2, ..., a_d$ drawn uniformly at random from $\mathbb{Z}_q^d$, a secret lattice vector $\mathbf{s} = \sum s_i a_i \in \mathbb{Z}_q^d$ and a noisy vector $\mathbf{e}$ drawn from a discrete Gaussian error distribution $\chi$ on $\mathbb{Z}_q$, the goal of the problem is to recover $\mathbf{s}$ from the approximated, noisy output $\mathbf{b} = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$. [Reg10]

$$
\begin{aligned}
a_1^{(1)} \cdot s_1 + \cdots + a_d^{(1)} \cdot s_d &\approx b^{(1)} \pmod q \\
a_1^{(2)} \cdot s_1 + \cdots + a_d^{(2)} \cdot s_d &\approx b^{(2)} \pmod q \\
&\vdots \\
a_1^{(n)} \cdot s_1 + \cdots + a_d^{(n)} \cdot s_d &\approx b^{(n)} \pmod q
\end{aligned}
$$

**Error distribution.** The error distribution $\chi$ is chosen to be a normal (Gaussian) distribution. This is then rounded to the nearest integer of standard deviation ($\alpha q$, where $\alpha > 0$ is typically $\frac{1}{poly(d)}$) and reduced modulo $q$, to ensure that the noise is small relative to the modulus.

**Modulus.** The modulus $q$ is typically chosen to be a polynomial of the vector dimension $d$. The choice of a polynomial modulus significantly increases the size of the input, which further affects the cryptographic application's efficiency in the favor of increasing the problem's hardness.

**Number of equations.** Researchers have drawn the conclusion that the number of equations does not result in any changes in hardness - on the contrary, the hardness is independent of it. This is due to a property discussed in [Reg10], where given a fixed polynomial number of equations, it is possible to generate an arbitrarily large number of additional equations, which can be further utilized.

The parameters provided to Learning with Errors, either number of vectors or vector dimension, usually determine the complexity of the encryption operation [MP13], or more specifically, the key size. Cryptographic applications often need at least $n$ vectors of $n$ dimensions, which lead to key sizes of $n^2$ [Reg10], which inherently means $O(n^2)$ space complexity. This aspect can represent a challenge as the need for a higher security

level increases, which makes the problem inefficient for practical cryptographic applications. The Ring Learning with Errors problem [LPR10] was introduced by Vladimir Lyubashevsky as an optimization of the initial LWE problem.

### 2.4 Ring Learning with Errors

Lyubashevsky's solution [LPR10] suggests a more lightweight and secure approach in implementing encryption algorithms by applying the ring structure over a finite field of moduli, onto the polynomial representation of public and private keys. With that being said, using polynomials within a ring structure instead of vectors and matrices should be more efficient storage-wise, leading to space complexities of $O(n)$.

Assuming that there exists some repeating "pattern" in the LWE samples, we deduce that: if we consider $d$ to be a power of 2, and if we assume that the $\mathbf{a}$ vectors arrive in blocks of $d$ samples $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_d \in \mathbb{Z}_q^d$, where $\mathbf{a}_1 = a_1, a_2, ..., a_d$ is still chosen uniformly at random , we can conclude that the remaining vectors $\mathbf{a}_i$ can be reproduced as $(a_i, ..., a_n, -a_1, .., -a_{i-1})$. This notation resembles a ring structure, where we write the rest of the $\mathbf{a}_i$ vectors in terms of $\mathbf{a}_1$, which reduces the complexity to $O(n)$ elements of $\mathbb{Z}_q$ as opposed to $O(n^2)$ for Learing with Errors.

Mathematically, this would imply trading the group of integers modulo $q$, $\mathbb{Z}_q^d$, for the ring of all polynomials over a finite field of moduli, $\mathbb{Z}_q/\langle x^d + 1\rangle$, which are all then reduced by $x^d + 1$ to ensure that the polynomial degree does not exceed $d - 1$ (this needs to go into the notation section). The additional requirements for this change to take place is having the dimension $d$ be a power of two, which further guarantees that $x^d + 1$ is irreducible over the rationals in order to provide a quotient ring structure.

