Efficient Modular Multiplication Hardware for Number Theoretic Transform on FPGA

Tolun Tosun, Selim Kırbıyık♦, Emre Koçer, Erkay Savaş, Ersin Alaybeyoğlu
Faculty of Engineering and Natural Sciences
Sabancı University
Istanbul, Turkey

***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 [1] and K^{2}RED [2]. 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 [3], 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.**

***Index Terms*—Modular Reduction, FPGA, Montgomery, K^{2}RED, DSP, FHE, NTT**

I. 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 [4]. With FHE, data can remain secure even when processed by third-party data centers.

Several FHE algorithms exist in the literature such as BFV [5], [6], CKKS [7] and TFHE [8]. Existing FHE algorithms are lattice-based schmes 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 2^{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 [3], Barrett [9], Plantard [10], and Montgomery-based methods like WLM [11], K^{2}RED. 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 and demonstrate its application in the context of FHE for the first time in the literature. Our study reveals that K^{2}RED can be effectively utilized in FHE implementations.
- We propose a novel variant of the WLM reduction algorithm [1], 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 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 needs to be used for different goals. An example analysis provided in the paper is the tradeoff between DSP and distributed logic use for a given algorithm. (Changed the sentence for readibility)

♦: Tolun Tosun and Selim Kırbıyık declares equal contribution

II. Background

A. 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., $\mathbf{a}_i$.
- 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 $\boldsymbol{a}_i$.

B. 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:

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

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 $\psi^n \equiv -1 \pmod{q}$. The forward NTT corresponds to the evaluation $\hat{\mathbf{a}}[i] = \boldsymbol{a}(\psi^{2i+1})$ for every coefficient $i < n$. The NTT can be efficiently implemented using butterfly circuits. Forward NTT is usually implemented with *Cooley-Tukey* (CT) [12] butterflies while the backward NTT is implemented with *Gentleman-Sande* (GS) [13] butterflies. For two coefficients $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$, the CT butterfly is defined as follows:

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

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 $O(n \log n)$.

C. Residue Number System (RNS)

FHE applications require performing large integer arithmetic due to security needs of the underlying computationally expensive Learning With Errors (LWE) [14] 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:

$$ab = \{a_i\}_i^{\lambda-1} \odot \{b_i\}_i^{\lambda-1} = \{a_i b_i \mod q_i\}_i^{\lambda-1} \quad (3)$$

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

TABLE I: The relationship between $\tilde{q}$ and n for 128-bit security [15], 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.

D. Modular Reduction Algorithms

1) Montgomery Reduction: Montgomery reduction [3] 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 [3]

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) 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 $\omega = 2 \log n$ where $q = 1 \pmod{\omega}$. As a result, the multiplication by q' is eliminated. The word-level Montgomery reduction for NTT friendly primes [1] is detailed in Algorithm 2.

3) K²RED: K²RED [2] is a reduction algorithm originally developed for Crystals-Kyber. K²RED requires the modulus to be a proth prime, which are in the form of $q = q_h 2^\omega + 1$

Algorithm 2 WLM; Word-Level Montgomery Reduction for NTT-friendly primes [1]

Input: modulus $q < 2^{\beta}$, word-size ω 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** $i = 0 \rightarrow \lceil \frac{\beta}{\omega} \rceil$ **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

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 [16]. Either the input operand needs to be pre-processed or the result needs to be post-processed to correct the $2^{-2\omega}$ term. Adaption of K^{2}RED in FHE context is unexplored.

Algorithm 3 K^{2}RED [2]

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

4) Barrett Reduction: Barrett reduction [9] 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 $2\beta \times \beta$. Also, the output of the Barrett reduction does not involve a constant factor compared to Montgomery reduction.

Algorithm 4 Barrett Reduction [9]

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	$3l$	421
64	17	$3l$	44
64	32	$2l$	16
64	17	$2l$	5
32	16	$3l$	80
32	15	$3l$	64
32	16	$2l$	7
32	15	$2l$	6

TABLE II: Number of proth-l primes for different bit-widths.

5) Plantard Reduction: Plantard reduction [10] is a relatively newer reduction algorithm, detailed in Algorithm 4. 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 $(a)(bq')$ 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 [10]

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

III. Proposed Modular Reduction Algorithms

A. Proth-l Primes

In this section, we present Proth-l primes, which are a subset of Proth primes. Recall that Proth primes are in the form of $q_h\omega + 1$ 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 $0 \leq l_2 \leq l_1 < \beta - 1$.

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 $0 \leq l_2 \leq l_1 < \beta - 1$ and $0 \leq l_3 < \beta - 1$.

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 II, 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 64 such primes are found and $64 \times 32 > 1800$. Also, recall from Section II-B 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.

B. 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 $q_h 2^{\omega} - 1$:

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

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

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

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 2 barrel. 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 is 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 III-A 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 $\log l_i = x$.

