



Deposited via The University of Sheffield.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/245360/>

Version: Published Version

Article:

Alhassani, A. and Benaissa, M. (2026) ML-KEM (CRYSTALS-Kyber) on FPGA using the residue number system. *Cryptography*, 10 (4). 47. ISSN: 2410-387X

<https://doi.org/10.3390/cryptography10040047>

Reuse

This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here:

<https://creativecommons.org/licenses/>

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Article

ML-KEM (CRYSTALS-Kyber) on FPGA Using the Residue Number System

Abdullah Alhassani *  and Mohammed Benaissa

School of Electrical and Electronic Engineering, The University of Sheffield, Sheffield S1 3JD, UK;
m.benaissa@sheffield.ac.uk

* Correspondence: ahmalhassani1@sheffield.ac.uk

Abstract

The NIST standardisation process for Post-Quantum Cryptography (PQC) has nominated the CRYSTALS-Kyber Key-Encapsulation Mechanism (KEM) scheme as the primary key establishment method. The algorithm was renamed as the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM). This work proposes an efficient design for ML-KEM on FPGA with built-in side-channel attack (SCA) protection. The design is based on combining two methodologies: the Residue Number System (RNS) arithmetic and the look-up tables implementation. At the arithmetic level in the number-theoretic transform (NTT) computation of the polynomial multiplication, the operations are spread across the RNS channels, and these computations are implemented using look-up tables. The use of look-up tables resulted in low-latency RNS implementation and higher performance. The proposed design, implemented on Xilinx Artix-7 FPGA, shows higher performance with a reasonable increase in area, whilst the experimental TVLA results demonstrate the design's SCA protection advantages.

Keywords: PQC; ML-KEM; CRYSTALS-Kyber; FPGA; RNS; SCA; NTT; BCM; TVLA

1. Introduction

The current development of quantum computing poses a serious threat to current public key protocols such as RSA and Elliptic Curve Cryptography (ECC). Shor's algorithm [1] running on a powerful quantum computer can break these cryptography schemes in polynomial time. Therefore, the National Institute of Standards and Technology (NIST) in the United States called for a Post-Quantum Cryptography (PQC) competition in 2016 [2], aiming to develop standards for PQC algorithms that resist both classical and quantum attacks. The primary algorithms were announced in 2024, and CRYSTALS-Kyber [3] was chosen for the KEM category. The institute renamed the algorithm as Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) [4].

ML-KEM is a lattice-based cryptography scheme, the security assumption of which is based on solving the hard problem of Module Learning With Error (MLWE) [5]. The scheme uses an NTT-based algorithm to perform polynomial multiplication. The adoption of NTT accelerates polynomial multiplication over quasilinear time compared to time-domain algorithms such as the schoolbook, which require quadratic time.

Research on side-channel analysis (SCA) countermeasures has emerged as an important aspect of the practical deployment of ML-KEM. Indeed, SCAs on ML-KEM have been demonstrated on both software [6–8] and hardware [9,10] implementations; SCA countermeasures for ML-KEM, including masking, have also been reported [11–13] but have been



Received: 23 May 2026

Revised: 4 July 2026

Accepted: 9 July 2026

Published: 10 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

shown in some cases to remain vulnerable and experience limitations from performance and resource overheads [14,15]. Therefore, there is a need to explore alternative approaches to SCA protection with reduced overheads.

The Residue Number System (RNS) has been proven to be a useful tool for side-channel attack countermeasures in RSA and ECC [16–18]. RNS also provides a framework for efficient computation by breaking down large operations into smaller ones that can be computed in parallel, thereby reducing the critical path requirements of the computations and energy consumption. However, performing modular arithmetic in RNS is not straightforward due to the requirement for reduction. A Montgomery modular reduction method for performing modular reduction in RNS based on base extension was first proposed in [19] and later improved in [20]. While this method eliminates the need for the costly conversion to and from the weighted number system to perform the modular reduction, it is still not practical for low-latency and small operand applications, as it requires an additional base and further conversion between these bases.

This work further extends the RNS methodology proposed in [21] to implement the whole polynomial multiplication of ML-KEM in the RNS domain, as opposed to just the integer modular multiplication part. Firstly, an efficient RNS moduli set is used for ML-KEM to decompose the modular arithmetic of the NTT into smaller computations that run in parallel. Then, adopting a look-up table-based architecture, a configurable RNS butterfly unit (RNS-BU) that computes the complete polynomial multiplication is proposed. A hardware accelerator is constructed utilising the proposed RNS-BU to construct a complete RNS-based NTT multiplier for ML-KEM, thus enabling a secure and efficient low-latency implementation.

The FPGA platform is used to benchmark the design and quantify resource utilisation and performance. The post-place-and-route implemented design on the Xilinx Artix-7 FPGA consumes less than 0.5% of the FPGA LUT and FF resources, around 7% of the device's BRAMs, and does not use DSP slices. The design computes a complete polynomial multiplication in 7.2 μ s.

The advocated SCA protection properties of the proposed methodology are confirmed using experimental power measurements of the implemented design on FPGA during operations, followed by analysis using the widely accepted test vector leakage assessment (TVLA) technique. Compared with the baseline unprotected design, the TVLA results obtained show that the proposed design does not exhibit SCA leakage and hence provides improved SCA protection.

To the best of our knowledge, this is the first work to explore RNS for the whole polynomial multiplication of ML-KEM. By adopting a look-up table architecture design, the complexity of the RNS arithmetic computation is transferred into FPGA BRAMs, leading to a low-complexity design. In addition, we believe this is the first work to explore an SCA countermeasure for ML-KEM using a combination of RNS and a look-up table-based RNS implementation utilising FPGA BRAMs configured as ROM to experimentally evaluate its side-channel leakage characteristics. Compared with our previous work [21], which focused on an efficient RNS implementation of ML-KEM polynomial multiplication, the present work extends the architecture by introducing an RNS system combined with a BRAM-based lookup-table implementation that performs the entire ML-KEM polynomial multiplication in the RNS domain aimed at reducing side-channel leakage. In addition, the proposed architecture incorporates a configurable butterfly unit supporting NTT, INTT, and BCM operations, and includes an experimental TVLA-based evaluation to assess its side-channel leakage resilience.

The remainder of the paper is structured as follows: Section 2 presents the background of the ML-KEM protocol, the NTT-based polynomial multiplication method, and the RNS.

Section 3 describes our contributions in this work, including our approach and proposed design. Section 4 presents the implementation results, compares them with prior works, and evaluates the SCA leakage of the proposed design. The conclusion and directions for future work are presented in Section 5.

2. Background

2.1. Notation

The ring of integers $\{0, 1, \dots, q-1\}$ is represented by $\mathbb{Z}_q$ where arithmetic operations on the elements of the ring are reduced with the modulus q . The quotient ring of polynomials is denoted by $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ where coefficients of the polynomials are integers in $\mathbb{Z}_q$ and operations on polynomials are reduced by the cyclotomic polynomial $x^n + 1$. The degree of a polynomial in $\mathbb{R}_q$ is denoted by n and has a maximum degree of $n-1$.

Lowercase letters such as a denote a polynomial in the normal domain. The forward NTT of a polynomial is written as $\hat{a} \leftarrow NTT(a)$ and inverse of the NTT is defined as $a \leftarrow INTT(\hat{a})$, where $\hat{a}$ denotes the polynomial in the NTT domain.

Bold lowercase letters, such as $\mathbf{s}$ represent a k -dimensional polynomial vector, and bold uppercase letters, such as $\mathbf{A}$ denote a $k \times k$ dimensional polynomial matrix. The transpose operation of matrix $\mathbf{A}$ (or vector $\mathbf{a}$) is indicated by $\mathbf{A}^T$ (or $\mathbf{a}^T$). Integer multiplication is represented by $\cdot$, and $\circ$ indicates matrix multiplication. Multiplication in the NTT domain, which is known as Base Case Multiplication (BCM), and formally known as Coefficient-Wise Multiplication (CWM), is written as $*$.

