Efficient Modular Multiplication Hardware for Number Theoretic Transform on FPGA

Tolun Tosun
Sabanci University
Istanbul, Turkey
toluntosun@sabanciuniv.edu

Selim Kırbıyık[*]
Sabanci University
Istanbul, Turkey
selim.kirbiyik@sabanciuniv.edu

Emre Koçer
Sabanci University
Istanbul, Turkey
kocer@sabanciuniv.edu

Erkay Savaş
Sabanci University
Istanbul, Turkey
erkays@sabanciuniv.edu

Ersin Alaybeyoğlu
Sabanci University
Istanbul, Turkey
ersin.alaybeyoglu@sabanciuniv.edu

Abstract

In this paper, we present a comprehensive analysis of various modular multiplication methods for Number Theoretic Transform (NTT) on FPGA. NTT is a critical and time-intensive component of Fully Homomorphic Encryption (FHE) applications while modular multiplication consumes a significant portion of the design resources in an NTT implementation. We study the existing modular reduction approaches from the literature, and implement particular methods on FPGA. Specifically Word-Level Montgomery (WLM) for NTT friendly primes [19] and K^{2}RED [3]. For improvements, we explore the trade-offs between the number of available primes in special forms and hardware cost of the reduction methods. We develop a DSP multiplication-optimized version of WLM, which we call WLM-Mixed. We also introduce a subclass of Proth primes, referred to as Proth-l primes, characterized by a low and fixed signed Hamming Weight. This special class of primes allows us to design multiplication-free shift-add versions of K^{2}RED and naive Montgomery reduction [20], referred to as K^{2}RED-Shift and Montgomery-Shift. We provide in-depth evaluations of these five reduction methods in an NTT architecture on FPGA. Our results indicate that WLM-Mixed is highly resource-efficient, utilizing only 3 DSP multiplications for 64-bit coefficient moduli. On the other hand, K^{2}RED-Shift and Montgomery-Shift offer DSP-free alternatives, which can be beneficial in specific scenarios.

Keywords

Modular Reduction, FPGA, Montgomery, K^{2}RED, DSP, FHE, NTT

1 Introduction

FHE is an advanced encryption technique that allows computations on encrypted data without needing to decrypt it first. FHE emerged in recent years by pioneering work from Gentry [12]. With FHE, data can remain secure even when processed by third-party data centers.

Several FHE algorithms exist in the literature such as BFV [4, 9], CKKS [13] and TFHE [5]. Existing FHE algorithms are lattice-based schemes which is based on polynomial ring arithmetic. The core operation is the polynomial multiplication in the ring of polynomials. For FHE, the degrees of these polynomials range from $n2^{12}$ to 2^{16}. Given the computational expense associated with operating on polynomials of such high degrees, there has been substantial research in the literature focused on accelerating FHE using FPGAs.

The state-of-art algorithm for polynomial multiplication is the well-known NTT, which reduces the complexity from $O(n^2)$ to $O(n \log n)$ compared to the naive school-book approach. The core component of the NTT is the butterfly unit, which primarily involves modular multiplication. There exists a variety of modular multiplication methods in the literature, such as Montgomery [20], Barrett [2], Plantard [25], and Montgomery-based methods like WLM [18], K^{2}RED [3]. These methods have different characteristics in terms of hardware complexity, latency, and throughput. In this paper, we study different modular reduction methods in the context of NTT implementations. Our contributions are outlined below.

- We evaluate existing modular reduction algorithms from the literature, focusing on their implementation in NTT for FHE applications on FPGAs. We propose an efficient architecture for K^{2}RED.
- We propose a novel variant of the WLM reduction algorithm [19], referred to as WLM-Mixed. This approach is particularly effective for 64-bit coefficient modulus, as it significantly reduces the number of DSP multiplications.
- We propose runtime configurable shift-add versions of the K^{2}RED and naive Montgomery reduction algorithms. These multiplication-free reduction algorithms are achieved by introducing a special subclass of Proth primes, referred to as Proth-l primes.
- We investigate the range of special primes utilized in this study, particularly in relation to the number of primes required for RNS representation in FHE applications. We analyze the trade-offs between the number of primes and the hardware cost of reduction algorithms.
- We implement the proposed modular reduction methods on AMD-Xilinx Alveo U280 (XCU280) FPGA, as well as the state-of-the-art techniques from the literature. We provide a comprehensive analysis of the performances of different modular multiplication methods studied in this paper on FPGA. According to characteristics of these algorithms, we present a general analysis for which modular multiplication suits the given parameter set. An example analysis provided in the paper is the trade-off between DSP and distributed logic use for a given algorithm.

[*] Tolun Tosun and Selim Kırbıyık declare equal contribution

2 Background

2.1 Notation

- Lowercase italic letters, such as a, represent integers. The logarithm function (log) is base-2 and returns the ceiling integer. Values of individiual bits of integers are shown using square brackets, e.g., $a[i]$.
- Bold lowercase letters, such as $\mathbf{a}$, denote vectors. Elements of vectors are accessed using sub-indices, e.g., $\mathbf{a}_i$.
- q denotes the integer modulus and β is an alias for $\log q$.
- The cyclotomic ring of polynomials $\mathbb{Z}_q[x]/(x^n+1)$ is denoted by $\mathcal{R}_{q,n}$. Polynomials are represented by bold lowercase italic letters, such as $\boldsymbol{a}(x)$. To simplify the narration, indeterminate x of polynomials are sometimes omitted. Polynomial coefficients are represented by sub-indices, such as $\boldsymbol{a}_i$.

2.2 Number Theoretic Transform (NTT)

NTT is the state-of-the-art method for polynomial multiplication. For two polynomials $\boldsymbol{a}(x), \boldsymbol{b}(x) \in \mathcal{R}_{q,n}$, multiplication using the NTT algorithm is performed as follows:

$$\boldsymbol{a}(x) \cdot \boldsymbol{b}(x) = \mathrm{iNTT}\Big(\mathrm{NTT}\big(\boldsymbol{a}(x)\big) \odot \mathrm{NTT}\big(\boldsymbol{b}(x)\big)\Big) \tag{1}$$

where $\odot$ represents element-wise multiplication of vectors in NTT domain. For NTT to be defined over $\mathcal{R}_{q,n}$, it is required that $q \equiv 1 \pmod{2n}$. In this context, there exists a primitive $2n$-th root of unity in $\mathbb{Z}_q$, denoted as ψ, such that $\psi^n \equiv -1 \pmod q$. For this condition to be satistifed, q must be in the form of $q_h 2^{\omega} + 1$ where $\omega \geq \log n + 1$. The forward NTT corresponds to the evaluation $\hat{\mathrm{a}}[i] = \boldsymbol{a}(\psi^{2i+1})$ for every coefficient $i < n$. The NTT can be efficiently implemented using butterfly circuits. Forward NTT is usually implemented with *Cooley-Tukey* (CT) [6] butterflies while the inverse NTT is implemented with *Gentleman-Sande* (GS) [11] butterflies. For two coefficients $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$, the CT butterfly is defined as follows:

$$(\mathrm{a}'_i,\ \mathrm{a}'_j) = (\boldsymbol{a}_i + \boldsymbol{a}_j\zeta,\ \boldsymbol{a}_i - \boldsymbol{a}_j\zeta) \tag{2}$$

where ζ is the twiddle factor, which is a power of ψ. NTT with CT or GS butterflies have $\log n$ stages and performs $n/2$ butterflies at each stage, resulting in a time complexity of $O(n \log n)$.

