Efficient Algorithm Level Error Detection for Number-Theoretic Transform used for Kyber Assessed on FPGAs and ARM

KASRA AHMADI, SAEED AGHAPOUR, MEHRAN MOZAFFARI KERMANI, AND REZA AZARDERAKHSH[*]

Polynomial multiplication stands out as a highly demanding arithmetic process in the development of post-quantum cryptosystems. The importance of the number-theoretic transform (NTT) extends beyond post-quantum cryptosystems, proving valuable in enhancing existing security protocols such as digital signature schemes and hash functions. CRYSTALS-KYBER stands out as the sole public key encryption (PKE) algorithm chosen by the National Institute of Standards and Technology (NIST) in its third round selection, making it highly regarded as a leading post-quantum cryptography (PQC) solution. Due to the potential for errors to significantly disrupt the operation of secure, cryptographically-protected systems, compromising data integrity, and safeguarding against side-channel attacks initiated through faults it is essential to incorporate mitigating error detection schemes. This paper introduces algorithm level fault detection schemes in the NTT multiplication using Negative Wrapped Convolution and the NTT tailored for Kyber Round 3, representing a significant enhancement compared to previous research. We evaluate this through the simulation of a fault model, ensuring that the conducted assessments accurately mirror the obtained results. Consequently, we attain a notably comprehensive coverage of errors. Furthermore, we assess the performance of our efficient error detection scheme for Negative Wrapped Convolution on FPGAs to showcase its implementation and resource requirements. Through implementation of our error detection approach on Xilinx/AMD Zynq Ultrascale+ and Artix-7, we achieve a comparable throughput with just a 9% increase in area and 13% increase in latency compared to the original hardware implementations. Finally, we attained an error detection ratio of nearly 100% for the NTT operation in Kyber Round 3, with a clock cycle overhead of 16% on the Cortex-A72 processor.

CCS Concepts: • **Security and privacy** → Embedded systems security.

Additional Key Words and Phrases: Fault detection, Fast Fourier Transform (FFT), Kyber, Negative Wrapped Convolution, Number-theoretic transform (NTT).

ACM Reference Format:
Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2024. Efficient Algorithm Level Error Detection for Number-Theoretic Transform used for Kyber Assessed on FPGAs and ARM. 1, 1 (September 2024), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Fast Fourier Transform (FFT) [11] algorithms which are used to compute the Discrete Fourier Transform (DFT) have various applications, ranging from digital signal processing to the efficient multiplication of large integers. When the

[*]K. Ahmadi, S. Aghapour, and M. Mozaffari-Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA. e-mails: {ahmadi1, aghapour, mehran2}@usf.edu.
R. Azarderakhsh is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA. e-mail: razarderakhsh@fau.edu.

Author's address: Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh.

Manuscript submitted to ACM

coefficients of the polynomial are specifically chosen from a finite field, the resulting transform is known as the Number Theoretic Transform (NTT) [12], and it can be computed using FFT algorithms designed for operations within this particular finite field. An efficient approach for polynomial multiplication, NTT holds significant importance in post-quantum cryptosystems, e.g., lattice-based cryptosystems which are regarded as a leading contender for quantum-secure public key cryptography, primarily because of its broad applicability and security proofs grounded in the worst-case hardness of established lattice problems.

In 2022, NIST unveiled the Round 3 outcomes of the Post-Quantum Cryptography standardization Process, which featured four chosen algorithms (CRYSTALS-KYBER , CRYSTALS-DILITHIUM, Falcon, SPHINCS+). These are called formally after Aug. 2024 with respect to Federal Information Processing Standards (FIPS) as FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard, FIPS 204, Module-Lattice-Based Digital Signature Standard, and FIPS 205, Stateless Hash-Based Digital Signature Standard while Falcon is yet to get a FIPS number [35], [36], and [37]. Out of the chosen algorithms, CRYSTALS-KYBER, also referred to as Kyber [7] and [35], stands out as the sole algorithm for both public key encryption and key establishment that relies on the challenge of the module learning with errors (MLWE) problem.

The NTT has proven to be a potent tool that facilitates the computation of this operation with quasi-linear complexity $O(n.lgn)$. Several recent studies have been conducted on optimizing the NTT [6], [17],[21], [22], [28], [29], [33], [48], and [50]. In addition to polynomial multiplication, the use of the NTT has the potential to significantly enhance existing schemes by improving their security parameters. NTT is widely used in signature schemes, hash functions, and identification schemes. As a result, incorporating efficient error detection mechanisms in the NTT for polynomial multiplication will not only improve the security and reliability but also mitigate the risk of fault attacks in the respective algorithms in post-quantum cryptography. Moreover, such schemes could at least alleviate the attack surface or be add-ons to other approaches. Kyber and Dilithium employ polynomial multiplication over $\mathbb{Z}[X]/(X^n + 1)$ utilizing the NTT. Integrating effective error detection schemes into the NTT used in Kyber will enhance security, reliability, and mitigate the risk of fault attacks in post-quantum cryptography algorithms.

1.1 Related works

Several previous studies have concentrated on the implementation and fault detection in different arithmetic components of both classical and post-quantum cryptography [9], [18], [30], [42], and [45]. We categorize relevant studies into three subsets. The initial category centers on fault attacks and countermeasures applied to post-quantum cryptographic schemes, particularly those extensively utilizing the NTT. Numerous prior studies have explored fault attacks with the objective of gaining access to or compromising secret keys [15], [19], [20], [39], [43], and [49]. Ravi *et al.* [41] presented the first fault injection analysis of the NTT. The authors found a significant flaw in the way the NTT is implemented in the pqm4 library [25]. The identified vulnerability, referred to as twiddle-pointer, is exploited to demonstrate practical and effective attacks on Kyber and Dilithium [3]. In [40], the authors proposed a fault attack aiming the NTT operation during the key generation and encapsulation routine of the Kyber. The fault attack is achieved by zeroize all the twiddle factors used in the NTT operation. When focusing on applying this technique to the NTT regarding secrets or errors, it can significantly diminish the randomness of the secret or error and reduce the entropy. Several other authors have reported employing fault injection on structured lattice-based schemes as a foundation for attacks [5], [8], and [34] threatening the security of implementations. Regarding active attacks, although some prior research addressed fault injection, Espitau et al. [16] introduced loop-abort faults in several lattice-based cryptosystems, including CRYSTALS-Kyber. In this attack, a fault is injected into the cryptosystem, causing the loop responsible for sampling random

Gaussian secret coefficients to terminate early. This early termination leads to the generation of lower-dimensional secrets, which can then be leveraged for a key recovery attack. In 2021, Pessl and Prokop [38] proposed an attack that involves a single instruction-skipping fault during the decoding process. Their fault simulations showed that at least 6,500 faulty decapsulation attempts are needed to fully recover the key for Kyber512 operating on a Cortex M4. In the same year, Hermelink et al. [20] combined fault injections with chosen-ciphertext attacks on CRYSTALS-Kyber, suggesting that their attack could bypass defenses such as decoder shuffling. Their findings demonstrated successful secret key recovery using 7,500 inequalities for Kyber-512, 10,500 for Kyber-768, and 11,000 for Kyber-1024. Delvaux [13] refined the side-channel attack (SCA) from [20], making it simpler to execute and more challenging to defend against. In [27], the authors proposed a new fault attack on the SCA-secure masked decapsulation algorithm for generic LWE-based KEMs and detailed the attack for Kyber. Jendral [23] presented an attack on CRYSTALS-Dilithium implementation on ARM Cortex-M4 using fault injection. Krahmer *et al.* [26] presented two key-recovery fault attacks on Dilithium's signing procedure.

