Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Lightweight Fault Detection Architecture for NTT on FPGA

Rourab Paul[1] | **Paresh Baidya**[2,3] | **Krishnendu Guha**[3]

[1]Computer Science and Engineering, Shiv Nadar University Chennai, Tamil Nadu, India

[2]Department of Mathematics, National Institute of Technology Jamshedpur, India

[3]Computer Science Engineering, SOA University, Bhubaneswar, Odisha, India

[4]Computer Science and Information Technology, University College Cork, Ireland

Correspondence

Corresponding author Rourab Paul, This is sample corresponding address.

Email: rourabpaul@snuchennai.edu.in

Abstract

Post-Quantum Cryptographic (PQC) algorithms are mathematically secure and resistant to quantum attacks but can still leak sensitive information in hardware implementations due to natural faults or intentional fault injections. The intent fault injection in side-channel attacks reduces the reliability of crypto implementation in future generation network security procesors. In this regard, this research proposes a lightweight, efficient, recomputation-based fault detection module implemented on a Field Programmable Gate Array (FPGA) for Number Theoretic Transform (NTT). The NTT is primarily composed of memory units and the Cooley–Tukey Butterfly Unit (CT-BU), a critical and computationally intensive hardware component essential for polynomial multiplication. NTT and polynomial multiplication are fundamental building blocks in many PQC algorithms, including Kyber, NTRU, Ring-LWE, and others. In this paper, we present a fault detection method called : Recomputation with a Modular Offset (REMO) for the logic blocks of the CT-BU using Montgomery Reduction and another method called Memory Rule Checkers for the memory components used within the NTT. The proposed fault detection framework sets a new benchmark by achieving high efficiency with significant low implementation cost. It occupies only 16 slices and a single DSP block, with a power consumption of just 3mW in *Artix*-7 FPGA. The REMO-based detection mechanism achieves a fault coverage of 87.2% to 100%, adaptable across various word sizes (w), fault bit counts (η), and fault injection modes. Similarly, the Memory Rule Checkers demonstrate robust performance, achieving 50.7% to 100% fault detection depending on η and the nature of injected faults.

KEYWORDS

Polynomial Multiplication, FPGA, Cooly-Tukey Butterfly, Memory, NTT, Modular Multiplication, Montgomery Reduction.

1 | INTRODUCTION

Faults in PQC hardware can occur naturally due to shrinking device dimensions, which increase the probability of errors caused by ion interference or electromagnetic radiation. These densely packed devices are particularly susceptible to internal faults induced by ion beam radiation in the Configurable Logic Block (CLB) of FPGAs[1]. Additionally, in side-channel attacks, adversaries deliberately introduce faults into PQC hardware and analyze variations in physical parameters, such as power consumption and execution time, to extract sensitive information. Therefore, mathematically secure PQC algorithms may still be vulnerable in hardware implementations to various side-channel attacks, such as power analysis, timing analysis, electromagnetic (EM) attacks and fault injection attacks. Specifically, fault injection attacks[2 3] exploit deliberate faults introduced into PQC hardware to expose intermediate states or reveal secret data. Barenghi et al.[4] discussed various cost-effective methods for injecting faults into existing cryptographic systems. Recently, Primas et al.[5] and Prasanna et al.[6] conducted a Soft Analytical Side-Channel Attack (SASCA) on the NTT, employing a probabilistic model utilizing power and timing data.

Preventing fault occurrences in PQC hardware provides a robust defence against both natural and intentional faults. To address various fault detection strategies, several fault detection approaches have been proposed in the literature. We categorize the existing work into two groups: (i) Fault detection solutions for various components of crypto algorithms, and (ii) Fault detection specifically targeting the NTT (Number Theoretic Transform).

1.1 Fault Detection in various Crypto Algorithms [7 8 9 10 11 12 13 14 15 16]

The modular exponentiation is a crucial operation in cryptography. Saeed et.al[7] proposed a recomputation based fault detection model for modular exponent $x^y mod\ n$ implemented on an ARM Cortex-A72 processors, AMD/Xilinx Zynq Ultrascale+ and Artix-7 FPGA. Their approach involves recomputing the modular exponent using encoded values of x and y. The proposed method achieves near 100% error detection accuracy with approximately 7% computational overhead and less than 1% area overhead compared to unprotected architectures. Saeed et.al[7] employed the popular Right-to-Left Exponentiation algorithm[17], where the number of iterations depends on the number of '1's in exponent y. This conditional operation makes the algorithm vulnerable[18] to timing analysis attacks. Canto et al.[8] presented various fault detection mechanisms design for finite-field operations, including addition, subtraction, multiplication, squaring, and inversion, specifically within the framework of the code-based McEliece cryptosystem. The proposed approaches used different error detection techniques, such as regular parity, interleaved parity, CRC-2, and CRC-8, to enhance the fault detection capabilities. These methods are applied to different elements of the Key Generator, focusing on improving error detection accuracy in operations such as multiplications and inversions in the finite field $GF(2^{13})$. Howe et al.[9] designed a fault detection module for error sampler used in lattice based cryptography. They introduced three methods: low-cost test, standard test, and expensive test designed for FPGA implementation to verify whether the output distribution of the error sampler matches its expected Gaussian or binomial shape. The study in[10] proposed three fault detection models for Multiply and Accumulate (MAC) unit of lattice-based Key Encapsulation Mechanisms (KEMs) and applied them to hardware accelerators of three NIST PQC finalists: FrodoKEM, SABER, and NTRU. The proposed schemes are based on the recomputation technique such as Recomputing with Shifted Operands (RESO), Recomputing with Negated Operands (RENO) and Recomputing with Scaled operands (RECO), implemented on a Kintex UltraScale+ device. Their implementation on FPGA devices demonstrates minimal overhead and significant error detection coverage, ensuring compatibility with other cryptographic systems utilizes hardware accelerators. Kermani et al.[11] introduced a novel error detection scheme for Galois Counter Mode (GCM) implemented on a 65nm Application Specific Integrated Circuit (ASIC) platform, specifically developed to improve data integrity verification. The proposed methodology improved compatibility with various block ciphers and finite field multipliers by employing a technique Re-computation of Swapped Ciphertext and Additional Authenticated Blocks (RESCAB). In this work, the primary computational unit, Galois Hash (GHASH) computed over the finite field $GF(2^{128})$, while the RESCAB module processed swapped inputs concurrently within another $GF(2^{128})$ instance. Then, fault detection achieved by comparing the outputs of GHASH and RESCAB. The architecture in[11], demonstrated significant design flexibility and reliability, as validated through hardware implementations and fault simulation analyzes. Cintas et al.[12] proposed an error detection mechanism for Goppa arithmetic units used in the McEliece cryptosystem. They used the algebraic structure of composite fields in this cryptosystem. Their approach included implementing a Parity Checker for various sub-blocks of McEliece. The proposed methods in[12] were not limited to arithmetic units but were also suitable for core functions of other public-key cryptosystems that relied on composite fields as their mathematical foundation. Additionally, the authors presented FPGA-based implementations of Goppa polynomial evaluation (GPE) and analyzed the performance overhead for different configurations. In[13], the authors investigated various techniques to strengthen the resilience of NTRUEncrypt hardware implementations, against fault analysis attacks. They used the cipher's algebraic properties, and proposed countermeasures based on error detection codes and spatial/temporal redundancies. A detailed evaluation of these methods was provided by comparing their error detection efficiency along with their impact on decryption throughput and hardware area. Ahmadi et al.[14] proposed a fault detection scheme for the window method in elliptic curve scalar multiplication (ECSM). They introduced refined algorithms and hardware implementations to address both permanent and transient errors. Using simulation-based fault injection, the schemes achieved extensive error coverage with less than 3% clock cycle overhead on Cortex-A72 processors and a 2% area increase on FPGAs. These results demonstrated efficient error detection with minimal resource overhead. Ahmadi et al.[15] addressed a research gap in fault detection for τNAF conversion and Koblitz curve cryptosystems. They proposed an algorithm-level fault detection scheme for the single τNAF conversion algorithm and developed two additional fault detection schemes for the double τNAF conversion algorithm. The feasibility of these methods was evaluated through implementation on ARMv7 and ARMv8 architectures. This paper[16] introduced fault-tolerant and error-detecting structures for elliptic curve scalar multiplication (ECSM). The proposed methods utilized recomputation, parallel computation, and encoding-decoding schemes to enhance error detection during ECSM operations. These schemes used scalar and point randomization techniques and were implemented on Xilinx Virtex 2000E FPGA. This work achieved a high error coverage with minimal computational overhead.

