Efficient, Error-Resistant NTT Architectures for CRYSTALS-Kyber FPGA Accelerators

Safiullah Khan, Ayesha Khalid, Ciara Rafferty, Yasir Ali Shah, Maire O'Neill, Wai Kong Lee and Seong Oun Hwang

October 24, 2023

Efficient, Error-Resistant NTT Architectures for CRYSTALS-Kyber FPGA Accelerators

Safiullah Khan[1,2], Ayesha Khalid[1], Ciara Rafferty[1], Yasir Ali Shah[1], Maire O'Neill[1],
Wai-Kong Lee[3], and Seong Oun Hwang[3]

[1]The Centre for Secure Information Technologies (CSIT), Queen's University Belfast, Belfast, United Kingdom
[2]Department of Computing and Mathematics, Manchester Metroploitan University, Manchester, United Kingdom
[3]Computer Engineering Department, Gachon University, Seongnam, Republic of Korea
Email: {s.khan, a.khalid, c.m.rafferty, y.shah}@qub.ac.uk; m.oneill@ecit.qub.ac.uk; {waikonglee, sohwang}@gachon.ac.kr

***Abstract*—The dawn of cost-effective miniaturised satellites is currently attracting venture capital in a never seen before ratio to launch mega-constellations of satellites for a diverse range of applications. These satellites are vulnerable to attacks by high-capability cyber-criminals (including quantum enabled adversaries), due to the critical data they transmit. Additionally, space missions have long lifespan and a long lead time in terms of development process, requiring a pre-emptive outlook to ensuring their safety. In 2016, National Institute of Standards and Technology (NIST) initiated the competition to standardise the post-quantum cryptography (PQC) schemes, announcing the first portfolio of chosen schemes in 2022. This work targets the only public key exchange (PKE) scheme among the winners of the NIST-PQC standardisation process, CRYSTALS-Kyber, and implements its core bottleneck operation, i.e., number theoretic transform (NTT) extensively used for the polynomial multiplication. To avoid data corruption due to space based radiations, a novel error-resistant model for NTT is presented based on hybrid protection mechanisms, i.e., the use of hamming codes for detection and correction of errors in the twiddle factors and the use of parity computed for all NTT coefficients for error detection. Benchmarking error protection overheads on a Xilinx Virtex-7 FPGA reports 16.4% and 10.8% degradation on the hardware efficiency when the hamming codes for twiddle factors and parity bit for NTT coefficients are used to mitigate errors, respectively. A total of 29.2% area overhead is benchmarked when compared to the standard unprotected NTT implementations.**

***Index Terms*—Post-quantum cryptography (PQC), Lattice-based cryptography (LBC), CRYSTALS-Kyber, Fault-tolerant architectures, Number theoretic transform (NTT), Error-resistant architectures.**

I. Introduction

By executing Shor's algorithm [1] and Grover's algorithm [2] on quantum computers, the traditional public key cryptography (PKC) (RSA [3] and elliptic curve cryptography (ECC) [4]) are susceptible to be compromised. Consequently, they need to be replaced by post-quantum cryptography (PQC). CRYSTALS-Kyber (Kyber) [5] being selected as a post-quantum key encapsulation mechanism (KEM) protocol belongs to the family of lattice-based cryptography (LBC), and the underlying hard problem for Kyber is module-learning with errors (M-LWE). M-LWE based schemes tend to have superior security [6] compared to Ring-LWE schemes [7].

The first author's current affiliation is Manchester Metropolitan University, however the work was carried out while he was the part of CSIT at Queen's University, Belfast.

Post-quantum cryptographic algorithms have been extensively implemented on the field programmable gate arrays (FPGA) platforms. The implementations have been extended to unconventional applications as well, such as remote sensing [8] for the satellites on space missions. Compared to conventional computational approaches, FPGAs offer several advantages, such as a large number of computational resources and high reconfigurability [9]. However, S-RAM based FPGAs are found to be sensitive to radiations that are present in satellite orbits at all levels, thus causing single event upsets (SEUs). SEU in the FPGAs can effect the multiplexers, latches, buffers, LUTs, control bits, and registers [10]. A single bit error introduced at any stage during the NTT computation can have a cascading impact on the entire encryption/decryption process, thus requiring some mitigation strategies.