The focus of the second category lies in offering fault detection methodologies pertaining to the algorithmic level of classical cryptographic schemes. In [2], the authors suggested an effective algorithmic-level error detection for the ECSM window method, aiming to identify both permanent and transient errors. In [1], the authors presented algorithm level error detection scheme designed for Montgomery ladder ECSM algorithm utilized in non-supersingular elliptic curves.

The third category focuses on developing fault detection schemes specifically for the NTT. In [4], the authors introduced a method for safeguarding the NTT against fault attacks. Mishra *et al.* [32] introduced a principle for countermeasures that is based on cryptographic guarantees rather than relying on ad hoc methods, aiming to offer measurable protection against the previously mentioned fault attacks on lattice-based schemes. In [47], the authors integrate the advantages of bit slicing, a software implementation technique where a datapath of an n-bit processor is treated as n parallel single-bit datapaths, to devise a fault countermeasure for the NTT used in Dilithium [14]. Sarker *et al.* [44] and [46], presented error detection architectures of the NTT based on recomputation with encoded operands. The authors achieved high error coverage with low area overhead. However, due to the recomputation process, their latency is doubled. We note that despite high error coverage, for fast implementations, these works might increase the total time to levels not acceptable. Additionally, Cintas-Canto et al. [10] introduced error detection schemes for lattice-based KEMs using recomputation techniques and implemented these schemes on FPGA. Below, we present our major contributions in this work.

1.2 Our Major Contributions

- Our proposed error detection schemes donot depend on recomputation; instead, it relies on algorithm-level coherency. As a result, it offers higher speed, lower latency, and reduced area compared to previous approaches.
- We have proposed an algorithm level fault detection scheme of the NTT multiplication using Negative Wrapped Convolution which is widely used in lattice-based cryptography. We achieved high error coverage with less area overhead and latency compared to previous works. This was achieved through the implementation of algorithm-level error detection for the NTT section and partial recomputation for pre-computation section.
- As the Negative Wrapped Convolution method does not work for the NTT multiplication used in Kyber Round 3, we have introduced an algorithm level error detection scheme for the NTT multiplication tailored for Kyber Round 3. Implementing algorithm level error detection for the NTT module enabled us to achieve substantial

error coverage with minimal overhead compared to previous works. This method can be integrated into Kyber Round 3 reference implementation.

- We performed simulations for single and burst fault injection using our proposed schemes. The simulation outcomes demonstrated that our approach is capable of detecting diverse types of faults with high error coverage and offers protection against the NTT fault attack presented in [41] and those with respective fault models.
- Our error detection method for Negative Wrapped Convolution was deployed on Xilinx/AMD Zynq Ultrascale+ , and Artix-7. The results of our implementations indicate that we can attain a significantly high level of error coverage with only a 13% increase in latency and a 9% area bloat. Our proposed error detection method for Kyber Round 3 is implemented on Intel Core-i7, and Cortex-A72 with additional overhead of 28% and 16% in terms of clock cycle overhead.

Our work is structured as follows: Section 2 discusses Negative Wrapped Convolution and the NTT used in Kyber. The presented error detection schemes on Negative Wrapped Convolution and the NTT tailored for Kyber are described in Section 3. Section 4 is divided into two subsections. First, the fault model in this work is discussed; next, the fault simulation is performed to assess error detection rate. We apply such fault detection schemes into the reference Kyber implementation and Negative Wrapped Convolution in Section 5 to benchmark the different overheads. Finally, our conclusions are presented in Section 6.

2 PRELIMINARIES

2.1 Number Theoretic Transform (NTT)

Multiplication of two polynomials f and g take quadratic complexity $O(n^2)$ utilizing school book algorithm. However, by using the NTT, it can be reduced to quasi-linear complexity $O(n.lgn)$. The NTT represents a specialized version of the Discrete Fourier Transform (DFT), utilizing a coefficient ring chosen from a finite field that encompasses the necessary roots of unity. We denote the ring $\mathbb{Z}[X]/(X^n+1)$ by R and the ring $\mathbb{Z}_q[X]/(X^n+1)$ by R_q. Consider ω to be a primitive n-th root of unity in $\mathbb{Z}_q$ which $\omega^n \equiv 1 \pmod q$ where $q \equiv 1 \pmod{2n}$, q is a prime number, and n is power of 2. The NTT of a polynomial $f \in R_q$ is defined as $\hat{f} = \text{NTT}(f) = \sum_{j=0}^{n-1} f[j]\omega^{ij} \pmod q$ and the inverse NTT $f = \text{NTT}^{-1}(\hat{f}) = \sum_{j=0}^{n-1} n^{-1}\hat{f}[j]\omega^{-ij} \pmod q$. The NTT of a sequence f is derived as in the form of matrix multiplication described in (1). The Cooley-Tukey butterfly algorithm can be used to efficiently implement the forward NTT.

$$\hat{f} = \text{NTT}(f) = \begin{bmatrix} \omega^0 & \omega^0 & ... & \omega^0 \\ \omega^0 & \omega^1 & ... & \omega^{n-1} \\ \omega^0 & \omega^2 & ... & \omega^{2(n-1)} \\ . & . & ... & . \\ \omega^0 & \omega^{n-1} & ... & \omega^{(n-1)^2} \end{bmatrix} * \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ . \\ f(n-1) \end{bmatrix}. \tag{1}$$

In Algorithm 1, the iterative NTT implementation derives the NTT of a specified polynomial f. The bit_reverse(k) function (presented in Line 5) rearranges the input k where the new placement of elements is determined by reversing the binary representation of the operand. Lines 9 and 10 perform the butterfly operation. Every butterfly module, taking inputs a and b and producing outputs c and d, executes the following butterfly computation: $c = a + b\,\omega^k$ and $d = a - b\,\omega^k$. Kyber's reference implementation uses Montgomery multiplication to achieve efficient modular multiplication and improve performance by converting numbers into Montgomery form.

Algorithm 1 Iterative NTT [51]

```
Input: $f \in \mathbb{Z}_q$ of length $n = 2^k$, $\omega$ is primitive $n$-th root of unity
Output: $\hat{f} = \mathrm{NTT}(f)$ in bit-reversed order
$\hat{f} = f$
for $s = k$ to 1:
    $m = 2^s$
    for $k = 0$ to $2^{k-s} - 1$:
        $\omega = \omega^{\textbf{bit-reverse}(k)\,.\,m/2}$
        for $j = 0$ to $m/2 - 1$:
            $u = \hat{f}[k.m + j]$
            $t \equiv \omega\,.\,\hat{f}[k.m + j + m/2] \bmod q$
            $\hat{f}[k.m + j] \equiv (u + t) \bmod q$
            $\hat{f}[k.m + j + m/2] \equiv (u - t) \bmod q$
return $\hat{f}$
```

Algorithm 2 The pre-process step for the NTT calculations

```
Input: $f \in \mathbb{Z}_q$ of length $n$, $\psi$ primitive $2n$-th root of unity
Output: $\tilde{f} = \text{pre-process}(f)$
for $i = 0$ to $n - 1$:
    $\tilde{f}[i] = f[i]\,.\,\psi^i$
return $\tilde{f}$
```

Algorithm 3 The Post-process step for the NTT calculations

```
Input: $f \in \mathbb{Z}_q$ of length $n$, $\psi$ primitive $2n$-th root of unity
Output: $\tilde{f} = \text{post-process}(f)$
for $i = 0$ to $n - 1$:
    $\tilde{f}[i] = f[i]\,.\,\psi^{-i}$
return $\tilde{f}$
```

2.2 Negative Wrapped Convolution

Let $f, g \in R_q$. Computing $h = f\,.\,g \bmod x^n + 1$ requires applying the NTT of length $2n$ and n zeros to be extended to f and g. This essentially doubles the input length and also necessitates an explicit reduction modulo $X^n + 1$. This problem can be resolved by utilizing Negative Wrapped Convolution, which prevents the doubling of input length [31]. Let ψ be a primitive $2n$-th root of unity in $\mathbb{Z}_q$ where $\psi^2 = \omega$.

