Error Detection Architectures for Hardware/Software Co-Design Approaches of Number-Theoretic Transform

Ausmita Sarker, *Student Member, IEEE*, Alvaro Cintas Canto, *Member, IEEE*, Mehran Mozaffari Kermani, *Senior Member, IEEE*, and Reza Azarderakhsh, *Member, IEEE*

***Abstract*—Number-theoretic transform (NTT) is an efficient polynomial multiplication technique of lattice-based post-quantum cryptography including Kyber which is standardized as the NIST key encapsulation mechanism (KEM) in 2022. Prominent NTT architectures have recently been implemented on hardware/software coprocessors. In this article, we introduce new error detection schemes embedded efficiently in the NTT accelerator architecture, detecting both transient and permanent faults. By encoding the operands with two approaches, i.e., negating and swapping, we detect the faults in such constructions after recomputing and decoding. Through simulation, our schemes show high error coverage for the stuck-at fault model. Moreover, we implement the schemes on field-programmable gate array (FPGA) and assure that acceptable overhead is achieved for performance and implementation metrics. The low overhead and high efficiency of our schemes make them suitable for various constrained usage models. Additionally, our schemes are also applicable to similar classical and post-quantum sub-blocks to obtain more reliable respective hardware constructions.**

***Index Terms*—Field-programmable gate array (FPGA), number-theoretic transform (NTT), post-quantum cryptography (PQC), recomputing with negated operands (RENO).**

I. Introduction

Hardware benchmarking is crucial to analyze the performance and security of the different post-quantum cryptographic (PQC) algorithms in NIST PQC standardization competition which is now finalized with Kyber as the key encapsulation mechanism (KEM) standard in 2022. To optimize performance, the inclusion of software implementation along with hardware is an emerging research topic of PQC [1], [2]. The software processor ranges from low-power embedded system processors, e.g., RISC-V and ARM Cortex-M4, to high-performance general-purpose processors, e.g., Intel core i7.

In the final round of the NIST PQC standardization project, lattice-based cryptography was one of the most extensively researched families including the decided standard, i.e., Kyber [3]. Number-theoretic transform (NTT), a finite field modification of the fast Fourier transform (FFT), is an elegant polynomial multiplication technique, essential to lattice-based cryptosystem. As polynomial multiplications are the most rigorous operations in such cryptosystems, applying NTT results in an efficient quasi-linear time complexity $O(n\log n)$, compared to the quadratic time complexity $O(n^2)$ of the schoolbook polynomial multiplication. Moreover, NTT is capable of enhancing the security parameters of various signature schemes, collision-resistant hash functions, and identification schemes. Because of such versatile applications of NTT, the performance improvement of NTT via hardware/software (HW/SW) co-design is gaining attention in the research community, hence the secure operation of NTT HW/SW design is crucial to boost the reliability and efficiency of such architectures.

Full hardware implementations of NTT have been extensively explored in prior works, such as [4], [5], and [6]. However, an HW/SW co-design variant has been proposed in [7] generating acceptable overheads with the ease of the benchmarking procedure. Such HW/SW co-designs provide the first glimpse into each candidate's suitability for hardware acceleration while achieving significant speed-up. The prior works also establish an open source code based on which optimized implementation protected against side channel and fault attacks can be built in future. These approaches reduce the development time by trading off a small increase in execution time.

In this article, we introduce error detection constructions for NTT HW/SW co-design approaches to detect natural and malicious faults. Former research works have explored error detection schemes on various public and symmetric-key cryptosystems [8], [9], [10], [11]. Few other works on error detection for PQC have been performed in previous works presented in [12], [13], and [14], including a fully hardware NTT algorithm to detect error in the butterfly architecture, an indispensable NTT component [15]. The main contributions of this article are as follows.

1) We introduce error detection schemes for the hardware accelerator of fast polynomial multiplication found in NTT.
2) The proposed error detection schemes are based on recomputing and decoding through two variants. We apply both schemes to different segments of the NTT accelerator architecture, where they are capable of detecting the faults injected with high error coverage.
3) We simulate the proposed schemes injecting stuck-at faults at inputs for both permanent and transient faults, to determine the error coverage.
4) The proposed error detection schemes are assessed and the results show acceptable error coverage. We implement our schemes on field-programmable gate array (FPGA) to derive the overhead and performance metrics.

