IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 13 March 2022; revised 15 June 2022; accepted 16 October 2022.
Date of publication 31 October 2022; date of current version 6 September 2023.

Digital Object Identifier 10.1109/TETC.2022.3217006

Error Detection Schemes Assessed on FPGA for Multipliers in Lattice-Based Key Encapsulation Mechanisms in Post-Quantum Cryptography

ALVARO CINTAS CANTO, (Member, IEEE), AUSMITA SARKER, (Member, IEEE), JASMIN KAUR, (Student Member, IEEE), MEHRAN MOZAFFARI KERMANI, (Senior Member, IEEE), AND REZA AZARDERAKHSH, (Member, IEEE)

Alvaro Cintas Canto is with the School of Technology and Innovation, Marymount University, Arlington, VA 22207 USA
Ausmita Sarker, Jasmin Kaur, and Mehran Mozaffari Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA
Reza Azarderakhsh is with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA
CORRESPONDING AUTHOR: MEHRAN MOZAFFARI KERMANI (mehran2@usf.edu)

This work was supported by Marymount University through the START under grant 2450100, and in part by the US National Science Foundation (NSF) through the award under Grant SaTC-1801488.

ABSTRACT Advances in quantum computing have brought the need for developing public-key cryptosystems secure against attacks potentially enabled by quantum computers. In late 2017, the National Institute of Standards and Technology (NIST) launched a project to standardize one or more quantum computer-resistant public-key cryptographic algorithms. Among the main post-quantum algorithm classes, lattice-based cryptography is believed to be quantum-resistant. The standardization efforts including that of the NIST which will be concluded in 2022-2024 also affirm the importance of such algorithms. In this work, we propose error detection schemes for lattice-based key encapsulation mechanisms (KEMs). As our case study, we apply such schemes to the hardware accelerators for three post-quantum cryptographic algorithms that have advanced to the third round of the NIST PQC standardization process, i.e., FrodoKEM, Saber, and NTRU. The merit of the proposed schemes is that they can be applied to other applications and cryptographic algorithms that use multiplications in their hardware accelerators. The schemes proposed in this paper are based on recomputing with shifted, negated, and scaled operands. Moreover, we implement our fault detection schemes on field-programmable gate array (FPGA) family Kintex Ultrascale+ device xcku5p-sfvb784-1LV-i to benchmark the overheads induced and the performance degradation of the proposed approaches when added to the original architectures. The results show acceptable overhead and high error coverage for all three studied NIST PQC finalists.

INDEX TERMS Fault detection, field -programmable gate array (FPGA), lattice-based, post-quantum cryptography

I. INTRODUCTION

Traditional public-key cryptosystems are believed to be vulnerable to attacks enabled by quantum computers [1]. In 2017, the National Institute of Standards and Technology (NIST) began the process to standardize quantum-resistant public-key cryptographic algorithms; algorithms that make cryptographic systems secure against both classical and quantum computers, and can work with communication protocols and networks that currently exist [2]. This effort is in its final stage as of 2022 and it is expected to conclude in 2022-2024.

Lattice-based cryptosystems have been used for many years in traditional known public-key schemes such as the RSA, Diffie-Hellman, or elliptic-curve cryptosystems. Lattice-based cryptography involves all cryptographic primitives that include lattices, either in the construction itself or in the security proof. Their high efficiency, strong security guarantees from worst-case hardness, and simplicity to implement, make lattice-based cryptography a promising quantum-resistant class. Currently, there are three finalists and two alternates which are lattice-based cryptosystems in the NIST standardization process.

Traditionally, hardware benchmarking was performed separated from software benchmarking; however, due to the mathematical complexity of post-quantum cryptography (PQC) algorithms and the hardware resources required, pure hardware implementations are difficult to achieve. There are only a few pure hardware implementations in the NIST PQC standardization process, e.g., [3] and [4]. The challenges of performing PQC algorithms purely hardware dedicated require new approaches [5]–[9]. The authors of [10] and [11] propose software/hardware codesigns to solve these challenges, developing different hardware accelerators for lattice-based key encapsulation mechanisms (KEMs). Such hardware accelerators involved many rewiring and operations, where multiplications are one the most costly and time-consuming processes.