Moreover, there exists a Pre-process module which is described in Algorithm 2. The Negative Wrapped Convolution of f and g is defined as $h = \text{post-process}(\mathrm{INTT}(\mathrm{NTT}(\tilde{f}) \bigcirc \mathrm{NTT}(\tilde{g})))$ where $\tilde{f} = \text{pre-process}(f)$, $\tilde{g} = \text{pre-process}(g)$, and the symbol $\bigcirc$ represents component-wise multiplication. We can achieve Post-process function by changing ψ to ψ^{-1}. Post-process function is described in Algorithm 3.

2.3 Polynomial Multiplication Utilizing the NTT in Kyber

To use Negative Wrapped Convolution method, we require that $2n \mid (q - 1)$. Regarding the parameters used in Kyber with the prime $q = 3329$ and $n = 256$ the base field $\mathbb{Z}_q$ contains 256-th roots of unity but not 512-th roots. Therefore, we cannot use the method used in [31]. To overcome this problem, the polynomial $X^{256} + 1$ of R factor into 128 polynomials

Algorithm 4 Kyber NTT

```
Input: $r \in \mathbb{Z}_q$ of length $n = 256$, $\omega = 17$ is primitive $n$-th root of unity
Output: NTT$(r)$ in bit-reversed order
1: $\widetilde{r} = r$
2: $j = 0$
3: $k = 0$
4: for ($s = 128; s >= 2; s = s/2$):
5:     for ($i = 0; i < 256; i = j + s$):
6:         $zeta = \omega^{(\textbf{bit-reverse}(k))}$
7:         $k = k + 1$
8:         for ($j = i; j < i + s; j + +$)
9:             $t = \text{fqmul}(zeta, \widetilde{r}[j + s])$
10:            $\widetilde{r}[j + s] = \widetilde{r}[j] - t \bmod q$
11:            $\widetilde{r}[j] = \widetilde{r}[j] + t \bmod q$
12: return $\widetilde{r}$
```

of degree 2 modulo q. The polynomial $X^{256} + 1$ can be written as

$$X^{256} + 1 = \prod_{i=0}^{127} (X^2 - \omega^{2i+1}) = \prod_{i=0}^{127} (X^2 - \omega^{2b_7(i)+1}),$$

where $b_7(i)$ is the bit reversal of the 7-bit i. Therefore, the NTT of f is a vector of 128 polynomials of degree one.

$$\hat{f} = \text{NTT}(f) = (\hat{f}_0 + \hat{f}_1 X, \hat{f}_2 + \hat{f}_3 X, ..., \hat{f}_{254} + \hat{f}_{255} X),$$

with

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2j} \omega^{(2br_7(i)+1)j}, \tag{2}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2j+1} \omega^{(2br_7(i)+1)j}. \tag{3}$$

We can compute $h = \hat{f} \circ \hat{g} = \text{NTT}^{-1}(\text{NTT}(f) \circ \text{NTT}(g))$ consisting of the 128 products of linear polynomials

$$\hat{h}_{2i} + \hat{h}_{2i+1} X = (\hat{f}_{2i} + \hat{f}_{2i+1} X)(\hat{g}_{2i} + \hat{g}_{2i+1} X) \bmod (X^2\text{-}\omega^{2b_7(i)+1}). \tag{4}$$

The iterative NTT implementation of Kyber is described in Algorithm 4. fqmul function (presented in Line 9 of Algorithm 4) performs multiplication of two operand and Montgomery reduction afterward.

3 PROPOSED ERROR DETECTION SCHEME

3.1 Negative Wrapped Convolution

In this section, we present our error detection schemes on the sub-block in the Negative Wrapped Convolution that uses the NTT and pre-process modules, $\text{NTT}(\tilde{f}) \bigcirc \text{NTT}(\tilde{g})$ where $\tilde{f}$ = pre-process(f) and $\tilde{g}$ = pre-process(g). We have devised an error detection scheme at the algorithm level on the component-wise NTT multiplication sub-block, $\text{NTT}(\tilde{f}) \bigcirc \text{NTT}(\tilde{g})$, as well as an error detection scheme for the pre-process function, involving careful utilization of recomputation using shifted operands. This error detection scheme can be applied to designs which require polynomial multiplication using the NTT. We used Negative Wrapped Convolution in our proposed error detection scheme.

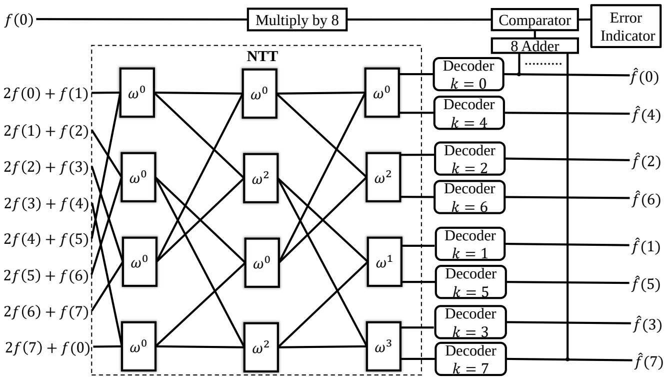

Fig. 1. Concurrent error detection scheme for the NTT operation.

3.1.1 Component-wise NTT Multiplication. Our focus is on $\mathrm{NTT}(\tilde{f}) \bigcirc \mathrm{NTT}(\tilde{g})$ operation in this section. Jou *et al.* [24] proposed an algorithm level error detection scheme on FFT network. Their proposed error detection scheme is depicted in Fig. 1. In order to achieve an efficient scheme, we have adapted the aforementioned approach to the component-wise NTT multiplication. The primary focus is to use an error detection scheme with small overhead and high error detection ratio with respect to efficient realizations of the NTT. From the NTT definition, we can use the relations presented below:

$$\hat{f}(0) = \sum_{j=0}^{n-1} \tilde{f}(j)\omega^0 = \sum_{j=0}^{n-1} \tilde{f}(j), \quad j = 0, 1, ..., n-1, \tag{5}$$

$$\hat{g}(0) = \sum_{j=0}^{n-1} \tilde{g}(j)\omega^0 = \sum_{j=0}^{n-1} \tilde{g}(j), \quad j = 0, 1, ..., n-1. \tag{6}$$

The result of the NTT multiplication for index 0 can be written as:

$$\hat{f}(0)\hat{g}(0) = \sum_{j=0}^{n-1} \tilde{f}(j) \sum_{j=0}^{n-1} \tilde{g}(j). \tag{7}$$

We note that (7) is used in our proposed error detection scheme. If we rotate the input sequence of (1) by one, then we get: $\begin{bmatrix} \hat{f}(0) \\ \hat{f}(1) \\ \hat{f}(2) \\ . \\ \hat{f}(N-1) \end{bmatrix} = \rho * \begin{bmatrix} \tilde{f}(1) \\ \tilde{f}(2) \\ . \\ \tilde{f}(N-1) \\ \tilde{f}(0) \end{bmatrix}$ or $\begin{bmatrix} \hat{f}(0)/\omega^0 \\ \hat{f}(1)/\omega^1 \\ \hat{f}(2)/\omega^2 \\ . \\ \hat{f}(n-1)/\omega^{(n-1)} \end{bmatrix} = \theta * \begin{bmatrix} \tilde{f}(1) \\ \tilde{f}(2) \\ . \\ \tilde{f}(n-1) \\ \tilde{f}(0) \end{bmatrix}$ where the symbol $*$ denotes matrix multiplication, $\rho = \begin{bmatrix} \omega^0 & \omega^0 & ... & \omega^0 \\ \omega^1 & \omega^2 & ... & \omega^0 \\ \omega^2 & \omega^4 & ... & \omega^0 \\ . & . & ... & . \\ \omega^{n-1} & \omega^{2(n-1)} & ... & \omega^0 \end{bmatrix}$ and $\theta = \begin{bmatrix} \omega^0 & \omega^0 & ... & \omega^0 \\ \omega^0 & \omega^1 & ... & \omega^{n-1} \\ \omega^0 & \omega^2 & ... & \omega^{2(n-1)} \\ . & . & ... & . \\ \omega^0 & \omega^{n-1} & ... & \omega^{(n-1)^2} \end{bmatrix}$.