2.3 Residue Number System (RNS)

FHE applications require performing large integer arithmetic due to security needs of the underlying computationally expensive Learning With Errors (LWE) [26] problem. RNS improves the efficiency of arithmetic operations in FHE, by representing large integers as a set of relatively smaller integers, called residues. Handling smaller integers reduces the complexity of modular arithmetic significantly. Let $a \in \mathbb{Z}_{\tilde{q}}$ and $\tilde{q} = \prod_i^{\lambda-1} q_i$. Then, the set of residues is defined as $\{a_i\}_i^{\lambda-1}$ where $a_i = a \pmod{q_i}$. By utilizing Chinese Remainder Theorem (CRT), addition and multiplication between two integers $a, b \in \mathbb{Z}_{\tilde{q}}$ can be performed in the RNS domain element-wise. Multiplication is performed as follows:

$$ab = \{a_i\}_{i=0}^{\lambda-1} \odot \{b_i\}_{i=0}^{\lambda-1} = \{a_i b_i \mod q_i\}_{i=0}^{\lambda-1} \tag{3}$$

$\log n$	$\log \tilde{q}$	λ	
		$\log q_i = 32$	$\log q_i = 64$
12	109	4	2
13	218	7	4
14	438	14	7
15	881	28	14
16	1761	55	28

Table 1: The relationship between $\tilde{q}$ and n for 128-bit security [1], showing the number of 32-bit and 64-bit primes q_i required with RNS.

Naturally, the isomorphism extends to the polynomials, $\mathcal{R}_{\tilde{q},n} \simeq \prod_i^{\lambda-1} \mathcal{R}_{q_i,n}$. The polynomial arithmetic involving the NTT is performed in the RNS domain, operating in each $\mathcal{R}_{q_i,n}$ independently. In practice, q_i are usually around 32 to 64 bits, depending on the implementation choices and requirements of FHE schemes.

2.4 Modular Reduction Algorithms

2.4.1 Montgomery Reduction. Montgomery reduction [20] is a widely used method for modular reduction in cryptographic applications, detailed in Algorithm 1. It requires two $\beta \times \beta$ multiplications and eliminates the division by using a modulus-dependent pre-computed factor, q'. The key idea is that by adding tq to the input a in Line 2, the lower β bits of $a + tq$ become 0. As a result, shifting it right by β bits reduces the bit-length of the result to β bits. The Montgomery reduction requires a final correction, as illustrated in Line 3. Note that the result of the Montgomery reduction includes a constant factor of $2^{-\beta}$.

Algorithm 1 Naive Montgomery Reduction [20]

```
Input: modulus $q < 2^{\beta}$, pre-computed factor $q' = -q^{-1} \pmod{2^{\beta}}$, operand $a < q^2$
Output: $b = a2^{-\beta} \pmod q$
1: $t \leftarrow q'a \pmod{2^{\beta}}$
2: $b \leftarrow (a + tq) \gg \beta$
3: if $b \geq q$ then $b \leftarrow b - q$
4: return $b$
```

2.4.2 Word-Level Montgomery Reduction for NTT Friendly Primes. Instead of performing the reduction entirely at once, a word-by-word reduction approach also exists. In this case, the word size ω-bit is reduced from the input operand at each iteration while $\lceil \beta/\omega \rceil$ iterations are performed. The pre-computed factor q' is computed as $-q^{-1} \pmod{\omega}$. This approach is particularly advantageous for the NTT friendly primes, as the pre-computed factor q' becomes -1. Consider the case $\omega = \log n + 1$ where $q = 1 \pmod{\omega}$. As a result, the multiplication by q' is eliminated. The word-level Montgomery reduction for NTT friendly primes [19] is detailed in Algorithm 2.

Algorithm 2 WLM; Word-Level Montgomery Reduction for NTT-friendly primes [19]

```
Input: modulus $q < 2^\beta$, word-size $\omega$ such that $q = 1 \pmod{\omega}$ and
    $q_h = q \gg \omega$, operand $a < q^2$
Output: $b = a2^{-\omega\lambda} \mod q$
1: $t \leftarrow a$
2: for $i = 0 \rightarrow \lceil \frac{\beta}{\omega} \rceil - 1$ do
3:     $t_l \leftarrow t \pmod{2^\omega}$, $t_h \leftarrow t \gg \omega$
4:     $t' \leftarrow -t_l \pmod{2^\omega}$
5:     $c \leftarrow t'[\omega - 1] \vee t[\omega - 1]$
6:     $t \leftarrow t_h + q_h t_l + c$
7: if $t \geq q$ then $b \leftarrow t - q$
8: else $b \leftarrow t$
9: return $b$
```

2.4.3 K^2RED. K^{2}RED [3] is a reduction algorithm originally developed for Crystals-Kyber. K^{2}RED requires the modulus to be a Proth prime, which are in the form of $q = q_h 2^\omega + 1$ where $\omega \geq \log q/2$. For Kyber, the multiplications by q_h in (Algorithm 3) Lines 2 and 4 can be efficiently performed by fixed shifts and summations as the modulus q is known in design time [22]. Either the input operand needs to be pre-processed or the result needs to be post-processed to correct the $2^{-2\omega}$ term.

Algorithm 3 K^{2}RED [3]

```
Input: a Proth prime modulus $q < 2^\beta$ where $q = 1 \pmod{\omega}$ for
    $\omega \geq \beta/2$ and $q_h = q \gg \omega$, operand $a < q^2$
Output: $b = a2^{-2\omega} \mod q$
1: $a_l \leftarrow a \pmod{2^\omega}$, $a_h \leftarrow a \gg \omega$
2: $t \leftarrow q_h a_l - a_h$
3: $t_l \leftarrow t \pmod{2^\omega}$, $t_h \leftarrow t \gg \omega$
4: $t' \leftarrow q_h t_l - t_h$
5: if $t' \geq q$ then $b \leftarrow t' - q$
6: else if $t' < 0$ then $b \leftarrow t' + q$
7: else $b \leftarrow t'$
8: return $b$
```

2.4.4 Barrett Reduction. Barrett reduction [2] is another classical reduction algorithm widely used in cryptographic applications, detailed in Algorithm 4. Similar to Montgomery reduction, it requires 2 multiplication and uses a pre-computed factor. Note that the multiplication in Line 1 is $2\beta \times \beta$, making the overall multiplication cost for Barrett reduction $(2\beta \times \beta) + (\beta \times \beta)$. Also, the output of the Barrett reduction does not involve a constant factor compared to the Montgomery reduction.

Algorithm 4 Barrett Reduction [2]

```
Input: modulus $q < 2^\beta$, pre-computed factor $q' = \lfloor 2^{2\beta}/q \rfloor$,
    operand $a < q^2$
Output: $b = a \pmod{q}$
1: $t \leftarrow (aq') \gg (2\beta)$
2: $b \leftarrow (a - tq)$
3: if $b \geq q$ then $b \leftarrow b - q$
4: return $b$
```

$\log q$	$\log q_h$	Proth-	#primes
64	32	$3l$	469
64	17	$3l$	53
64	32	$2l$	16
64	17	$2l$	5
32	16	$3l$	95
32	15	$3l$	80
32	16	$2l$	7
32	15	$2l$	7

Table 2: Number of proth-l primes for different bit-widths.

2.4.5 Plantard Reduction. Plantard reduction [25] is a relatively newer reduction algorithm, detailed in Algorithm 5. Similar to Barrett and Montgomery, Plantard reduction also uses a pre-computed factor q'. However, the multiplication bq' in Line 1 can be pre-computed, making Plantard reduction particularly advantageous for constant multiplications, such as those involving twiddle factors in butterfly circuits. As a drawback, the multiplication $(a) \cdot (bq')$ needs to be performed with double-word precision, a $2\beta \times \beta$ multiplication. The overall multiplication complexity for the case of multiplication by a constant is $(2\beta \times \beta) + (\beta \times \beta)$.