II. Preliminaries

Ideal lattices are defined by $\mathbb{R}_q = ([\mathbb{Z}_q/p\mathbb{Z}[x]]/[x^n+1])$ as a ring of polynomial, with $n-1$ degree and coefficients in $\mathbb{Z}_q$. Also, n is a power of 2, and q is a prime number where $q \equiv 1 \bmod 2n$. We can define multiplication of two polynomials $a(x),\ b(x) \in \mathbb{Z}_q$, as: $a(x).b(x) = \sum_{i=0}^{n-1}\sum_{i=0}^{n-1} a_i b_j x^{i+j} \mod f(x)$.

Algorithm 1 shows the steps for deriving the iterative NTT. NTT is a discrete Fourier transform, defined in a finite field, $\mathbb{Z}_q$. For a

Manuscript received 22 July 2021; revised 27 July 2022 and 5 October 2022; accepted 28 October 2022. Date of publication 1 November 2022; date of current version 20 June 2023. This work was supported by the U.S. National Science Foundation (NSF) under Award SaTC-1801488. This article was recommended by Associate Editor D. Atienza. *(Corresponding author: Mehran Mozaffari Kermani.)*

Ausmita Sarker and Mehran Mozaffari Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: asarker@usf.edu; mehran2@usf.edu).

Alvaro Cintas Canto is with the School of Technology and Innovation, Marymount University, Arlington, VA 22207 USA (e-mail: acintas@marymount.edu).

Reza 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).

Digital Object Identifier 10.1109/TCAD.2022.3218614

Algorithm 1 Iterative-NTT

Input: $a \in \mathbb{Z}_q[x]$ of length $n = 2^k$ with $k \in \mathbb{N}$ and a primitive n-th root of unity $\omega \in \mathbb{Z}_q$
Output: $y = \text{NTT}_\omega(a)$

$A \leftarrow$ Bit-reverse(a); $m \leftarrow 2$
while $m \leq N$ **do**
  $s \leftarrow 0$
  while $s < N$ **do**
    for i to $m/2 - 1$ **do**
      $N \leftarrow i.n/m$; $a \leftarrow s + i$; $b \leftarrow s + i + m/2$
      $c \leftarrow A[a]$; $d \leftarrow A[b]$
      $A[a] \leftarrow c + \omega^{N \bmod n} d \bmod q$
      $A[b] \leftarrow c - \omega^{N \bmod n} d \bmod q$
    end for
    $s \leftarrow s + m$
  end while
  $m \leftarrow m.2$
end while
return A

given primitive nth root of unity in $\mathbb{Z}_q$, $A(x)$ and $B(x)$ are polynomials under $\mathbb{Z}_q$, where both are generic forward NTT_ω (a) and NTT_ω (b), respectively: $A_i = \text{NTT}^n_\omega(a(x))_i = \sum_{j=o}^{n-1} a_j \omega^{ij} \bmod q$, $i = 0, 1, , \ldots, n-1$.

The NTT exists if and only if the block length n divides $p-1$ for every prime factor p of q, where q is a prime and n is a power of 2. Inverse NTT (INTT) is similar to computing NTT, replacing ω with ω^{-1} and introducing n^{-1}, i.e., $a_i = \text{INTT}^n_\omega(A(x))_i = n^{-1} \sum_{j=o}^{n-1} A_j \omega^{-ij} \bmod p$, $i = 0, 1, \ldots, n-1$. As p is a prime, the inverse of n, i.e., n^{-1}, can be computed in mod p, where $n.n^{-1} \equiv 1 \bmod p$. Applying NTT and INTT to compute polynomial multiplication reduces the time complexity from $O(n^2)$, i.e., schoolbook polynomial multiplication, to $O(n\log n)$.