This work presents error-resistant NTT architectures by combining hamming codes and the computation of parity bits. Hamming codes have been applied to the twiddle factors in order to detect and correct any SEU occurred. A single parity bit is computed for the intermediate layers of NTT that can detect any error occurred. The correction can be performed by resetting the computations for NTT, making the architecture lightweight. Thus a novel architecture combining the advantages of Hamming codes for efficiency and parity bits for area-conservation is presented in this work.

The main contribution of our work is as follows:

- We propose an efficient error detection and mitigation strategy for the most compute-intensive part for standardised post-quantum algorithm, CRYSTALS-Kyber. The ability to detect and correct the error utilising hamming codes combined with parity computations for error detection, has been employed to impart error-resistant characteristics to the NTT architectures.
- We implemented our proposed architectures on the Virtex-7 FPGA platform and carefully benchmarked the overhead in terms of computational resources. Implementation results indicate that incorporation of error-resistant

attributes incur an area-overhead of 29.2%, compared to the standard implementations.

The rest of the paper is organised as follows: Section II gives an overview of the previous works. Section III introduces the Kyber KEM protocol and NTT in Kyber. The proposed NTT implementations and the error-resistant schemes are discussed in Section IV. Section V presents the implementation results and discussion. Finally, the paper concludes in Section VI.

II. Literature Review

Several works have been proposed in the literature relating to the efficient FPGA implementation of the core modules in Kyber. Modular multiplication, being one of the main components, has been discussed in [11]–[13] along with the proposal of efficient implementation strategies. In addition, efficient implementations of NTT has remained a prime focus. A novel 2×2 butterfly unit incorporating the K2-RED reduction algorithm is presented in [13]. Authors in [14] propose lightweight, balanced, and high performance architectures for NTT/INTT. A fully pipelined architecture has been proposed by [15], however the proposed architecture is resource-intensive.

A limited number of works have focused on the error-resistant architectures, especially for the PQC. SEU detection for the substitution phase of AES by employing block memory and combinational logic is discussed in [16] with the goal to provide area-efficient architectures. A fault-detection mechanism for secure hash standard (SHS) and keyed-hash message authentication code (HMAC), by employing the hamming codes, is discussed in [17]. A modified version of TMR, known as partial TMR, is presented in [18], where TMR is applied to most critical sections with the help of an automated tool. Authors in [19] discuss error detection architectures for NTT to encounter transient and permanent faults.

In summary, the literature contains numerous works on optimizing the Kyber implementation, however, error-resistant Kyber architectures are not well-studied. The proposed architecture focus on integrating error-resistant features with efficient NTT implementations in Kyber that are suitable for space applications.

III. Preliminaries

A. CRYSTALS-Kyber

Kyber is standardised as post-quantum KEM and it consists of three main operations: Key Generation, Key Encapsulation, and Key Decapsulation. The mathematical hard problem underlying Kyber is M-LWE; Kyber belongs to the LBC family in PQC. For LBC, the core operation is polynomial multiplication, where $pk = sk \circ A + e$ is the main operation in the key generation. The modulus for Kyber is chosen to be 3329, selected in such a manner to ensure the accelerated performance for polynomial multiplication when executed via the NTT algorithm. Each of the polynomials for Kyber consist of $n = 256$ coefficients. For detailed insights into the working of Kyber KEM, further information can be found in [5].

B. NTT in Kyber

To accelerate the polynomial multiplication, NTT is often employed in LBC. For Kyber, the prime q is selected to be 3329, where $q - 1 = 2^8 \cdot 13$, the base field $\mathbb{Z}_q$ comprises primitive 256-th roots of unity. For any polynomial in Kyber, denoted as $f = \{f_0 + f_1 x + \cdots + f_{255} x^{255}\} \in R_q$ and primitive root of unity ω mod q, the NTT operations can be defined as:

$$NTT(f) = \hat{f} = \hat{f}_0 + \cdots + \hat{f}_{255} x^{255} \tag{1}$$

with

$$\hat{f}_{2i} = \sum_{j=0}^{127} f_{2i} \omega^{(2\mathbf{br}_7(i)+1)j} \tag{2}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{127} f_{2i+1} \omega^{(2\mathbf{br}_7(i)+1)j} \tag{3}$$

where $\mathbf{br}_7$ means bit reversal of the unsigned 7-bit integer i.