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.

C. K^{2}RED with Barrel Shifters

In this section, we present 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. 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-3l primes by removing l_3 terms. Using Proth-3l provides additional primes at the cost of increased area. Algorithm 7 requires 6 barrel shifters for Proth-3l primes and 4 barrel shifters for Proth-2l primes.

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

D. 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 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 Γ_A, Γ_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: $\quad \delta_i \leftarrow \omega - \omega_i$
5: $\quad t_h \leftarrow t \gg \omega_i$, $t_l \leftarrow t \pmod{2^{\omega_i}}$
6: $\quad t' \leftarrow -t_l \pmod{2^{\omega_i}}$
7: $\quad c \leftarrow t'[\omega_i - 1] \vee t[\omega_i - 1]$
8: $\quad t \leftarrow t_h + \big((q_h t_l) \ll \delta_i\big) + c$
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 3. the number of DSP multiplications is significantly greater for the classical word-level Montgomery reduction (Algorithm 2) using general NTT-friendly primes. Consider ther case $n = 2^{16}$ therefore $\omega = 17$. Then, 4 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.

IV. 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), WLM-Mixed (Algorithm 8), K^{2}RED (Algorithm 3) from the literature and provide the architectural details in this section. We do not implement naive Montgomery reduction, Barrett reduction and Plantard reduction as these algorithm are theoretically more costly than the implemented algorithms, as detailed in Section V-A.

A. Architecture

Evaluated algorithms are implemented with Verilog HDL. The implementations are highly parametric such that the parameters $\log q$, $\log q_h$, or enabling l_3 for shift-add designs, can be provided in design-time to generate the corresponding hardware.

In Algorithm 2, Algorithm 3, and Algorithm 8, 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 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 [1].

The implementations are pipelined. Note that the pipeline steps are not illustrated in Figure 1 and Figure 2. Specifically, we place 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. In a similar manner, 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 barrel shifters and additions are performed in different ccs while these are performed in single cc for ρ_B. This demonstrates thetrates the 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. 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.

B. Parameter Selection

In this section, we explain the parameter selection for each algorithm. Since WLM supports general NTT-friendly primes, we implement it from ω ranging from 13 to 17, which

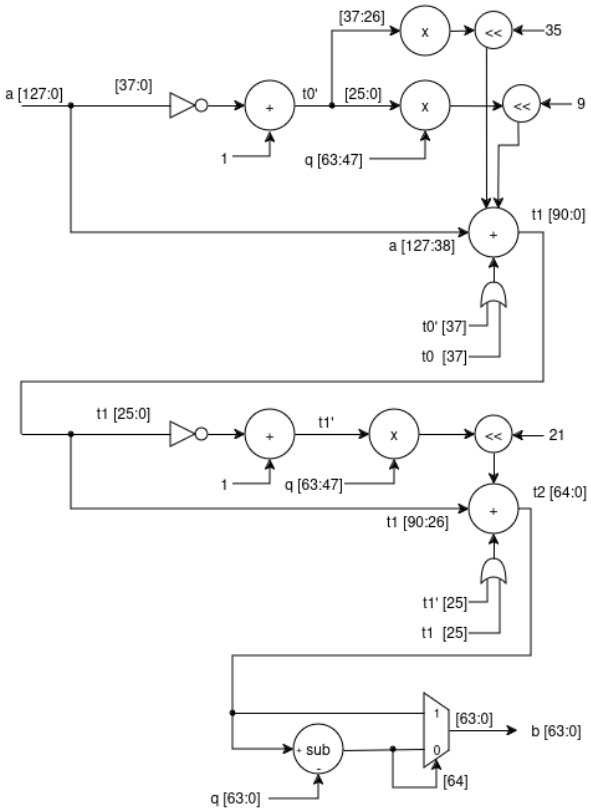


Fig. 1: WLM-Mixed Architecture for $\log q = 64$ and $\log q_h = 17$. DSP operand sizes are $\Gamma_A = 26$, $\Gamma_B = 17$.