Based on the design from [7], when $\log_2(n)$ is odd, the signal X allows the signals A, B, C, and D to pass directly to the SIPO unit. On the contrary, for even $\log_2(n)$, the multiplexers with the select signal X can be eliminated. The NTT hardware architecture loads four coefficients per clock cycle and places them into registers A, B, C, and D. When the multiplexer select $s = 0$, the circuit operates in the MUL mode only, whereas $s = 1$ performs NTT operation on the circuit.

III. Proposed Error Detection Scheme

In this section, we present our schemes designed to detect fault injections on the HW/SW co-design architecture of NTT. Our framework is based on constructing recomputing schemes, which incur low overhead and high error detection rate. We aim at detecting the presence of faults, both permanent and transient, on the most computationally exhaustive mathematical operations of the NTT accelerator architecture, i.e., the multiplication. Generally speaking, we incorporate our proposed error detection schemes, which perform encoded and decoded rounds of operation, and then compare the recomputed output with regular output (i.e., without any error detection circuit). As our recomputing schemes fall under concurrent error detection (CED) schemes, the recomputed output will be consistent with the regular round of output if no fault is present. We perform error detection on all four registers, i.e., A, B, C, and D, using recomputing with the encoded operands scheme.

A. Fault Model

Stuck-at fault models are popular for malicious fault injections, preferably, single-bit fault injections. However, technological constraints might complicate such injection for the attacker, where multiple-bit faults can be utilized. Repeated comparison of faulty and fault-free outputs can compromise the secret key by deriving the last subkey. Stuck-at faults can occur by both natural and malicious faults, thus our fault model covers both types of faults, which can be detected by the error detection schemes proposed in the following.

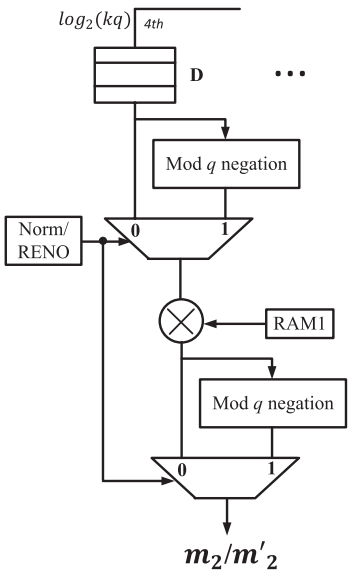

Fig. 1. Proposed construction for RENO on D.

B. Proposed Recomputing Scheme for Operand D

We propose error detection schemes which will result in low-complexity architectures while augmenting on top of the original constructions. In the hardware implementation of the NTT unit of hardware accelerator presented in the work of [7], we have applied recomputing with negated operands (RENO) schemes on the coefficient of line D, as shown in Fig. 1. In the original operation (Norm cycle of multiplexer), the data of register D is multiplied with RAM1, and the output m_2 is reduced using the Montgomery reduction. In our proposed RENO cycle, we negated the data of register D, which is mod q negation in modular arithmetic, and multiply $(-D)$ with RAM1. The encoded operand is $-D \times$ RAM1. According to Fig. 1, we again perform mod q negation on the product of this multiplication, obtaining the output m'_2, which should be consistent with the Norm cycle output, m_2, in a fault-free scenario. We compare the outputs from both the cycles m_2 and m'_2 using a comparator and any discrepancy flags presence of faults.

C. Proposed Recomputing Scheme for Operands B and A

In lines A and B of the same NTT unit, the select $s = 0/1$ determines the input of the multiplication with RAM2, which is B or A, during multiplication or NTT operation, respectively. Using our recomputing scheme (Fig. 2), we can detect faults during both these operations for both A and B.

During the Norm operation, i.e., at $s = 0$, the multiplication output is $B \times$ RAM2, defined as out_2. In our proposed RENO cycle of multiplexer, we negate the operand B using mod q negation. In the same line of logic as the previous scheme, we negate the product of $-B$ and RAM2, stored as out'_2, and compare it with out_2, as shown in Fig. 2. The error flag will be high to demonstrate any presence of faults.