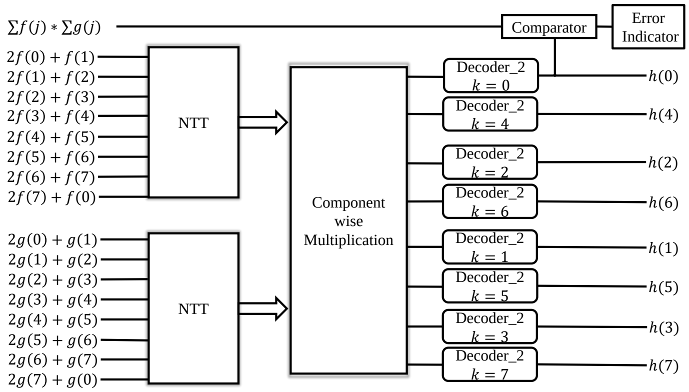

Fig. 2. Proposed algorithm level error detection scheme for the NTT multiplication module using Negative Wrapped Convolution.

From $\theta * \begin{bmatrix} \alpha\tilde{f}(0) \\ \alpha\tilde{f}(1) \\ \alpha\tilde{f}(2) \\ . \\ \alpha\tilde{f}(n-1) \end{bmatrix} + \theta * \begin{bmatrix} \beta\tilde{f}(1) \\ \beta\tilde{f}(2) \\ . \\ \beta\tilde{f}(n-1) \\ \beta\tilde{f}(0) \end{bmatrix}$, we get the following:

$$\begin{bmatrix} \hat{f}(0)(\alpha+\beta\omega^{-0}) \\ \hat{f}(1)(\alpha+\beta\omega^{-1}) \\ \hat{f}(2)(\alpha+\beta\omega^{-2}) \\ . \\ \hat{f}(n-1)(\alpha+\beta\omega^{-(n-1)}) \end{bmatrix} = \theta * \begin{bmatrix} \alpha\tilde{f}(0)+\beta\tilde{f}(1) \\ \alpha\tilde{f}(1)+\beta\tilde{f}(2) \\ . \\ \alpha\tilde{f}(n-2)+\beta\tilde{f}(n-1) \\ \alpha\tilde{f}(n-1)+\beta\tilde{f}(0) \end{bmatrix}, \tag{8}$$

where α and β in (8) are scalars which can be selected arbitrarily. Let us denote $\tilde{f}^m$ and $\tilde{g}^m$, respectively, as the inputs which are shifted by m. In other words, we can achieve:

$$\hat{f} = \frac{1}{(\alpha+\beta\omega^{-k})} NTT(\alpha\tilde{f} + \beta\tilde{f}^1), \tag{9}$$

$$\hat{g} = \frac{1}{(\alpha+\beta\omega^{-k})} NTT(\alpha\tilde{g} + \beta\tilde{g}^1). \tag{10}$$

Let us denote h as the component-wise NTT multiplication of $\tilde{f}$ and $\tilde{g}$. We can get:

$$h = NTT(\tilde{f}) \bigcirc NTT(\tilde{g}) = \frac{1}{(\alpha+\beta\omega^{-k})^2} NTT(\alpha\tilde{f} + \beta\tilde{f}^1) \bigcirc NTT(\alpha\tilde{g} + \beta\tilde{g}^1). \tag{11}$$

The proposed error detection scheme is depicted in Fig. 2. The Decoder_2 sub-block multiplies its input with $\frac{1}{(\alpha+\beta\omega^{-k})^2}$, which is the inverse of $(\alpha+\beta\omega^{-k})^2$ in $\mathbb{R} = \mathbb{Z}[X]/(X^n+1)$. The scheme for error detection involves comparing $h(0)$, obtained from (11), with $\sum_{j=0}^{n-1} \tilde{f}(j) \sum_{j=0}^{n-1} \tilde{g}(j)$ obtained from (7).

3.1.2 Pre-processor. The pre-processor module performs element-by-element multiplication on two input arrays. In proposing the error detection scheme, we have applied an additional recomputing with shifted operands. As shown in

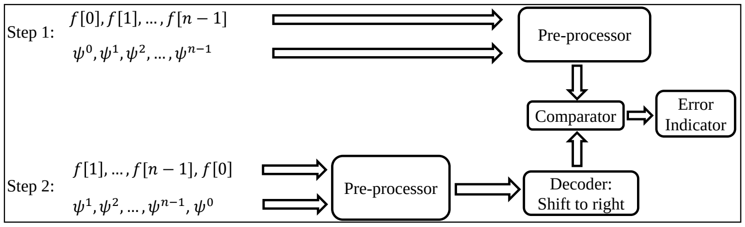
Step 1:
$f[0], f[1], ..., f[n-1]$
$\psi^0, \psi^1, \psi^2, ..., \psi^{n-1}$
Pre-processor
Comparator
Error Indicator
Step 2:
$f[1], ..., f[n-1], f[0]$
$\psi^1, \psi^2, ..., \psi^{n-1}, \psi^0$
Pre-processor
Decoder: Shift to right

Fig. 3. Proposed error detection scheme for pre-process module through recompuation with shifted operands.

Fig. 3, the encoding and decoding modules constitute shifting which is free in hardware. The pre-processing module considers a slight overhead linked to the NTT multiplication. As a result, the error detection scheme employing recomputation does not notably impact the overall performance.

3.2 Kyber

To apply the Negative Wrapped Convolution method, it is necessary that $2N \mid (q-1)$. For the Kyber parameters with the prime $q = 3329$ and $N = 256$, the base field $\mathbb{Z}_q$ includes 256-th roots of unity but lacks 512-th roots. Therefore, Negative Wrapped Convolution cannot be used in Kyber. We proposed below our error detection scheme on the Kyber NTT module (equations (2) and (3)).

3.2.1 *NTT Module.*

By expressing equations (2) and (3) as a matrix multiplication, we obtain the following relation where $N = 256$ with respect to Kyber parameters:

$$f'_{even} = \begin{bmatrix} \hat{f}(0) \\ \hat{f}(2) \\ . \\ \hat{f}(252) \\ \hat{f}(N-2) \end{bmatrix} = \partial * f_{even}, \text{ and } f'_{odd} = \begin{bmatrix} \hat{f}(1) \\ \hat{f}(3) \\ . \\ \hat{f}(253) \\ \hat{f}(N-1) \end{bmatrix} = \partial * f_{odd}, \text{ where } f_{even} = \begin{bmatrix} f(0) \\ f(2) \\ . \\ f(252)) \\ f(N-2) \end{bmatrix},$$

$$f_{odd} = \begin{bmatrix} f(1) \\ f(3) \\ . \\ f(253) \\ f(N-1) \end{bmatrix}, \text{ and } \partial = \begin{bmatrix} 1 & \omega & \omega^2 & \dots & \omega^{127} \\ 1 & \omega^{2b(1)+1} & \omega^{2(2b(1)+1)} & \dots & \omega^{127(2b(1)+1)} \\ 1 & \omega^{2b(2)+1} & \omega^{2(2b(2)+1)} & \dots & \omega^{127(2b(2)+1)} \\ . & . & . & \dots & . \\ 1 & \omega^{2b(127)+1} & \omega^{2(2b(127)+1)} & \dots & \omega^{127(2b(127)+1)} \end{bmatrix}.$$