2.2. ML-KEM

ML-KEM [4] is a lattice-based cryptography scheme constructed using module lattices and is based on the Learning with Error (LWE) hard problem. The scheme first establishes a public-key encryption (K-PKE) scheme that is secure against the indistinguishability under Chosen-Plaintext Attack (IND-CPA) model. The K-PKE consists of three algorithms: key generation, encryption, and decryption. Then, a modified Fujisaki–Okamoto (FO) transform [22] is used to transform the constructed PKE into a KEM that is secure against the indistinguishability under adaptive Chosen Ciphertext Attack (IND CCA2) model. The ML-KEM, as shown in Figure 1, consists of key generation, encapsulation, and decapsulation.

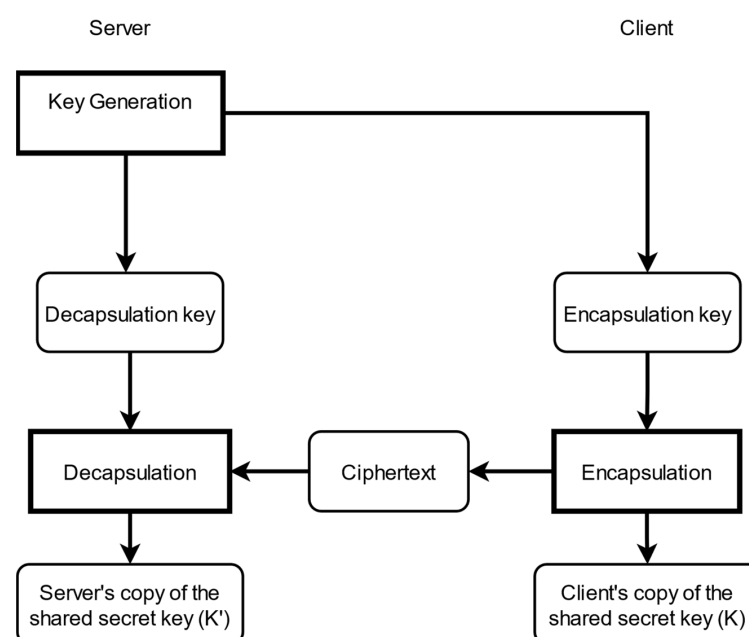


Figure 1. Key establishment using the KEM method (adapted from [4]).

ML-KEM uses polynomials in $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ to represent the scheme elements. The degree of the polynomials is selected as $n = 256$, and the modulus is $q = 3329$. ML-KEM has three versions: ML-KEM-512, ML-KEM-768, and ML-KEM-1024. These represent the three NIST security levels: 1, 3, and 5, respectively. The level of security can be adjusted according to the module lattice dimension k , taking the values of 2, 3, and 4 for the three security levels in order. A simplified version of the K-PKE algorithms is presented as follows, and for detailed specifications, please refer to the protocol standard presented in [4]:

- **K-PKE.KeyGen:** At the first side and using a random seed ρ , elements of the public matrix $\hat{A} \in \mathbb{R}_q^{k \times k}$ are sampled in the NTT domain from a uniform distribution. Next, using a centred binomial distribution (CBD) sampler, a secret s and an error $e \in \mathbb{R}_q^k$ are sampled, and then their NTT is computed. Finally, the public key is constructed as $pk = (\rho, \hat{t})$, where $\hat{t} = \hat{A} \circ \hat{s} + \hat{e}$, and the secret key is $sk = \hat{s}$.
- **K-PKE.Encrypt:** On the other side, using the same seed, the public matrix $\hat{A}$ is sampled. Next, using a CBD sampler, polynomials $r, e_1 \in \mathbb{R}_q^k$ and $e_2 \in \mathbb{R}_q$ are sampled, followed by the computation of the NTT for r . Then, a random message m is generated and encoded. Finally, the ciphertext is generated as $ct = (u, v)$, where $u = INTT(\hat{A}^T \circ \hat{r}) + e_1$ and $v = INTT(\hat{t}^T \circ \hat{r}) + e_2 + m$.
- **K-PKE.Decrypt:** On the first side, the message is recovered from the received ciphertext as $m = v - INTT(\hat{s}^T \circ NTT(u))$.

2.3. Polynomial Multiplication in ML-KEM

Polynomial multiplication is the most time-consuming operation in ML-KEM, where the NTT-based algorithm is utilised instead of the traditional time-domain multiplication. Using the NTT reduces the time complexity of the multiplication from n^2 to $O(n(\log n))$. The NTT is a generalisation of the Discrete Fourier Transform (DFT) and is defined over the ring $\mathbb{Z}_q$. The forward NTT transformation of a polynomial $a(x) = a_0 + a_1x + a_2x^2 \dots a_{n-1}x^{n-1}$ can be written as follows:

$$\hat{a}(x) = \sum_{j=0}^{n-1} a_j \omega_n^{ij} \text{ mod } q, i \in [0, n-1] \quad (1)$$

where the symbol ω , known as the twiddle factor constant, indicates the n -th root of unity that satisfies $\omega^n \equiv 1(\text{mod } q)$ and $\omega^i \neq 1(\text{mod } q) \forall i < n$. Also, the prime modulus q must satisfy the condition $q \equiv 1(\text{mod } n)$. Polynomials in the NTT domain can be converted back into the normal domain using the INTT transform, where $\omega^{-1}(\text{mod } q)$ is used instead of ω , and the result is scaled with n^{-1} .

Given that polynomial multiplication in ML-KEM is conducted within the ring $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, the utilisation of the negative wrapped convolution (NWC) allows for the elimination of the requirement to double the inputs and execute additional polynomial reduction by $x^n + 1$. Applying the NWC requires scaling the inputs by powers of the $2n$ -th root of unity; the outputs are scaled by powers of the inverse $2n$ -th root of unity. However, in [23], a new approach was introduced to compute polynomial multiplication using the NTT without scaling the inputs and outputs. Also, the new method does not necessitate the existence of the $2n$ -th root of unity; it only constrains the modulus to the n -th root of unity $q \equiv 1(\text{mod } n)$. The new variant is referred to as incomplete NTT and generates $\frac{n}{2}$ polynomials of degree 1. The NTT and INTT algorithms were adjusted accordingly, as outlined in Algorithms 1 and 2, respectively. In these algorithms, the function $br_7(k)$ represents the bit reversal operation of an unsigned 7-bit integer k . In addition, the NTT uses the Cooley–Tukey (CT) [24] butterfly structure, while the INTT utilises the Gentleman–Sande (GS) [25] butterfly as proposed in [26] to avoid bit-reversal operations.

Algorithm 1 ML-KEM NTT using CT butterfly

Input $a(x) \in \mathbb{R}_q$ in normal order, $\omega \in \mathbb{Z}_q$ and $n = 2^l$
Output $\hat{a}(x) \in \mathbb{R}_q$ in bit-reversed order

```

1:  $k = 1$ 
2: for ( $i = 1$ ;  $i \leq l - 1$ ;  $i = i + 1$ ) do
3:    $m = 2^{l-i}$ 
4:   for ( $s = 0$ ;  $s < n$ ;  $s = s + m$ ) do
5:     for ( $j = s$ ;  $j < s + m$ ;  $j = j + 1$ ) do
6:        $W = \omega^{br_7(k)} \bmod q$ 
7:        $X = a[j], Y = a[j + m]$ 
8:        $T = (W \cdot Y) \bmod q$ 
9:        $a[j] = (X + T) \bmod q$ 
10:       $a[j + m] = (X - T) \bmod q$ 
11:     end for
12:    $k = k + 1$ 
13:   end for
14: end for

```

Algorithm 2 ML-KEM INTT using GS butterfly

Input $\hat{a}(x) \in \mathbb{R}_q$ in bit-reversed order, $\omega^{-1} \in \mathbb{Z}_q$ and $n = 2^l$
Output $a(x) \in \mathbb{R}_q$ normal in order

```

1:  $k = 0$ 
2: for ( $i = l - 1$ ;  $i \geq 1$ ;  $i = i - 1$ ) do
3:    $m = 2^{l-i}$ 
4:   for ( $s = 0$ ;  $s < n$ ;  $s = s + 2m$ ) do
5:     for ( $j = s$ ;  $j < s + m$ ;  $j = j + 1$ ) do
6:        $W = \omega^{br_7(k)+1} \bmod q$ 
7:        $X = \hat{a}[j], Y = \hat{a}[j + m]$ 
8:        $\hat{a}[j] = (X + Y) \div 2 \bmod q$ 
9:        $\hat{a}[j + m] = (X - Y) \cdot W \div 2 \bmod q$ 
10:    end for
11:    $k = k + 1$ 
12:   end for
13: end for