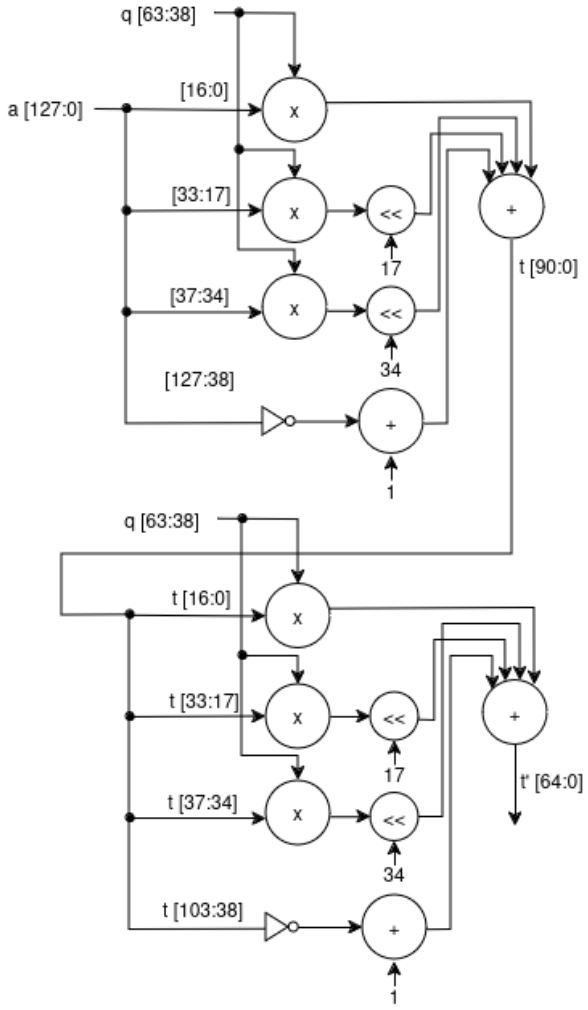


Fig. 2: K^{2}RED Architecture for $\log q = 64$ and $\log q_h = 26$. DSP operand sizes are $\Gamma_A = 26$, $\Gamma_B = 17$. Correction step in Lines 5-7 of Algorithm 3 is skipped.

corresponds to ring sizes $n = 2^{12}$ to $n = 2^{16}$, respectively. Recall that finding a primitive $2n$-th root of unity requires $2 \log n \geq \omega$. 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 III-D. For 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 implement a classical configuration where $\log q_h = 32$ for 64-bit as well as a DSP-optimized parameter selection where $\log q_h = 38$. With the latter, the number of DSPs is reduces to 6 from 8. Note that, $\log q - \log q_h = 26 \leq \Gamma_A$ and $\lceil 64/\Gamma_B \rceil = 3$, leading to 3 DSP multiplications for Line 2 and Line 4 of Algorithm 3. For the 32-bit case, we include $\log q_h = 15$ which corresponds to $n = 2^{16}$ and $\log q_h = 16$ to cover $\log n \leq 16$. Recall that $\log q_h$ must be satisfy $\log q_h \geq \beta/2$ with respect to the definition of Proth primes.

For the K^{2}RED-Shift and Montgomery-Shift, we implement $\log q_h = 17$ and $\log q_h = 32$ for 64-bit. Recall from Section III 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 include $\log q_h = 15$, which corresponds to $n = 2^{16}$, and $\log q_h = 16$ to cover $\log n \leq 16$. Our shift-add based reduction implementations support 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 I that the number of available primes is a crucial factor for the RNS representation.

V. 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 4). 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 [17]. ATP is calculated as

Average Latency $(\mu s) \times (\text{LUT} + \text{FF}/2 + 100 \times \text{DSP} + 300 \times \text{BRAM})$. We target the AMD-Xilinx Alveo U280[1] (XCU280) FPGA for our evaluations, using Vivado 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 ($\Gamma_A = 26$, $\Gamma_B = 17$).

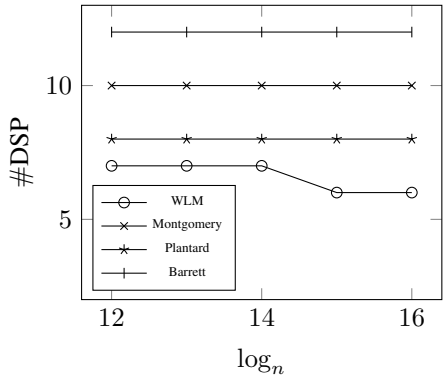
A. Theoretical Analysis of Montgomery, Barrett and Plantard

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 28 DSPs in total. The DSP usage is counted in the same manner for the compared algorithms. Note that we did not include K^{2}RED and WLM-Mixed in the theoretical analysis as provide a practical anaylsis in the next section. This section aims to provide a rationale for comparing WLM-Mixed with WLM.

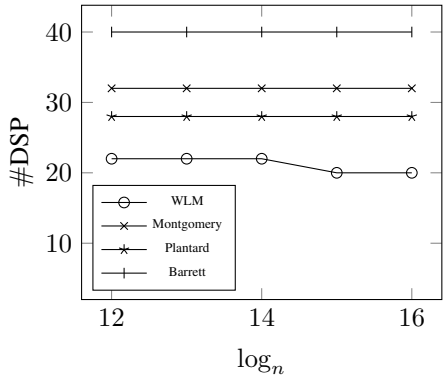
B. Standalone Implementation Evaluations

Next, we evaluate the resource-efficiency of implemented reduction algorithms in a standalone manner. Table III presents the ATP results for both 32-bit and 64-bit. Observe that the proposed WLM-Mixed leads to the lowest ATP scores for the 64-bit class, as it is designed for minimizing the number of DSP multiplications. It achieves $2.45\times$ and $1.39\times$ lower ATP compared 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 also align 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.