1.2 Fault Detection for NTT [19 20 21 22 6 23]

NTT is used as both forward and inverse transforms and is one of the most widely used components in many post-quantum cryptography (PQC) algorithms. The recent NIST standardization of Kyber as Module Lattice-based Key Encapsulation Mechanism (ML-KEM) and Dilithium as Module Lattice-based Digital Signature Algorithm (ML-DSA) includes the Number Theoretic Transform (NTT) and KECCAK as the most critical hardware components. The NTT block consists of three main components: (i) Random Access Memories (RAMs) to store polynomial coefficients, error polynomials and secret vectors, (ii) Read-Only Memories (ROMs) to store constants such as twiddle factors, and (iii) a CT-BU to perform the core computations of the NTT. Khan et al.[19] implemented a fault-tolerant memory modules (RAMs and ROMs) for storing polynomial coefficients and twiddle factors of the NTT using Hamming codes and parity bits, resulting in a 29.2% overhead on a Virtex-7 FPGA. They implemented four fault-tolerant variants for the memory modules of the NTT used in Kyber: (i) Area Optimized (A-O) using Hamming Code, (ii) Run-Time Optimized (R-O) using Hamming Code, (iii) Area Optimized (R-O) using Hamming Code combined with Parity and (iv) Run-Time Optimized (R-O) using Hamming Code combined with Parity. Sven et al.[20] implemented a fault-tolerant model for the Number Theoretic Transform (NTT) that can resist faults in multiplication with a twiddle factor and the addition in a butterfly operation on the ARM Cortex-M4 platform. They used interpolation and, evaluation and inverse NTT method to detect the faults inside the CT-BU. They have taken a polynomial coefficient $f(X)$, and its inverse NTT is $f(w^j)$. If the Eq. 1 is satisfied, it indicates that no fault occurred during the NTT computation.

$$f(u) = evaluation(NTT^{-1}(f(w^j))) = Interpolation(f(w^j)) \quad (1)$$

Polynomial Evaluation and Interpolation is a method that evaluates a polynomial at powers of a primitive root of unity during the forward NTT and recovers the original polynomial coefficients through interpolation during the inverse NTT, all over a finite field. However, the interpolation and evaluation processes introduce additional 3160 and 2979 clock cycles, respectively, resulting in a significant 72% timing overhead for Dilithium.

Sarker et al.[21] implemented a fault detection scheme for the HW/SW co-design of NTT on Spartan-6 and Zynq UltraScale+ platforms, resulting in 12.74% resource overhead and approximately 20% power overhead. They used recomputing with negative operands (RENO). Depending on the placement of the RENO block in the logic path, Sarker et al.[21] proposed three versions of the NTT. To the best of our knowledge, no existing fault detection solution in the literature has addressed both the logic unit of the NTT (CT-BU) and the memory units separately.

Apart from the fault detection schemes proposed in [19], [20], and [21], there are a few NTT implementations that incorporate certain precautions to make side-channel attacks on the NTT more difficult. Jati et al.[22] implemented a side channel attack protected configurable Kyber processor on Artix-7 FPGA. Different components of the configurable Kyber processor employ different fault detection methodologies. For example, alongside the original state machine, a duplicated inverted-logic state machine was used to verify the control flow integrity of the Kyber processor. Specifically for the NTT operation in the Kyber processor, they employed a randomized memory addressing technique. Instead of using linear increments for the memory addresses of the RAMs and ROMs in the CT-BU, the addresses were randomized. This randomization of memory access in the CT-BU helps decorrelate the relationship between power consumption and NTT iterations, thereby significantly improving resistance against side-channel attacks on the NTT. Rafael et al.[23] proposed a locally masked NTT scheme on Artix-7 FPGA in which the input and output are masked with random twiddle factors. This approach effectively prevents the leakage of computational patterns, thereby enhancing resistance against side-channel attacks. Ravi et al.[24] implemented both local masking and memory access randomization techniques in their NTT design on the ARM Cortex-M4 platform to enhance resistance against side-channel attacks.

To the best of our knowledge, existing literature lacks any lightweight fault detection scheme that independently targets both the logic component (CT-BU) and the memory subsystem of the NTT. In this paper we propose two fault detection schemes for CT-BU and Memory Units (RAMs and ROMs) of the NTT core. Our fault detection scheme for the CT-BU targets Montgomery reduction unit. It is based on the RECO method but is neither RESO nor RENO; rather, it is a recomputation with a modular offset (`REMO`). It is important to note that the proposed `REMO` based fault-tolerant model can serve as a generic fault-resistant method for any Montgomery multiplication and reduction. To the best of our knowledge, the `REMO`-based fault detection in the CT-BU of the NTT is the first of its kind. We also propose a memory address rule checker for the RAMs and ROMs used in the NTT to detect any ambiguities in memory addressing. The contribution of the paper can be summerized as:

- This paper introduces a novel fault detection technique based on a modified word-wise Montgomery reduction algorithm, targeted for the Cooley-Tukey Butterfly Unit (CT-BU) in the NTT architecture. Unlike existing methods, the proposed `REMO` scheme integrates fault detection into the core arithmetic logic, offering protection without requiring significant implementation cost. The customizable word size (w) provides significant flexibility in usage of slices without compromising the fault detection efficiency. Although it is designed for the CT-BU, the versatility of this approach extends seamlessly to any hardware architecture performing modular polynomial multiplication using Montgomery reduction,
- This work also introduces the Memory Rule Checker (`Memory RC`), a lightweight and effective mechanism to ensure structural integrity in memory access patterns. Unlike traditional memory protection techniques that focus solely on content integrity, the proposed `Memory RC` monitors and validates the correctness of address sequences during all read/write operations of polynomial coefficients (Generate Matrix for Kyber), error vectors, secret vectors, etc in RAM and twiddle factor retrievals from ROM, across both forward and inverse NTT executions. To the best of our knowledge, this is the first approach that systematically targets address-level faults in memory-intensive cryptographic datapaths.
- The proposed lightweight fault detection framework introduces negligible implementation overhead across leading post quantum cryptographic schemes, including Kyber, Crystals-Dilithium, Falcon, and NTRU. Uniquely designed for modular arithmetic and memory bound operations, this model delivers high fault detection accuracy, even under diverse fault scenarios and varying numbers of corrupted bits. This is a robust protection with minimal overhead across multiple PQC standards in resource-constrained FPGA deployments.