Hence, for Kyber, the polynomial with $n = 256$ can be split into two polynomials, each with 128 coefficients. The NTT can be computed independently.

IV. Proposed Error-Resistant NTT Architecture

A. Proposed Modular Reduction Architecture

The butterfly core for the NTT computation consists of a modular reduction operation. Several modular reduction algorithms can be implemented, however for NTT typically Barrett reduction algorithm is employed as it is suitable for operating on smaller mod values. Given two positive integers a and q, computing a mod q requires a division by q, where the Barrett reduction estimates $1/q$ to replace expensive division operation by multiplication and shifting operations, where $1/q$ is approximated as $x/2^k$. x can be taken as $\lfloor 2^k/q \rfloor$. k depends on a and should also satisfy the error approximation to be less than 1, $k = \log_2(a)$. Algorithm 1 demonstrates the working of Barrett Reduction.

Figure 1 shows the implementation of modified Barrett reduction in hardware. Since the reduction architecture is designed to work with a single mod q, the hardware design can be modified. Mod q, and parameters k and x are pre-computed and stored inside the registers. Two multiplications involved in the reduction process employ the standard school-book approach and are computed in series and constitute the major hardware footprint for the reduction process. By pre-computing, substantial hardware resources and time utilisation can be saved, making the butterfly architecture efficient.

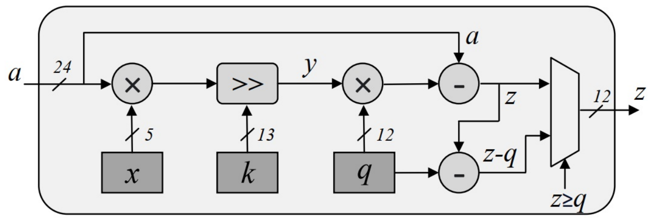

Fig. 1. Modified modular reduction using Barrett reduction algorithm

Algorithm 1 The Barrett Reduction Algorithm

```
1: Input: $a$
2: Output: $a$ mod $q$
3: Pre-computed: $k = \log_2(a)$, $x = \lfloor 2^k/q, \rfloor$, $q = 3329$
4: $y \leftarrow (a \times x) >> k$;
5: $z \leftarrow a - y \times q$;
6: if $z \geq q$ then
7:     $z \leftarrow z - q$
8: end
9: return $z$
```

B. NTT in Butterfly core

The compute-intensive operation in terms of hardware footprint and time utilisation is polynomial multiplication, thus acting as the bottle-neck operation in Kyber. To achieve efficiency for the execution of polynomial multiplication, NTT is computed for the operands of multiplication. For NTT computation, the proposed architecture utilises two modular reduction operations being executed in parallel inside the butterfly core. A single butterfly core is iterated during each clock cycle with different values of inputs, that can take two input coefficients u and v and a twiddle factor ω and outputs the updated coefficients. The butterfly core includes a multiplication (between twiddle factor ω and one of the inputs v) to generate an intermediate result, an addition and a subtraction of the intermediate result to the other input u, followed by the reduction process. This process executes the Cooley-Tookey configuration and computes the important butterfly step as $u + vw$ and $u - vw$.

In order to start the NTT computation, the coefficients are first loaded into the registers. A register bank with n registers, each of $\lceil log_2(q) \rceil$-bit hold the input coefficients. One coefficient is loaded into the register bank in each clock cycle thus requiring n clock cycles. This is followed by loading the twiddle factors w, that require $n/2$ clock cycles. Thus the pre-processing requires $3n/2$ clock cycles, where the data is stored in register banks before the start of computation. The novel implementation strategy, proposed in this work suggests to use the register banks as they are the fastest memory elements available in an FPGA and data access can be performed in a single clock cycle. In addition, register banks use a small amount of FPGA resources compared to other memory options, making them suitable for designs with limited available resources.