If we rotate the input sequence of f'_{even} and f'_{odd} by one, by using the symmetric and periodicity properties ($\omega^{k+N/2} = -\omega^k$ and $\omega^{k+N} = \omega^k$) of NTT, we get:

$$f''_{even} = \begin{bmatrix} \frac{\hat{f}(0)-2f(0)}{\omega^{2b(0)+1}} \\ \frac{\hat{f}(2)-2f(0)}{\omega^{2b(1)+1}} \\ . \\ \frac{\hat{f}(252)-2f(0)}{\omega^{2b(126)+1}} \\ \frac{\hat{f}(N-2)-2f(0)}{\omega^{2b(\frac{N-2}{2})+1}} \end{bmatrix} = \partial * \begin{bmatrix} f(2) \\ f(4) \\ . \\ f(N-2) \\ f(0) \end{bmatrix}, \tag{12}$$

$$f_{odd}^{''} = \begin{bmatrix} \frac{\hat{f}(1)-2f(1)}{\omega^{2b(0)+1}} \\ \frac{\hat{f}(3)-2f(1)}{\omega^{2b(1)+1}} \\ . \\ \frac{\hat{f}(253)-2f(1)}{\omega^{2b(126)+1}} \\ \frac{\hat{f}(N-1)-2f(1)}{\omega^{2b(\frac{N-2}{2})+1}} \end{bmatrix} = \partial * \begin{bmatrix} f(3) \\ f(5) \\ . \\ f(N-1) \\ f(1) \end{bmatrix}. \tag{13}$$

From $\alpha f_{even}^{'} + \beta f_{even}^{''}$ and $\alpha f_{odd}^{'} + \beta f_{odd}^{''}$, we get the following:

$$\begin{bmatrix} \alpha\hat{f}(0) + \frac{\beta(\hat{f}(0)-2f(0))}{\omega^{2b(0)+1}} \\ \alpha\hat{f}(2) + \frac{\beta(\hat{f}(2)-2f(0))}{\omega^{2b(1)+1}} \\ . \\ \alpha\hat{f}(252) + \frac{\beta(\hat{f}(252)-2f(0))}{\omega^{2b(126)+1}} \\ \alpha\hat{f}(N-2) + \frac{\beta(\hat{f}(N-2)-2f(0))}{\omega^{2b(127)+1}} \end{bmatrix} = \partial * \begin{bmatrix} \alpha f(0) + \beta f(2) \\ \alpha f(2) + \beta f(4) \\ . \\ \alpha f(252) + \beta f(N-2) \\ \alpha f(N-2) + \beta f(0) \end{bmatrix}, \tag{14}$$

$$\begin{bmatrix} \alpha\hat{f}(1) + \frac{\beta(\hat{f}(1)-2f(1))}{\omega^{2b(0)+1}} \\ \alpha\hat{f}(3) + \frac{\beta(\hat{f}(3)-2f(1))}{\omega^{2b(1)+1}} \\ . \\ \alpha\hat{f}(253) + \frac{\beta(\hat{f}(253)-2f(1))}{\omega^{2b(126)+1}} \\ \alpha\hat{f}(N-1) + \frac{\beta(\hat{f}(N-1)-2f(1))}{\omega^{2b(127)+1}} \end{bmatrix} = \partial * \begin{bmatrix} \alpha f(1) + \beta f(3) \\ \alpha f(3) + \beta f(5) \\ . \\ \alpha f(253) + \beta f(N-1) \\ \alpha f(N-1) + \beta f(1) \end{bmatrix}. \tag{15}$$

Let us denote f_{even}^{1} and f_{odd}^{1} , respectively, as the f_{even} and f_{odd} that are shifted by 1. If we call equations (2) and (3) NTT_Kyber, we can achieve:

$$\hat{f}(2k) = \frac{1}{(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)} (\text{NTT\_Kyber}(\alpha f(2k) + \beta f(2k)^1) + \frac{2\beta f(0)}{\omega^{2b(k)+1}}), \quad k = 0, 1, ..., 127, \tag{16}$$

$$\hat{f}(2k+1) = \frac{1}{(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)} (\text{NTT\_Kyber}(\alpha f(2k+1) + \beta f(2k+1)^1) + \frac{2\beta f(1)}{\omega^{2b(k)+1}}), \quad k = 0, 1, ..., 127. \tag{17}$$

We can use the relation presented below for our error detection scheme:

$$128(f(0) + f(1)) \bmod q = \sum_{j=0}^{n-1} \hat{f}(j) \bmod q, \quad q = 3329. \tag{18}$$

The proposed error detection scheme is depicted in Fig. 4. The Decoder_3 sub-block multiplies its input with $\frac{1}{(\frac{\beta}{\omega^{2b(k)+1}}+\alpha)}$, which is the inverse of $(\frac{\beta}{\omega^{2b(k)+1}} + \alpha)$ in $\mathbb{R} = \mathbb{Z}[X]/(X^n + 1)$. The scheme for error detection involves comparing $128(f(0) + f(1)) \bmod q$, with $\sum_{j=0}^{n-1} \hat{f}(j) \bmod q$.

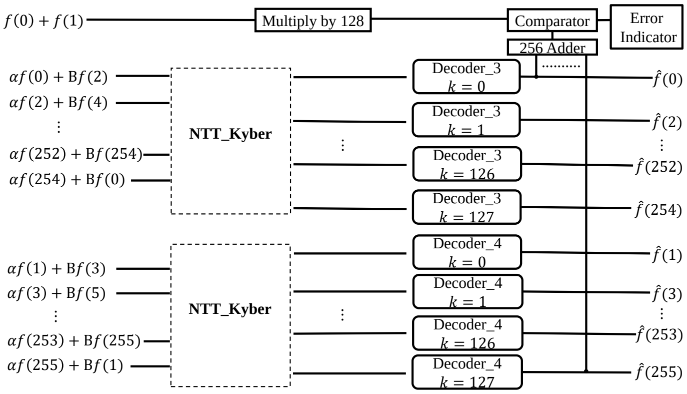

Fig. 4. Proposed algorithm level error detection scheme for the NTT utilized in Kyber.

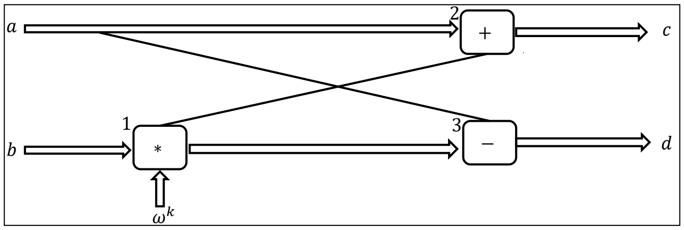

Fig. 5. The utilized fault model in this work for the butterfly sub-block.

4 ERROR COVERAGE SIMULATION RESULTS

4.1 Fault Model

The butterfly module is a key element in the NTT that enables efficient modular transformations, much like its function in the FFT. It processes two input elements by calculating their sum and difference. Its ability to handle modular arithmetic and root of unity multiplications is essential for enhancing the performance of the NTT.

As depicted in Fig. 5, faults can happen at modules {1,2, or 3} marked in the figure, which may result in erroneous outputs of a butterfly sub-block used in the NTT module. Fault occurrence follows a normal distribution within butterfly modules in Fig. 2 and Fig. 4. Furthermore, as we provide fault detection schemes over the NTT multiplication, faults can happen during component wise multiplication module. The NTT process involves numerous butterfly operations, where faults may occur at any of these operations. Within each butterfly, faults can happen at positions {1, 2, or 3}. In our fault model, no two faults are allowed to occur in the same butterfly operation. In the normal mode fault injection method, if the NTT process contains N butterfly operations and F represents the total number of faults, each butterfly operation has a probability of $\frac{F}{N}$ of being faulty. Moreover, if a butterfly is faulty, the probability of the fault occurring at position {1, 2, or 3} is $\frac{1}{3}$.