Algorithm 5 Plantard Modular Multiplication [25]

```
Input: modulus $q < 2^\beta$, pre-computed factor $q' = q^{-1}$
    $\mod (2^{2\beta})$, operands $a, b < q$
Output: $c = ab(-2^{-2\beta}) \mod q$
1: $c' \leftarrow (abq' \pmod{2^{2\beta}}) \gg \beta$
2: $c \leftarrow ((c' + 1)q) \gg \beta$
3: if $c = q$ then $c = 0$
4: return $c$
```

3 Proposed Modular Reduction Algorithms

3.1 Proth-l Primes

In this section, we introduce Proth-l primes, which are a subset of Proth primes. Recall that Proth primes are in the form of $q_h 2^\omega + 1$ where $\log q_h \leq \omega$. Proth-l primes require that the number of non-zero terms in the signed binary representation of q_h is small. To clarify the number of non-zero terms, we provide the following definitions:

Definition: Proth-2l prime. A Proth prime q is also a Proth-2l prime if $q = 2^{\beta-1} + (2^{l_1} - 2^{l_2})2^\omega + 1$ where $0 \leq l_2 \leq l_1 < \beta - \omega - 1$.

Definition: Proth-3l prime. A Proth prime q is also a Proth-3l prime if $q = 2^{\beta-1} + (2^{l_1} - 2^{l_2} + 2^{l_3})2^\omega + 1$ where $0 \leq l_2 \leq l_1 < \beta - \omega - 1$ and $0 \leq l_3 < \beta - \omega - 1$.

We write Proth-l to refer both Proth-$2l$ or Proth-$3l$ primes. Recall that the RNS representation requires use of multiple primes of relatively smaller sizes (in practice, usually 32-bit or 64-bit). Therefore, it is essential to know the number of Proth-l primes that can be found for these parameters. As shown in Table 2, the number of Proth-l primes is sufficient for FHE applications that employ 32-bit or 64-bit RNS arithmetic. For instance, consider the ring dimension $n = 2^{16}$ which requires $\log \tilde{q} \approx 1800$ for 128-bit security level. This case can be achieved by Proth-$3l$ primes and 32-bit RNS arithmetic, as 80 such primes are found (when $\log q_h = 15$) and $80 \times 32 > 1800$. Also, recall from Section 2.2 that NTT with $n = 2^{16}$ requires $\omega \geq 17$ as a $2n$-th root of unity is needed. In the next two sections, we present modular reduction algorithms that replace multiplications with shift-adds using Proth-l primes.

3.2 Montgomery with Barrel Shifters

In this section, we present a shift-add variant of naive Montgomery reduction algorithm (Algorithm 1) for Proth-l primes, referred to as Montgomery-Shift.

For a β-bit Proth prime q which is of the form $q = q_h 2^{\omega} + 1$, the Montgomery factor q' is $q_h 2^{\omega} - 1$:

$$q'q = (q_h 2^{\omega} + 1)(q_h 2^{\omega} - 1) = q_h^2 2^{2\omega} - 1 = -1 \pmod{2^{\beta}} \quad (4)$$

Recall that $2^{2\omega} \geq 2^{\beta}$ is satisfied for Proth primes. For a Proth-$3l$ prime q:

$$q' = 2^{\beta-1} + (2^{l_1} - 2^{l_2} + 2^{l_3})2^{\omega} - 1 \quad (5)$$

As a result, the multiplications in Line 1 and Line 2 of Algorithm 1 can be implemented with 3 barrel shifters for Proth-$3l$ primes. Algorithm 6 presents the revised Montgomery reduction algorithm. Similarly, for Proth-$2l$ primes, the multiplications can be achieved using two barrel shifters. We would like to note that the shifts with l_1, l_2 and l_3 are run-time configurable while the shift with β and ω are known at the design-time. The bit-lengths of l_1, l_2, and l_3 are $\log(q_h - 1)$, which is a critical factor for the hardware cost of the shifts in Line 2 and Line 3. For example, when $9 < \log q_h \leq 17$, $\log l_i = 4$. Similarly, when $17 < \log q_h \leq 33$, $\log l_i = 5$. Recall from Section 3.1 that the number of Proth-l primes increase with $\log q_h$, which is a trade-off between the number of primes and the hardware cost of the shifts. When $\log q_h = 2^x + 1$ the number of primes is maximized for $\log l_i = x$.

3.2.1 *Further Discussion on Barrett and Plantard.*

We would like to note that the optimization presented in this section does not apply to the Barrett and Plantard algorithms. For Plantard reduction, q' is computed modulo $2^{2\beta}$ which will avoid disappearance of the term $q_h^2 2^{2\omega}$ in Equation (4). For Barrett reduction, although the factor q' is approximately β-bit, the result of the division by a Proth-l modulus q does not lead to a term with low signed Hamming weight.

3.3 K^{2}RED with Barrel Shifters

In this section, we present the shift-add variant of the run-time configurable version of the K^{2}RED algorithm (Algorithm 3) for Proth-l primes, referred to as K^{2}RED-Shift.

The trade-off between the number of primes and the hardware cost, through $\log q_h$ and using l_3, directly applies to K^{2}RED-Shift.

Algorithm 6 Montgomery-Shift; Montgomery Reduction with Shift-Adds

```
Input: a proth-$l$ prime modulus $q = 2^{\beta-1} + (2^{l_1} - 2^{l_2} + 2^{l_3})2^{\omega} + 1$,
    operand $a < q^2$
Output: $b = a2^{-\beta} \pmod{q}$
1: $a_l \leftarrow a_l \pmod{2^{\beta}}$, $a_h \leftarrow a \gg \beta$
2: $t \leftarrow \big(a_l \ll (\beta - 1)\big) + \Big(\big((a_l \ll l_1) - (a_l \ll l_2) + (a_l \ll l_3)\big) \ll \omega\Big) - a_l \pmod{2^{\beta}}$
3: $t' \leftarrow \big(t \ll (\beta-1)\big) + \Big(\big((t \ll l_1) - (t \ll l_2) + (t \ll l_3)\big) \ll \omega\Big) + t$
4: $t'_h \leftarrow t' \gg \beta$, $c \leftarrow t'[\beta - 1] \vee a_l[\beta - 1]$
5: $b' \leftarrow (a_h + t'_h + c)$
6: if $b' \geq q$ then $b \leftarrow b' - q$
7: else $b \leftarrow b'$
8: return $b$
```

Algorithm 7 K^{2}RED-Shift; K^{2}RED with Shift-Adds

```
Input: proth-$l$ prime modulus $q = 2^{\beta-1} + (2^{l_1} - 2^{l_2} + 2^{l_3})2^{\omega} + 1$
    where $\omega \geq \beta/2$, operand $a \leq (q - 1)^2$
Output: $b = a2^{-2\omega} \pmod{q}$
1: $a_l \leftarrow a \pmod{2^{\omega}}$, $a_h \leftarrow a \gg \omega$
2: $t \leftarrow ((a_l \ll \beta-1-\omega)+(a_l \ll l_1)+(a_l \ll l_2))-((a_l \ll l_2)+a_h)$
3: $t_l \leftarrow t \pmod{2^{\omega}}$, $t_h \leftarrow t \gg \omega$
4: $t' \leftarrow ((t_l \ll \beta-1-\omega)+(t_l \ll l_1)+(t_l \ll l_2))-((t_l \ll l_2)+t_h)$
5: if $t' \geq q$ then $b \leftarrow t' - q$
6: else if $t' < 0$ then $b \leftarrow t' + q$
7: else $b \leftarrow t'$
8: return $b$
```

In Algorithm 7 the terms l_1, l_2, l_3 are run-time configurable and terms β and $\log q_h$ are design-time configurable, as in Montgomery-Shift. Recall that this flexibility allows for efficient reduction using barrel shifters with a specified range of primes which we can utilize in NTT. Although Algorithm 7 explicitly uses Proth-$3l$ primes, it trivially applies to Proth-$2l$ primes by removing l_3 terms. Using Proth-$3l$ provides additional primes at the cost of increased area. Algorithm 7 requires six barrel shifters for Proth-$3l$ primes and four barrel shifters for Proth-$2l$ primes.