The organization of the article is as follows: Sections 2 and 3 provide detailed descriptions of the proposed fault detection methods: `REMO` for the CT-BU and the Memory Rule Checker for the memory units, respectively. The detailed hardware architecture of the NTT, including the CT-BU, RAMs, and ROMs, along with the `REMO` and Memory Rule Checker, is presented in Section 4, while the results are discussed in Section 5. Finally, the conclusions are provided in Section 6.

2 | FAULT DETECTION IN CT-BU

The CT-BU is the most computationally intensive hardware block in any polynomial multiplication, as well as both forward and reverse NTT operations. As shown in Equ. 2 and Equ. 3, the CT-BU involves two major consecutive steps shown separately in line 9 and 10 of Algorithm 1.

$$U = \alpha[j + k] \tag{2}$$

$$V = \alpha[j + k + \frac{i}{2}] \times \omega \ mod \ q \tag{3}$$

Here the U and V are used to calculate the coefficients of inverse NTT $\overline{\alpha}$, ω is a primitive n^{th} root of unity and q is a prime number. The calculation of V involves a modular reduction on the product of a polynomial coefficient α and the primitive n^{th} root of unity ω, which is the most hardware-intensive and latency-critical operation in the NTT transformation. We use popular Montgomery reduction technique which allows an efficient hardware implementation of modular multiplication without computing modular reduction operation on the product of α and ω. However, the Montgomery reduction operation remains the most hardware-intensive process in the CT-BU and NTT transformation. We have chosen Montgomery reduction over Barrett reduction because Barrett requires more intermediate computations than Montgomery in word-wise form on FPGA. Muller et al.[25] show that Montgomery reduction often outperforms Barrett in FPGA implementations, especially for word-wise operations.

2.1 | REMO Method

In our fault detection method, rather than computing the Montgomery reduction on all l bits of the polynomial coefficient $\alpha(x)$ at once, we operate on smaller, fixed word sizes of w bits from the total l bits of $\alpha(x)$ where $w \leq l$. This modification of Montgomery reduction is required for three reasons.

- The partial recomputation technique for fault detection requires intermediate data.
- Lower value of w reduces the overhead of hardware resources as $w \leq l$.
- The tunable w can adjust the speed, resource usage, and power consumption of the design, offering architectural flexibility in optimization.

Algorithm 1 Iterative NTT Algorithm

```
Input: α(x), ω, q
Output: ᾱ(x)
for i = 2 to l by 2 × i do
    for j = 0 to i/2 − 1 do
        for k = 0 to n − 1 step i do
            ω_i = ω[2^((i−1)k)]
            U = α[j + k]
            V = MMRFD(α[j + k + i/2], ω, q)
            ᾱ[j + k] = U + V
            ᾱ[j + k + i/2] = U − V
        end for
        ω = ω · ω_i
    end for
end for
return ᾱ
```

2.2 | Modified Montgomery Reduction with REMO

The proposed fault detection method for CT-BU is recomputation based technique where encoding inputs α of Montgomery reduction can detect permanent and transient errors. This paper proposes a word-wise modified Montgomery reduction that takes two operands: the multiplicand α, the multiplier β, a modular base q, and the modular inverse q'. Here $\alpha = \alpha[j+k+\frac{i}{2}]$ and $\beta = \omega$. For the hardware implementation of Montgomery reduction, the number of bits l in α, β, q, and q' must be divisible by the word size w. As shown in Algorithm 2, lines 6 and 7 pad $w-p$ zeros to α and β to ensure that l is divisible by w. As shown in lines 9 and 10, the zero-padded α and β are then stored in α' and β', respectively. This modified Montgomery reduction involves two primary computations of μ_i and γ_i. Here μ_i depends on γ_i, aw_i and β'; γ_i depends on μ_i, aw_i and β' where $aw_i = (\alpha'_{iw+w-1}..\alpha'_{iw})$. The proposed fault detection method, which relies on recomputation, calculates additional values μ_i^f and γ_i^f using an encoded form of aw_i, denoted aw_i^f. As shown in modified Montgomery Reduction for Fault Detection (MMRFD) Algorithm 2, line 20, the value of aw_i^f is $(\alpha'_{iw+w-1}..\alpha'_{iw}) + K.q$. If there is no fault in $\alpha[j+k+\frac{i}{2}]$ or in ω, then γ_i^f with aw_i^f and γ_i with aw_i will be the same, where $\beta = \omega$. Conventional Montgomery reduction on l bits calculates $\alpha.\beta.R^{-1} \mod q$. In encoded form it calculates $(\alpha + k.q).\beta.R^{-1} \mod q$. Both these result must be the same as $k.q.\beta.R^{-1} \mod q$ term will be canceled out as k and R^{-1} are integers. However, the proposed modified word-wise Montgomery reduction operates on w bits instead of l bits in each loop iteration. This modified word-wise approach still computes the same intermediate values in γ for both the encoded and non-encoded forms.

Lemma 1. *If we divide l bit α' in w bit word-wise (segments), each word of α' can be expressed as: $aw_i = (\alpha'_{iw+w-1}..\alpha'_{iw})$ and the encoded word of α' is represented as: $aw_i^f = (\alpha'_{iw+w-1}..\alpha'_{iw}) + K.q$ where K is a constant and q is the modulus. Then γ_i and γ_i^f computed in the i^{th} loop from aw_i and aw_i^f respectively must be same.*

Proof. From algorithm 2, line 18 and 19 We have

$$\mu_i = [(\gamma_{w-1}, .., \gamma_0) + aw_i.(\beta'_{w-1}, .., \beta'_0)].q' \ \% \ 2^w$$
$$\gamma_i = [\gamma_i + aw_i(\beta'_{l-1}, .., \beta'_0) + \mu_i q]/2^w$$

Algorithm 2 Modified Montgomery Reduction for Fault Detection in Hardware with REMO: MMRFD(α, β, q)

```
Input: α(x) = (α_{l−1}, ..., α_0), β(x) = (β_{l−1}, ..., β_0), q = (q_{l−1}, ..., q_0)
       with R = w^l, gcd(q, w)
Output: γ, f
p ← l mod w
if p ≠ 0 then
    α′ ← Pad with (w − p) zeros ∥ α(x)
    β′ ← Pad with (w − p) zeros ∥ β(x)
else
    α′ ← α(x)
    β′ ← β(x)
end if
γ ← (0, ..., 0)
γ^f ← (0, ..., 0)
μ ← (0, ..., 0)
f ← (0, ..., 0)
for i = 0 to (l + p)/w − 1 do
    aw_i ← (α′_{iw+w−1}, ..., α′_{iw})
    μ_i ← ((γ_{w−1}, ..., γ_0) + aw_i · (β′_{w−1}, ..., β′_0)) · q′ mod 2^w
    γ_i ← (γ_i + aw_i · β + μ_i · q) /2^w
    aw_i^f ← (α′_{iw+w−1}, ..., α′_{iw}) + K · q
    μ_i^f ← ((γ^f_{w−1}, ..., γ^f_0) + aw_i^f · (β′_{w−1}, ..., β′_0)) · q′ mod 2^w
    γ_i^f ← (γ_i^f + aw_i^f · β + μ_i^f · q) /2^w
    if γ_i ≠ γ_i^f then
        f_i ← 1
    else
        f_i ← 0
    end if
end for
return γ, f
```