```

Two incomplete NTT polynomials can be multiplied using the Base Case Multiply (BCM) algorithm, formally known as Coefficient-Wise Multiplication (CWM). The operation is performed over the ring $\mathbb{Z}_q[x] / (x^2 - \omega^{2br_7(i)+1})$ for $i = 0, \dots, 127$ and two polynomials $\hat{a}$ and $\hat{b}$; their BCM is computed as follows:

$$\begin{aligned}
 \hat{c} &= BCM(\hat{a}, \hat{b}); \\
 &= \hat{a} * \hat{b}; \\
 &= (\hat{a}_{2i+1}X + \hat{a}_{2i}) \cdot (\hat{b}_{2i+1}X + \hat{b}_{2i}) \bmod (x^2 - \omega^{2br_7(i)+1}); \\
 &= (\hat{a}_{2i+1} \cdot \hat{b}_{2i} + \hat{a}_{2i} \cdot \hat{b}_{2i+1})X + \hat{a}_{2i} \cdot \hat{b}_{2i} + \hat{a}_{2i+1} \cdot \hat{b}_{2i+1} \cdot \omega^{2br_7(i)+1}.
 \end{aligned} \tag{2}$$

For each pair of coefficients, five modular multiplications and two modular additions are required. Therefore, for the 128 pairings, the total cost is 640 modular multiplications and 256 modular additions.

2.4. Residue Number System

The RNS is a non-positional number system which is based on the Chinese Remainder Theorem (CRT). The numbers in RNS are represented by pairwise coprime numbers, called moduli $\{m_1, m_2, \dots, m_n\}$. The dynamic range of the system can be defined as the product of all moduli, $M = m_1 \cdot m_2 \cdot \dots \cdot m_n$. An integer number X in the range of $0 \leq X < M$ can have a unique representation as a vector $(x_1, x_2, \dots, x_n)$, where $x_i = |X|_{m_i} = X \bmod m_i$.

Weighted numbers are converted into RNS representation using a forward converter. Then, arithmetic operations such as addition, subtraction, and multiplication of two numbers can be performed independently in each RNS channel with respect to its modulus. This advantage allows a large integer operation to be performed in parallel on smaller residues where no carry propagation occurs. Hence, the critical path delay requirement is lowered, leading to a high-speed computation. However, division and modular reduction operations are complex in RNS as magnitude comparison cannot be applied due to the non-positional nature of the RNS. To apply such an operation, a conversion back to the weighted system using a reverse converter is required. This involves computing the CRT of the RNS representation, and the integer X can be recomputed as follows:

$$X = \left| \sum_{i=1}^n \left| x_i \cdot M_i^{-1} \right|_{m_i} \cdot M_i \right|_M \quad \text{where } M_i = \frac{M}{m_i} \quad (3)$$

This costly operation is inefficient and limits the adoption of RNS in modular arithmetic applications. An attempt to perform modular reduction in RNS was initially presented in [19] and later enhanced in [20]. This method utilises the classical Montgomery reduction [27] to perform modular reduction in RNS and requires a base extension. As a result, another base and conversion between these bases are required. Despite this complexity, this method can benefit applications where arithmetic involves bit sizes larger than 32 bits. However, it is generally not advantageous for small operand sizes (e.g., less than 32 bits), such as those encountered in ML-KEM. Another issue with this approach is that it is not beneficial for low-latency applications due to the conversion between RNS bases.

3. The Proposed Design

3.1. The Proposed RNS

The NTT/INTT computation involves butterfly operations: modular additions, modular subtraction, and modular multiplication. Selecting the RNS moduli set must ensure a valid dynamic range to avoid overflow during computations. The dynamic range of addition and subtraction is $2q$, while that for multiplication is q^2 . This work uses look-up tables to compute RNS operations, aiming for a low-latency, SCA-resistant design. The dynamic range of addition and subtraction can be implemented with a reasonable number of look-up tables; however, multiplication requires a large number of look-up tables, which is inefficient.

Multiplication can be replaced with addition using isomorphism mapping between the multiplicative group $g_n = \{1, 2, 3, \dots, q-1\}$ where the multiplication was performed using modulo q and the additive group $k_n = \{0, 1, 2, \dots, q-2\}$. Here, addition was performed for modulo $q-1$. This mapping is constructed using the formula $g_n = \alpha^{k_n}$, where α represents

the primitive root of q . This enables the multiplication of two integers to be performed as follows:

$$|g_n \cdot g_j|_q = \alpha^{|k_n + k_j|_{q-1}} \quad (4)$$

where $|a|_q$ represents the least positive residue of a : modulo q .

To conduct modular multiplication, firstly, index k_i for the two numbers being multiplied is determined. Next, the sum of the two indices using modulo $q - 1$ is calculated, and then the index operation is reversed to obtain the result. With this simplification, all NTT operations require only a dynamic range of $2n$. Furthermore, the NTT operations can now be performed in a modulus different from the prime modulus of ML-KEM with the restriction that it must cover the same dynamic range, i.e., $2n$.

The modulus in ML-KEM is $q = 3329$. To meet the dynamic range requirement, a composite modulus that is twice the primary modulus is needed. This composite modulus can be divided into smaller, relatively prime RNS sub-moduli. In this work, the sub-moduli 7, 31, and 32 were chosen to satisfy the condition $7 \cdot 31 \cdot 32 > 2q$, as proposed in our previous work [21]. NTT/INTT/BCM operations were then performed in these three RNS channels, and the results were reconstructed for the next operation. As in [21], a look-up table methodology was adopted to implement the RNS computations. As a result, it took three clock cycles to perform the NTT/INTT/BCM operations: one for conversion into RNS, one for the operation itself, and finally, one for reconstructing the results. This methodology leads to a low-latency RNS implementation.

The modular addition and subtraction operations involve using residue tables, addition/subtraction tables, and addition/subtraction correction reconstruction tables. Modular multiplication, however, due to the isomorphic transformation of multiplication into addition, involves index, addition, and reconstruction tables. A conversion back to the weighted number is also necessary before performing any other butterfly operation.

3.2. Configurable RNS Butterfly (RNS-BU)

The proposed RNS butterfly is shown in Figure 2 and can be configured to perform NTT, INTT, and BCM operations. NTT is computed using the CT structure, while INTT is computed using the GS structure to avoid the bit reversal operation. In BCM mode, the RNS-BU bypasses the CT and GS butterfly-specific data paths and reuses the shared modular multiplication and modular addition resources to perform the coefficient-wise operations required by Equation (2). The operating mode is selected by the control unit through configuration signals. All operations are performed in look-up tables implemented in FPGA BRAM slices configured as Read-Only Memory (ROM) with dual-mode configuration. The dual-port BRAM organisation allows two independent operations of the same stage to be processed in parallel. Each RNS modular arithmetic operation requires three clock cycles: one cycle for forward conversion into the RNS domain, one cycle for the RNS-domain arithmetic operation, and one cycle for reconstruction to the weighted representation. A complete butterfly consists of two sequential arithmetic stages, for example, modular multiplication followed by modular addition/subtraction. Since each stage requires three clock cycles to perform the corresponding RNS arithmetic operation, the total latency of a complete butterfly is six clock cycles.

The generation of the required RNS tables for the proposed configurable RNS Butterfly unit involves the following:

1. Submodular residue tables: These tables are generated straightforwardly for each sub-moduli by computing the modulo of the corresponding sub-moduli of the input.

- Submodular index tables: The primitive root α is selected as 3, taking into account the modulus of ML-KEM. Then, using the mapping $g = \left| 3^k \right|_{3329}$ the mapping table as shown in Table 1 is generated for $k = 0, 1, \dots, q - 2$.