3.4 Mixed-Radix Word-level Montgomery Reduction

In this section, we present an improved word-level Montgomery reduction algorithm that significantly reduces the number of DSP multiplications compared to Algorithm 2. This is achieved by using a mixed-radix approach with two reduction iterations and the regular Proth primes. The term mixed-radix indicates that the iterations are performed with varying word sizes. Algorithm 8 details the approach, referred to as WLM-Mixed. The word sizes for each iteration, ω_0 and ω_1, are determined based on the DSP operand sizes and $\log q_h$. Notice that the assignments in Line 1 aims to fit the multiplication in the second iteration into a single DSP by selecting the appropriate word size ω_1. Then, ω_0 is set accordingly. Lines 3-8 perform the reduction iterations as in Algorithm 2. The shift by δ_i

is needed to align the multiplication output with the one in Line 6 of Algorithm 2. Note that for Algorithm 2, $\omega + \log q_h = \beta$. Similarly, for Algorithm 8, the relationship $\omega_i + \log q_h + \delta_i = \omega + \log q_h = \beta$ holds.

The requirement for q to be a Proth prime ensures that $\omega_0 \leq \omega$ (see Line 1 of Algorithm 8). Otherwise, the first iteration would require explicit computation of the Montgomery factor q' and therefore the existing algorithm definition would be incorrect.

Algorithm 8 WLM-Mixed; DSP-Optimized Mixed-Radix Word-Level Montgomery Reduction with two iterations

```
Input: a Proth prime modulus $q < 2^\beta$; DSP operand bit-lengths $\Gamma_A$, $\Gamma_B$ such that $\Gamma_A \geq \Gamma_B$; $\log q_h$ such that for $\omega = \beta - \log q_h$, $q = 1 \pmod{\omega}$, $\log q_h \leq \Gamma_B$, $\omega \geq \beta/2$ and $q_h = q \gg \omega$; operand $a < q^2$
Output: $b = a2^{-\beta} \mod q$
1:  $\omega_1 \leftarrow \min(\Gamma_A, \omega)$, $\omega_0 \leftarrow \beta - \omega_1$
2:  $t \leftarrow a$
3:  for $i = 0 \rightarrow 2$ do
4:      $\delta_i \leftarrow \omega - \omega_i$
5:      $t_h \leftarrow t \gg \omega_i$, $t_l \leftarrow t \pmod{2^{\omega_i}}$
6:      $t' \leftarrow -t_l \pmod{2^{\omega_i}}$
7:      $c \leftarrow t'[\omega_i - 1] \vee t[\omega_i - 1]$
8:      $t \leftarrow t_h + \left((q_h t_l) \ll \delta_i\right) + c$
9:  if $t \geq q$ then $b \leftarrow t - q$
10: else $b \leftarrow t$
11: return $b$
```

As an example, consider 26×17-bit DSP multipliers ($\Gamma_A = 26, \Gamma_B = 17$), and a 64-bit Proth prime q where $\log q_h = 17$. Then, the second reduction iteration is performed with $\omega_1 = 26$ and the second iteration is performed with $\omega_1 = 38$. Notice that both are feasible since $\log q - \log q_h = 47 \geq 26, 38$. The first iteration involves a 17×38-bit multiplication, which can be executed using 2 DSPs, while the second iteration involves a 17×26-bit multiplication, utilizing 1 DSP. Consequently, the total number of DSP multiplications is three. The number of DSP multiplications is significantly greater for the classical word-level Montgomery reduction (Algorithm 2) using general NTT-friendly primes. Consider the case $n = 2^{16}$, therefore $\omega = 17$. Then, four iterations are needed where a 47×17-bit multiplication is performed in each iteration, resulting in 8 DSP multiplications in total. For lower ring dimensions, the difference in the number of DSP multiplications becomes even more significant.

Naturally, the WLM-mixed approach limits the range of primes. Specifically, the non-zero bits in the prime modulus q are constrained by the DSP operand sizes. However, there are still enough Proth primes that satisfy these constraints for FHE applications. For example, when $\log q = 64$ and $\log q_h = 17$, there are 2986 such primes available, which is far more than what is needed for FHE applications.

4 Implementation

In this section, we provide implementation details for the proposed algorithms, WLM-Mixed (Algorithm 8), K^{2}RED-Shift (Algorithm 7) and Montgomery-Shift (Algorithm 6). We also implement WLM (Algorithm 2) and K^{2}RED (Algorithm 3) from the literature and provide the architectural details in this section. We do not implement the naive Montgomery reduction, Barrett reduction, and Plantard reduction, as these algorithms are theoretically more costly than the algorithms implemented, as detailed in Section 5.1. The implementations are developed using SystemVerilog (and Verilog) HDL, designed with a high degree of parameterization. Parameters such as $\log q$, $\log q_h$, and the option to enable l_3 for shift-add designs, can be specified at design time to generate the corresponding hardware. The complete implementations are publicly accessible at https://github.com/cisec-su/modmul-hdl.

4.1 Architecture

In WLM, K^{2}RED, and WLM-Mixed, the multiplications are performed using a multiply-accumulate approach. Initially, the operands are partitioned according to the DSP word sizes to generate partial products. These partial products are then accumulated along with the corresponding terms specific to each algorithm. For example, in WLM-Mixed (Algorithm 8), the partial products from $q_h t_l$ are summed together with t_h and the carry c. Architectures implemented for WLM-Mixed and K^{2}RED for $\log q = 64$ are detailed in Figure 1 and Figure 2, respectively. The architecture implemented for WLM implemented for WLM follows [19].

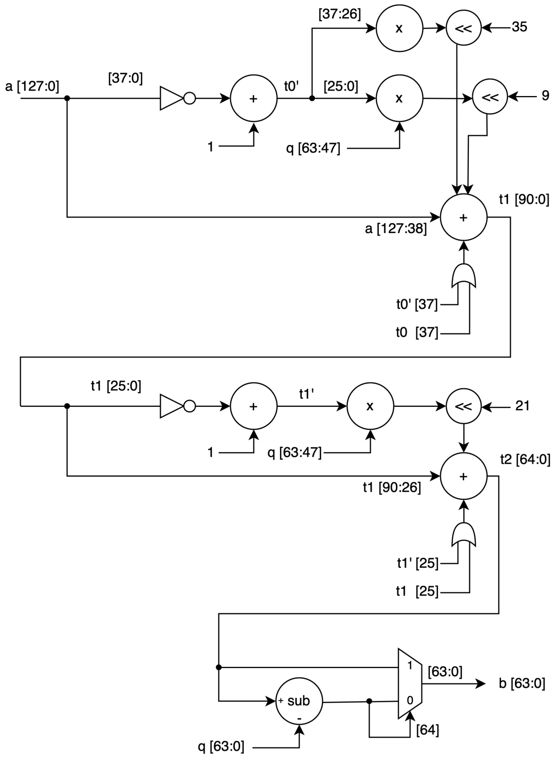


Figure 1: WLM-Mixed Architecture for $\log q = 64$ **and** $\log q_h = 17$**. DSP operand sizes are** $\Gamma_A = 26$, $\Gamma_B = 17$**.**

The implementations are pipelined. Note that the pipeline steps are not illustrated in Figure 1 and Figure 2. Specifically, we place