Now replace the aw_i by aw_i^f

$$\begin{aligned}
\mu_i &= [(\gamma_{w-1}, .., \gamma_0) + \underbrace{(aw_i + K.q)}_{aw_i^f}.(\beta'_{w-1}, .., \beta'_0)].q' \ \% \ 2^w \\
&= [(\gamma_{w-1}, .., \gamma_0) + aw_i.(\beta'_{w-1}, .., \beta'_0) + k.q.(\beta'_{w-1}, .., \beta'_0)]q' \% 2^w \\
&= [(\gamma_{w-1}, .., \gamma_0) + aw_i.(\beta'_{w-1}, .., \beta'_0)]q' + k.\overbrace{q.q'}^{-1}.(\beta'_{w-1}, .., \beta'_0) \% 2^w \\
&= [(\gamma_{w-1}, .., \gamma_0) + aw_i.(\beta'_{w-1}, .., \beta'_0)]q' - k.(\beta'_{w-1}, .., \beta'_0) \% 2^w
\end{aligned}$$

Therefore,

$$\mu_i + k.(\beta'_{w-1}, .., \beta'_0) = [(\gamma_{w-1}, .., \gamma_0) + aw_i.(\beta'_{w-1}, .., \beta'_0)]q' \% 2^w$$

Now,

$$\begin{aligned}
\gamma_i^f &= [\gamma_i^f + aw_i.(\beta'_{l-1}, .., \beta'_0) + \mu_i.q]/2^w \\
&= [\gamma_i + \underbrace{(aw_i + K.q)}_{aw_i^f}.(\beta'_{l-1}, .., \beta'_0) + \underbrace{[(\gamma_{w-1}, ., \gamma_0) + aw_i.(\beta'_{w-1}, ., \beta'_0)]q' - k.(\beta'_{w-1}, ., \beta'_0)]}_{u_i}.q]/2^w \\
&= [\gamma_i + aw_i.(\beta'_{l-1}, .., \beta'_0) + \mu_i.q]/2^w + q.\underbrace{\frac{k}{2^w}.[(\beta'_{l-1}, .., \beta'_0) - (\beta'_{w-1}, .., \beta'_0)]}_{t} \\
&= [\gamma_i + aw_i.(\beta'_{l-1}, .., \beta'_0) + \mu_i.q]/2^w + q.t \\
&= \gamma_i + q.t
\end{aligned}$$

As t is an integer, after applying the Montgomery transformation, the final value of γ_i^f is:

$$\gamma_i^f = \gamma_i^f.R \ \% \ q = (\gamma_i + q.t).R \ \% \ q = \gamma_i$$

□

2.3 | Hardware Architecture of REMO: γ_i^f & γ_i

The values of γ_i and γ_i^f shown in lines 19 and 22 of Algorithm 2 are computed by two hardware blocks named γ_i *Gen* and γ_i^f REMO, respectively. The γ_i *Gen* uses three multipliers ($\times$) and two adders (+), whereas the γ_i^f REMO requires four multipliers ($\times$) and three adders (+). The γ_i *Gen* and γ_i^f REMO use one right shifter to shift α' in each clock cycle, generating word-wise values aw_i. Here ω' is β' and ω'_0 $\beta'_{w-1}, .., \beta'_0$ of Algorithm 2. Modulus and division operations are computationally expensive on FPGA hardware. Therefore, the modulus operations: mod 2^w in Algorithm 2, as shown in line 18 and line 21, are implemented by retaining only the least significant w bits of μ_i and μ_i^f. On the other hand, the division operations: $/2^w$ in Algorithm 2, as shown in line 19 and line 22, are implemented by discarding the least significant w bits of γ_i and γ_i^f. These two approaches make *MMRFD* significantly lightweight. The Fig 1 shows the details hardware architecture of γ_i *Gen* and REMO: γ_i^f.

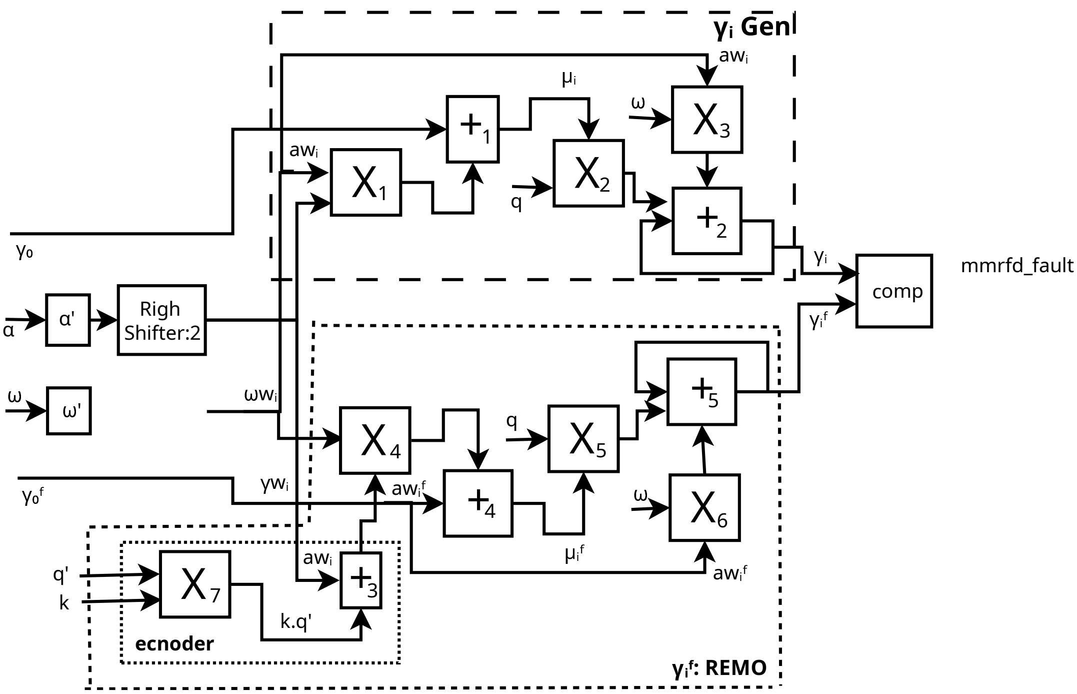

FIGURE 1 Hardware Architecture of REMO

2.3.1 | γ_i *Gen*

The γ_i *Gen* calculates lines 18 and 19 of Algorithm 2. The $\times_1$, $\times_2$ and $\times_3$ in γ_i *Gen* are used to multiply aw_i with ω'_0, μ_i with q and ω with aw_i respectively. The $+_1$ and $+_2$ adders are used to compute μ_i and γ_i respectively. If $p = 0$, the $+_2$ block provides the final output at the γ_i register after $\frac{l}{w}$ cycles. If $p \neq 0$, the $+_2$ block provides the final output at the γ_i register after $\frac{l}{w} + 1$ cycles.