As previous research has shown, lattice-based cryptography is vulnerable to differential fault analysis attacks (DFA), which is a type of side-channel attack where faults are induced into the cryptographic implementation to reveal data. The side-channel security aspect of lattice-based cryptography has received limited attention, but it is necessary to study it since most of these attacks allow to recover the secret key making the entire system vulnerable and not secure enough. In [12], the authors perform successfully partial key exposure attacks on BIKE, Rainbow and NTRU. The authors of [13] propose side-channel attacks on BLISS lattice-based signatures that can yield a full key recovery using branch tracing. In [14], researchers work with polynomials over the ring $\frac{\mathbb{Z}_q[x]}{x^n+1}$ used in the RLWE-based public-key encryption scheme and prove that they can recover the entire key by using the leakage coming from the Number Theoretic Transform (NTT) of such scheme. The research work in [15] shows DFAs in CRYSTALS-Dilithium and qTESLA. In particular, they prove that up to 65.2% of the execution time of Dilithium is vulnerable to DFAs leading to fully key recovery. There has been some previous research providing countermeasures against side-channel attacks for lattice-based cryptographic systems such as [16] and [17]. In [16], the authors propose a practical fault analysis attack against NTRUEncrypt using polynomial inversions in $\frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^n-1}$, providing countermeasures to such attacks based on checksums and spatial/temporal redundancy in [17]. A major drawback with checksums is the potential for high hardware overhead when high error coverage is achieved. In [18]–[20], more countermeasures against side-channel attacks are proposed.

However, to the best of our knowledge, in this work, we propose for the first time error detection schemes based on recomputing for the hardware accelerators of FrodoKEM, Saber, and NTRU. Such schemes have the advantage of covering more faults than checksum signatures, they successfully detect both permanent and transient faults, they provide an acceptable overhead for such high error detection coverage, and they are usable by major lattice-based post-quantum algorithms advanced to the final round of the NIST PQC standardization process. Additionally, we calculate the performance degradation and the overheads of the proposed error detection schemes. This is done by embedding the fault detection architectures into the original constructions. The benchmark has been done using Xilinx FPGA family Kintex Ultrascale+ device xcku5p-sfvb784-1LV-i to assess the efficiency of the proposed schemes.

II. PRELIMINARIES

In this paper, we propose error detection architectures for hardware accelerators of the lattice-based KEMs proposed in [10] and [11], i.e., FrodoKEM, Saber, and NTRU. Some of the benefits of using KEMs instead of the traditional approach of public-key encryption are that the length of the message is not limited, they can provide integrity protection, and the symmetric keys are unrelated, avoiding mathematical properties. We will go over some important aspects of FrodoKEM, Saber, and NTRU algorithms; however, if the reader is interested in a more detailed explanation of such cryptographic algorithms, please refer to [21], [22], and [23], respectively.

FrodoKEM, Saber, and NTRU are based on the Learning with Errors (LWE), Module Learning with Rounding (Mod-LWR), and Shortest Vector problems, respectively. The most time-consuming operation of such cryptographic algorithms is matrix-by-matrix multiplication for the case of FrodoKEM, matrix-by-vector and vector-by-vector multiplications (or polynomial multiplication since its matrix and vector elements are polynomials) for Saber, and polynomial multiplication in NTRU. The multiplications carried by each of these cryptosystems are modulo a power of two, expressed as q. Other important parameters of these lattice-based cryptographic algorithms are:

1) For FrodoKEM, n stands for the matrix dimensions, B is the number of bits encoded in each matrix entry, and σ specifies the standard deviation.
2) For Saber, n is the degree of the polynomial ring, l is the rank, and μ stands for the coefficients of the secret vectors.
3) For NTRU, n is the degree of the polynomial P and kcb is the number of bytes in a ciphertext for the KEM.

The FrodoKEM matrices are formed by elements of the ring $\mathbb{Z}_q$, Saber matrices and vectors are formed by elements of the ring $\frac{\mathbb{Z}_q[x]}{x^n+1}$, and NTRU elements use three different rings: $\frac{\mathbb{Z}_q[x]}{x^n-1}$, $\frac{\mathbb{Z}_q[x]}{(x^n-1)/(x-1)}$, and $\frac{\mathbb{Z}_3[x]}{(x^n-1)/(x-1)}$. To eliminate the need for padding, FrodoKEM uses SHAKE as the hash-based function, Saber uses SHAKE, SHA3-256, and SHA3-512, and NTRU uses SHA3-256. In Table 1, we provide the different parameter sets for the FrodoKEM, Saber, and NTRU algorithms.