With that being said, the R-LWE equivalent of the original LWE problem would be:

$$a^{(i)}(x) \cdot s(x) + e^{(i)}(x) \approx b^{(i)}(x) \pmod{x^d + 1, q}$$

**Hardness.** Solving the R-LWE problem implies a solution to the worst-case lattice problems, which are restricted to the family of *ideal lattices* [LPR10]. Ideal lattices are mathematical structures that satisfy a certain symmetry condition: if $(a_1, a_2, ..., a_d)$ is a lattice vector, then so is the ring representation of it $(x_2, ..., x_n, -x_1)$. Therefore, we can conclude that no changes over the hardness would occur in the context of Ring-Learning with Errors.

### 2.5 Polynomial Multiplication and Modular Reduction

Ring-Learning with Errors brings optimizations to Learning with Errors only space-wise through the ring structure and polynomial representation. However, multiplying polynomials still yields $O(n^2)$ time complexity in case of naive polynomial multiplication. This approach will do modular multiplication of all polynomial coefficients with each other. To reduce the complexity to quasi-linear time, Number Theoretic Transform (NTT) is applied and thus, the time complexity drops to $O(nlogn)$. Polynomial multiplication needs two types of multiplication:

1. Multiplication between polynomial coefficients and *twiddle factors*.

2. Element-wise multiplication between polynomial coefficients

In both cases, modular reduction techniques are applied. However, since modular reduction algorithms such as Montgomery and Barrett require precomputation steps, it is crucial to find reduction optimization methods in order to streamline the NTT-step. If this transformation is done efficiently, the performance of polynomial multiplication will increase significantly, and resources such as area, power and time are therefore preserved.

## 2.6 Montgomery Reduction

Montgomery Reduction [Mon85] is an efficient method that performs modular reduction after integer multiplication ($x \cdot y \ mod \ Q$). Its efficiency comes from replacing traditional long division with simple bitwise operations, as well as from precomputing the input values into the "Montgomery form".

**Montgomery Form.** We firstly define a modulus $Q > 1$ such that $\{x \in \mathbb{Z} \mid x \equiv r \ mod \ Q\}$. Then we select a radix (number base) $R = 2^n$, where $n = machine\ word\ size$, and $gcd(Q, R) = 1$, with $R > Q$. This choice of $R$ ensures that the computations are inexpensive by making the shifting operation possible. If $x$ is an integer, its Montgomery form is:

$$xR \mod Q.$$

This conversion should have no impact over addition or subtraction, because the result format in Montgomery form is 1:1 as its integer counterpart due to the distributive law. However, this is not the case for integer multiplication. Let $x,\ y$ be integers in the range $[0,\ Q-1]$. The Montgomery form of $x \cdot y \ mod \ Q$ is:

$$(xR \ mod \ Q)(yR \ mod \ Q) \ mod \ Q \ = \ (xyR)R \ mod \ Q.$$

We notice that after the multiplication, the result contains an extra $R$ that needs to be eliminated. A simple method to achieve this would be division by $R$, but the result is not divisible by $R$ due to the modulo operation. In this case, we can employ the modular inverse of $R$, which is $R^{-1}$ with the condition that $RR^{-1} - QQ' = 1$, and $Q'$ is the set of integers satisfying $0 < R^{-1} < Q$ and $0 < Q' < R$. By introducing the inverse, $R^{-1}$, the additional $R$ is reduced as follows:

$$(xR \ mod \ Q)(yR \ mod \ Q)R^{-1} \equiv (xR)(yR)R^{-1} \equiv (xyR)RR^{-1} \equiv (xy)R \ mod \ Q.$$

Thus, we have converted the multiplication between $x, y$ in Montgomery form.

**Algorithm 1** Montgomery Reduction (REDC)

**Require:** $T, Q, R = 2^n$, where $\gcd(R, Q) = 1$.
**Output:** $TR^{-1} \mod Q$

```
m ← (T mod R)Q' mod R
t ← (T + m · Q)/R
if t ≥ Q then
    t ← T − Q
end if
return t
```

**Reduction Algorithm.** Furthermore, we will discuss how to apply modular reduction to an integer in Montgomery form by outlining the reduction algorithm below.