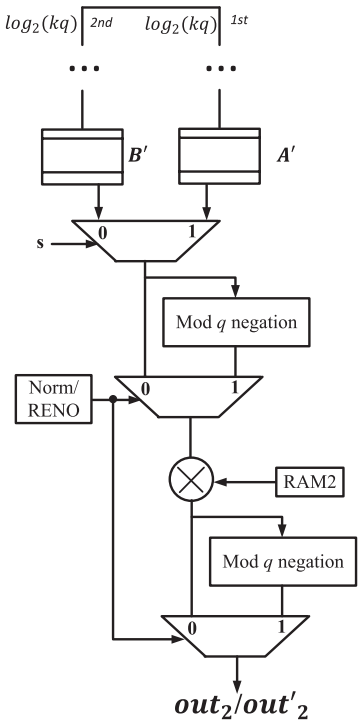

Fig. 2. Proposed construction for RENO on B and A.

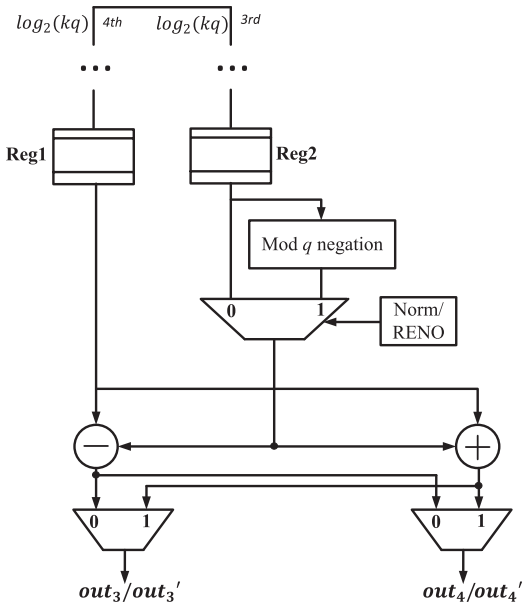

Fig. 3. Proposed construction for recomputing on C and D.

Moreover, during $s = 1$, the NTT operation is performed, which multiplies A with RAM2 during the Norm cycle. Our RENO operation will negate the input A using mod q negation and perform the multiplication. Another mod q reduction on this product should be consistent with the Norm cycle output; otherwise, the proposed construction will detect any deviations due to fault injection.

D. Proposed Recomputing Scheme for Operands C and D

Finally, we perform error detection on both C and D of the NTT unit in the hardware accelerator, through the last registers of their respective lines, Reg2 and Reg1, as shown in Fig. 3. During Norm cycle, where no error detection schemes are inserted, line D performs $\text{out}_3 = \text{Reg1} - \text{Reg2}$, whereas line C provides $\text{out}_4 = \text{Reg1} + \text{Reg2}$ as output, depicted by the boldface dashed lines of Fig. 4(a). We can check injection of faults in these operations by incorporating both recomputing with negated and swapped operands. In our RENO operation, as shown by the boldface dashed lines in Fig. 4(b), we negate Reg2 using a modular negation, which is then fed at both the subtractor and adder of lines D and C, respectively. In a fault-free scenario, the output at this stage from line D and C are $\text{out}'_3 = \text{Reg1} + \text{Reg2}$ and $\text{out}'_4 = \text{Reg1} - \text{Reg2}$, respectively. The final result is computed after swapping the outputs of these intermediate steps, which should be consistent with the Norm cycle output, in a fault-free scenario. We compare Norm/RENO output of line D, i.e., $\text{out}_3/\text{out}'_3$ and $\text{out}_4/\text{out}'_4$, using a comparator.

IV. Error Coverage and FPGA Assessments

In this section, we present the results of our error simulations and ASIC assessments using Xilinx Vivado and VHDL for three of our architectures to assess the overhead. We have injected faults on the basis of our fault models and observed the error flag in the error simulation section. We implemented our proposed techniques on Zynq UltraScale+ and Spartan-7 FPGA families which provide us the performance metrics, i.e., area, delay, and power overhead.