III. PROPOSED FAULT DETECTION SCHEMES

A. FAULT MODEL

One can observe several fault models based on different facets of the attacks. These models are based on the number of bits compromised, the location of the faults, how the faults are introduced, and how long the faults last. An attacker may not be able to flip precisely one bit to capture crucial information due to technological limitations. However, to save time and effort, the

TABLE 1. Different parameter sets for FrodoKEM, Saber, and NTRU.

Algorithm	Security Level	n	q	Other Parameters
Frodo-640	1	640	2^{15}	$B = 2, \sigma = 2.8$
Frodo-976	3	976	2^{16}	$B = 3, \sigma = 2.3$
Frodo-1344	5	1344	2^{16}	$B = 4, \sigma = 1.4$
LightSaber-KEM	1	256	2^{13}	$l = 2, \mu = 10$
Saber-KEM	3	256	2^{13}	$l = 3, \mu = 8$
FireSaber-KEM	5	256	2^{13}	$l = 4, \mu = 6$
ntruhps2048509	-	509	2^{11}	$kcb = 699$
ntruhps2048677	1	677	2^{11}	$kcb = 930$
ntruhps4096821	3	821	2^{12}	$kcb = 1230$
ntruhrss701	1	701	2^{13}	$kcb = 1138$

attacker tries to introduce as few errors as possible (preferably single faults of different intensities). There are two main types of faults depending on the duration of the faults: Transient faults, which are the most common ones, lasting one or a few clock cycles and when the system retries to perform the affected operation, the fault disappears; and permanent faults, which last longer than transient faults, and the system will have to be reset or the value where the fault injection took place will have to be overwritten for the device to output the correct values. The error detection schemes derived in this work are based on recomputing with shifted, negated, and scaled operands, which consider both transient faults and permanent internal faults (and the proposed schemes are oblivious of these). Additionally, these techniques can identify numerous stuck-at faults (both stuck-at one and stuck-at zero situations), adjacent (for interleaved cases), single or multiple stuck-at faults.

In this paper, we propose error detection architectures for hardware accelerators of the lattice-based KEMs FrodoKEM, Saber, and NTRU. We provide recomputing to the different multiplication modules that each KEM uses, i.e., matrix-by-matrix, matrix-by-vector, vector-by-vector, and polynomial multiplications. These multiplications are performed by the units 4*MAC* and *MAC*. 4*MAC* is used in the hardware accelerator of FrodoKEM, while the unit *MAC* is used in the hardware accelerators of Saber and NTRU. The *MAC* and 4*MAC* units are essentially performing vector-by-vector multiplications. However, they can by used to perform matrix-by-matrix, matrix-by-vector, and polynomial multiplications. For example, to perform matrix-by-vector multiplication in the hardware accelerator of FrodoKEM, the unit 4*MAC* takes one row of matrix A, which is multiplied by the coefficients of vector B, and the coefficients of vector C are added to obtain the first bits of ACC. Subsequently, the other rows of A are being used in parallel to obtain $ACC = AB + C$. On the other hand, the unit *MAC* basically performs one of the iterations of the unit 4*MAC* obtaining $ACC = AB + C$. We note that A, B, and C are considered vectors throughout this paper for simplicity; however, for the matrix-by-vector example that we just proposed, A is considered a matrix. In the next subsections, we will use the unit *MAC* to derive the different error detection schemes, but the proposed schemes are applicable to the 4*MAC* unit as well, which means that all of them can be used for the hardware accelerators of FrodoKEM, Saber, and NTRU.