The input $T$ is any integer, or, in our case, the product between $(x,\ y)$ in Montgomery form. $Q$ is the modulus, $R = 2^n$ is the Montgomery radix or base. The condition $gcd(Q, R) = 1$ ensures that Montgomery reduction can be applied, as it requires that $R$ has a modular inverse modulo $Q$. The integer $Q'$ can be computed similarly to $R'$, by using the Extended Euclidean Algorithm.

The intermediate value $m$ is computed to ensure that $T + mQ$ is divisible by $R$. Knowing that $Q \cdot Q' \equiv -1\ mod\ R$, we can infer that $Q \cdot Q' + 1$ is a multiple of $R$. By applying $Q'$ to m, we ensure that it will "undo" $Q\ mod\ R$ from the computation of $t$, such that $(T + m \cdot Q)\ mod\ R = 0$. To validate this, we extend $T + mQ$ as follows:

$$T + mQ \equiv T + ((T\ mod\ R)Q' \mod R)Q \equiv T + T \cdot \underbrace{QQ'}_{QQ' \equiv -1 \mod R} \equiv T - T \equiv 0 \mod R$$

**Hardware implementation.** Even though the REDC algorithm delivers the desired result in an optimal manner, implementing it in hardware could lead to a few issues, among which the most obvious one is the computation of $N'$. In this regard, we are also presenting an iterative method that uses addition, shifts and subtractions.

Let $R = 2^n$, $gcd(R, Q) = 1$, let $Q$ be odd. The multiplicands are numbers in binary representation, $x, y$, where $x = (x_{n-1}, x_{n-2}, ..., x_0)_2$ and $0 \leq y \leq Q$. We define an accumulator value whose parity bit determines whether or not the intermediate sum needs to be adjusted by adding $Q$ or not. In both cases, we divide the result by 2, or we shift right by 1. After $n$ iterations, the attained result is $xyR^{-1} \mod Q$. [Mon85], [RAF+15]

**Algorithm 2** Montgomery Reduction Hardware Implementation

**Require:** $X, Y, Q$ and $Q$ is odd, $X, Y < Q$
**Output:** $Acc \leftarrow X \times Y \times R^{-1} \mod Q$

```
1: Acc ← 0
2: for i = 0 to n − 1 do
3:    Acc ← Acc + x_i × Y
4:    if r_0 = 0 then
5:       Acc ← Acc/2
6:    else
7:       Acc ← (Acc + Q)/2
8:    end if
9: end for
10: return Acc
```

## 2.7 Barrett Reduction

Barrett's reduction [Bar86] started out as a technique provided by Paul Barrett in 1986 to obtain a high-speed implementation of the RSA encryption algorithm on a digital signal processing chip. This technique achieved a two and a half second encryption time on average for 512 bit exponent and modulus, on a first generation digital signal processor (DSP). A digital signal processor is a multiplier accumulator (MAC) and a fast microprocessor that facilitates fast multiplication due to the multiplier unit it includes in addition to the arithmetic logic unit (ALU).

The algorithm is known for its efficiency due to replacing division by cheaper multiplication and by precomputing terms while using bitshift operations. Required are a **fixed** modulus $M$ and a radix $R = 2^{2k}$, where $k$ is the length of the modulus. The radix and the modulus play a crucial role in the precomputation step. In addition, the input multiplicands, $x, y$ must be elements of $[0, M-1]$.

Taking into consideration all previous input constraints, we seek to retrieve the remainder of the operation $(x \cdot y \mod M)$. The schoolbook method presented below [Roy] first computes the quotient $q$, which is further used to compute the remainder $r$. However, this method is not suitable for hardware implementation due to the division operation which is very expensive.

$$\begin{aligned} t &= x \cdot y \\ q &= \lfloor t/m \rfloor \\ r &= t - q \cdot m. \end{aligned}$$

In this regard, Barrett's algorithm introduces the precomputation step where the floating point term $\frac{1}{m}$ is approximated by truncating the least $2k$ bits found after (.), and converted to an integer by a shift to the left by the radix $R$. The approximation step is shown in Figure 4.