During each clock cycle, two coefficients are read from the register bank. For the first level, the distance between the coefficients is $n/2^l$, where l is the level number during computation. For first level, $l = 1$, the difference would be 128, so the first coefficients are $(0, 128)$ and so on. The input is provided to the butterfly core and the output of the first level is stored in registers sequentially. The same process is repeated for all inputs, thus requiring $n/2$ clock cycles to process the first level of NTT computation. Once all of the results are stored in the register banks, the computation for the second layer starts in a much similar fashion, however, the distance between the coefficients, $n/2^l$ would be 64, since $l = 2$ for this layer. The updated values of the layer are then stored back in the previous registers, so only two register banks are required to complete the whole process. In this way a total of $l \times n/2$ clock cycles are required to complete the NTT computation. This architecture for NTT computation can save a substantial amount of hardware footprint. The twiddle factors are also stored in registers than can hold $n/2$ values each of $\lceil log_2(q) \rceil$-bit values. This constitutes the overall architecture of the NTT for the proposed design. Figure 2 shows the architecture for the butterfly core and the register banks employed during the NTT computation.

C. Error-Resistant NTT Architectures

Once the basic architecture for the NTT computation is designed, the error-resistance characteristics are then incorporated. In order to mitigate the effect of SEU, the hardware architectures for the NTT, described in the previous section, have been modified. The proposed error-resistant architecture is shown in the Figure 3 that utilises hamming codes to detect and correct any SEU introduced in the twiddle factors, with the help of extra bits introduced. In addition, for the intermediate data stored between updating the internal layers of NTT, the parity bits is computed to detect any bit flip. The aim of this work is to incorporate the error-resistance properties within the NTT that can pose minimum overhead in terms of hardware footprint and computation time.

1) Error detection and correction of twiddle factors using hamming codes: The twiddle factors must be secure from SEU. A single error introduced during the NTT computation has the potential to cascade throughout the final output in a domino effect, thus requiring the mechanisms to negate the effect of SEU. To overcome this problem, the proposed scheme suggests to compute the hamming weights for each twiddle factor and store them along with the original value of the twiddle factors. The hamming codes have the capability to detect and correct the SEU, ideally desired for such applications. The hamming code bits are computed in such a way that specific data bits are chosen and the total number of 1 bits, including the hamming bits itself, must be even. The twiddle factors are stored in $n/2$ registers, where each register holds $\lceil log_2(q) \rceil$-bit value. The number of additional bits required must satisfy $2^p \geq m + p + 1$, where p-bits are minimum acceptable for the error detection and correction. p is typically determined with trial and error method, starting from the smallest p value to the point where the equation is satisfied. All the hamming weights for the twiddle factors are computed on the fly and stored in the registers along with twiddle factors. This operation does not pose any overhead in terms of computation time as the operations are pipelined with NTT computation operations. Figure 3 shows the registers that holds the twiddle factors along with the desired width.

The butterfly core is modified to be capable of error detection and correction before the twiddle factors are incorporated in the computation. For this purpose, when the twiddle factors are read from the register bank, they are read along with