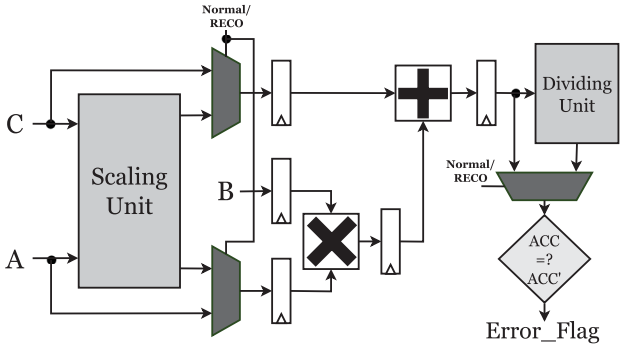

FIGURE 1. *MAC* **unit with the proposed RECO scheme.**

B. RECO

The error detection schemes provided in this paper aim to provide a high level of security at the expense of low overheads, which are needed for high-performance low-energy deeply-embedded systems like wearable medical devices. The first scheme presented is based on recomputing with scaled operands (RECO), shown in Figure 1. RECO has been used extensively in research and one of its main benefits is that RECO is a superset of recomputing with shifted operands (RESO) and recomputing with negated operands (RENO) (presented in the next subsections), providing flexibility depending on the scaling integer used. By scaling, the operands are multiplied by an integer t. If t is -1, it would be negation, and if t is 2, 4, 8, etc., it would be shifting by 1, 2, 3, etc. However, not all scalings are efficient: For example, scaling by 3 needs shift and add but by 4 needs just two shifts in hardware.

As mentioned above, in the recomputation step for RECO, the operands are scaled by being doubled, quadrupled, or multiplied by a specific factor t. After the computing step is finished and ACC is obtained, the *RECO* signal is selected to start the recomputing step. In this step, operands A and B are multiplied by t by using the *Scaling Unit* as depicted in Figure 3. The operand C is then multiplied by t^2 by also using the *Scaling Unit* since both operands A and B have a common factor of t. To obtain ACC', the addition of the scaled operands AB with the scaled operand C is divided by $2 \cdot t$ using the *Dividing Unit* to obtain

$$ACC' = \frac{(t \cdot A \cdot t \cdot B + t^2 \cdot C)}{t^2}. \quad (1)$$

The result of the computed step (ACC) is finally compared with the result of the recomputed step to detect if any faults are present in the system. For the RECO scheme, the sizes of the registers for operands A and B are $n + t_{LengthInBits}$-bits long, while the size of the operand C register is $n + t^2_{LengthInBits}$-bits, where n is the size of the original operands. For example, if $A = 010$, $B = 001$, $C = 011$, and $t = 3$, three new registers

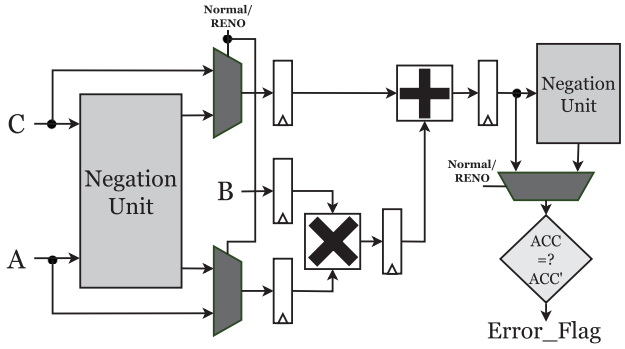

FIGURE 2. *MAC* unit with the proposed RENO scheme.

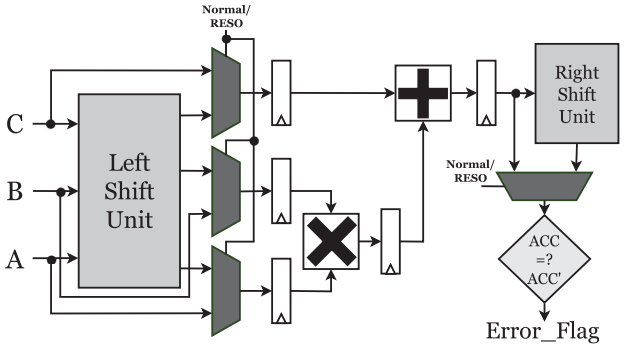

FIGURE 3. *MAC* unit with the proposed RESO scheme.

are created so when A, B, and C are scaled, $A' = 00110$, $B' = 00100$, and $C' = 0011011$ are obtained. ACC will be 101 and $ACC' = \frac{(t\cdot A\cdot t\cdot B + t^2\cdot C)}{2\cdot t} = \frac{00110\cdot 00011 + 0011011}{1001} = \frac{101101}{0110} = 101$. Therefore, $ACC = ACC'$.