2.3.2 | $REMO : \gamma_i^f$

The `REMO`: γ_i^f calculates lines 21 and 22 of Algorithm 2. The $\times_4$, $\times_5$ and $\times_6$ in `REMO`: γ_i^f are used to multiply aw_i^f with ω'_0, μ_i^f with q and ω with aw_i^f respectively. The additional multiplier and $\times_7$ is used to compute k, q' in the encoder block. The $+_4$ and $+_5$ adders are used to compute μ_i^f and γ_i^f respectively. One additional adder $+_3$ is used to calculate the encoded aw_i^f. If $p = 0$, the $+_5$ block provides the final output at the γ_i^f register after $\frac{l}{w}$ cycles. If $p \neq 0$, the $+_2$ block provides the final output at the γ_i^f register after $\frac{l}{w} + 1$ cycles. The comparator block (*comp*) compares γ_i^f and γ_i. If they are not equal, the signal *mmrfd_fault* is asserted.

3 | FAULT DETECTION IN MEMORY UNITS

NTT uses two types of memory units: RAMs and ROMs. RAMs are used to read and write polynomial coefficients during forward and inverse NTT operations. The constant terms in Learning With Errors (LWE)-based PQC algorithms, such as twiddle factors, are stored in ROM. Our fault detection method for the RAMs and ROMs of the NTT uses `Memory RC`, which is primarily constructed using `i-k RC` for RAMs and `i-j RC` for ROMs. These are presented in the next two sections based on the Kyber standard.

3.1 | Memory Address Protection in RAMs : `i-k RC`

We have implemented Kyber-768, in which Forward NTT operations are applied during key generation, decryption, and encryption to the secret vector samples s, the ephemeral secret vector r, the error samples e, and the vector of polynomials u, which is the result of a matrix-vector multiplication with the public key matrix A. The Inverse NTT (INTT) operations are performed on $\hat{u}$, $\hat{us}$, $\hat{tr}$, $\hat{A^T} \circ \hat{r}$, and $\hat{t^T} \circ \hat{r'}$. Here $\hat{u} = NTT(Decompress(cipher))$, $\hat{us} = \hat{u}.\hat{s}$, $\hat{tr} = \hat{t}.\hat{r}$. $\hat{A^T}$ is the Generate matrix during encryption. For further details on Kyber, please refer to FIPS 203[26]. Our Kyber implementation uses one RAM for each polynomial, resulting in a total of 10 RAMs used for the key generation, decryption, and encryption processes.

Block	NTTs	Kyber-512 (k = 2)			Kyber-768 (k = 3)			Kyber-1024 (k = 4)		
		# NTT Call	Total Hits on RAMs	Total ROM Hits	# NTT Call	Total Hits on RAMs	Total ROM Hits	# NTT Call	Total Hits on RAMs	Total ROM Hits
KeyGen	NTT(s)	2	4096	2048	3	6144	3072	4	8192	4096
	NTT(e)	2	4096	2048	3	6144	3072	4	8192	4096
	INTT	0	0	0	0	0	0	0	0	0
Encap	NTT(r)	2	4096	2048	3	6144	3072	4	8192	4096
	INTT($\hat{A^T} \circ \hat{r}$)	2	4096	2048	3	6144	3072	4	8192	4096
	INTT($\hat{t^T} \circ \hat{r}$)	1	2048	1024	1	2048	1024	1	2048	1024
Decap	NTT(r)	2	4096	2048	3	6144	3072	4	8192	4096
	NTT(u)	2	4096	2048	3	6144	3072	4	8192	4096
	INTT($\hat{u}$)	2	4096	2048	3	6144	3072	4	8192	4096
	INTT($\hat{us}$)	1	2048	1024	1	2048	1024	1	2048	1024
	INTT($\hat{tr}$)	1	2048	1024	1	2048	1024	1	2048	1024
total		17	**34,816**	**17,408**	24	**49,152**	**24,576**	31	**63,488**	**31,744**

TABLE 1 Pattern of Memory hits in Kyber Variants using i, j & k

In our Kyber, i and k indices are used to generate the addresses of 10 RAMs during read, write operations. In Table 1, we present the number of RAM hits using the address bus j during the key generation, decryption, and encryption processes for different variants of Kyber. The read and write operation on RAMs are considered as RAM hits. The data output of the RAMs is denoted by α. The i and k patterns in each loop iteration during forward and inverse NTT is structured and hierarchical, where the upper bound k are based on i. Therefore, we establish rules for each *CT-BU* iteration: $k \leq s_i$, where s_i is initialized to $n-1$ and is right-shifted by one bit every 256 iterations. The $i-k$ RC takes i and k indices from $i-j-k$ *Gen* and check the above mentioned rule. If the rule is not violated, the *ram fault* is set to 0; otherwise, it is set to 1.

3.2 | Memory Address Protection in ROM : `i-j RC`

Kyber needs a ROM to store Twiddle factors (ω), which required to read from ROM in each iteration of forward and inverse NTT. For more details about ω, please refer to FIPS 203[26]. In our Kyber variants, the i and j patterns in each loop iteration during forward and inverse NTT are also structured and hierarchical, where the upper bound of j depends on i, specifically $j \leq 2^i - 1$. The $i-j$ RC takes i and j indices from $i-j-k$ *Gen* and check the above mentioned rule. If the rules is not violated, the *rom fault* is set to 0; otherwise, it is set to 1.

4 | HARDWARE ARCHITECTURE OF NTT WITH CT-BU, REMO& MEMORY RC

As shown in Fig. 2, the CT-BU unit has 3 pipeline stages: (i) For buffering ω and U (ii) Calculation of V and (iii) Update elements of point-wise representation. Our $CT-BU$ inside NTT iterates $log_2 \times \frac{n}{2} - 1$ times where $n-1$ is the degree of input polynomial. For our Kyber $n = 256$. Therefore, the NTT in our Kyber iterates 1024 times. Line 8 of Algorithm 1 represents the 2^{nd} pipeline stage of the CT-BU, which computes the Montgomery modular reduction of q on the product of ω and $\alpha[j+k+\frac{i}{2}]$.

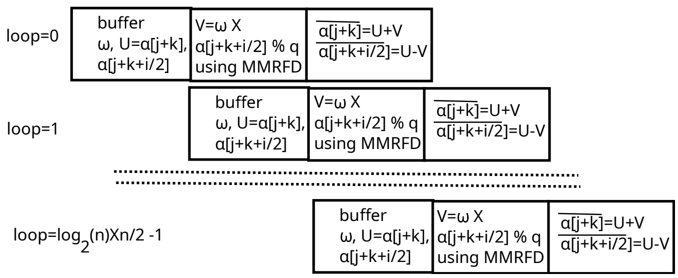

FIGURE 2 Pipeline Stages of CT-BU

As shown in Fig. 3, the polynomial coefficients and twiddle factors ω are stored in RAMs and ROM, which can be accessed by various hardware blocks such as the polynomial multiplier, polynomial adder, NTT and etc. The specific hardware block that accesses the memory depends on the requirements of the PQC algorithm. The *addr*, *rd_en*, *wr_en* and *din* of the memory block can be accessed by different hardware blocks by selecting inputs of the muxes controlled by *Control Unit*. Similarly using *demux*, different hardware blocks can read α from RAMs through *dout*. Then, α is connected to the $CT-BU$ for the U and V calculation (line 7 and 8 of Algorithm 1). The calculation of V depends on the *MMRFD* block which follows Algorithm 2 to multiply α and ω in a w word-wise fashion. The γ block shown in Fig. 3, takes w bits named aw_i at a time from the α. The γ^f block shown in Fig. 3, takes w bits named aw_i^f at a time from the α. The *encoder* block calculates aw_i^f (line 20 of Algorithm 2) and sends it to the γ^f block. The γ block executes u_i and γ_i (line 18 and line 19 of Algorithm 2). The γ^f block executes u_i^f and γ_i^f (line 21 and line 22 of Algorithm 2). The modules 2^w operation to calculate u_i and u_i^f in the γ block and γ^f block, respectively, are performed by restricting the size of u_i and u_i^f to w bits. The division by 2^w operation to calculate γ_i and γ_i^f in the γ block and γ^f block, respectively, are performed by removing w bits form the right side of the size of γ_i and γ_i^f. These two bitwise operations replace resource- and latency-intensive modulus and division operations, significantly reducing slice usage and delay. The *comp* block compares γ and γ^f. If both values match, $mmrfd\_fault = 0$ and V is used to calculate $\alpha[j+k]$ and $\alpha[j+k+\frac{i}{2}]$. Otherwise, $mmrfd\_fault = 1$ is initiated. The $\alpha[j+k]$ and $\alpha[j+k+\frac{i}{2}]$ are calculated by the *adder* and *sub* blocks respectively.