In the burst mode fault injection method, once a faulty butterfly operation is randomly chosen, all subsequent butterfly operations will also be faulty.

Table 1. Negative Wrapped Convolution achieved error detection ratios of the proposed schemes with 1, 2, 4, 8, and 16 faults occurrence with normal fault injection method for one million samples

Parameters		[1]$n = 256$,[2] $\omega = 3,844$, [3]$q = 7,681$	
Component		**Pre-process**	**NTT Multiplication**
Number of faults	1	99.7%	53%
	2	99.9%	70.6%
	4	100%	90.9%
	8	100%	99%
	16	100%	99.9%

[1] n = Polynomial size, [2] ω = Twiddle factor, [3] q = Prime number

4.2 Fault Simulation

To assess the error coverage, we employed Python3 to perform simulations, applying the described fault models and error detection techniques to both the NTT and pre-process modules.

4.2.1 Negative Wrapped Convolution. The simulations encompassed a million samples, using parameters identical to the standard Kyber Round 1. As shown in Table 1, with increase in the number of faults, we can attain an error detection ratio close to 100% for both modules. Higher error coverage is achievable for identifying burst errors.

The table provides data comparing the effectiveness of the proposed fault injection on two cryptographic components: the Pre-process stage and the NTT Multiplication stage, based on the number of faults injected (1, 2, 4, 8, or 16). For the Pre-process component, for a single fault, we already get high error coverage of 99.7%. As the number of injected faults increases, the ratio quickly reaches 100%. In contrast, the NTT Multiplication stage is more difficult to protect, but eventually we get to close to 100%. With one injected fault, 53% is the detection ratio. However, the success rate increases as more faults are injected: 70.6% for 2 faults, 90.9% for 4, 99% for 8, and 99.9% for 16 faults.

4.2.2 Kyber. In the standard Kyber Round 3 NTT implementation, there are 896 butterfly operations, corresponding to the polynomial size of 256. A fault can potentially occur at any of these 896 butterfly operations. The simulations involved one million samples, using the same parameters as in Kyber Round 3.

As indicated in Table 2, increasing the number of faults leads to an error detection ratio approaching 100% in both normal and burst modes. In Kyber's NTT implementation, the error detection ratios for various fault occurrences were analyzed using one million samples, with parameters set to $n = 256$, $\omega = 17$, and $q = 3329$. The results show that as the number of injected faults increases, the error detection ratio improves significantly. With just 1 fault, 74.9% of the errors were detected. This detection rate jumps to 93.45% with 2 faults. With 4 faults, the detection ratio reaches 99.49%, and by injecting 8 faults, it further increases to 99.95%. Finally, with 16 faults, the error detection ratio hits 100%, indicating complete detection of faults. This trend highlights the robustness of the error detection mechanism in Kyber's NTT when subjected to increasing numbers of faults. A higher number of faults leads to near-perfect or perfect error detection, suggesting the scheme becomes more effective at fault identification as fault intensity increases.

The NTT in Kyber was also tested using a burst fault model, with error detection ratios evaluated over one million samples. In the burst fault model, multiple consecutive bits in the data are corrupted simultaneously. The number of burst faults was varied, and the error detection ratios achieved by the proposed fault detection scheme are as follows:

- With 2 burst faults, the scheme detected errors with a success rate of **93.82%**.

Table 2. NTT in Kyber achieved error detection ratios of the proposed schemes with normal and burst fault injection methods for one million samples

Parameters		$n = 256, \omega = 17$ $q = 3,329$		
Fault injection method		Normal mode	Burst mode	
Number of faults	1	74.9%	2	93.82%
	2	93.45%	3	98.3%
	4	99.49%	4	99.42%
	8	99.95%	5	99.76%
	16	100%	6	99.8%

- When 3 burst faults occurred, the detection rate increased to **98.3%**.
- With 4 burst faults, the detection ratio improved further to **99.42%**.
- At 5 burst faults, the scheme achieved a near-perfect detection rate of **99.76%**.
- For 6 burst faults, the detection rate was nearly flawless, reaching **99.8%**.

The trend shows that as the number of burst faults increases, the error detection capability of the scheme improves, approaching almost 100% accuracy with 5 or more faults. This indicates the robustness of the detection method, particularly when dealing with higher levels of fault occurrences.

5 XILINX/AMD FPGA AND SOFTWARE IMPLEMENTATION RESULTS

5.1 Negative Wrapped Convolution

To confirm the efficiency of our presented approach, we chose to assess its performance by applying it to the NTT multiplication involving a 256-degree polynomial. We conducted a benchmark for implementation on different FPGAs, i.e., Xilinx/AMD Zynq Ultrascale+ and Artix-7. The Xilinx/AMD Zynq Ultrascale+ is a high-performance, multi-core SoC (System on Chip) that integrates programmable logic with ARM Cortex-A53 processors. It is designed for complex applications requiring significant processing power, such as AI, 5G, and automotive systems. It offers advanced features like multi-core processing, high-speed I/O, and power optimization.

In contrast, the Xilinx Artix-7 is a lower-cost, mid-range FPGA aimed at applications that need efficient power usage and moderate performance, such as communications, video processing, and embedded systems. It does not include integrated processors, relying solely on programmable logic.

Key differences include the Zynq Ultrascale+'s integrated ARM processors and higher performance capabilities, whereas the Artix-7 focuses on power efficiency and simpler, more cost-effective designs.

The results clearly demonstrate that our proposed efficient error detection schemes maintain a modest overhead while effectively achieving a high level of fault detection. We utilized the High-Level Synthesis (HLS) Vitis development environment of AMD/Xilinx to transform our proposed schemes into Register Transfer Level (RTL) hardware descriptions. The generated IP imported to Vivado to report power, utilization, and latency. Tables 3 and 4 demonstrate the results of our implementations and the derivations for area, timing, power, and energy. In the Vivado synthesis tool context, area is determined by combining Slices and DSPs, with a conversion ratio where one DSP is deemed equivalent to 100 Slices. Let us first go over Table 4. This table compares the implementation of two designs (our scheme vs. baseline) on the AMD/Xilinx Zynq Ultrascale+ FPGA (xczu4ev-sfvc784-2-i) in terms of area, power, and performance. Our scheme, which includes error detection, uses slightly more hardware resources than the baseline, with higher values for LUTs (1,469 vs. 1,435), FFs (1,188 vs. 1,122), CLBs (343 vs. 340), and DSPs (33 vs. 30). Both designs

Table 3. AMD/Xilinx Zynq Ultrascale+, xczu4ev-sfvc784-2-i Implementation Results

Strategy		**Area Effort**		**Timing Effort**	
Scheme		Our scheme	Baseline work	Our scheme	Baseline work
Area	LUTs	1,469	1,435	3,207	3,165
	FFs	1,188	1,122	2,227	2,161
	CLBs	343	340	733	683
	DSPs	33	30	54	51
Power (W) @ 140 MHz		0.46	0.45	0.52	0.51
Timing	Latency [CCs]	3,703	3,438	2,241	1,979
	Total time [ns]	26,439	24,547	16,000	14,130
Energy (nJ)		12,161	11,046	8,320	7,206

Our Scheme: The design which includes error detection scheme.
Baseline work: The design which does not include any error detection scheme.

have similar power consumption (0.46W vs. 0.45W), but our scheme incurs higher latency (3,703 vs. 3,438 clock cycles), total time (26,439 ns vs. 24,547 ns), and energy consumption (12,161 nJ vs. 11,046 nJ) due to the added complexity of error detection. These are reasonable overheads for providing error detection.