C. RENO

The next variant is based on RENO. The schematic of this variant is provided in Figure 2. During the computation step (*Normal* signal), the original *MAC* operation is performed without any modifications to obtain ACC. Next, the *RESO* signal is selected for the recomputation step, where the operands A and C are negated through the *Negating Unit* to obtain $-A$ and $-C$. The *Negating Unit* performs the two's complement of A and C, where they are inverted and added with a single '1' bit. Then, the result of multiplying $-AB$ is stored, C is subtracted (which is the same as doing the addition of C negated), and ACC' is calculated by performing a last negation, obtaining

$$ACC' = -(-AB - C). \tag{2}$$

To detect if any faults are present in the system, ACC' is compared with the precomputed ACC value by using XOR gates. In this variant, the sizes of the operand registers are n, where n is the bit-size of the inputs. For example, if $A = 010$, $B = 001$ and $C = 011$, three new registers are created so when A and C are negated, $A' = 110$, $B' = 001$, and $C' = 101$ are obtained. ACC will be 101 and $ACC' = -(-AB - C) = -(110 \cdot 001 + 101) = -(1011) = 101$. Therefore, $ACC = ACC'$.

D. RESO

Lastly, this variant is based on RESO and it is shown in Figure 3. The *select* signal of the different multiplexers determines if the system is performing the original *MAC* operation (*Normal* signal) or the recomputing scheme (*RESO* signal). In the initial computation step, the *Normal* signal is selected and ACC is calculated by performing by multiplying the inputs A and B first and adding input C after, such as $ACC = AB + C$. Once the output ACC is calculated and stored in a register, the *RESO* signal is selected so the system can start the recomputation step, where the operands A, B, and C are shifted k-bits by the *Left Shift Unit*. The shifted operands are then stored in registers to obtain a pipelined implementation. Once the multiplication of the operands A and B is calculated, C is added and the result is shifted right by k-bits through the *Right Shift Unit* to obtain

$$ACC' = shift_{2r}((shift_l(A))(shift_l(B)) + (shift_{2\,l}(C))). \tag{3}$$

ACC is then compared to ACC' to detect any faults. In RESO, if the operands are of size n-bits, then each register needs to have a size of $(n + k)$-bits since the operands are being shifted k-bits. For example, if $A = 010$, $B = 001$ and $C = 011$, three new registers are created so when A, B, and C are shifted, $A' = 0100$, $B' = 0010$, and $C' = 01100$ are obtained. ACC will be 101 and $ACC' = shift_{2r}((shift_l(A))(shift_l(B)) + (shift_{2\,l}(C))) = shift_{2r}(0100 \cdot 0010 + 01100) = shift_{2r}(10100) = 101$. Therefore, $ACC = ACC'$.

E. COST OF THE PROPOSED ERROR DETECTION SCHEMES

For the sake of comparison in terms of area (occupied slices), delay (ns), and power (mW) with the clock frequency of 50 MHz between RECO, RENO, and RESO, we have added them to the original *MAC* and 4*MAC* units architectures and implemented them on Xilinx FPGA family Kintex Ultrascale + device xcku5p-sfvb784-1LV-i using the Vivado tool. As we can see in Table 2, the most suitable error detection scheme for deeply-constrained devices is RESO due to the use of shifting, which has no cost in hardware. The worst proposed scheme in terms of area, delay, and overhead is for RECO when used with a non-power of 2 scaling integer since it not only needs shifting, but also addition (for the RECO implementations, we have used a scaling integer of 3). The proposed schemes have an error coverage of close to 100%; therefore, RESO (or RECO when t is a power of 2) is the best choice as not only has high error coverage, but also an acceptable hardware cost. In the next section, more in-depth implementations are carried out along with an analysis on the error detection capabilities.

IV. ERROR COVERAGE AND FPGA IMPLEMENTATIONS

The proposed error detection schemes not only have to provide a high error coverage percentage, but they also need to achieve acceptable overheads for deeply-embedded systems. The hardware accelerators of Saber and NTRU (ntruhps4096821) use

TABLE 2. An illustrative example with FPGA implementation results for the original *MAC* and 4*MAC* architectures using RECO, RENO, and RESO.