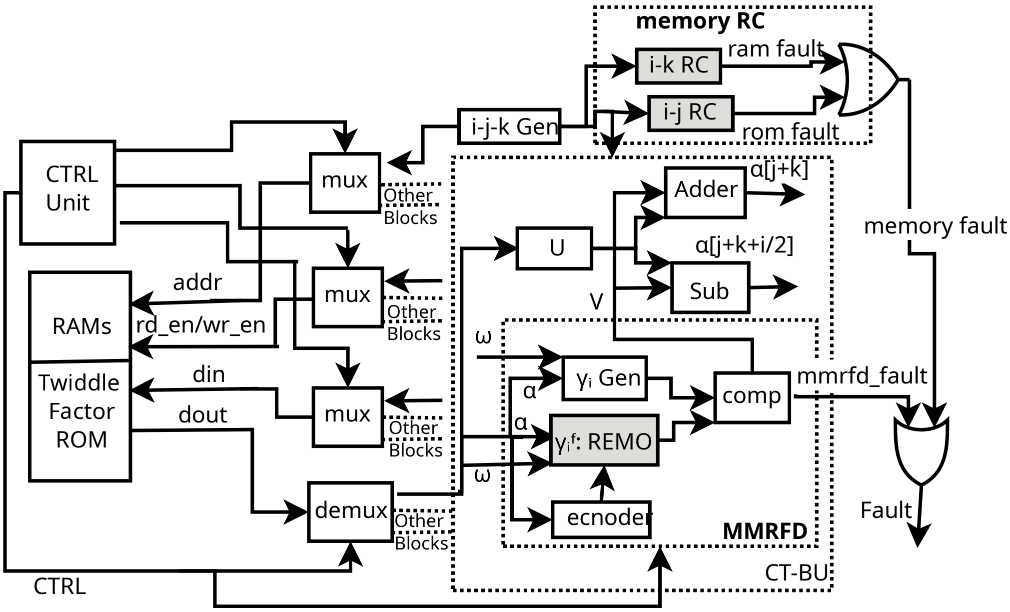

FIGURE 3 NTT Architecture With CT-BU, REMO & Memory RC

Architecture	n, q	SECs	Slices	LUTs	FFs	DSPs/ BRAMS	Power (mW)
Kyber Montgomery (Baseline)	256,	173	73	242	100	1/0	104
Kyber Montgomery (Protected)	3329	287	89	275	139	2/0	107
CRYSTALS-Dilithium Montgomery (Baseline)	256,	150	50	121	111	1/0	100
CRYSTALS-Dilithium Montgomery (Protected)	8380417	274	74	171	148	2/0	102
Falcon Montgomery (Baseline)	512,	136	36	84	70	1/0	99
Falcon Montgomery (Protected)	12289	248	48	128	103	2/0	101
NTRU Montgomery (Baseline)	2048,	132	32	84	78	1/0	106
NTRU Montgomery (Protected)	12289	249	49	122	120	2/0	110
Artix-7 (xc7a100tcsg324-3), w=4, clock=100MHz							

TABLE 2 Overhead of REMO & Memory RC in Different PQC Algorithms

5 | RESULTS & DISCUSSIONS

This section first covers the fault detection scheme's overheads, then its coverage.

5.1 | Overheads

The design is implemented on an *Artix*-7 (*xc*7*a*100*tcsg*324-3) FPGA using the *Vivado* 22.02 tool and the VHDL language. The proposed fault detection model is implemented in Kyber, Crystal Dilithium, Falcon and NTRU algorithms.The implementation costs of the proposed Fault Detection (*FD*) model, which includes both `REMO` and `Memory RC`, for the Kyber, CRYSTALS-Dilithium, Falcon, and NTRU algorithms are presented in Table 2. These costs are measured in terms of slices, LUTs, flip-flops, DSPs, and power consumption. Additionally, we calculate the Slice Equivalent Cost (SEC) as defined in Equ. 4, following the method described in[27].

$$SEC = Slices + DSPs \times 100 + BRAMS \times 200 \quad (4)$$

Our `REMO` model for the Kyber standard, with $n = 256$ and $q = 3329$, utilizes only 8 slices (comprising 15 LUTs and 8 Flip-Flops) and a single DSP block, as shown in Table 3. Additionally, `Memory RC` consumes 8 more slices. Table 4 shows a comparison of error coverage and overheads in terms of area, delay, and energy with existing fault detection literature. The number of extra slices and DSP required for *MMR* to detect faults using `REMO` are 8 and 1 respectively. The `Memory RC` also required 8 slices. Therefore, the total Area Overhead (*AO*) of fault detection of *NTT* is shown in Equ. 5.

$$AO = \frac{SEC\ of\ REMO + SEC\ of\ memory\ RC}{SEC\ of\ CT - BU} \times 100 \qquad (5)$$
$$= \frac{108 + 8}{1356} \times 100 = 8.5\%$$

As shown in Table 4, we compare our fault detection model with other fault detection solutions designed for NTT and other different computational units of cryptographic algorithms. While the solutions different computational units of cryptographic algorithms are not directly comparable to our proposed fault detection method, this comparison provides an overview of the implementation cost required to achieve a given level of error coverage. As shown in Table 4, this fault detection module has a slice overhead of 8.5% compared to the unprotected NTT. The fault detection hardware consumes only 3mW of power, resulting in a 1.8% power overhead compared to the unprotected NTT. The proposed fault detection method runs in parallel with the NTT component. Adopting this proposed fault detection logic into the NTT does not impact the critical path, clock period, or the number of clock cycles required for the baseline NTT. Consequently, the implementation incurs a 0% delay, 8.5% slice overhead and 1.8% energy overhead, which is highly reasonable and competitive with the existing literature for detecting 87.2% to 100% fault occurrences. It is to be noted that the our fault detection unit operates with a delayed clock compared to the main *NTT*. This is done to enable the detection of both transient and permanent faults.