[1]https://www.xilinx.com/products/boards-and-kits/alveo/u280.html
[2]https://www.xilinx.com/products/design-tools/vivado.html
[3]https://docs.amd.com/r/en-US/ug958-vivado-sysgen-ref/DSP48E2



(a) $\log q = 32$



(b) $\log q = 64$

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

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 III also presents the maximum supported ring dimension n for each algorithm and parameter, which is particularly important for Montgomery-Shift and K^{2}RED using Proth-l primes. It is computed based on the number of available primes considering the RNS representation (see Table I) and the availability of the $2n$-th root of unity (see Section II-B).

VI. 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 [1] and K^{2}RED [2] significantly in 64-bit modulus, as it only requires 3 DSP multiplications by design. Therefore, our study reveals that

	$\log q$	$\log q_h$	# 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
pipe. conf.			l_3 en.						
K^2RED-Shift (Algorithm 7)									
ρ_A	64	17	✓	44	16	1259	979	0	476	3.67
ρ_B	64	17	✓	44	16	1295	409	0	370	4.04
ρ_B	64	17	✗	5	13	1079	397	0	454	2.81
ρ_A	64	32	✓	421	≥ 16	1287	1071	0	500	3.64
ρ_B	64	32	✓	421	≥ 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	✓	64	16	611	537	0	588	1.49
ρ_B	32	15	✓	64	16	617	245	0	434	1.7
ρ_B	32	15	✗	6	13	510	232	0	555	1.12
ρ_A	32	16	✓	64	16	595	537	0	555	1.55
ρ_B	32	16	✓	80	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	✓	44	16	1114	954	0	526	3.02
ρ_B	64	47	✓	44	16	1002	591	0	416	3.11
ρ_B	64	47	✗	5	13	671	583	0	416	2.31
ρ_A	64	32	✓	421	≥ 16	1545	1101	0	500	4.19
ρ_B	64	32	✓	421	≥ 16	1391	642	0	416	4.1
ρ_B	64	32	✗	16	15	945	632	0	416	3.02
ρ_A	32	17	✓	64	16	665	558	0	625	1.49
ρ_B	32	17	✓	64	16	603	329	0	434	1.76
ρ_B	32	17	✗	6	13	411	321	0	476	1.2
ρ_A	32	16	✓	80	16	667	564	0	588	1.61
ρ_B	32	16	✓	80	16	639	332	0	454	1.77
ρ_B	32	16	✗	7	13	420	324	0	476	1.22

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

TABLE III: Implementation results for different reduction Algorithms.

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. On the other hand, we presented multiplication-free variants of the Naive Montgomery algorithm and K^2RED, namely Montgomery-Shift and K^2RED-Shift. To construct modular reduction through shifts and adds, we presented a special subclass of Proth primes, which we referred to as Proth-l primes. These algorithms offer opportunities for further trading DSPs with programmable logic considering target platforms with low number of multiplication resources. 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] A. C. Mert, E. Öztürk, and E. Savaş, "Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 353–362, 2019.

[2] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed ntt-based polynomial multiplication accelerator for post-quantum cryptography," in *2021 IEEE 28th symposium on computer arithmetic (ARITH)*, pp. 94–101, IEEE, 2021.

[3] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[4] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

[5] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Annual cryptology conference*, pp. 868–886, Springer, 2012.

[6] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[7] M. K. J. H. Cheon, A. Kim and Y. Song., "Homomorphic encryption for arithmetic of approximate numbers," in *Asiacrypt 2017*, pp. 409–437, Springer, 2017.

[8] I. Chilotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," in *Journal of Cryptology*, Springer, 2019.

[9] P. 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*, pp. 311–323, Springer, 1986.

[10] T. Plantard, "Efficient word size modular arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[11] A. C. Mert, E. Karabulut, E. Öztürk, E. Savaş, M. Becchi, and A. Aysu, "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)*, pp. 346–351, IEEE, 2020.

[12] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[13] W. M. Gentleman and G. Sande, "Fast fourier transforms: for fun and profit," in *Proceedings of the November 7-10, 1966, fall joint computer conference*, pp. 563–578, 1966.

[14] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[15] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[16] D. N. Nguyen, H. L. Pham, V. T. D. Le, D. K. Lam, T. H. Tran, Y. Nakashima, *et al.*, "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)*, pp. 1–6, IEEE, 2024.

[17] Z. Ye, R. C. Cheung, and K. Huang, "Pipentt: A pipelined number theoretic transform architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 10, pp. 4068–4072, 2022.