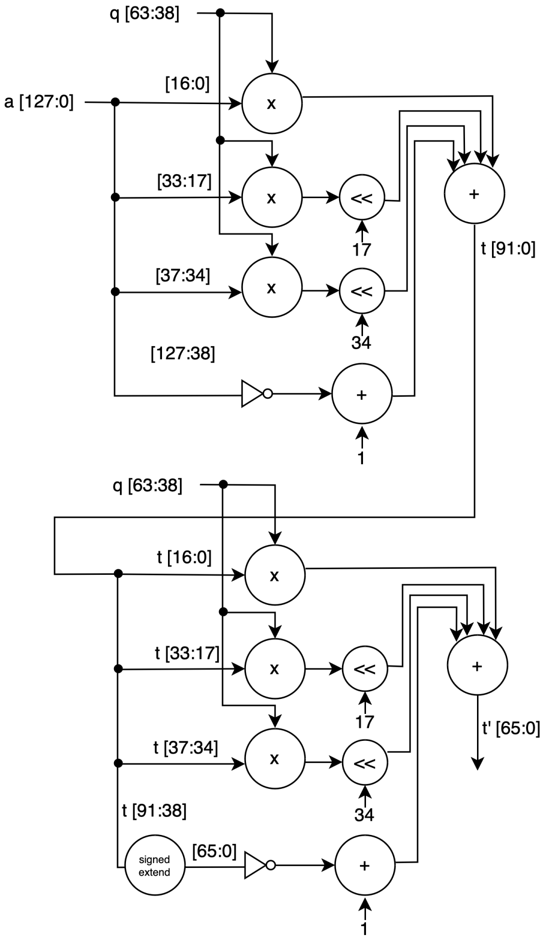
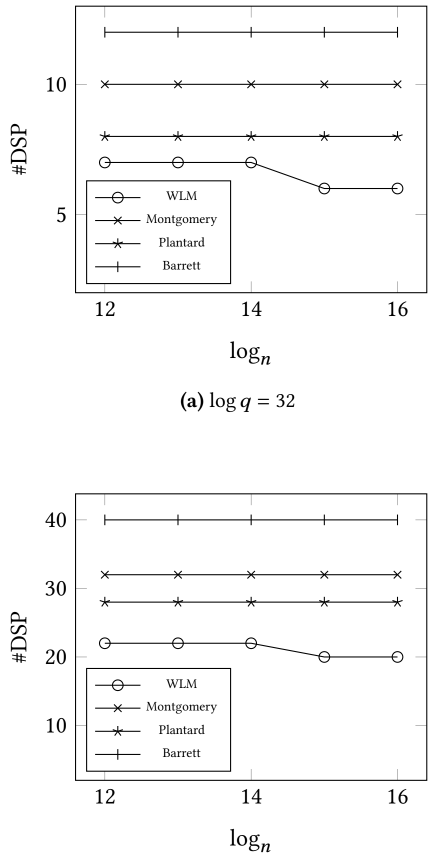


Figure 2: K^{2}RED Architecture for $\log q = 64$ and $\log q_h = 26$. DSP operand sizes are $\Gamma_A = 26$, $\Gamma_B = 17$. The correction step in Lines 5-7 of Algorithm 3 is skipped.



(a) $\log q = 32$

(b) $\log q = 64$

Figure 3: DSP multiplication counts compared for Montgomery (Algorithm 1), Barrett (Algorithm 4), Plantard (Algorithm 5), WLM (Algorithm 2).

Flip-Flops (FFs) to the partial products from the computation of $q_h t_l$ in Line 6, the output of summation t in Line 6, and b in Lines 7-8 of Algorithm 2 for WLM. As a result, the latency of WLM is $2\lceil \beta/\omega \rceil + 1$ clock cycles (ccs). We use the same pipeline strategy for WLM-Mixed and the latency is equal to 5 ccs. Similarly, for K^{2}RED, we put FFs to the partial products from $q_h a_l$ and the output of summation t in Line 2; the partial products from $q_h t_l$ and the output of summation t' in Line 4; and b in Lines 6-8 of Algorithm 3, resulting in 5 ccs latency. For K^{2}RED-Shift and Montgomery-Shift, we implement two pipeline configurations. ρ_A, ρ_B. For ρ_A, the shifts and additions are performed in different ccs while these are performed in single cc for ρ_B. This leads to a trade-off between speed and area. For Montgomery-Shift and ρ_A, we put FFs for t in Line 2; t' in Line 3; b' in Line 4; and b in Lines 6-7 of Algorithm 6, resulting in 4 ccs latency. Due to additional FFs, Latency of Montgomery-Shift is increased to 6 cc by using ρ_A. Similarly, for K^{2}RED-Shift and ρ_B we put FFs for t in Line 2; t' in Line 4; b' in Line 4; and b in Lines 5-7 of Algorithm 7, resulting in 3 ccs latency. Using ρ_A increases the latency of K^{2}RED-Shift to 5.

5 Evaluation

In this section, we evaluate the resource efficiency of the proposed reduction algorithms in addition to the studied ones from the literature. First, we provide a theoretical analysis of the DSP usage by naive Montgomery (Algorithm 1), Barrett (Algorithm 4) and Plantard (Algorithm 5). Then, we provide practical results for the implemented reduction algorithms, namely WLM (Algorithm 2), WLM-Mixed (Algorithm 8), K^{2}RED (Algorithm 3), K^{2}RED-Shift (Algorithm 7) and Montgomery-Shift (Algorithm 6). We use the Area-Time-Product (ATP) as our assessment metric, which is widely used in the literature [27]. ATP is calculated as Average Latency (μs) × (LUT + FF/2 + 100 × DSP + 300 × BRAM). We target the AMD-Xilinx Alveo U280[1] (XCU280) FPGA for our evaluations, using Vivado

[1] https://www.xilinx.com/products/boards-and-kits/alveo/u280.html

$\log q$	$\log q_h$	# primes	max. $\log n$	LUT	FF	DSP	Freq. (MHz)	ATP
			WLM (Algorithm 2)					
64	47	*	16	487	1177	8	416	4.5
64	48	*	15	518	1195	8	416	4.59
64	49	*	14	566	1450	10	434	5.26
64	50	*	13	579	1478	10	434	5.33
64	51	*	12	583	1506	10	434	5.37
32	47	*	16	136	226	2	526	0.85
32	48	*	15	130	210	2	526	0.82
32	49	*	14	152	315	3	526	1.15
32	50	*	13	156	320	3	526	1.17
32	51	*	12	160	325	3	526	1.18
			WLM-Mixed (Algorithm 8)					
64	17	✈	16	315	522	3	476	1.83
32	15	*	16	131	211	2	526	0.82
			K^2RED (Algorithm 3)					
64	26	✈	≥ 16	411	424	6	476	2.56
64	32	✈	≥ 16	432	483	8	476	3.09
32	15	*	16	151	201	2	588	0.76
32	16	*	15	154	210	2	588	0.78

✈: More than 1K. *: All NTT primes w.r.t. max. $\log n$.

Table 3: Implementation results for DSP-based reduction Algorithms.

pipe. conf.	$\log q$	$\log q_h$	l_3 en.	# primes	max. $\log n$	LUT	FF	DSP	Freq. (MHz)	ATP
			K^2RED-Shift (Algorithm 7)							
ρ_A	64	17	✓	53	16	1259	979	0	476	3.67
ρ_B	64	17	✓	53	16	1295	409	0	370	4.04
ρ_B	64	17	✗	5	13	1079	397	0	454	2.81
ρ_A	64	32	✓	469	≥ 16	1287	1071	0	500	3.64
ρ_B	64	32	✓	469	≥ 16	1340	465	0	344	4.56
ρ_A	64	32	✗	16	15	945	953	0	588	2.41
ρ_B	64	32	✗	16	15	1048	449	0	416	3.05
ρ_A	32	15	✓	80	16	611	537	0	588	1.49
ρ_B	32	15	✓	80	16	617	245	0	434	1.7
ρ_B	32	15	✗	7	13	510	232	0	555	1.12
ρ_A	32	16	✓	80	16	595	537	0	555	1.55
ρ_B	32	16	✓	95	16	598	248	0	416	1.73
ρ_A	32	16	✗	7	13	462	466	0	714	0.97
ρ_B	32	16	✗	7	13	488	235	0	526	1.15
			Montgomery-Shift (Algorithm 6)							
ρ_A	64	47	✓	53	16	1114	954	0	526	3.02
ρ_B	64	47	✓	53	16	1002	591	0	416	3.11
ρ_B	64	47	✗	5	13	671	583	0	416	2.31
ρ_A	64	32	✓	469	≥ 16	1545	1101	0	500	4.19
ρ_B	64	32	✓	469	≥ 16	1391	642	0	416	4.1
ρ_B	64	32	✗	16	15	945	632	0	416	3.02
ρ_A	32	15	✓	80	16	665	558	0	625	1.49
ρ_B	32	15	✓	80	16	603	329	0	434	1.76
ρ_B	32	15	✗	7	13	411	321	0	476	1.2
ρ_A	32	16	✓	95	16	667	564	0	588	1.61
ρ_B	32	16	✓	95	16	639	332	0	454	1.77
ρ_B	32	16	✗	7	13	420	324	0	476	1.22

Table 4: Implementation results for shift-add-based reduction algorithms.