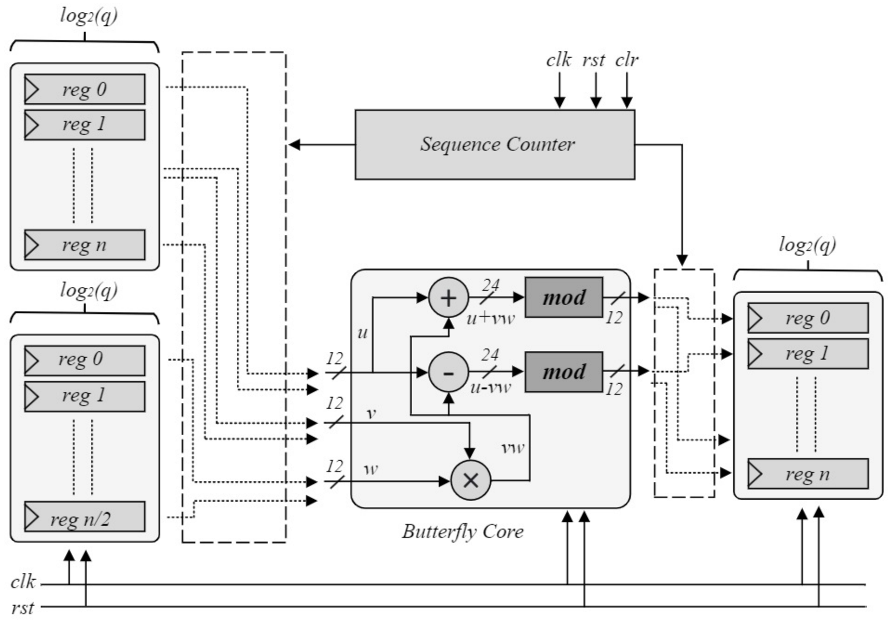

Fig. 2. Basic hardware architecture for the NTT/INTT computation

the hamming weights. Inside the butterfly core, the hamming codes are computed again using the *h.compute* module and compared to the received hamming codes. In case of detection of any mismatch, the butterfly core has the capability to detect the position of the erroneous bit and then correct it just by flipping that bit by using the *h.correct* module. The corrected output is then processed for the standard NTT computations. This enhances the data integrity and reliability during the NTT computation. Figure 3 shows the internal architecture of the modified butterfly core along with the *h.compute* and *h.correct* modules.

2) *Error detection and correction of coefficients using parity bits:* During the NTT computation, the coefficients are stored in the register banks, that are updated and stored in another register bank. The coefficients are updated by shifting between both register banks. In order to avoid any SEU for the coefficients stored in the registers, a relatively simple yet efficient approach is employed. Instead of computing hamming weights for all of the intermediate layers that can pose a certain area and time overhead, the proposed architecture suggests to compute a single bit parity for each of the data stored in between the layers. The register width in this case would increase to $\lceil log_2(q) \rceil + 1$ bit, as indicated in the Figure 3. Thus only a single bit register is increased in terms of the area overhead. The computation of parity bits does not pose any overhead on time as the operation of computation of parity bits are pipelined with the storing the coefficients in the registers.

The butterfly core has the additional capabilities to detect SEU for the coefficients of NTT. For this purpose, the butterfly core computes the parity of each coefficient before the start of the computation, using the *parity* module, which is then compared to the parity received. In case of a mismatch a signal rst_1 or rst_2 is generated that can act as a reset signal for the NTT computation. This can reset the whole computation, thus the process is repeated. Figure 3 shows the architecture for the NTT computation with the error-resistant capabilities incorporated.

V. Implementation Results and Discussion

Implementations of the proposed architectures are flexible to be implemented for any FPGA family and are independent of any specific hardware. The architectures are designed in Verilog HDL, and synthesis and implementations have been performed in Vivado for the Virtex-7 platform. The designs have been tested for functional verification and results have been generated for post-place and route (PAR) simulations.

The basic evaluation metrics include the hardware area utilisation and time consumption while some other important metrics like throughput (TP) and throughput/area (TP/A) are also measured. TP is the number of bits that the specific architecture can process in certain time while TP/A considers the area consumption as well. Table I summarises the implementation results for the proposed strategies.

TABLE I
Results implemented on FPGA (Virtex-7)

	#LUT	#DSP	Time *ns*	Freq. *MHz*	TP *Mbps*	TP/A *TP/kL*	%+A
Basic Architecture							
A-O	7.9k	0	16.57	60.32	206.82	26.17	-
R-O	7.8k	6	13.86	72.11	247.26	31.69	-
Hamming codes							
A-O	9.2k	0	16.34	61.17	209.75	22.79	16.4
R-O	9.3k	6	13.59	73.54	252.17	27.11	19.2
Hamming codes + Parity bits							
A-O	10.2k	0	16.84	59.37	203.71	19.97	10.8
R-O	11.3k	6	14.06	71.12	243.86	21.58	21.5