Architecture	Scheme	Area (CLBs)	Delay (ns)	Power (mW)
Saber *MAC*	RECO	266	0.426	19.572
	RENO	65	0.424	5.194
	RESO	19	0.424	4.956
NTRU *MAC*	RECO	257	0.426	19.818
	RENO	62	0.424	5.946
	RESO	18	0.424	4.851
FrodoKEM 4*MAC*	RECO	1207	0.473	22.376
	RENO	302	0.467	7.166
	RESO	87	0.461	5.476

256 and 821 *MAC* units, respectively, while the hardware accelerator of FrodoKEM uses 8 4*MAC* units. In Figure 4, an overall structure of the proposed recomputing schemes embedded in FrodoKEM's hardware accelerator is presented. As it is shown in Figure 4, a total of 8 error flags, denoted as *EF* with index 1-8, are obtained by XORing the original output *ACC* with the output of the recomputing block.

To confirm that the proposed error detection schemes are overhead-aware and that they provide high error coverage, the RESO schemes are embedded into every *MAC* and 4*MAC* unit in the Saber, NTRU, and FrodoKEM hardware accelerators and implemented on Xilinx FPGA family Kintex Ultrascale+ device xcku5p-sfvb784-1LV-i. The error coverage is evaluated by simulating fault injection in Xilinx and using Verilog as our hardware design entry. We note that this is a fairly recent FPGA family; nevertheless, because our schemes are independent of the utilized FPGA device/family/vendor, we expect similar results on other FPGAs as well. Due to technological limitations, an adversary may not be able to flip precisely one bit to capture sensitive information. Therefore, we inject three types of faults in our simulations, i.e., single, 2-bit, and multiple-bit faults. More than 7×10^5 simulations are performed for the three different hardware accelerators, obtaining a close to 100% error coverage (all the injected faults are detected). We assume that the comparators are hardened, i.e., the comparators are fault free and not compromised, and that the inputs are not compromised

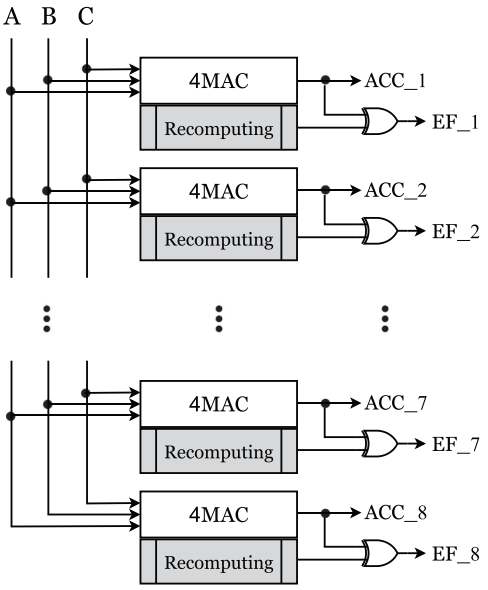

FIGURE 4. Proposed recomputing schemes embed in FrodoKEM's hardware accelerator.

prior to the execution of the *MAC* and 4*MAC* units. The proposed detection schemes compare the original output to produce an error flag. If this error flag signals '1,' a fault has been detected.

Moreover, we present the overhead results in terms of area (CLBs), delay, and power consumption (at the frequency of 50 MHz) for the implementations of the original *MAC* and 4*MAC* units with our proposed error detection based on recomputing in Table 3. For the implementations, we create internal inputs from a seed using a shifter, and the outputs are ORed once calculated. To obtain the area, CLBs are read from Vivado's location utilization report. We utilize the Timing Constraints Wizard function in Vivado to calculate the delay, setting a primary clock period restriction of 20 ns, which corresponds to a frequency of 50 MHz. The total on-chip power, which is the power consumed internally within the FPGA and is calculated by summing device static power and design power, is also reported. Recomputing schemes decrease the throughput greatly due to computing the operations twice; however, by using pipelined implementations, the throughput does not decrease as much. From Table 3, we can see that the overheads of the implemented architectures are acceptable.

TABLE 3. Overheads of the proposed error detection schemes based on RESO for the *MAC* and 4*MAC* units found in the saber, NTRU, and FrodoKEM hardware accelerators on Xilinx FPGA family kintex ultrascale+ device xcku5p-Sfvb784-1LV-I.

Architecture	Area (CLBs)	Delay (ns)	Power (mW) @50 MHz
Saber *MAC* units	3,991	4.544	0.491
Saber *MAC* units using Recomputing	5,451 (36.6%)	5.833 (28.3%)	0.497 (1.2%)
NTRU *MAC* units	11,820	5.088	0.630
NTRU *MAC* units using Recomputing	16,503 (39.6%)	5.939 (16.7%)	0.650 (3.2%)
FrodoKEM 4*MAC* units	640	4.742	0.451
FrodoKEM 4*MAC* units using Recomputing	822 (28.4%)	6.291 (32.7%)	Negligible Over.