2023.2[2] for synthesis and implementation. The XCU280 FPGA features 1303680 Look-Up Table (LUT)s, 2607360 FFs, 9024 DSPs[3], and 2016 BRAM36E1s. The DSP48E2 unit supports 26 × 17 unsigned multiplication with accumulation ($\Gamma_A = 26$, $\Gamma_B = 17$).

5.1 Theoretical Analysis of Montgomery, Barrett and Plantard

Figure 3 shows that the required number of DSP multiplications are significantly smaller for WLM compared to naive Montgomery, Barrett and Plantard. Recall that it eliminates the multiplication by a pre-computed factor such as q' which is the main reason behind its superiority. The number of DSP multiplications are counted by assuming standard tiling. In particular, to perform a $\beta \times \beta$-bit multiplication, $\lceil \beta/\Gamma_A \rceil \lceil \beta/\Gamma_B \rceil$ DSP multiplications are needed. Moreover, for multiplications involving q, only q_h is considered as an operand as the lower bits of NTT friendly primes is fixed. For instance, the multiplication tq in Line 2 of Algorithm 1 is considered as a $\beta \times \log q_h$-bit multiplication. Also note that we report the DSP usage for a modular multiplication, involving the multiplication part as well. Recall that the Plantard reduction is particularly effective for multiplication by a constant situation, such as in a butterfly circuit of NTT. Therefore, to make a fair comparison, we include the cost of integer multiplication. For instance, for Montgomery, the total cost of modular multiplications is the summation of $\beta \times \beta$ for the integer multiplication part, and $(\beta \times \beta) + (\beta \times \log q_h)$ for the reduction part which is based on Algorithm 1. In this case, the integer multiplication requires 12 DSP multiplications while the reduction part requires 12 + 8 DSP multiplications, leading to 32 DSPs in total for $\beta = 64$. The DSP usage is counted in the same manner for the compared algorithms. Note that we did not include K^2RED and WLM-Mixed in the theoretical analysis, as a practical analysis is provided in the next section. This section aims to provide a rationale for these algorithms with WLM.

5.2 Parameter Selection

In this section, we explain the parameter selection for evaluated algorithms. Since WLM supports general NTT-friendly primes, we

[2] https://www.xilinx.com/products/design-tools/vivado.html

[3] https://docs.amd.com/r/en-US/ug958-vivado-sysgen-ref/DSP48E2

Reduction	$\log n$	PE	$\log q_h$	pipe.	l_3	LUT	FF	DSP	BRAM	Freq.	Lat.	ATP $\cdot 10^{-3}$
				conf.	en.	$\cdot 10^{-3}$	$\cdot 10^{-3}$			(MHz)	(μs)	
WLM	12	16	19			14	10	112	17	434	3.60	113.71 (**1.07×**)
WLM-Mixed	12	16	15			14	9	96	17	434	3.59	105.78 (**1.00×**)
K^{2}RED	12	16	15			14	11	96	17	416	3.75	111.73 (**1.06×**)
K^{2}RED-Shift	12	16	16	ρ_A	✗	17	13	64	17	400	3.91	119.51 (**1.13×**)
Montgomery-Shift	12	16	16	ρ_B	✗	17	10	64	17	400	3.91	115.56 (**1.09×**)
WLM	12	32	19			26	22	224	32.5	357	2.23	135.24 (**1.17×**)
WLM-Mixed	12	32	15			25	19	192	32.5	370	2.15	117.62 (**1.02×**)
K^{2}RED	12	32	15			25	21	192	32.5	384	2.07	115.52 (**1.00×**)
K^{2}RED-Shift	12	32	16	ρ_A	✗	34	26	128	32.5	384	2.07	125.37 (**1.08×**)
Montgomery-Shift	12	32	16	ρ_B	✗	34	20	128	32.5	384	2.06	120.39 (**1.04×**)
WLM	12	64	19			42	39	448	64.5	312	1.32	142.89 (**1.20×**)
WLM-Mixed	12	64	15			40	34	384	64.5	312	1.32	127.55 (**1.07×**)
K^{2}RED	12	64	15			41	39	384	64.5	333	1.23	123.47 (**1.03×**)
K^{2}RED-Shift	12	64	16	ρ_A	✗	57	49	256	64.5	357	1.15	124.9 (**1.05×**)
Montgomery-Shift	12	64	16	ρ_B	✗	58	39	256	64.5	357	1.15	119.46 (**1.00×**)
WLM	14	16	17			14	11	112	50	400	18.01	565.49 (**1.10×**)
WLM-Mixed	14	16	15			13	10	96	50	384	18.72	530.5 (**1.03×**)
K^{2}RED	14	16	15			15	12	96	50	434	16.56	514.18 (**1.00×**)
K^{2}RED-Shift	14	16	15	ρ_A	✓	20	14	64	50	303	23.76	824.72 (**1.60×**)
Montgomery-Shift	14	16	15	ρ_B	✓	21	11	64	50	370	19.44	657.52 (**1.28×**)
WLM	16	16	15			13	11	96	65	416	78.73	2233.78 (**1.01×**)
WLM-Mixed	16	16	15			13	9	96	65	400	82.01	2275.68 (**1.03×**)
K^{2}RED	16	16	15			14	11	96	65	434	75.45	2214.43 (**1.00×**)
K^{2}RED-Shift	16	16	15	ρ_A	✓	19	14	64	65	370	88.57	2973.24 (**1.34×**)
Montgomery-Shift	16	16	15	ρ_B	✓	17	10	64	65	400	82.01	2403.14 (**1.09×**)

BRAM is not included in ATP computation as it is equivalent for all designs.

Table 5: NTT Implementation results for $\log q = 32$ **with different reductions.**

set $\omega = \log n + 1$ which is necessary for finding a primitive $2n$-th root of unity. For WLM-Mixed, we set $\log q_h = 17$ ($\omega = 47$) as $\Gamma_B = 17$. As a result, the reduction requires only 3 DSPs while a sufficient number of primes are available as discussed in Section 3.4. For the 32-bit case, we set $\log q_h = 15$ to cover ring dimensions up to $n = 2^{16}$. The WLM-Mixed implementation also supports $\log q_h = 16$, but we skip it as the ATP results would be theoretically very similar to the previous case, with no effective difference in the number of primes.

For K^{2}RED, we employ a classical configuration where $\log q_h = 32$ for 64-bit as well as a DSP-optimized parameter selection where $\log q_h = 26$. With the latter, the number of DSPs is reduced to 6 from 8. Note that, $\log q_h = 26 \leq \Gamma_A$ and $\lceil(\log q - \log q_h), /\Gamma_B\rceil = 3$, leading to 3 DSP multiplications for Line 2 and Line 4 of Algorithm 3. For the 32-bit case, we set $\log q_h = 15$ which corresponds to $n = 2^{16}$ and $\log q_h = 16$ to cover $n \leq 2^{15}$. Recall that $\log q_h$ must satisfy $\log q_h \leq \beta/2$ with respect to the definition of Proth primes.