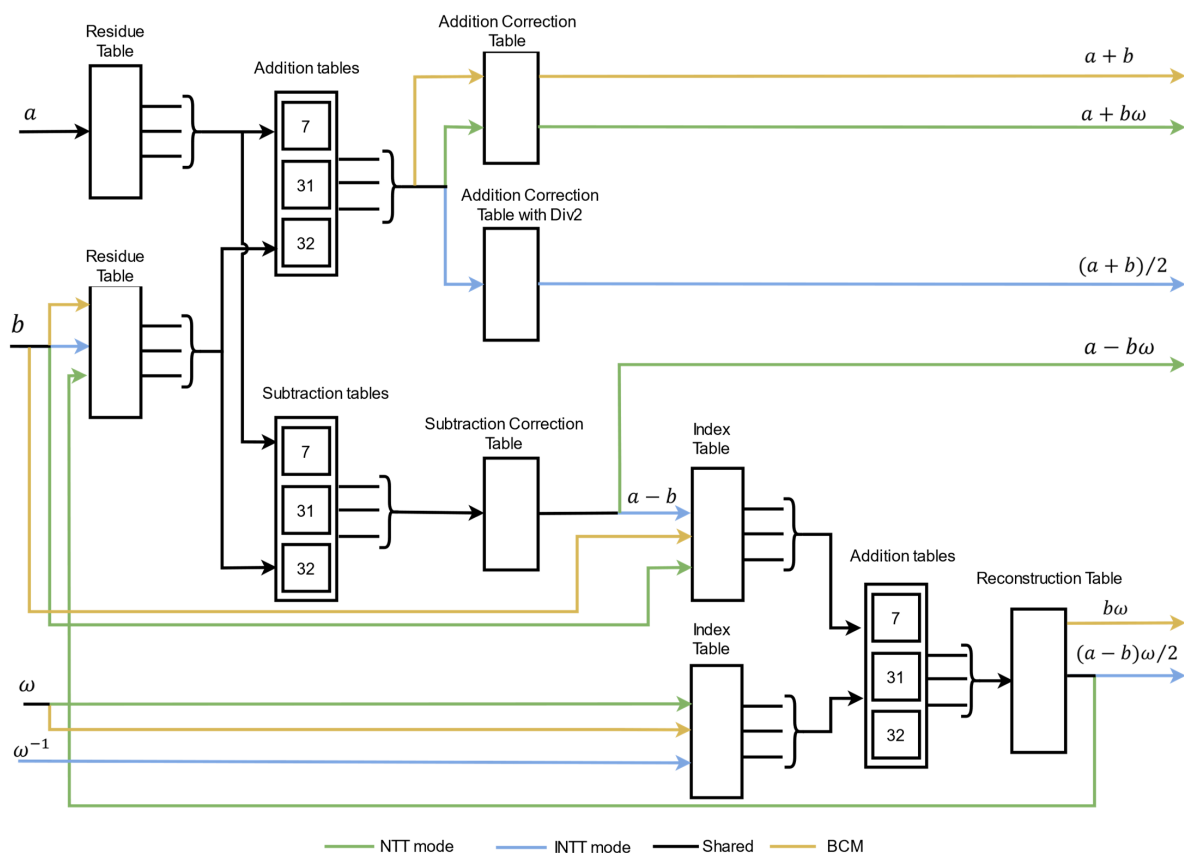
Table 1. Index mapping table.

k	0	1	2	3	...	200	201	...	3326	3327
g	1	3	9	27	...	1242	397	...	370	1110

Next, the table's contents and addresses are swapped, and the table's contents are reordered according to the value of g . Then, the modulo of the three sub-moduli is computed for each value of k to create the index table for each sub-moduli, as shown in Table 2:

Table 2. Submodular index tables.

g	1	2	3	...	370	371	...	3327	3328
k	0	1134	1	...	3326	2306	...	2798	1664
$ k _7$	0	0	1	...	1	3	...	5	5
$ k _{31}$	0	18	1	...	9	12	...	8	21
$ k _{32}$	0	14	1	...	30	2	...	14	0

**Figure 2.** The proposed configurable RNS butterfly.

- Submodular addition and subtraction tables: To facilitate submodular addition and subtraction, two tables are generated for each sub-moduli. First, the input residues are added to create the submodular addition table, and then the modulo operation

is computed with the corresponding sub-moduli. The subtraction table is computed similarly. The addresses of these tables are formed by concatenating the two residues to be added or subtracted.

4. Multiplication reconstruction table: The residues from the submodular addition operations are used to determine the address of the final modular multiplication result. Given the proposed RNS, this table is constructed using a 3D structure, where residues from paths q_1 , q_2 , and q_3 determine the page address, column address, and row address sequentially. The values of this table are computed in three steps as follows:
 - a. Submodular reconstruction: Let the residues r_1, r_2 , and r_3 result from the submodular addition operations, and the sub-moduli set be represented by q_1, q_2 , and q_3 . The weighted form of a given number can be recovered using the CRT as follows:

$$r = \left| \sum_{j=1}^3 \hat{q}_j \cdot \left| \frac{r_j}{\hat{q}_j} \right|_{q_j} \right|_{q_1 \cdot q_2 \cdot q_3} \quad \text{where } \hat{q}_j = \frac{q_1 \cdot q_2 \cdot q_3}{q_j} \quad (5)$$

This can be written as follows:

$$r = \left| r_1 \cdot \hat{q}_1 \cdot \left| \frac{1}{\hat{q}_1} \right|_{q_1} + r_2 \cdot \hat{q}_2 \cdot \left| \frac{1}{\hat{q}_2} \right|_{q_2} + r_3 \cdot \hat{q}_3 \cdot \left| \frac{1}{\hat{q}_3} \right|_{q_3} \right|_{q_1 \cdot q_2 \cdot q_3} \quad (6)$$

This can be simplified further as follows:

$$r = \left| \begin{array}{l} r_1 \cdot q_2 \cdot q_3 \cdot \left| \frac{1}{q_2 \cdot q_3} \right|_{q_1} + \\ r_2 \cdot q_1 \cdot q_3 \cdot \left| \frac{1}{q_1 \cdot q_3} \right|_{q_2} + \\ r_3 \cdot q_1 \cdot q_2 \cdot \left| \frac{1}{q_1 \cdot q_2} \right|_{q_3} \end{array} \right|_{q_1 \cdot q_2 \cdot q_3} \quad (7)$$

Given our chosen sub-moduli system, and computing the following modular inverse, the following is obtained:

$$\begin{aligned} \left| \frac{1}{q_2 \cdot q_3} \right|_{q_1} &= \left| \frac{1}{992} \right|_7 = 3 \\ \left| \frac{1}{q_1 \cdot q_3} \right|_{q_2} &= \left| \frac{1}{224} \right|_{31} = 9 \\ \left| \frac{1}{q_1 \cdot q_2} \right|_{q_3} &= \left| \frac{1}{217} \right|_{32} = 9 \end{aligned} \quad (8)$$

The equation is now written as follows:

$$r = |r_1 \cdot 2976 + r_2 \cdot 2016 + r_3 \cdot 1953|_{6944} \quad (9)$$

- b. Modulus overflow correction: The sub-moduli q_1, q_2 , and q_3 have no overflow, and only the modulus $q - 1$ requires overflow correction as follows:

$$r_i = |r|_{q-1} \quad (10)$$

- c. Inverse index look-up: The following mapping is used to invert the operation and recover the modular multiplication result:

$$x_i = |\alpha^{r_i}|_q \quad (11)$$

Following these three steps, the whole contents of the reconstruction table are generated. Theoretically, multiplication by zero is prohibited using this modular multiplication method; however, this specific scenario is handled through manual coding. The sub-

moduli $q_2 = 31$ route is selected to identify the operation of multiplying by zero. This is achieved by storing the value 31 in index zero, as this number is not an acceptable outcome within the sub-modulo 31 system. Furthermore, the addition table corresponds to row and column 31 and includes the value 31. Consequently, this results in zero output from the reconstruction table, as column address 31 consists entirely of zeros.

5. Addition and subtraction correction tables: Addition and subtraction correction tables are used to transform a number back into its weighted form after submodular addition and subtraction operations and correct the results, respectively. It is similar to the 3D structure of the multiplication reconstruction table, but uses a slightly different procedure to generate the contents. It follows the CRT to recover the result and only requires modulus overflow correction as follows:

$$r_i = |r|_q \quad (12)$$

The only difference between the addition and subtraction table is in the dynamic range, where addition has a dynamic range of $0, 1, \dots, 2n$ and subtraction is $-n, \dots, 0, \dots, n$.