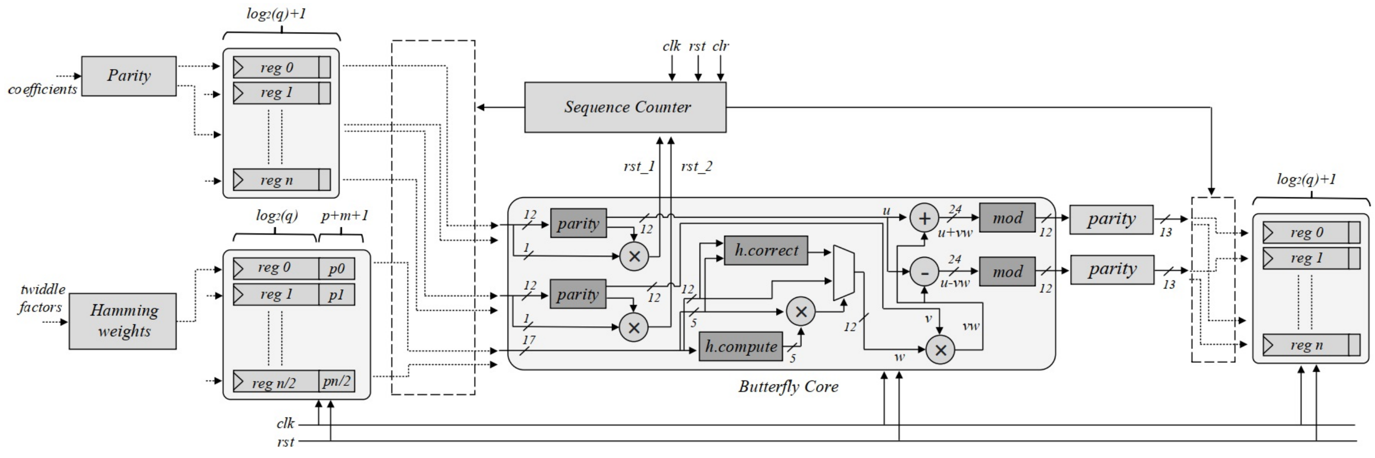

Fig. 3. Error-Resistant NTT/INTT computation architecture

Aside from the internal architectural optimisations discussed in the previous section, the synthesis optimisations provided by the Vivado tool are also applied to the proposed architectures. As a result, an area-optimised (A-O) architecture and a run-time optimised (R-O) architecture are presented for the proposed strategies.

Firstly, the area utilisation in terms of number of LUTs and DSP blocks is discussed. For the basic architecture of NTT, without any error-resistance method incorporated, the proposed architecture consumes 7.9k LUT for area-optimised design while 7.8k LUTs + 6 DSP blocks for the run-time optimised design. The area-optimised design is mapped into the general fabric of FPGA without the need of any DSP blocks. Once the hamming codes are incorporated to the twiddle factors, the area consumption has been increased by 16.4% (in terms of LUTs). This is primarily due to the fact that more register width is required to hold the values (twiddle factors + hamming weights). For the next design, when the parity bits are also added to detect the error for the intermediate layers, the area further increases by 10.8% (in terms of LUTs). This area increase is caused by addition of parity bit in the registers. The butterfly core has also been modified to include the error detection and correction part. Compared to the initial unprotected area consumption to the final architecture, there is an increase of 29.2% area utilisation.

Next the computation time and operating frequency are discussed. For all the proposed architectures, the clock cycles consumption is always $l \times n/2$. The additional operations for the SEU detection and mitigation are pipelined with the internal NTT computations, making the whole operations to be performed in constant clock cycles. By introducing this pipelined architecture, operations are overlapped in different pipeline stages, thus introducing the same critical path delay. Frequency, being the inverse, remains relatively similar for all the architectures for the area-optimised architectures as well as the run-time optimised architectures. The same can be seen for the run-time optimised architectures as well.

Some important parameters as already defined are the TP and TP/A. Since the overall time for the proposed architectures is constant (constant clock cycles utilisation and critical path) the TP remain relatively unchanged for each of the optimisation strategy. The TP/A however varies as the designs utilise different hardware. TP/A ratios are better for the basic architecture mainly due to the fact that they utilise less area compared to the error-resistant designs. The percentage increase in area (%+A) for each architecture compared to the basic implementations is also provided in the Table I. Summarising the implementation results, we can say that the overall error-resistant design requires almost 29.2% more hardware area compared to the basic architecture for the NTT. The possible limitations of the proposed architectures comes from the area overhead. On the other hand, the designs are pipelined in such a way to make the overall architectures independent of the time consumption making them run in constant time.