For the K^{2}RED-Shift and Montgomery-Shift, we set $\log q_h = 17$ and $\log q_h = 32$ for 64-bit. Recall from Section 3 that the cost of shift operations as well as the number of available primes for both algorithms directly depends on $\log q_h$. For the 32-bit case, same with the K^{2}RED, we set $\log q_h = 15$, which corresponds to $n = 2^{16}$, and $\log q_h = 16$ to cover $n \leq 2^{15}$. We include both Proth-$2l$ and Proth-$3l$ primes, as it also leads to a trade-off between the available primes and resource consumption. Recall from Table 1 that the number of available primes is a crucial factor for the RNS representation.

5.3 Standalone Implementation Evaluations

Next, we evaluate the resource-efficiency of implemented reduction algorithms in a standalone manner. Table 3 and Table 4 presents the ATP results for DSP-based and shift-add-based approaches, respectively. We report the maximum frequency that can be implemented using Vivado. Observe that the proposed WLM-Mixed leads to the

Reduction	$\log n$	PE	$\log q_h$	pipe.	l_3	LUT	FF	DSP	BRAM	Freq.	Lat.	ATP $\cdot 10^{-3}$
				conf.	en.	$\cdot 10^{-3}$	$\cdot 10^{-3}$			(MHz)	(μs)	
WLM	12	16	51			34	32	352	34	333	4.71	402.81 (**1.35×**)
WLM-Mixed	12	16	17			28	22	240	34	322	4.85	310.58 (**1.04×**)
K^{2}RED	12	16	26			30	22	288	34	333	4.70	330.71 (**1.11×**)
K^{2}RED-Shift	12	16	17	ρ_A	✗	38	27	192	34	333	4.70	334.79 (**1.12×**)
Montgomery-Shift	12	16	17	ρ_B	✗	35	22	192	34	344	4.54	299.09 (**1.00×**)
WLM	12	32	51			67	61	704	65	270	2.97	502.89 (**1.37×**)
WLM-Mixed	12	32	17			57	43	480	65	270	2.95	376.2 (**1.03×**)
K^{2}RED	12	32	26			60	43	576	65	263	3.03	421.92 (**1.15×**)
K^{2}RED-Shift	12	32	17	ρ_A	✗	75	53	384	65	263	3.03	427.1 (**1.16×**)
Montgomery-Shift	12	32	17	ρ_B	✗	70	44	384	65	285	2.79	366.71 (**1.00×**)
WLM	12	64	51			139	123	1408	129	238	1.76	601.5 (**1.36×**)
WLM-Mixed	12	64	17			120	86	960	129	238	1.73	450.55 (**1.02×**)
K^{2}RED	12	64	26			121	87	1152	129	250	1.65	463.03 (**1.04×**)
K^{2}RED-Shift	12	64	17	ρ_A	✗	155	107	768	129	250	1.65	471.71 (**1.06×**)
Montgomery-Shift	12	64	17	ρ_B	✗	142	86	768	129	243	1.69	443.63 (**1.00×**)
WLM	14	16	49			36	33	352	113.5	344	20.90	1849.13 (**1.22×**)
WLM-Mixed	14	16	17			31	24	240	113.5	322	22.32	1512.35 (**1.00×**)
K^{2}RED	14	16	26			32	24	288	113.5	333	21.60	1601.42 (**1.05×**)
K^{2}RED-Shift	14	16	32	ρ_A	✗	38	29	192	113.5	303	23.76	1720.62 (**1.37×**)
Montgomery-Shift	14	16	32	ρ_B	✗	41	23	192	113.5	285	25.20	1819.62 (**1.20×**)
WLM	16	16	47			31	30	320	129	333	98.43	7759.74 (**1.16×**)
WLM-Mixed	16	16	17			30	22	240	129	322	101.70	6664.72 (**1.00×**)
K^{2}RED	16	16	26			30	22	288	129	333	98.42	6945.1 (**1.04×**)
K^{2}RED-Shift	16	16	17	ρ_A	✓	44	30	192	129	333	98.42	7812.04 (**1.17×**)
Montgomery-Shift	16	16	17	ρ_B	✓	41	22	192	129	303	108.25	7777.88 (**1.16×**)

BRAM is not included in ATP computation as it is equivalent for all designs.

Table 6: NTT Implementation results for $\log q = 64$ **with different reductions.**

lowest ATP scores for the 64-bit class, as it is designed for minimizing the number of DSP multiplications. It achieves 2.45× and 1.39× lower ATP compared to WLM and K^{2}RED. On the other hand, for 32-bit class, K^{2}RED and WLM-Mixed leads to comparable results while K^{2}RED slightly performs better. The performance of WLM is aligned with these two algorithms for $\log q_h = 15$ and $\log q_h = 16$. However, with $\log q_h \geq 17$, the number of iterations for WLM increases to 3 (see Algorithm 2), increasing its ATP.

It is worth noting that while the shift-add methods, Montgomery-Shift and K^{2}RED-Shift, result in higher ATPs, they can be advantageous in scenarios where the system has a limited number of available DSPs and the implementer prefers to trade DSPs for LUTs. The ATP performances of Montgomery-Shift and K^{2}RED-Shift are comparable.

Table 3 and Table 4 also presents the maximum supported ring dimension n for each algorithm and parameter, which is particularly important for Montgomery-Shift and K^{2}RED-Shift to minimize the hardware cost for the desired n. It is computed based on the number of available primes considering the RNS representation (see Table 1) and the availability of the $2n$-th root of unity (see Section 2.2).

5.4 NTT Evaluations

We also evaluated the reduction algorithms within an NTT architecture. Specifically, we provide ATP results from the NTT implementation using different modular reduction algorithms inside the butterfly modules. Table 5 and Table 6 presents the results for $\log q = 32$ and $\log q = 64$, respectively. The implementations are performed using various numbers of processing elements (PEs), n and q. Note that the number of PEs defines the number of parallel executed butterfly circuits. For algorithm-specific configurations such as $\log q_h$, using Proth-$2l$ or Proth-$3l$, or the pipeline configurations, we selected the best set from Table 3 and Table 4. For instance, for Montgomery-Shift $\log n = 12$ and $\log q = 64$, we employed Proth-$2l$ and $\log q_h = 17$ as it is the best configuration regarding resource consumption based on Table 4. For Montgomery-Shift, we opted for the ρ_B pipeline configuration. Although ρ_A yielded better results,

Work	Reduction	Device	$\log n$	$\log q$	PE	LUT	FF	DSP	BRAM	Freq.	Lat.	ATP
						$\cdot 10^{-3}$	$\cdot 10^{-3}$			(MHz)	cc / μs	$\cdot 10^{-3}$
[19]	WLM	VC707	12	32	-	80	-	952	325.5	200	- / 0.18	49.5
[7]	Barrett	ZCU106	12	32	-	3.3	1.5	42	29.5	180	- / 136.58	2335.5
[21]	Montgomery	Virtex-7	12	24	16	14.6	6.5	80	12	121	1543 / 12.8	377
[15]	Barrett	Virtex-7	12	32	8	6.8	6.4	88	24	228	3081 / 13.5	351
[24]	Montgomery	XCVU37P	12	28	8	7.9	3.9	32	24	272	- / 11.34	229.6
this work	WLM-Mixed	XCU280	12	32	16	14.9	9.6	96	17	434	1565 / 3.6	124.14
[14]	eDARM	XCVU37P	12	60	-	74.5	61.4	288	697.5	250	951 / 3.8	1304.4
[17]	WLM	Virtex-7	12	60	32	99.3	-	992	176	125	972 / 7.7	1935
[27]	K^{2}RED	Virtex-7	12	60	-	17	11	286	24.5	150	- / 27.5	1607.4
this work	Montgomery-Shift	XCU280	12	64	16	35.5	22.5	192	34	344	1564 / 4.54	141.78
[14]	eDARM	XCVU37P	14	60	-	74.5	61.4	288	697.5	250	4340 / 17.3	5938.2
[23]	Barrett	XCU250	14	60	-	36.4	3.6	336	60	200	- / 29.3	2631.1
[10]	eDARM	XCU250	14	60	-	99.7	56.6	384	135	388	8267 / 21.8	4510.4
this work	WLM-Mixed	XCU280	14	64	16	31	24	240	113.5	322	7201 / 22.32	2272.44
[8]	Barrett	XCU250	16	60	32	148.5	90.9	564	137	200	536832 / 2684	782251.8
[14]	eDARM	XCVU37P	16	60	-	74.5	61.4	288	697.5	264	16628 / 66.51	22829.6
[16]	Barrett	XCU250	16	60	-	274.4	83.1	630	1084	250	16552 / 66.2	46614.7
this work	WLM-Mixed	XCU280	16	64	16	30.4	22.4	240	129	322	32805 / 101.70	10600.33