**Modulus $m = 7069$ is 13 bits long. Hence $k = 13$.**

$$\begin{aligned}\frac{1}{m} &= 0.000141462724572075258169472 34_{10} \\ &= 0.000000000000010010100010101011000111\ldots_2 \\ &\approx 0.00000000000000100101000101012 \quad \text{Truncate least } 2k = 26 \text{ bits}\end{aligned}$$

$$\begin{aligned}\mu &= 0.00000000000000100101000101012 \ll 26 \quad \text{multiply by } 2^{26} \\ &= 10010100010101_2 \\ &= 9493_{10}\end{aligned}$$

Figure 4: Approximation step in Barrett's reduction [Roy]

The value of the precomputed term will not change as long as the value of the modulus remains constant [HGG07]. Direct division is as well avoided in the approximation step of the quotient (line 3 in the algorithm) through the division by the radix, which represents a shift to the right in hardware implementations.

**Algorithm 3** Barrett Reduction

**Require:** $X, Y, M,\ X, Y < M$
**Output:** $X \cdot Y \mod M$

1: $T \leftarrow X \cdot Y$
2: $\mu \leftarrow \lfloor (1/M) \cdot 2^{2k} \rfloor$
3: $q \leftarrow \lfloor (T \cdot \mu)/2^{2k} \rfloor$
4: $r \leftarrow T - q \cdot M$
5: **if** $r \geq M$ **then**
6: $\quad r \leftarrow r - M$
7: **end if**
8: **return** $r$

**Mathematically,** the formula for finding the remainder of $(x \cdot y \mod M)$ is given by:

$$r = T - \left\lfloor \frac{T \cdot \lfloor \frac{2^{2k}}{m} \rfloor}{2^{2k}} \right\rfloor \cdot M$$

Which is further equal to:

$$r = T - \left\lfloor \frac{T \cdot \mu}{R} \right\rfloor \cdot M$$

Where $T = x \cdot y$ and R is the radix.

### 2.8 Shift & Add

Shift & Add is a classic multiplication method which is mainly employed in in the context of hardware implementation. This method is at the heart of many algorithms, ranging from cryptographic applications to graphics and image processing due to the lightweight approach it has over the multiplication. The main idea stands in leveraging bitshift operations which are equivalent to either multiplication or division. Therefore, given two integers $x, y$ in binary representation, the algorithm is stated as follows:

**Algorithm 4** Shift & Add multiplication

**Require:** $X, Y$ in binary representation
**Output:** $X \cdot Y$
1: $Prod \leftarrow 0$ (because it is computed as a sum)
2: **for** $i = bitlength(X) - 1$ **to** 0 **do**
3: **if** $X[i] = 1$ **then**
4: $Prod = Prod + X[i] << i$
5: **end if**
6: **end for**
7: **return** $Prod$

We start from the least significant bit of either of the numbers, in our case $x$. If the current bit is 1, we shift the value to the left by $i$ and we add it to the $Prod$ variable. Otherwise, if the bit value is 0, nothing is done. At the end, the value of the product is returned.

Nonetheless, this algorithm is not only well-suited for cutting down the effort of multiplication. In [SWA12], the reduction step speed is increased significantly when using shifts and additions in combination with Mersenne/Pseudo-Mersenne primes for moduli ($M_p = 2^p - 1$). This improvement stems from the fact that Mersenne primes are numbers that are one less than a power of two, and powers of two are particularly efficient for bitshift operations. Thus, this special kind of modulus is a good candidate for fast modular multiplication implementation on FPGA or ASIC.

## 3 Conclusion

In this paper, we explore different implementations and designs of the Montgomery and Barrett modular multipliers, on FPGA and ASIC, for lattice-based cryptosystems. From [KP06] we learn that separated and interleaved designs of the Montgomery algorithm yield the same results on an FPGA as the word size increases. Additionally, for word lengths up to 16 bits, Barrett's reduction with a precomputed inverse of the modulus performs better. In [SWA12], the speed of long-integer modular multiplication is improved by replacing the reduction step in a modified Barrett algorithm with shifts and additions, leveraging the properties of Mersenne and Pseudo-Mersenne primes.

For FHE ASIC implementations, bit parallel designs are more efficient than bit-serial implementations in terms of area- and power consumption. A Montgomery multiplier with

a constrained modulus yields the best results area- and power-wise by limiting the range of possible modulus values. Under the circumstances of post-quantum cryptographic schemes, the Number Theoretic Transform is optimized using algorithms that employ vector integer instructions in their implementation. This detail facilitates parallelization and pipelining, which streamline the modular reduction process. Finally, we can infer that for both FPGA and ASIC platforms, parallel designs for modular multipliers manage to preserve the most resources.