6. Addition correction table with modular division by 2: The computation of the INTT involves scaling the result by n^{-1} ; as a result, 256 modular multiplications are required. This scaling can be eliminated by computing a modular division by 2 on the output of the GS butterfly during the INTT butterfly operations, as proposed in [28]. This work follows this method. One more addition correction table is used, and its contents are generated by performing modular division by 2 based on the results from Equation (12) as follows:

$$a = a/2 \bmod q \quad (13)$$

Then, modular division by 2 is performed for the subtraction results of the INTT on the precomputed values of the twiddle factors ω^{-1} , and stored as $\left(\frac{\omega^{-1}}{2}\right) \bmod q$ as follows:

$$\frac{(a-b)w}{2} \equiv (a-b) \cdot \frac{w}{2} \quad (14)$$

The results are stored in the same location as ω^{-1} ; therefore, there is no need to add another table. The modular multiplication procedure is summarised in Algorithm 3.

Algorithm 3 The proposed modular multiplication

	Two integers a and b
Input	Modulus q , Primitive root of the modulus α Sub-moduli set q_1, q_2, q_3
Output	$c = a \cdot b _q$
1:	Find index k for a and b using $g = \alpha^k _q$
2:	Compute submodular residues for $k(a), k(b)$:
3:	$a_1 = k(a) _{q_1}, a_2 = k(a) _{q_2}, a_3 = k(a) _{q_3}$
4:	$b_1 = k(b) _{q_1}, b_2 = k(b) _{q_2}, b_3 = k(b) _{q_3}$
5:	Compute residues additions:
6:	$r_1 = a_1 + b_1 _{q_1}$
7:	$r_2 = a_2 + b_2 _{q_2}$
8:	$r_3 = a_3 + b_3 _{q_3}$
9:	Compute CRT: $r = CRT(r_1, r_2, r_3)$
10:	Correct modulus overflow: $r_i = r _{q-1}$
11:	Inverse index look-up: $c = \alpha^{r_i} _q$

Modular addition, modular addition with division by 2, modular subtraction, and modular subtraction with division by 2 use similar procedures; as an example, modular addition with division by 2 is summarised in Algorithm 4.

Algorithm 4 The proposed modular addition with Div2

Input Two integers a and b , Modulus q
Sub-moduli set q_1, q_2, q_3

Output $c = |(a + b) \div 2|_q$

- 1: Compute submodular residues for a and b :
- 2: $a_1 = |a|_{q_1}, a_2 = |a|_{q_2}, a_3 = |a|_{q_3}$
- 3: $b_1 = |b|_{q_1}, b_2 = |b|_{q_2}, b_3 = |b|_{q_3}$
- 4: Compute residues additions:
- 5: $r_1 = |a_1 + b_1|_{q_1}$
- 6: $r_2 = |a_2 + b_2|_{q_2}$
- 7: $r_3 = |a_3 + b_3|_{q_3}$
- 8: Compute CRT: $r = CRT(r_1, r_2, r_3)$
- 9: Correct modulus overflow: $r_i = |r|_q$
- 10: Compute modular division by 2: $c = |r_i \div 2|_q$

3.3. Multiplier Architecture

The top-level architecture of the proposed RNS NTT multiplier is shown in Figure 3. The architecture consists of the proposed RNS butterfly unit, two RAMs to store the coefficients, and a ROM for storing the twiddle factors. Also, it has a control unit that incorporates the address generator module to generate the read and write addresses and the required control signals.

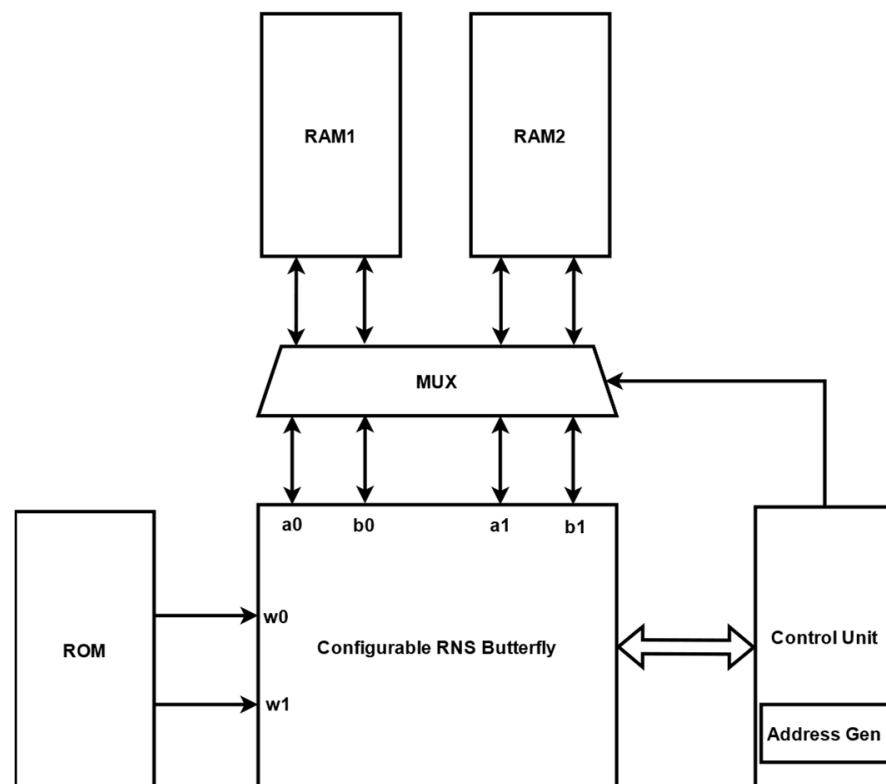


Figure 3. Top-level architecture.

The proposed RNS-BU processes two NTT/INTT operations at the same time, requiring four coefficients (two from each polynomial) to be read and written simultaneously. To achieve this, two memories, RAM1 and RAM2, are used. These memories are constructed using FPGA BRAMs and configured as dual-port block RAM, as shown in Figure 4. The coefficients are 12 bits each and are packed as 24 bits spread across the two RAMs. The second half of each RAM is used for the second polynomial, and the coefficients are stored in the opposite order. This arrangement allows for efficient access during the BCM operation.

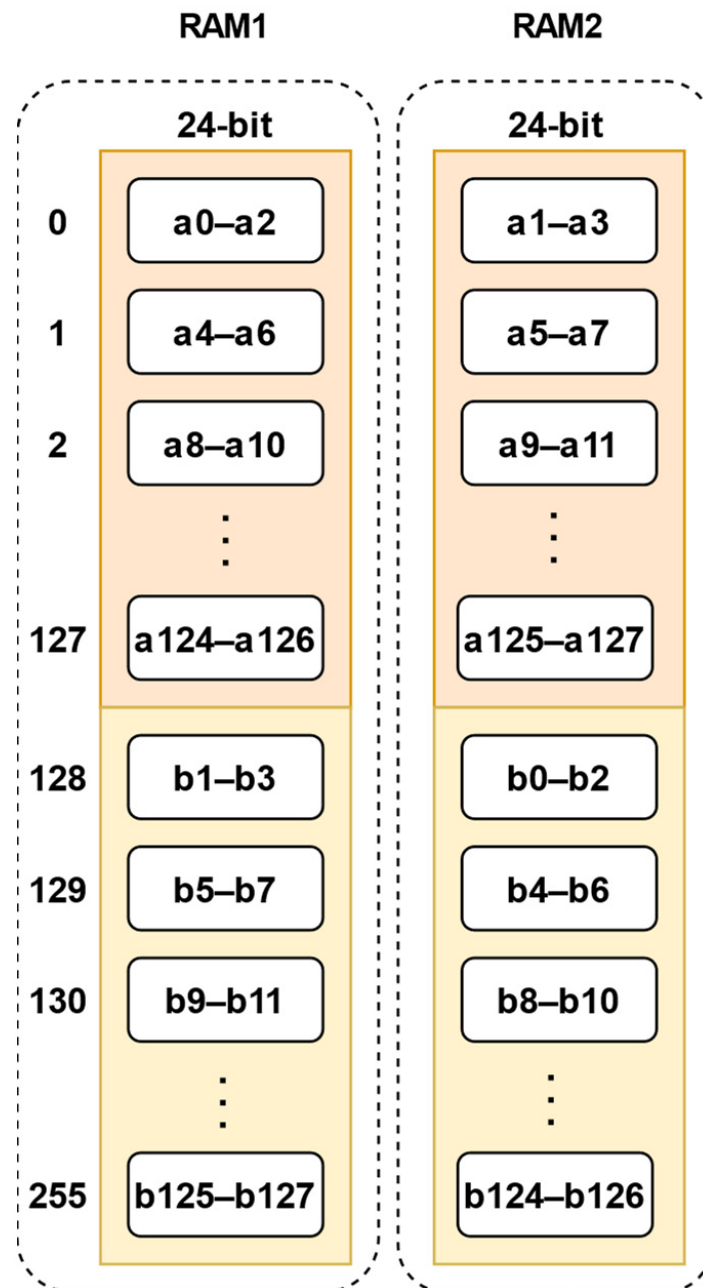


Figure 4. Memory organisation.