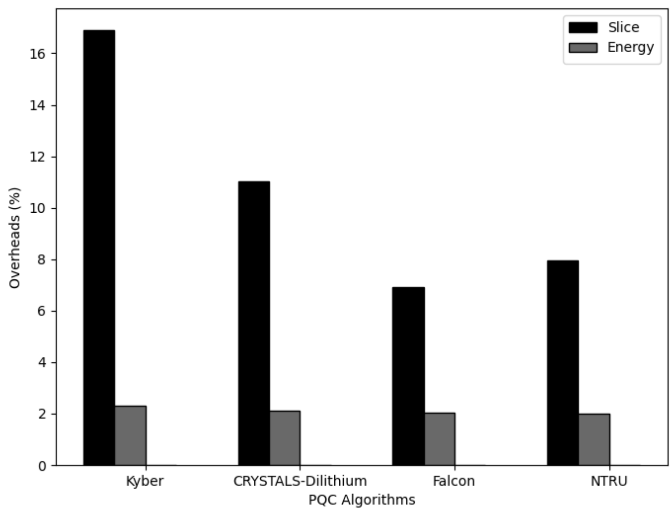

FIGURE 4 Overhead of REMO & Memory RC for Different PQC Algorithms

5.2 | Error Coverage

To measure the error coverage of the proposed `REMO` and `Memory RC` schemes, we simulated the fault injection process using Python on a *i*5 processor with 8 GB of RAM, running Ubuntu 24.04. Both the simulation process uses two fault modes: random faults and burst faults. The random mode flips η number of bits randomly, whereas the burst mode flips η number of consecutive bits. In both fault modes, bit flips mean that '0'(s) are turned into '1'(s) and '1'(s) are turned into '0'(s).

Block Names	SEC	Slices	LUTs	FFs	DSPs	BRAMs	Power (mW)	Critical Path (ns)
Kyber-768 (baseline)	3395	1795	6008	4404	2	7	425	9.79
NTT/INTT (baseline)	1356	556	1711	1204	2	3	163	9.39
CT BU (baseline)	685	285	778	207	2	1	127	9.39
MMR (baseline)	173	73	242	100	1	0	104	9.87
REMO	108	8	15	8	1	0	2	9.11
Memory RC	8	8	18	31	0	0	1	7.64
Artix-7 (xc7a100tcsg324-3), n=256, q=3329, l=12, w=4, clock=100MHz								

TABLE 3 Overhead of REMO & Memory RC for Kyber Standard

Work	Type of Fault Detection	Target HW	Baseline (for Overhead)	Overhead (%) Area	Overhead (%) Delay	Overhead (%) Energy	(%) Error Coverage
8	CRC5	sub, add of McEliece crypto	McEliece crypto	18.33	11.25	∼0	>99.9
7	REMO	x^y *mod n*	x^y *mod n*	0.8	0.27	0.65	97.1-100
9∗	look up table based output distribution check low cost/ standard/ expensive	CDT Error Sampler	CDT Error Sampler	9.09/ 77.4/ 18.2	NR	NR	NR
10	RESO	MAC unit of Saber/ NTRU/ FrodoKEM	Saber/ NTRU/ FrodoKEM	36.6/39.6/ 28.4	28.3/16.7/ 32.7	1.2/3.2/ ∼0	>99.9
11	Recomputing with swapped ciphertext	Galois Counter Mode	AES-GCM	4.9/6.7	NR	NR	100
13	Spatial duplication	NTRU	NTRU	6.22	NR	NR	100
12	1/2/3-bit parity	Goppa Arithmetic	McEliece	9.8/11.3/9.6	1.4/0.8/1	2.7/2.7/2.7	100
22	Randomized Memory Addr.	NTT	Kyber	NR	NR	NR	NR
23	Local Masked	NTT	NTT	NR	NR	NR	NR
24	Randomized Memory Addr. + Local Masked	NTT	NTT	NR	NR	NR	NR
19	Hamming A-O/ R-O	NTT Memory	Kyber	16.4/ 19.2	0/ 0	NR	NR
19	Hamming+ Parity A-O/ R-O	NTT Memory	Kyber	10.8/ 21.5	1.6/ 1.44	NR	NR
20	Polynomial Evaluation and Interpolation	CT-BU	Dilithium	NR	72	NR	NR
21	RENO Spartan7 v1/v2/v3	Butterfly Unit	NTT	20.2/15.3/21.5	8.46/15.88/13.71	15.6/7.6/11.2	99.51/99.67 /99.41
21	RENO Zynq v1/v2/v3	Butterfly Unit	NTT	24/7.5/17	9.32/19.66/ 21.78	20.47/13.27/ 17.26	99.51/99.67 /99.41
Our*	**REMO + Memory RC**	**CT-BU & Memories**	**CT-BU & Memories**	**16.9**	**0**	**2.3**	**87.2-100 (REMO), 50.7-100 (Memory RC)**
			NTT	**8.5**	**0**	**1.8**	
			Kyber-768	**3.18**	**0**	**0.7**	

Note: NR = Not Reported; ∗ =Eq. 4 is used to calculate area overhead.

TABLE 4 Overhead Comparison with literature

5.3 | Error Coverage of REMO

This simulation method utilized 1.5 million samples. The random and burst fault modes inject faulty bits into α, ω, and both α and ω. Table 5 shows that the fault detection efficiency varies from 87.2% to 100%, depending on the word size w, the number of faulty bits η and the fault mode. From Table 5 three conclusions can be drawn:

- The size of w has minimal impact on fault detection efficiency but significantly affects the area, as measured by SECs.
- As the number of faulty bits increases, the fault detection efficiency improves.
- The fault detection efficiency is higher in random mode compared to burst mode.

w	# fault bits(η)	Fault Detection Efficiency(%)						SECs of REMO (Eq. 4)
		fault in α		fault in ω		fault in α & ω		
		random	burst	random	burst	random	burst	
2	1	87.24	-	98.33	-	98.89	-	103
	3	96.72	93.67	100	99.99	100	100	
	5	99.03	96.83	100	100	100	100	
	11	99.95	99.61	100	99.99	100	100	
	17	99.99	99.95	100	100	100	100	
	23	99.99	99.99	100	100	100	100	
4	1	87.21	-	98.26	-	98.89	-	108
	3	95.94	90.34	99.01	99.97	100	100	
	5	98.3	93.68	100	100	100	100	
	11	99.68	97.63	100	99.99	100	100	
	17	99.87	99.27	100	100	100	100	
	23	99.98	99.79	100	100	100	100	
8	1	87.2	-	97.65	-	98.87	-	179
	3	94.43	88.51	99.99	99.81	100	100	
	5	96.55	89.98	100	99.99	100	100	
	11	98.19	94.28	100	100	100	100	
	17	99.03	97.1	100	99.99	100	100	
	23	99.85	98.33	100	99.99	100	100	
l=24, sample size=1.5 million								

TABLE 5 Error Detecting Efficient for η bit Random & Burst Flipping using REMO

5.4 | Error Coverage of Memory RC

To measure the error coverage of the `Memory RC`, we executed the key generation, decryption and encryption processes of Kyber-768 a total of 300 times. Each execution of Kyber-768 results in 49, 152 RAM accesses (as shown in Table 1) and 24, 576 ROM accesses to read the twiddle factors. Among these 300 executions, faults were injected using three different modes, with 100 executions allocated to each mode: (i) faults in both RAM and ROM, (ii) faults only in ROM, and (iii) faults only in RAM. Table 6 shows that the fault detection efficiency varies from 50.7% to 100%, depending on the number of faulty bits η and the fault mode.