Comparison of the proposed architectures directly with other works in the literature is not possible. The architecture of AES presented in [16] that can detect the SEU for substitution phase of AES requires almost 80% more area in terms of number of LUTs utilised. For the 256-bit SHA and HMAC implementations presented in [17] an area overhead of 106% and 60% can be seen when the registers are encoded all the time. The work proposed in [19] can detect the transient and permanent faults for the NTT, by recomputing and decoding through two variants, where the best results can obtain error detection with 12.74% LUTs overhead and 15.88% more computation time. To the best of our knowledge, our proposed work remains the first to employ a hybrid protection mechanism employing efficient error correction codes along with the lightweight parity bits computation to detect and correct the SEUs for the NTT in Kyber.

VI. Conclusion

This research proposes error-resistant FPGA implementations for the bottle-neck operation in CRYSTALS-Kyber, i.e., NTT. The proposed architecture employs hamming codes and parity bits to detect as well as correct SEU occurring in the radiation environment for space applications. The overhead in terms of area consumption has been computed to be 29.2%. Further, the performance analysis has been presented in terms of TP and TP/A ratio. As a future direction, the same strategies can be exported to design CRYSTALS-Kyber accelerators capable to detect and correct the SEUs.

Acknowledgement

This research was funded in part by EPSRC Quantum Communications Hub (EP/T001011/1).

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[4] V. S. Miller, "Use of elliptic curves in cryptography," in *Conference on the theory and application of cryptographic techniques*, pp. 417–426, Springer, 1985.

[5] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-kyber (version 3.0) – submission to round 3 of the nist post-quantum project," *NIST, Tech. Rep*, 2020.

[6] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 1–23, Springer, 2010.

[7] R. Lindner and C. Peikert, "Better key sizes (and attacks) for LWE-based encryption," in *Cryptographers' Track at the RSA Conference*, pp. 319–339, Springer, 2011.

[8] A. El Makhloufi, N. Tagmouti, N. Chekroun, S. El Adib, J. A. Sobrino, and N. Raissouni, "AES/FPGA encryption module integration for satellite remote sensing systems: LST-SW case," in *2020 3rd International Conference on Advanced Communication Technologies and Networking (CommNet)*, pp. 1–7, IEEE, 2020.

[9] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Availability analysis for satellite data processing systems based on SRAM FPGAs," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 3, pp. 977–989, 2016.

[10] P. Graham, "Consequences and categories of SRAM FPGA configuration SEUs," *Proc. Military and Aerospace Programmable Logic Devices, Sept. 2003*, 2003.

[11] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-kyber on FPGA," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 328–356, 2021.

[12] C. Zhang, D. Liu, X. Liu, X. Zou, G. Niu, B. Liu, and Q. Jiang, "Towards efficient hardware implementation of NTT for kyber on FPGAs," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.

[13] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "High-speed NTT-based polynomial multiplication accelerator for post-quantum cryptography," in *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*, pp. 94–101, IEEE, 2021.

[14] F. Yaman, A. C. Mert, E. Öztürk, and E. Savaş, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-Kyber PQC scheme," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1020–1025, IEEE, 2021.

[15] D.-e.-S. Kundi, Y. Zhang, C. Wang, A. Khalid, M. O'Neill, and W. Liu, "Ultra high-speed polynomial multiplications for lattice-based cryptography on FPGAs," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2022.

[16] S. Ghaznavi and C. Gebotys, "A SEU-resistant, FPGA-based implementation of the substitution transformation in AES for security on satellites," in *2008 10th International Workshop on Signal Processing for Space Communications*, pp. 1–5, IEEE, 2008.

[17] M. Juliato and C. Gebotys, "A quantitative analysis of a novel SEU-resistant SHA-2 and HMAC architecture for space missions security," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 3, pp. 1536–1554, 2013.

[18] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2274–2280, 2008.

[19] A. Sarker, A. C. Canto, M. M. Kermani, and R. Azarderakhsh, "Error detection architectures for hardware/software co-design approaches of Number-Theoretic Transform," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.