Table 7: Comparison to NTT implementations from Literature

the performance of the two configurations was comparable. For WLM, we always set $\log q_h = \log q - \log n - 1$, assuming it supports all NTT-friendly primes for a given n. Similar to the previous section, we report the smallest frequency that can be implemented for each design. Observe that the NTT implementation statistics mainly align with our initial observation from the previous section. Notably, Montgomery-Shift exhibits low ATP scores for $\log n = 12$. This is primarily due to the reduced hardware cost associated with this ring dimension, as fewer primes are required. Additionally, NTT with Montgomery-Shift achieves higher operating frequencies compared to the DSP-based reductions, WLM-Mixed and K^{2}RED. For ring dimensions $\log n = 14$ and $\log n = 16$ multiplication-based methods become the superior choice as WLM-Mixed leads to the lowest ATP scores for $\log q = 64$.

5.5 Comparison with Literature

We also compare our results with those reported in the literature, as shown in Table 7. The main goal in presenting this comparison is to demonstrate that our NTT architecture, which we benchmark the reduction algorithms, is comparable with existing designs in the literature, rather than claiming superiority. The reductions used in the referenced works generally employ the reduction techniques studied in this paper. Additionally, [10, 14] utilize a specialized reduction method called eDARM. This technique uses primes of the form $q = 2^\beta - \delta$, where $\delta \leq 2^{\lfloor 2\beta/3 \rfloor}$ and β is even. To illustrate the efficiency of the proposed WLM-Mixed method compared to eDARM, consider the case where $\log q = 64$ and $\log n = 16$, resulting in $\delta \leq 2^{24}$. The eDARM method requires 3 $\log \delta' \times \beta$-bit multiplications, where $\delta' \leq 2^{24-\log n-1} = 2^7$ that takes the advantage of the fact that lower $\log n + 1$ bits of q is fixed for NTT friendly primes. This can be implemented using 9 DSPs, as $\lceil \log \delta' / \Gamma_B \rceil \times \lceil \beta / \Gamma_A \rceil = 3$.

6 Conclusion

In this paper, we studied the resource efficiency of various modular reduction algorithms, targeting NTT implementations on FPGA. Particularly, we explored the optimization opportunities through the trade-offs between the number of primes available in special forms and the hardware costs. Our proposed WLM-Mixed outperformed WLM [19] and K^{2}RED [3] significantly in 64-bit modulus, as it only requires 3 DSP multiplications by design. On the other hand, we introduced multiplication-free variants of the Naive Montgomery algorithm and K^{2}RED, termed Montgomery-Shift and K^{2}RED-Shift, respectively. These algorithms demonstrated low ATP scores for small ring dimensions, such as $n = 2^{12}$. This is because reducing the number of required primes directly impacts the hardware cost. Our study reveals that significant improvements are possible by reducing the number of free bits in the modulus while leaving sufficient number of free bits to meet the requirements of RNS representation. Although we concentrated on FPGA implementations, our methodology applies to ASICs as well. We leave the in-depth analysis of ASIC implementations of our proposed algorithms for future work.

References

[1] Martin R Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology* 9, 3 (2015), 169–203.

[2] Paul Barrett. 1986. 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, 311–323.

[3] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *2021 IEEE 28th symposium on computer arithmetic*

(ARITH). IEEE, 94–101.

[4] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual cryptology conference*. Springer, 868–886.

[5] Ilaria Chilotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2019. TFHE: Fast Fully Homomorphic Encryption over the Torus. In *Journal of Cryptology*. Springer.

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

[7] Stefano Di Matteo, Matteo Lo Gerfo, and Sergio Saponara. 2023. VLSI Design and FPGA implementation of an NTT hardware accelerator for Homomorphic seal-embedded library. *IEEE Access* (2023).

[8] Phap Duong-Ngoc, Sunmin Kwon, Donghoon Yoo, and Hanho Lee. 2022. Area-efficient number theoretic transform architecture for homomorphic encryption. *IEEE Transactions on Circuits and Systems I: Regular Papers* 70, 3 (2022), 1270–1283.

[9] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[10] Yue Geng, Xiao Hu, and Zhongfeng Wang. 2024. GS-MDC: High-Speed and Area-Efficient Number Theoretic Transform Design. *IEEE Transactions on Circuits and Systems II: Express Briefs* (2024).

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

[12] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[13] M. Kim J. H. Cheon, A. Kim and Y. Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Asiacrypt 2017*. Springer, 409–437.

[14] Stefanus Kurniawan, Phap Duong-Ngoc, and Hanho Lee. 2023. Configurable memory-based NTT architecture for homomorphic encryption. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 10 (2023), 3942–3946.

[15] Si-Huang Liu, Chia-Yi Kuo, Yan-Nan Mo, and Tao Su. 2023. An Area-Efficient, Conflict-Free, and Configurable Architecture for Accelerating NTT/INTT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023).

[16] Rella Mareta, Ardianto Satriawan, Phap Ngoc Duong, and Hanho Lee. 2024. A Bootstrapping-Capable Configurable NTT Architecture for Fully Homomorphic Encryption. *IEEE Access* (2024).

[17] Ahmet Can Mert, Emre Karabulut, Erdinç Öztürk, Erkay Savaş, and Aydin Aysu. 2020. An extensive study of flexible design methods for the number theoretic transform. *IEEE Trans. Comput.* 71, 11 (2020), 2829–2843.

[18] Ahmet Can Mert, Emre Karabulut, Erdinç Öztürk, Erkay Savaş, Michela Becchi, and Aydin Aysu. 2020. A flexible and scalable NTT hardware: Applications from homomorphically encrypted deep learning to post-quantum cryptography. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 346–351.

[19] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. 2019. Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 2 (2019), 353–362.

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

[21] Jianan Mu, Yi Ren, Wen Wang, Yizhong Hu, Shuai Chen, Chip-Hong Chang, Junfeng Fan, Jing Ye, Yuan Cao, Huawei Li, et al. 2022. Scalable and conflict-free NTT hardware accelerator design: Methodology, proof and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[22] Dinh Nhat Nguyen, Hoai Luan Pham, Vu Trung Duong Le, Duc Khai Lam, Thi Hong Tran, Yasuhiko Nakashima, et al. 2024. HyperNTT: A Fast and Accurate NTT/INTT Accelerator with Multi-Level Pipelining and an Improved K2-RED Module. In *2024 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*. IEEE, 1–6.

[23] Tuy Tan Nguyen, Jisu Kim, and Hanho Lee. 2023. CKKS-Based Homomorphic Encryption Architecture using Parallel NTT Multiplier. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.

[24] Rogério Paludo and Leonel Sousa. 2022. NTT architecture for a linux-ready RISC-V fully-homomorphic encryption accelerator. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 7 (2022), 2669–2682.

[25] Thomas Plantard. 2021. Efficient Word Size Modular Arithmetic. *IEEE Transactions on Emerging Topics in Computing* 9, 3 (2021), 1506–1518. https://doi.org/10.1109/TETC.2021.3073475

[26] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)* 56, 6 (2009), 1–40.

[27] Zewen Ye, Ray CC Cheung, and Kejie Huang. 2022. PipeNTT: A pipelined number theoretic transform architecture. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 10 (2022), 4068–4072.