6 | CONCLUSION

In this manuscript, we present a light wight modified Montgomery reduction integrated within a CT-BU for fault detection, capable of addressing both permanent and transient faults. It uses the `REMO` method. The results demonstrate that our fault detection scheme achieves a high fault detection rate with minimal resource and power overhead, without affecting the critical path of the original design. Although the fault detection scheme proposed in this paper is specifically designed for Montgomery reduction within the CT-BU, it can also be applied to any hardware implementing polynomial multiplication with modular

TABLE 6 Error Detecting Efficient for η bit Random & Burst Flipping using Memory RC for Kyber Standard

# Faulty Bits (η)	Fault Detection Efficiency(%)					
	Faulty addr in RAMs and ROM (j & k)		Faulty addr in ROM (j only)		Faulty addr in RAMs (k only)	
	Random (%)	Burst (%)	Random (%)	Burst (%)	Random (%)	Burst (%)
1	87.79	-	53.63	-	50.07	-
2	98.57	93.78	67.53	58.13	66.53	56.12
3	99.93	98.16	75.33	63.54	75.01	62.35
4	100	100	80.27	69.34	79.99	68.76
5	100	100	83.41	75.37	83.41	75.11
6	100	100	85.81	81.44	85.81	81.24
7	100	100	87.5	87.5	87.5	87.5

reduction, where Montgomery reduction is utilized. The `Memory RC` can detect between 50.7% and 100% of faults in the memory units used in the NTT. To the best of our knowledge, our fault detection method has one of the lowest slice overheads among existing fault-tolerant techniques in the literature on PQC. It is important to note that we have not explored fault detection methods for the contents of the memory units, as several efficient techniques such as hamming codes, parity bits, and CRC are already well-established in the literature[19] for protecting memory contents. The code of this work is uploaded to GitHub [‡].
Acknowledgment This publication has emanated from research conducted with the financial support of Taighde Éireann - Research Ireland under Grant number 13/RC/2077_P2 at CONNECT: the Research Ireland Centre for Future Networks.

References

1. Nikodem M. Error Prevention, Detection and Diffusion Algorithms for Cryptographic Hardware. In: 2007:127-134
2. Ravi P, Chattopadhyay A, D'Anvers JP, Baksi A. Side-channel and Fault-injection attacks over Lattice-based Post-quantum Schemes (Kyber, Dilithium): Survey and New Results. *ACM Trans. Embed. Comput. Syst.*. 2024;23(2). doi: 10.1145/3603170
3. Xagawa K, Ito A, Ueno R, Takahashi J, Homma N. Fault-Injection Attacks Against NIST's Post-Quantum Cryptography Round 3 KEM Candidates. In: Tibouchi M, Wang H., eds. *Advances in Cryptology – ASIACRYPT 2021*Springer International Publishing 2021; Cham:33–61.
4. Barenghi A, Breveglieri L, Koren I, Naccache D. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proceedings of the IEEE.* 2012;100(11):3056-3076. doi: 10.1109/JPROC.2012.2188769
5. al. eP. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In: Springer International Publishing 2017; Cham:513–533.
6. al. eP. Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems.* 2020;2020(3):307–335. doi: 10.13154/tches.v2020.i3.307-335
7. Aghapour S, Ahmadi K, Kermani MM, Azarderakhsh R. Efficient Fault Detection Architectures for Modular Exponentiation Targeting Cryptographic Applications Benchmarked on FPGAs. 2024.
8. Canto AC, Kermani MM, Azarderakhsh R. Reliable Constructions for the Key Generator of Code-based Post-quantum Cryptosystems on FPGA. *J. Emerg. Technol. Comput. Syst.*. 2022;19(1). doi: 10.1145/3544921
9. Howe J, Khalid A, Martinoli M, Regazzoni F, Oswald E. Fault Attack Countermeasures for Error Samplers in Lattice-Based Cryptography. In: 2019:1-5
10. al eACC. Error Detection Schemes Assessed on FPGA for Multipliers in Lattice-Based Key Encapsulation Mechanisms in Post-Quantum Cryptography. *IEEE Transactions on Emerging Topics in Computing.* 2023;11(3):791-797. doi: 10.1109/TETC.2022.3217006
11. Kermani MM, Azarderakhsh R. Reliable Architecture-Oblivious Error Detection Schemes for Secure Cryptographic GCM Structures. *IEEE Transactions on Reliability.* 2019;68(4):1347-1355. doi: 10.1109/TR.2018.2882484
12. Cintas Canto A, Kermani MM, Azarderakhsh R. Reliable Architectures for Composite-Field-Oriented Constructions of McEliece Post-Quantum Cryptography on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* 2021;40(5):999-1003. doi: 10.1109/TCAD.2020.3019987
13. Kamal AA, Youssef AM. Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks. *Journal of Cryptographic Engineering.* 2013;3(4):227–240. doi: 10.1007/s13389-013-0061-7

[‡] https://github.com/rourabpaul1986/NTT

14. Ahmadi K, Aghapour S, Kermani MM, Azarderakhsh R. Efficient error detection schemes for ECSM window method benchmarked on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.* 2023;32(3):592–596.

15. Ahmadi K, Aghapour S, Kermani MM, Azarderakhsh R. Error Detection Schemes for τ NAF Conversion within Koblitz Curves Benchmarked on Various ARM Processors. *Authorea Preprints.* 2023.

16. Dominguez-Oviedo A, Hasan MA. Error detection and fault tolerance in ECSM using input randomization. *IEEE Transactions on dependable and secure computing.* 2008;6(3):175–187.

17. Menezes AJ, Van Oorschot PC, Vanstone SA. *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press, 1996.

18. Kocher PC. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: . 1109. Springer 1996:104-113

19. al. eSk. Efficient, Error-Resistant NTT Architectures for CRYSTALS-Kyber FPGA Accelerators. In: 2023:1-6

20. Bauer S, Santis FD, Koleci K, Aghaie A. A Fault-Resistant NTT by Polynomial Evaluation and Interpolation. Cryptology ePrint Archive, Paper 2024/788; 2024.

21. al. eA, Canto . Error Detection Architectures for Hardware/Software Co-Design Approaches of Number-Theoretic Transform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* 2023;42(7):2418-2422. doi: 10.1109/TCAD.2022.3218614

22. Jati A, Gupta N, Chattopadhyay A, Sanadhya SK. A Configurable CRYSTALS-Kyber Hardware Implementation with Side-Channel Protection. *ACM Trans. Embed. Comput. Syst.*. 2024;23(2). doi: 10.1145/3587037

23. al. eR. Hardware Implementation and Security Analysis of Local-Masked NTT for CRYSTALS-Kyber. Cryptology ePrint Archive, Paper 2024/1194; 2024.

24. al. eR. On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT: A Performance Evaluation Study over Kyber and Dilithium on the ARM Cortex-M4. In: Springer-Verlag 2020; Berlin, Heidelberg:123–146

25. Müller R, Meier W, Wildfeuer CF. Area Efficient Modular Reduction in Hardware for Arbitrary Static Moduli. 2023.

26. National Institute of Standards and Technology . Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication 203; 2024.

27. Liu W, Fan S, Khalid A, Rafferty C, O'Neill M. Optimized Schoolbook Polynomial Multiplication for Compact Lattice-Based Cryptography on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.* 2019;27(10):2459-2463. doi: 10.1109/TVLSI.2019.2922999

28. al. eJA. Kyber: NIST Post-Quantum Cryptography Standardization. 2022. NIST PQC Standardization Round 3, available from: https://csrc.nist.gov/Projects/post-quantum-cryptography.

29. Aghapour S, Ahmadi K, Kermani MM, Azarderakhsh R. Efficient Fault Detection Architectures for Modular Exponentiation Targeting Cryptographic Applications Benchmarked on FPGAs. 2024.