Moreover, as seen in Table 5, it summarizes the implementation results of two designs (our scheme vs. baseline) on the AMD/Xilinx Artix-7 FPGA (xc7a75ti-ftg256-1L), comparing their area, power, and timing performance. Our scheme, which includes error detection, uses more resources than the baseline, with higher values for LUTs (1,569 vs. 1,530), FFs (1,693 vs. 1,597), and DSPs (33 vs. 30). Both designs have similar power consumption, but our scheme consumes slightly more (0.2W vs. 0.19W). However, our scheme has increased latency (3,749 vs. 3,482 clock cycles), total time (26,767 ns vs. 24,861 ns), and energy consumption (5,353 nJ vs. 4,723 nJ), while providing fault detection not found in the baseline design. Our proposed error detection designs achieve a maximum operational frequency of around 140 MHz across all the FPGAs. Our proposed error detection scheme imposes a maximum overhead of 9% in additional area and introduces a latency increase of up to 13% in clock cycles, at most considering different designs. Our simulation and implementation code is publicly available in our github[1].

5.1.1 Implementation Optimization. Although HLS has shifted the design entry level of abstraction from RTL to C/C++, practical implementation often requires significant source code rewriting to make it HLS-ready. We have carefully taken this into effort as discussed here. This includes the incorporation of pragmas to attain satisfactory performance. Our intended area and timing efforts implementation was realized by strategically inserting the following pragmas into our program.

[1] https://github.com/KasraAhmadi/NTT_Error_Detection

Table 4. AMD/Xilinx Artix-7, xc7a75ti-ftg256-1L Implementation Results

Strategy		**Area Effort**		**Timing Effort**	
Scheme		Our scheme	Baseline work	Our scheme	Baseline work
Area	LUTs	1,569	1,530	2,956	2,939
	FFs	1,693	1,597	2,979	2,883
	SLICEs	673	656	1,240	1,253
	DSPs	33	30	54	51
Power (W) @ 140 MHz		0.2	0.19	0.28	0.26
Timing	Latency [CCs]	3,749	3,482	2,267	2,003
	Total time [ns]	26,767	24,861	16,186	14,301
Energy (nJ)		5,353	4,723	4,532	3,718

5.1.2 Pre-calculate the decoder module values. To enhance the efficiency of the proposed error detection scheme, we have the option to pre-calculate the decoder module in (11), specifically computing $\frac{1}{(\alpha+\beta\omega^{-k})^2}$ for various values of the parameter k and save them in memory.

5.1.3 Utilizing task-level pipelining. We perform loop unrolling on the outer loop in Algorithm 1 (Line 2) and transform the remaining code into a function named "stage," illustrating each stage in the NTT architecture. We undertook this approach to leverage task-level pipelining, allowing functions and loops to operate simultaneously. This enhances the concurrency of the RTL implementation, resulting in an overall increase in design throughput.

5.1.4 Pipelining the stage function. This pragma was inserted to facilitate instruction-level pipelining, aiming to enhance throughput and clock frequency within each NTT stage function. It is important to acknowledge that this optimization entails the trade-off of utilizing extra digital resources.

5.2 Kyber

We implemented our proposed error detection scheme for the official Kyber NIST submission (Round 3) on both x86-64 architecture and a 12th Gen Intel Core i7-12800H 2.4GHz processor, as well as on more resource-constrained devices, such as the 2.4GHz quad-core ARM Cortex-A72 (Raspberry Pi). We utilized the PAPI library to evaluate the performance of the software implementation and measure the overhead introduced by our error detection scheme. The PAPI library provides a standardized interface for accessing performance counters in CPUs, enabling researchers/developers to measure and analyze the performance of applications efficiently. Table 5 demonstrates the results of our implementation using standard Kyber Round 3 implementation as baseline work on 2 different architectures. The code for our simulation and implementation is accessible on our GitHub. On the ARM Cortex-A72, the baseline work requires 8,859 clock cycles to execute NTT operation in Kyber Round 3, whereas our scheme takes 10,316 clock cycles, an increase of approximately

Table 5. Software Implementation Results For the NTT in Kyber

	ARM v8 Cortex-A72 @ 1.5GHz	Intel Core-i7 @ 2.4GHz
Baseline work	8,859 clock cycles	5,814 clock cycles
Our Scheme	10,316 clock cycles	7,443 clock cycles

Our Scheme: The design which includes error detection scheme.
Baseline work: The design which does not include any error detection scheme.

16.5%. This suggests that the additional error detection or enhancements come with a performance cost in terms of clock cycles.

Similarly, on the Intel Core-i7, the baseline work completes in 5,814 clock cycles, while our scheme takes 7,443 clock cycles, an increase of about 28%. The performance overhead on the Intel Core-i7 is higher compared to the ARM v8, indicating that the performance impact of the error detection or other improvements is more significant on this processor.

5.2.1 *Implementation Optimization.*

According to equations (16) and (17), the overhead introduced by our error detection scheme depends on the encoding and decoding modules. The encoding module involves one addition and one shift operation. For the decoder, we can store coefficients ($\frac{1}{(\frac{\beta}{\omega^{2b(k)+1}}+\alpha)}$ and $\frac{2\beta}{\omega^{2b(k)+1}}$) in memory to eliminate additional computations. The decoding module comprises two multiplications and one addition.

6 CONCLUSION

In this paper, we proposed an efficient algorithm level error detection scheme over polynomial multiplication using the NTT and Negative Wrapped Convolution and the NTT operation in Kyber Round 3. We have aimed at achieving low hardware overhead and low latency, suitable for deeply-embedded systems. The presented scheme effectively safeguards cryptographic algorithms that employ the NTT multiplication. By performing simulations, we demonstrated that the proposed error detection scheme achieved extensive coverage of faults. Moreover, we implemented our proposed error detection schemes on two Xilinx/AMD FPGAs and two CPUs. Regarding overhead and latency, the implementation led to minimal additional expenses in hardware and software. With high error coverage and acceptable overhead, the proposed schemes in this work are suitable for resource-constrained and sensitive usage models.

ACKNOWLEDGMENTS

This work was supported by the US National Science Foundation (NSF) through the award SaTC-1801488.

REFERENCES

[1] Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2024. Efficient Error Detection Cryptographic Architectures Benchmarked on FPGAs for Montgomery Ladder. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2024), 1–0. https://doi.org/10.1109/TVLSI.2024.3419700

[2] Kasra Ahmadi, Saeed Aghapour, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2024. Efficient Error Detection Schemes for ECSM Window Method Benchmarked on FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 3 (2024), 592–596. https://doi.org/10.1109/TVLSI.2023.3341147

[3] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2019. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round* 2, 4 (2019), 1–43.

[4] Sven Bauer, Fabrizio De Santis, Kristjane Koleci, and Anita Aghaie. 2024. A Fault-Resistant NTT by Polynomial Evaluation and Interpolation. *Cryptology ePrint Archive* (2024).

[5] Nina Bindel, Johannes Buchmann, and Juliane Krämer. 2016. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 63–77.

[6] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography. In *2021 IEEE 28th symposium on computer arithmetic (ARITH)*. IEEE, 94–101.

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

[8] Leon Groot Bruinderink and Peter Pessl. 2018. Differential fault attacks on deterministic lattice signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (2018), 21–43.

[9] Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Reliable constructions for the key generator of code-based post-quantum cryptosystems on FPGA. *ACM Journal on Emerging Technologies in Computing Systems* 19, 1 (2022), 1–20.

[10] Alvaro Cintas Canto, Ausmita Sarker, Jasmin Kaur, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. 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* 11, 3 (2022), 791–797.

[11] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. 1967. What is the fast Fourier transform? *Proc. IEEE* 55, 10 (1967), 1664–1674.