3.4. RNS as Potential SCA Countermeasure Analysis

Side-channel attacks vary, including timing attacks, simple power analysis (SPA), and differential power attacks (DPAs). These attacks measure time and the device's physical properties during algorithm computations, which involve using secret keys to find information about the secret key in question. Power consumption varies depending on the secret

key, and this variation is exploited by measuring power consumption during computations for different inputs while keeping the same key and ultimately recovering the secret key. The same concept also applies to timing attacks. The proposed design provides resistance to these kinds of attacks based on the following assumptions:

1. Using RNS for NTT computations spreads the computation across several channels, making measurements of the device's physical properties difficult. The RNS multi-channel representation is expected to reduce data-dependent switching activity and may contribute to reduced side-channel leakage. Using RNS as a side-channel countermeasure has been proven to be effective in RSA and ECC [16–18].
2. The proposed method uses look-up tables stored in FPGA BRAMs configured as ROM to perform the RNS computations. Compared with conventional arithmetic circuits based on adders and multipliers, this implementation style can reduce data-dependent switching activity during computation and, consequently, may reduce observable side-channel leakage. Previous studies have reported that BRAM-based implementations can serve as effective building blocks for side-channel countermeasures and may exhibit lower leakage than conventional arithmetic-based implementations [29–31].
3. Using a BRAM-based implementation for RNS-BU ensures that all RNS arithmetic computations are performed at a constant time, thus preventing attacks based on differences in timing for computations.

4. Results and Comparison

This section presents the FPGA implementation results and the SCA leakage evaluation of the proposed design. The Xilinx Vivado 2022 tool was employed to synthesise and implement the design, and the results were captured after the place-and-route process to ensure more accuracy. The design is coded in VHDL and implemented on the Xilinx Artix-7 XC7A200 FPGA for comparison with existing works. The reported polynomial multiplication time corresponds to the complete polynomial multiplication operation, i.e., $\text{INTT}(\text{NTT}(A) * \text{NTT}(B))$, for ML-KEM polynomials of degree ($n = 256$) and modulus ($q = 3329$). Since these parameters are identical across all ML-KEM parameter sets (ML-KEM-512, ML-KEM-768, and ML-KEM-1024), the reported polynomial multiplication performance is independent of the selected security level. The execution time is derived from the total latency of the NTT, BCM, and INTT stages and the maximum post-place-and-route operating frequency.

4.1. Comparison with Related Works

There is no previous RNS implementation for the entire polynomial multiplication of ML-KEM with which to compare the proposed design. Therefore, recent FPGA implementation works have been considered for comparison, ranging from lightweight design to higher-level performance, as shown in Table 3. The proposed design consumes 571 LUTs, 485 FFs, 19 BRAMs, and 245 slices. The FPGA utilisation is less than 0.5% for the LUTs and FFs and around 7% for the BRAMs. The design does not consume any DSP slices, as all the NTT operations are performed in look-up tables. It runs at a 236 MHz clock frequency and computes the whole polynomial multiplication in 7.2 μs . It should be noted that the comparison in this work focuses on NTT-based FPGA implementations of ML-KEM polynomial multiplication, as this is the multiplication approach adopted by ML-KEM. Alternative time-domain polynomial multiplication techniques, such as Toom–Cook multiplication, represent different design approaches. A dedicated implementation and leakage evaluation would be required under the same target platform to enable a fair comparison.

Table 3. FPGA implementation results and comparison with related works.

Ref.	SCA Protection	AREA					Freq. (MHz)	Latency (CC)	PMUL Time ¹	ATP-LUT ²	ATP-ENS ³
		LUT	FF	DSP	BRAM	Slices		NTT/INTT/BCM			
This Work	Yes	571	485	0	19	245	236	458/458/328	7.2	4.1	28.6
[32]	No	737	290	6	4	371	115	474/602/1289	24.7	18.2	46.3
[33]	No	1579	1058	2	3	527	161	448/448/256	9.9	15.7	13.5
[34]	No	880	999	2	1.5	345	229	448/448/256	7	6.1	6.1
[35]	No	784	441	2	3	259	200	313/313/256	6	4.7	6.5
[36]	No	1154	1031	2	0	445	300	456/456/265	5.4	6.3	3.7
[37]	No	1549	788	4	16	635	159	457/457/140	6.6	10.3	28.2
[38]	No	1070	1071	5	10.5	401	278	306/306/256	4.2	4.5	12.9

¹ The time in microseconds (μ s) required to perform complete polynomial multiplication, i.e., $\text{INTT}(\text{NTT}(A) * \text{NTT}(B))$. ² ATP-LUT: Area-Time Product considering LUTs. $\text{ATP-LUT} = \text{Total Time of computing complete polynomial multiplication in milliseconds (ms)} \times \text{number of LUTs}$. ³ ATP-ENS: Area-Time Product considering Equivalent Number of Slices (ENSs). $\text{ATP-ENS} = \text{Total Time of computing complete polynomial multiplication in milliseconds (ms)} \times \text{ENS}$. $\text{ENS} = \text{Number of Slices} + \text{DSP} \times 120 + \text{BRAM} \times 196$ [36].

Although the compared designs do not provide the same level of side-channel protection, the results indicate that the proposed architecture achieves competitive resource utilisation and performance despite incorporating an SCA countermeasure. The proposed RNS design has the fewest LUTs and slices. The ATP-LUT metric also contributes to the presented design's area efficiency and surpasses all prior works. This work transfers the area complexity of the SCA countermeasure into the FPGA BRAMs; therefore, only BRAM's utilisation is increased while other area metrics are decreased. Despite this, the ATP-ENS, which includes the BRAM usage, produces a reasonable figure, especially considering that other works do not have an SCA countermeasure. The ATP-ENS produces a better result than the work in [32], similar to the work in [37], and lower than the others.

The method presented in this work demonstrates an efficient low-latency RNS implementation compared to the standard approach. Forward conversion, modular RNS arithmetic, and reverse conversion typically involve complex operations such as CRT, resulting in a high-latency design. However, this work simplifies these operations by utilising look-up tables, enabling them to be executed in only three clock cycles, with one clock cycle allocated to each operation.

SCAs against ML-KEM exploit different operations in the NTT-based polynomial multiplication, including NTT, INTT, and BCM [6,8,10,14]. These operations include modular multiplication, modular addition, modular subtraction, and modular division by 2. The work in [21] explored RNS for integer modular multiplication only to speed up the NTT computation; hence, this work proposes a complete RNS implementation to counter these SCAs.

An SCA countermeasure typically has a computation overhead of 2.5 to 6 times for a given design [39]. For example, the masked RISK-V codesign presented in [13] has a 5.66 computation overhead for the ML-KEM decapsulation procedure at the NIST level I parameters set. In Table 4, the proposed design is first compared against the partially protected design referenced in [21], given that, to the best of our knowledge, there is no other work using RNS for the full polynomial multiplication. This comparison gives a reasonable estimation of the required overhead for the full RNS design. In addition, Table 4 provides additional context by comparing this method to both masked and unmasked implementations of the FPGA design for polynomial multiplication using the schoolbook method, with parameters similar to those of ML-KEM [39].

Table 4. Comparison of SCA countermeasure implementation costs.

Ref.	AREA LUT/FF/DSP/BRAM	Freq. (Mhz)	Latency (cc)	Area Ratio ¹	Time Ratio ²
This work	571/485/0/19	236	1702	2.3	1.4
[21]	590/671/0/7.5	342	1702		
[39] masked	25,214/13,280/0/1	130	1020	1.6	5.8
[39] unmasked	17,429/5083/0/1	250	336		

¹ The area increase ratio was measured as the ENSs of the protected design/ENSs of the unprotected design. ² The time increase ratio was measured as the time of polynomial multiplication of the protected design/time of the unprotected design.