A. Fault Simulations

Using VHDL, we simulate the error coverage of our proposed work. We inject three types of stuck-at faults, i.e., 1) single; 2) two-bit; and 3) multiple-bit faults, all of which are injected at the input of the algorithm. Technological constraints might make it difficult for the adversary to flip just one bit and collect confidential information, which is why we are considering multiple-bit faults as well. We simulate both stuck-at 0 and stuck-at 1 faults. We injected 36 867 faults at each of our schemes at the input (a total of 110 601 faults). For the schemes presented in Sections III-B–III-D, the error coverage rates are 99.51%, 99.67%, and 99.41%, respectively.

Our schemes provide high error coverage and detect permanent and transient faults. Assuming the comparators are hardened, i.e., the comparators are not compromised, the schemes can detect fault injections successfully, according to the simulation results. However, compromised comparison units, i.e., voters, will deteriorate the error coverage, which can be solved by using other fault-tolerant techniques.

B. FPGA Assessments and Comparisons

In this section, we present the results of our FPGA assessments using Xilinx Vivado and VHDL with Zynq UltraScale+ (xczu4eg-fbvb900-1LV-i) as well as Spartan-7 (xc7s100fgga676-1IL) FPGA families. We choose the parameters used in NewHope, a well-known PQC algorithm, where, $n = 512$, $q = 12\,289$, and $k = 3$. The overheads are presented in Table I, the proposed RENO operations on A, B, and D as well as the recomputing scheme incorporated on C and D. The benchmarking is performed for the error detection architectures and also for the original constructions. As shown in Table I, the area is presented in LUTs and flip-flops (FFs), and the power consumption is calculated at 50-MHz frequency. Here, the term delay refers to maximum working frequency.

From Table I, the area overhead for Zynq UltraScale+ is 27.44%, 14.63%, and 18.25% for RENO on D, B, and A, and recomputing on C and D, respectively. We can also notice the delay overhead being the minimum of 9.32% for Fig. 1, whereas the lowest power overhead being 13.27% for the scheme of Fig. 3. The area overhead is lower