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

[13] Jeroen Delvaux. 2021. Roulette: A diverse family of feasible fault attacks on masked kyber. *Cryptology ePrint Archive* (2021).

[14] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.

[15] Mohamed ElGhamrawy, Melissa Azouaoui, Olivier Bronchain, Joost Renes, Tobias Schneider, Markus Schönauer, Okan Seker, and Christine van Vredendaal. 2023. From MLWE to RLWE: A Differential Fault Attack on Randomized & Deterministic Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023, 4 (2023), 262–286.

[16] Thomas Espitau, Pierre-Alain Fouque, Benoit Gerard, and Mehdi Tibouchi. 2018. Loop-abort faults on lattice-based signature schemes and key exchange protocols. *IEEE Trans. Comput.* 67, 11 (2018), 1535–1549.

[17] Yue Geng, Xiao Hu, Minghao Li, and Zhongfeng Wang. 2023. Rethinking parallel memory access pattern in number theoretic transform design. *IEEE Transactions on Circuits and Systems II: Express Briefs* 70, 5 (2023), 1689–1693.

[18] Ying Guo, Wenfen Liu, Wen Chen, Qingwen Yan, and Yongcan Lu. 2024. ECLBC: A Lightweight Block Cipher With Error Detection and Correction Mechanisms. *IEEE Internet of Things Journal* 11, 12 (2024), 21727–21740. https://doi.org/10.1109/JIOT.2024.3376527

[19] Daniel Heinz and Thomas Pöppelmann. 2022. Combined fault and DPA protection for lattice-based cryptography. *IEEE Trans. Comput.* 72, 4 (2022), 1055–1066.

[20] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. 2021. Fault-enabled chosen-ciphertext attacks on Kyber. In *Progress in Cryptology–INDOCRYPT 2021: 22nd International Conference on Cryptology in India, Jaipur, India, December 12–15, 2021, Proceedings 22*. Springer, 311–334.

[21] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. 2023. Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023), 60–88.

[22] Vincent Hwang, Jiaxiang Liu, Gregor Seiler, Xiaomu Shi, Ming-Hsien Tsai, Bow-Yaw Wang, and Bo-Yin Yang. 2022. Verified NTT multiplications for NISTPQC KEM lattice finalists: Kyber, SABER, and NTRU. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2022), 718–750.

[23] Sönke Jendral. 2024. A Single Trace Fault Injection Attack on Hedged CRYSTALS-Dilithium. *Cryptology ePrint Archive* (2024).

[24] J-Y Jou and Jacob A. Abraham. 1988. Fault-tolerant FFT networks. *IEEE transactions on computers* 37, 5 (1988), 548–561.

[25] Matthias J Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2019. pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. (2019).

[26] Elisabeth Krahmer, Peter Pessl, Georg Land, and Tim Güneysu. 2024. Correction fault attacks on randomized crystals-dilithium. *Cryptology ePrint Archive* (2024).

[27] Suparna Kundu, Siddhartha Chowdhury, Sayandeep Saha, Angshuman Karmakar, Debdeep Mukhopadhyay, and Ingrid Verbauwhede. 2024. Carry your fault: A fault propagation attack on side-channel protected lwe-based kem. *arXiv preprint arXiv:2401.14098* (2024).

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

[29] Bin Li, Yunfei Yan, Yuanxin Wei, and Heru Han. 2023. Scalable and parallel optimization of the number theoretic transform based on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023).

[30] Fabiano Libano, Paolo Rech, and John Brunhaver. 2023. Efficient error detection for matrix multiplication with systolic arrays on FPGAs. *IEEE Trans. Comput.* 72, 8 (2023), 2390–2403.

[31] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. 2008. SWIFFT: A modest proposal for FFT hashing. In *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers 15*. Springer, 54–72.

[32] Nimish Mishra and Debdeep Mukhopadhyay. 2024. Probabilistic algorithms with applications to countering fault attacks on lattice based post-quantum cryptography. *Cryptology ePrint Archive* (2024).

[33] Jianan Mu, Huajie Tan, Jiawen Wu, Haotian Lu, Chip-Hong Chang, Shuai Chen, Shengwen Liang, Jing Ye, Huawei Li, and Xiaowei Li. 2023. Energy-Efficient NTT design with one-bank SRAM and 2-D PE Array. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–2. https://doi.org/10.23919/DATE56975.2023.10137059

[34] Koksal Mus, Saad Islam, and Berk Sunar. 2020. QuantumHammer: a practical hybrid attack on the LUOV signature scheme. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1071–1084.

[35] NIST. 2024. https://csrc.nist.gov/pubs/fips/203/final, Last accessed on 2024-09-17.

[36] NIST. 2024. https://csrc.nist.gov/pubs/fips/204/final, Last accessed on 2024-09-17.

[37] NIST. 2024. https://csrc.nist.gov/pubs/fips/205/final, Last accessed on 2024-09-17.

[38] Lukas Prokop and Peter Peßl. 2021. Fault attacks on CCA-secure lattice KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 2 (2021), 37–60.

[39] Prasanna Ravi, Anupam Chattopadhyay, Jan Pieter DAnvers, and Anubhab Baksi. 2024. Side-channel and fault-injection attacks over lattice-based post-quantum schemes (Kyber, Dilithium): Survey and new results. *ACM Transactions on Embedded Computing Systems* 23, 2 (2024), 1–54.

[40] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. 2019. Exploiting determinism in lattice-based signatures: practical fault attacks on pqm4 implementations of NIST candidates. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 427–440.

[41] Prasanna Ravi, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. 2023. Fiddling the twiddle constants-fault injection analysis of the number theoretic transform. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023).

[42] Jan Richter-Brockmann, Pascal Sasdrich, Florian Bache, and Tim Güneysu. 2020. Concurrent error detection revisited: Hardware protection against fault and side-channel attacks. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 1–11.

[43] Sayandeep Saha, Manaar Alam, Arnab Bag, Debdeep Mukhopadhyay, and Pallab Dasgupta. 2023. Learn from your faults: leakage assessment in fault attacks using deep learning. *Journal of Cryptology* 36, 3 (2023), 19.

[44] Ausmita Sarker, Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Error detection architectures for hardware/software co-design approaches of number-theoretic transform. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 7 (2022), 2418–2422.

[45] Ausmita Sarker, Mehran Mozaffari Kermani, and Reza Azarderakhsh. 2022. Efficient error detection architectures for postquantum signature Falcon's sampler and KEM SABER. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 6 (2022), 794–802.

[46] Ausmita Sarker, Mehran Mozaffari-Kermani, and Reza Azarderakhsh. 2018. Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 3 (2018), 738–741.

[47] Richa Singh, Saad Islam, Berk Sunar, and Patrick Schaumont. 2024. Analysis of EM fault injection on bit-sliced number theoretic transform software in dilithium. *ACM Transactions on Embedded Computing Systems* 23, 2 (2024), 1–27.

[48] Tolun Tosun and Erkay Savas. 2024. Zero-Value Filtering for Accelerating Non-Profiled Side-Channel Attack on Incomplete NTT based Implementations of Lattice-based Cryptography. *IEEE Transactions on Information Forensics and Security* (2024).

[49] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. 2021. Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part II 27*. Springer, 33–61.

[50] Cong Zhang, Dongsheng Liu, Xingjie Liu, Xuecheng Zou, Guangda Niu, Bo Liu, and Quming Jiang. 2021. Towards efficient hardware implementation of NTT for kyber on FPGAs. In *2021 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 1–5.

[51] Neng Zhang, Qiao Qin, Hang Yuan, Chenggao Zhou, Shouyi Yin, ShaoJun Wei, and Leibo Liu. 2019. NTTU: An area-efficient low-power NTT-uncoupled architecture for NTT-based multiplication. *IEEE Trans. Comput.* 69, 4 (2019), 520–533.