The proposed design uses fewer LUTs and FFs but requires more Block RAMs than the work in [21], resulting in an increase in the area ratio by a factor of 2.3. In terms of time efficiency, the proposed design experiences a slight increase in the time ratio of 1.4. On the other hand, the masking method in [39] has a smaller area ratio of 1.6 compared to the unmasked version, but this comes at the cost of a significant increase in the time ratio of 5.8. Given that the work in [39] has similar polynomial parameters, this design is 8% faster and 52% smaller than the masked version in [39].

This analysis shows that an RNS-based methodology for the hardware design and implementation of ML-KEM can be an efficient tool for SCA protection with reduced additional overheads whilst maintaining performance speed. In the next section, an experimental evaluation of SCA leakage in the design is presented to support this analysis.

4.2. Side-Channel Leakage Evaluation

This section presents the results of the experimental evaluation of the proposed design's effectiveness against SCA. The power traces were captured using the ChipWhisperer Husky board, while the design was implemented on the ChipWhisperer CW305 target board and configured with an Artix-7 FPGA (XC7A100T). The design operated at a frequency of 5 MHz, and power measurements were taken using an SMA connector on the CW305 board, connected to the FPGA power supply. Power was measured using a 0.1 Ω shunt resistor with an amplification of 20 dB using the CW305 on-board low-noise amplifier. The gain for the Husky was set to 40 dB, and the ADC's sampling clock on the Husky board was synchronised with the target clock of CW305. This setup allowed one sample to be taken during each clock cycle. In synchronous sampling, capturing one sample per clock cycle is sufficient to detect power changes, as stated in [40].

The experiment employed a standard method to assess SCA leakage, known as the test vector leakage assessment (TVLA) [41]. The non-specific fixed-vs-random TVLA test was used, requiring the power traces to be recorded for two sets: one with fixed inputs (T0) and the other with random inputs (T1). The test requires the n number of power traces to be recorded for each set, followed by computing the mean μ and the variance s^2 of the samples for each set. Then, Welsh's t -test can be computed on each point in the collected traces to identify any significant differences between the two sets. The resulting t -value from Welsh's t -test is computed as follows:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \quad (15)$$

According to [42], a design can be considered secure against SCA if the t -value falls within the range $(-4.5, +4.5)$. However, if the t -value falls outside this threshold, there is a likelihood of SCA leakage with 99.999% confidence.

To validate the experimental setup, a baseline, unprotected design was implemented, utilising the modular arithmetic described in [43]. The SCA targeted the NTT, INTT, and BCM operations; therefore, the leakage evaluation was conducted separately for each operation, with a total of 1000 power traces collected for each one. The evaluation results for both the proposed and baseline designs are presented in Figure 5. In the figure, the red line indicates the t-value threshold. The baseline design fails the test at a considerable number of points across all three algorithms. By contrast, the proposed design successfully surpasses this, thereby validating the previous analysis. The TVLA results show that no statistically significant first-order leakage was detected under the evaluated conditions using 1000 power traces. Nevertheless, additional traces and higher-order analyses may provide further insight into the leakage characteristics of the proposed architecture.

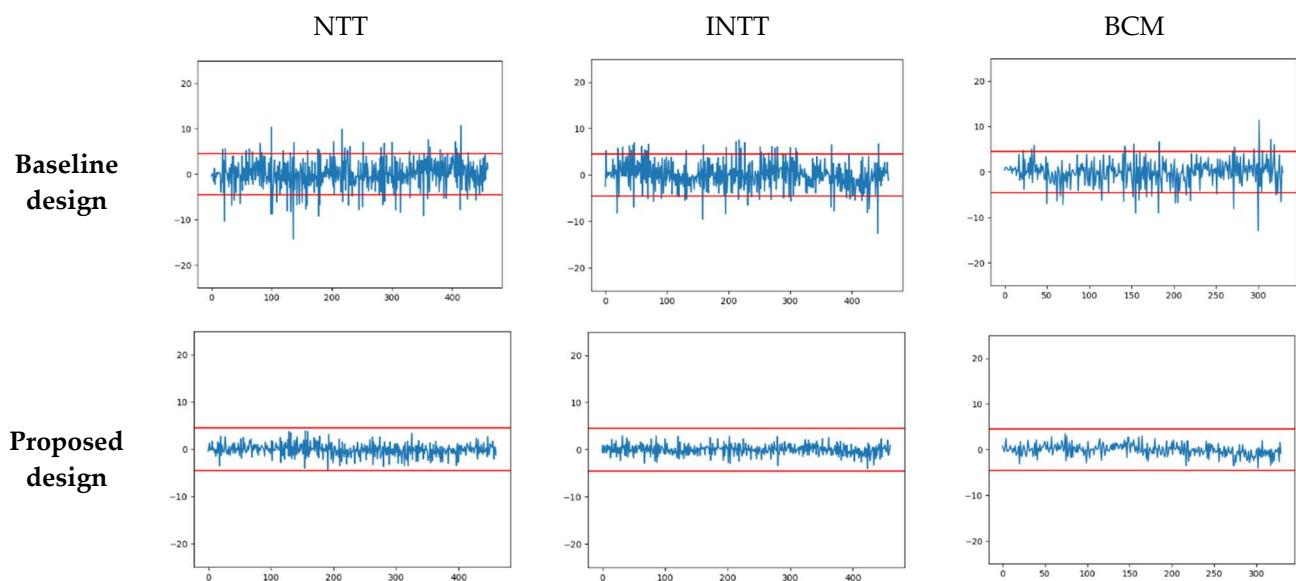


Figure 5. TVLA results for NTT, INTT, and BCM operations of both the unprotected baseline design and the proposed design.

5. Conclusions and Future Work

In this work, a methodology that combines RNS arithmetic with a look-up table architecture was applied to carry out the polynomial multiplication of the ML-KEM algorithm for efficiency and SCA protection purposes. Based on an efficient RNS moduli set, a configurable RNS butterfly unit was integrated to build an RNS NTT hardware multiplier targeting FPGA platforms. The RNS computations are offloaded into FPGA BRAM slices, resulting in a low-complexity and low-latency design. Considering the implementation cost of the SCA countermeasure, the FPGA implementation results show that the proposed design achieves competitive results in terms of the area and time metrics compared to state-of-the-art designs.

The SCA resistance evaluation of the proposed design through TVLA demonstrated the attractiveness of the proposed design methodology as a promising tool to support SCA protection designs of PQC.

Future work will consider end-to-end design for the whole ML-KEM protocol.

Author Contributions: Conceptualisation, A.A. and M.B.; methodology, A.A.; software, A.A.; validation, A.A. and M.B.; formal analysis, A.A. and M.B.; investigation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, M.B.; supervision, M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data supporting the conclusions of this study are presented within the article. The source code is available from the corresponding author upon reasonable request.