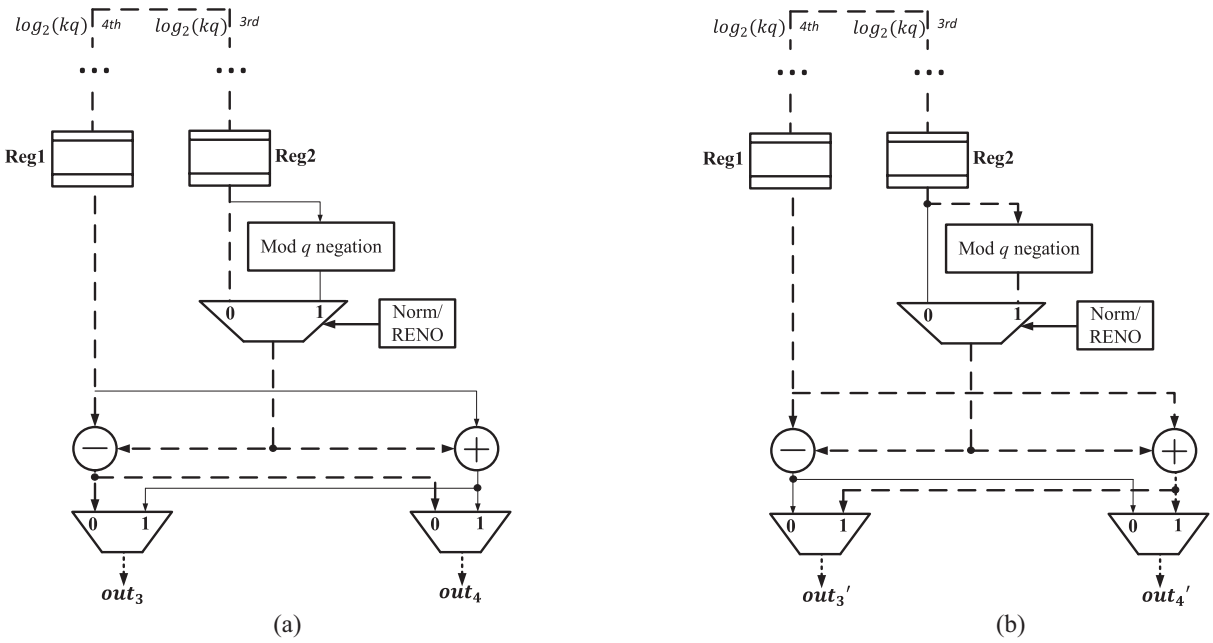

Fig. 4. Hardware architecture of recomputing on C and D. (a) Norm cycle, the select of multiplexer is normal. (b) RENO cycle, the select of multiplexer is RENO.

TABLE I
IMPLEMENTATION RESULTS FOR FPGA THROUGH ZYNQ ULTRASCALE+ (XCZU4EG-FBVB900-1LV-I) AND SPARTAN-7 (XC7S100FGGA676-1IL) FOR THE THREE PROPOSED ARCHITECTURES, I.E., (A) RENO ON D, (B) RENO ON B AND A, AND (C) RECOMPUTING ON C AND D FOR $n = 512$, $q = 12\,289$, AND $k = 3$

Architecture	Zynq UltraScale+ (xczu4eg-fbvb900-1LV-i)				Spartan-7 (xc7s100fgga676-1IL)			
	Area		Delay	Power	Area		Delay	Power
	LUT	FF	(ns)	(W)	LUT	FF	(ns)	(W)
Original (a)	277	235	15.78	1.27	429	398	13.07	1.03
RENO (a)	353	285	17.25	1.53	523	469	14.18	1.19
	(27.44%)	(21.28%)	(9.32%)	(20.47%)	(21.91%)	(17.84%)	(8.46%)	(15.62%)
Original (b)	265	212	19.87	1.13	376	344	17.39	0.96
RENO (b)	303	238	23.77	1.28	424	417	20.15	1.03
	(14.63%)	(12.26%)	(19.66%)	(13.27%)	(12.74%)	(21.22%)	(15.88%)	(7.62%)
Original (c)	389	345	40.31	7.82	318	293	30.12	6.33
RENO (c)	460	399	49.09	9.17	395	336	34.25	7.04
	(18.25%)	(15.65%)	(21.78%)	(17.26%)	(24.21%)	(14.68%)	(13.71%)	(11.22%)

in flip flops compared to LUTs for all the schemes. Table I also shows that RENO on B and A provides lower overall overheads than other two schemes. To conclude, the overheads are acceptable and can be incorporated for both high-performance and low-complexity architectures.

To the authors' knowledge, this is the first work to explore fault detection on the NTT HW/SW co-design approach. Thus, there is no existing article to compare with our implementation results. Some previous works have performed recomputing on NTT butterfly unit [15]. However, the RENO presented in [15] requires a decoding stage, which adds additional hardware cost, compared to the RENO we presented in Fig. 3 with no extra decode stage. The same logic applies to the recomputing presented in [14]. Such hardware improvement in the RENO scheme can be beneficial for resource-constrained hardware applications.

In absence of any compensation, the total time of the recomputed scheme will be twice that of the original, i.e., $2n$ cycles. To improve the data path delay and throughput, we incorporate subpipelining, which reduces the path delay by doubling the frequency. While such a technique incurs higher area overhead, we can highly compensate the throughput degradation via subpipelining. By inserting registers in the respective locations, the timing paths can be broken into approximately equal halves in a subpipelined architecture, thus improving throughput and frequency degradation. As a result, we can improve the efficiency and throughput of recomputing schemes using pipelines, with the tradeoff of slightly higher area overhead. Our proposed architectures are standard-cell library oblivious, as a result we expect similar results for ASIC platforms as well.