## References


[BAM21] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. "High-Speed NTT-based Polynomial Multiplication Accelerator for Post-Quantum Cryptography". In: *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. 2021, pp. 94–101. DOI: 10.1109/ARITH51176.2021.00028.

[Bar86] Paul Barrett. "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor". In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1986, pp. 311–323.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 353–367.

[Eic] Maria Eichlseder. *Cryptography - L8 - Asymmetric Cryptography - Post Quantum Cryptography*. Accessed: 2024-11-30. URL: https://seafile.iaik.tugraz.at/seafhttp/files/f2f6fbbf-20c1-4123-ae6b-20e71e4cefbc/L8_Asymmetric_Crypto_PostQuantum.pdf.

[GCB+22] Gamze Gürsoy, Eduardo Chielle, Charlotte M Brannon, Michail Maniatakos, and Mark Gerstein. "Privacy-preserving genotype imputation with fully homomorphic encryption". In: *Cell systems* 13.2 (2022), pp. 173–182.

[HGG07] William Hasenplaugh, Gunnar Gaubatz, and Vinodh Gopal. "Fast Modular Reduction". In: *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*. 2007, pp. 225–229. DOI: 10.1109/ARITH.2007.18.

[KP06] Yinan Kong and Braden Phillips. "Comparison of Montgomery and Barrett modular multipliers on FPGAs". In: *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*. IEEE. 2006, pp. 1687–1691.

[LN16] Patrick Longa and Michael Naehrig. "Speeding up the number theoretic transform for faster ideal lattice-based cryptography". In: *Cryptology and Network Security: 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings 15*. Springer. 2016, pp. 124–139.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On ideal lattices and learning with errors over rings". In: *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer. 2010, pp. 1–23.

[LS19] Vadim Lyubashevsky and Gregor Seiler. *NTTRU: Truly Fast NTRU Using NTT*. Cryptology ePrint Archive, Paper 2019/040. 2019. URL: https://eprint.iacr.org/2019/040.

[Mon85] Peter L Montgomery. "Modular multiplication without trial division". In: *Mathematics of computation* 44.170 (1985), pp. 519–521.

[MP13] Daniele Micciancio and Chris Peikert. "Hardness of SIS and LWE with small parameters". In: *Annual cryptology conference*. Springer. 2013, pp. 21–39.

[MR07] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *SIAM Journal on Computing* 37.1 (2007), pp. 267–302. DOI: 10.1137/S0097539705447360. eprint: https://doi.org/10.1137/S0097539705447360. URL: https://doi.org/10.1137/S0097539705447360.

[RAF+15] Antonius P. Renardy, Nur Ahmadi, Ashbir A. Fadila, Naufal Shidqi, and Trio Adiono. "Hardware implementation of montgomery modular multiplication algorithm using iterative architecture". In: *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)*. 2015, pp. 99–102. DOI: 10.1109/ISITIA.2015.7219961.

[Reg] Oded Regev. *Lattices in Computer Science*. Accessed: 2024-11-27. URL: https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/introduction.pdf.

[Reg10] Oded Regev. "The learning with errors problem". In: *Invited survey in CCC* 7.30 (2010), p. 11.

[Roy] Sujoy Sinha Roy. *Selected Topics - Cryptography on HW Platform - Modular Arithmetic Techniques*. Accessed: 2024-12-12. URL: https://seafile.iaik.tugraz.at/seafhttp/files/f2f6fbbf-20c1-4123-ae6b-20e71e4cefbc/L8_Asymmetric_Crypto_PostQuantum.pdf.

[SNG+23] Deepraj Soni, Mohammed Nabeel, Homer Gamil, Oleg Mazonka, Brandon Reagen, Ramesh Karri, and Michail Maniatakos. "Design Space Exploration of Modular Multipliers for ASIC FHE accelerators". In: *2023 24th International Symposium on Quality Electronic Design (ISQED)*. 2023, pp. 1–8. DOI: 10.1109/ISQED57927.2023.10129292.

[SWA12] Suhas Sreehari, Huapeng Wu, and Majid Ahmadi. *Application of new classes of Mersenne primes for fast modular reduction for large-integer multiplication*. 2012.