Acknowledgments: The authors gratefully acknowledge the support of George T London and Kwashie A Andoh in conducting the experimental SCA measurements and leakage evaluation of the FPGA implementation of the designs.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* **1997**, *26*, 1484–1509. [\[CrossRef\]](#)
- Chen, L.; Jordan, S.; Liu, Y.-K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; The National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016.
- Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation. *NIST PQC Round* **2021**, *2*, 1–43.
- FIPS 203; Module-Lattice-Based Key-Encapsulation Mechanism Standard. National Institute of Standards and Technology: Department of Commerce: Washington, DC, USA, 2024.
- Langlois, A.; Stehlé, D. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.* **2015**, *75*, 565–599.
- Xu, Z.; Pemberton, O.; Roy, S.S.; Oswald, D.; Yao, W.; Zheng, Z. Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems With Chosen Ciphertexts: The Case Study of Kyber. *IEEE Trans. Comput.* **2021**, *71*, 2163–2176.
- Tanaka, Y.; Ueno, R.; Xagawa, K.; Ito, A.; Takahashi, J.; Homma, N. Multiple-Valued Plaintext-Checking Side-Channel Attacks on Post-quantum KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2023**, *2023*, 473–503. [\[CrossRef\]](#)
- Mujdei, C.; Wouters, L.; Karmakar, A.; Beckers, A.; Mera, J.M.B.; Verbauwhede, I. Side-Channel Analysis of Lattice-Based Post-quantum Cryptography: Exploiting Polynomial Multiplication. *ACM Trans. Embed. Comput. Syst.* **2024**, *23*, 27. [\[CrossRef\]](#)
- Ji, Y.; Dubrova, E. A Side-Channel Attack on a Hardware Implementation of CRYSTALS-Kyber. In *Proceedings of the 2023 IEEE European Test Symposium (ETS), Venice, Italy, 22–26 May 2023*; IEEE: Piscataway, NJ, USA, 2023; pp. 1–5.
- Rodriguez, R.C.; Bruguier, F.; Valea, E.; Benoit, P. Correlation Electromagnetic Analysis on an FPGA Implementation of CRYSTALS-Kyber. In *Proceedings of the 2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Valencia, Spain, 18–21 June 2023*; pp. 217–220.
- D’ANvers, J.-P.; Van Beirendonck, M.; Verbauwhede, I. Revisiting Higher-Order Masked Comparison for Lattice-Based Cryptography: Algorithms and Bit-Sliced Implementations. *IEEE Trans. Comput.* **2023**, *72*, 321–332.
- Bos, J.W.; Gourjon, M.; Renes, J.; Schneider, T.; Van Vredendaal, C. Masking Kyber: First- and Higher-Order Implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 173–214. [\[CrossRef\]](#)
- Fritzmant, T.; Van Beirendonck, M.; Roy, D.B.; Karl, P.; Schamberger, T.; Verbauwhede, I.; Sigl, G. Masked Accelerators and Instruction Set Extensions for Post-Quantum Cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 414–460. [\[CrossRef\]](#)
- Bock, E.A.; Banegas, G.; Brzuska, C.; Chmielewski, L.; Puniamurthy, K.; Šorfi, M. Breaking DPA-Protected Kyber via the Pair-Pointwise Multiplication. In *International Conference on Applied Cryptography and Network Security*; Springer: Cham, Switzerland, 2024; pp. 101–130.
- Hamburg, M.; Hermelink, J.; Primas, R.; Samardjiska, S.; Schamberger, T.; Streit, S.; Strieder, E.; Van Vredendaal, C. Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 88–113. [\[CrossRef\]](#)
- Bajard, J.-C.; Imbert, L.; Liardet, P.-Y.; Teglia, Y. Leak Resistant Arithmetic. In *Cryptographic Hardware and Embedded Systems—CHES 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 62–75.
- Perin, G.; Imbert, L.; Torres, L.; Maurine, P. Electromagnetic Analysis on RSA Algorithm Based on RNS. In *Proceedings of the 2013 Euromicro Conference on Digital System Design (DSD), Los Alamitos, CA, USA, 15 October 2013*; pp. 345–352.
- Pettenghi, H.; Ambrose, J.A.; Chaves, R.; Sousa, L. Method for designing multi-channel RNS architectures to prevent power analysis SCA. In *Proceedings of the 2014 IEEE International Symposium on Circuits and Systems (ISCAS), Melbourne, VIC, Australia, 1–5 June 2014*; pp. 2233–2236.
- Posch, K.; Posch, R. Modulo reduction in residue number systems. *IEEE Trans. Parallel Distrib. Syst.* **1995**, *6*, 449–454. [\[CrossRef\]](#)
- Bajard, J.-C.; Didier, L.-S.; Kornerup, P. An RNS Montgomery modular multiplication algorithm. *IEEE Trans. Comput.* **1998**, *47*, 766–776. [\[CrossRef\]](#)
- Alhassani, A.; Benaissa, M. High-Speed Polynomials Multiplication HW Accelerator for CRYSTALS-Kyber. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 6105–6113. [\[CrossRef\]](#)

22. Fujisaki, E.; Okamoto, T. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Proceedings of the 19th Annual International Cryptology Conference, Los Angeles, CA, USA, 15–19 August 1999*; Springer: Berlin/Heidelberg, Germany, 1999.
23. Lyubashevsky, V.; Seiler, G. NTTTRU: Truly Fast NTRU Using NTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2019**, *2019*, 180–201. [\[CrossRef\]](#)
24. Cooley, J.W.; Tukey, J.W. An algorithm for the machine calculation of complex fourier series. *Math. Comput.* **1965**, *19*, 249–259. [\[CrossRef\]](#)
25. Gentleman, W.M.; Sande, G. Fast fourier transforms—For fun and profit. In *AFIPS Conference Proceedings—1966 Fall Joint Computer Conference*; AFIPS: Arlington, VA, USA, 1966.
26. Longa, P.; Naehrig, M. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *International Conference on Cryptology and Network Security*; Springer: Cham, Switzerland, 2026; pp. 124–139.
27. Montgomery, P.L. Modular Multiplication Without Trial Division. *Math. Comput.* **1985**, *44*, 519–521. [\[CrossRef\]](#)
28. Zhang, N.; Yang, B.; Chen, C.; Yin, S.; Wei, S.; Liu, L. Highly Efficient Architecture of NewHope-NIST on FPGA using Low-Complexity NTT/INTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2020**, *2020*, 49–72. [\[CrossRef\]](#)
29. Shah, S.; Velegali, R.; Kaps, J.-P.; Hwang, D. Investigation of DPA Resistance of Block RAMs in Cryptographic Implementations on FPGAs. In *Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs, Cancun, Mexico, 13–15 December 2010*; IEEE: Piscataway, NJ, USA, 2010; pp. 274–279.
30. Bhasin, S.; Danger, J.-L.; Guilley, S.; He, W. Exploiting FPGA block memories for protected cryptographic implementations. In *Proceedings of the 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, Darmstadt, Germany, 10–12 July 2013.
31. Sasdrich, P.; Mischke, O.; Moradi, A.; Güneysu, T. Side-Channel Protection by Randomizing Look-Up Tables on Reconfigurable Hardware. In *Constructive Side-Channel Analysis and Secure Design*; Springer: Cham, Switzerland, 2015.
32. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari-Kermani, M. Instruction-Set Accelerated Implementation of CRYSTALS-Kyber. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 4648–4659. [\[CrossRef\]](#)
33. Yufei, X.; Shuguo, L. A Compact Hardware Implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-KYBER on FPGA. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 328–356. [\[CrossRef\]](#)
34. Dang, V.B.; Mohajerani, K.; Gaj, K. High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber. *IEEE Trans. Comput.* **2022**, *72*, 306–320.
35. Guo, W.; Li, S. Split-Radix Based Compact Hardware Architecture for CRYSTALS-Kyber. *IEEE Trans. Comput.* **2023**, *73*, 97–108. [\[CrossRef\]](#)
36. Ni, Z.; Khalid, A.; Liu, W.; O’Neill, M. Towards a Lightweight CRYSTALS-Kyber in FPGAs: An Ultra-lightweight BRAM-free NTT Core. In *Proceedings of the 2023 IEEE International Symposium on Circuits and Systems (ISCAS), Monterey, CA, USA, 21–25 May 2023*; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2023; pp. 1–5.
37. Guo, W.; Li, S.; Kong, L. An Efficient Implementation of KYBER. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 1562–1566. [\[CrossRef\]](#)
38. Sun, J.; Bai, X. A High-Speed Hardware Architecture of an NTT Accelerator for CRYSTALS-Kyber. *Integr. Circuits Syst.* **2024**, *1*, 92–102. [\[CrossRef\]](#)
39. Aikata, A.; Basso, A.; Cassiers, G.; Mert, A.C.; Roy, S.S. Kavach: Lightweight masking techniques for polynomial arithmetic in lattice-based cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2023**, *2023*, 366–390. [\[CrossRef\]](#)
40. O’Flynn, C.; Chen, Z. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *J. Cryptogr. Eng.* **2015**, *5*, 53–69.
41. Gilbert Goodwill, B.J.; Jaffe, J.; Rohatgi, P. A testing methodology for side-channel resistance validation. In *Proceedings of the NIST Non-Invasive Attack Testing Workshop*, Nara, Japan, 25–27 September 2011.
42. Schneider, T.; Moradi, A. Leakage assessment methodology: A clear roadmap for side-channel evaluations. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2015.
43. Yaman, F.; Mert, A.C.; Ozturk, E.; Savas, E. A Hardware Accelerator for Polynomial Multiplication Operation of CRYSTALS-KYBER PQC Scheme. In *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 1–5 February 2021; pp. 1020–1025.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.