V. CONCLUSION

In this article, we present error detection schemes for the NTT found on HW/SW co-design constructions. We prove through our error simulation that our designs ensure high error coverage with low overheads. We implement the proposed schemes on FPGA Zynq UltraScale+ as well as Spartan-7, and our performance metrics add acceptable hardware overhead, e.g., 12.74%, 8.46%, and 7.62% being the best case area, delay, and power overheads, respectively, for

Spartan-7 FPGA family. HW/SW codesign approaches are receiving popularity due to flexibility, shorter development time, and easier benchmarking process. Hence, error detection schemes are crucial for the secure operation of PQC algorithms under adversarial attacks. As our schemes are platform oblivious, we expect similar results for ASIC for both permanent and transient faults. The error detection schemes can be employed on compact and resource constraint devices, e.g., Internet of Nano-Things and deeply embedded systems.

REFERENCES

[1] F. Farahmand, D. T. Nguyen, V. B. Dang, A. Ferozpuri, and K. Gaj, "Software/hardware codesign of the post quantum cryptography algorithm NTRUEncrypt using high-level synthesis and register-transfer level design methodologies," in *Proc. Int. Conf. Field Program. Logic Appl.*, Sep. 2019, pp. 225–231.

[2] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," Int. Assoc. Cryptol. Res., Lyon, France, Rep. 2020/795, Jul. 2021.

[3] J. Bos et al., "CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Security Privacy*, 2018, pp. 353–367.

[4] A. C. Mert, E. Karabulut, E. Ozturk, E. Savas, and A. Aysu, "An extensive study of flexible design methods for the number theoretic transform," *IEEE Trans. Comput.*, vol. 71, no. 11, pp. 2829–2843, Nov. 2022.

[5] T. Poppelmann and T. Guneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Proc. Progr. Cryptol. LATINCRYPT*, 2012, pp. 139–158.

[6] D. D. Chen et al., "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Trans. Circuits Syst. I, Reg2. Papers*, vol. 62, no. 1, pp. 157–166, Jan. 2015.

[7] D. T. Nguyen, V. B. Dang, and K. Gaj, "High-level synthesis in implementing and benchmarking number theoretic transform in lattice-based post-quantum cryptography using software/hardware codesign," in *Proc. Appl. Reconfig. Comput. Architect. Tools. Appl.*, Mar. 2020, pp. 247–257.

[8] M. M. Kermani, R. Azarderakhsh, A. Sarker, and A. Jalali, "Efficient and reliable error detection architectures of hash-counter-hash tweakable enciphering schemes," *ACM Trans. Embedded Computing Syst.*, vol. 17, no. 2, pp. 1–19, May 2018.

[9] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, "Security analysis of concurrent error detection against differential fault analysis," *J. Cryptol. Eng.*, vol. 5, no. 3, pp. 153–169, Dec. 2014.

[10] M. Yasin, B. Mazumdar, S. S. Ali, and O. Sinanoglu, "Security analysis of logic encryption against the most effective side-channel attack: DPA," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Oct. 2015, pp. 97–102.

[11] S. Saha, U. Kumar, D. Mukhopadhyay, and P. Dasgupta, "An automated framework for exploitable fault identification in block ciphers," *J. Cryptol. Eng.*, vol. 9, no. 3, pp. 203–219, May 2019.

[12] M. M. Kermani, R. Azarderakhsh, and A. Aghaie, "Fault detection architectures for post-quantum cryptographic stateless hash-based secure signatures benchmarked on ASIC," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 2, pp. 1–19, Dec. 2016.

[13] A. Sarker, M. M. Kermani, and R. Azarderakhsh, "Error detection architectures for ring polynomial multiplication and modular reduction of ring-LWE in $\frac{\mathbb{Z}_p[x]}{x^n+1}$, benchmarked on ASIC," *IEEE Trans. Rel.*, vol. 70, no. 1, pp. 362–370, Mar. 2021.

[14] A. Sarker, M. M. Kermani, and R. Azarderakhsh, "Fault detection architectures for inverted binary ring-LWE construction benchmarked on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 4, pp. 1403–1407, Apr. 2021.

[15] A. Sarker, M. M. Kermani, and R. Azarderakhsh, "Hardware constructions for error detection of number-theoretic transform utilized in secure cryptographic architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 3, pp. 738–741, Mar. 2019.