TABLE 4. Worst-case overhead comparison of the presented schemes with other fault detection works.

Work	Fault Detection Scheme	Worst-Case Overhead %			Error Coverage %
		Area	Delay	Power	
[17]	Spatial duplication	6.22	Not given	Not given	100
[18]	Recomputing with swapped ciphertext and additional authenticated blocks (plain/pipelined)	4.9/6.7	Not given	Not given	>99.9
[19]	Normal parity/2-bit parity/3-bit parity	9.78/11.35/9.57	1.39/0.84/1.00	2.74/2.74/2.74	100
[20]	Statistical tests (low cost/standard/expensive)	8/44/85	Not given	Not given	Not given
[24]	CRC-5	18.33	11.25	≃0	>99.9
This work	RESO (Saber/NTRU/FrodoKEM)	36.6/39.6/28.4	28.3/16.7/32.7	1.2/3.2/Negl.	>99.9

The area overhead of the error detection schemes based on recomputing on top of the Saber hardware accelerator is 36.6%, while the delay increments a 28.3%, and the power goes up by 1.2%. The area overhead added into the NTRU hardware accelerator is 39.6%, while the delay increments a 16.7%, and the power goes up by 3.2%. Lastly, the area overhead added into the FrodoKEM hardware accelerator is 28.4%, while the delay increments a 32.7%, and the power overhead is negligible. These overhead increases are due to additional CLBs, clock-cycles, and power usage that are needed to calculate the recomputed outputs, which are then compared to original output to detect if a fault has occurred. Moreover, the worst-case scenario in terms of area overhead is 39.6% for the hardware accelerator of the lattice-based KEM NTRU and a worst-case scenario in terms of delay overhead of less than 33% for the FrodoKEM hardware accelerator. Furthermore, the power overheads added to the original architectures are less than 4%, obtained by the NTRU hardware accelerator. As it is shown, the proposed error detection schemes add acceptable overheads considering that they provide a close to 100% error coverage.

A. COMPARISONS WITH OTHER SCHEMES FOR PQC

To the best of the authors' knowledge, there is no previous research on these types of error detection schemes for the *MAC* and 4*MAC* units found in the hardware accelerators of Saber, NTRU, and FrodoKEM. However, for a qualitative comparison, we will go over some case studies to verify that the overheads incurred are acceptable. In [24], authors performed concurrent error detection based on cyclic redundancy check (CRC) signatures for finite field multipliers on FPGA with the Luov cryptosystem (a multivariate-based PQC algorithm) as a case study, obtaining worst-case area and delay overheads of approximately 18.3% and 11.3%, respectively. Moreover, error detection architectures for redundant arithmetic-based inversion in $GF(2^8)$ are presented in [18], obtaining a worst-case area overhead of 35.6%. Additionally, Table 4 shows a comparison on the implementation cost and error detection capability between the proposed methods and other previous works. These and other prior works on classical and post-quantum cryptography demonstrate that the proposed techniques have equivalent overheads to other fault-detection architectures, resulting in a reasonable overhead. These degradations are acceptable due to the error detection offered to original architectures that lack the ability to identify faults.

V. CONCLUSION

Software/hardware codesigns solve the challenge of performing PQC algorithms purely hardware dedicated by developing different hardware accelerators for lattice-based KEMs. Such hardware accelerators are vulnerable to differential fault analysis attacks and it is extremely important to provide countermeasures against such attacks. In this work, we have derived different error detection schemes based on recomputing with negated, shifted, and scaled operands for the FrodoKEM, Saber, and NTRU hardware accelerators. Moreover, we embedded the proposed fault detection architectures into the original constructions to calculate the performance degradation and the overheads of the proposed schemes. This is done by using Xilinx FPGA Kintex Ultrascale+ xcku5p-sfvb784-1LV-i, showing a high error coverage. The proposed error detection schemes have at most 39.6% area overhead, obtained by the hardware accelerator of the lattice-based KEM NTRU, a worst-case scenario in terms of delay overhead of less than 33% obtained by the FrodoKEM hardware accelerator, and worst-case scenario of less than 4% in terms of power overhead, obtained by the NTRU hardware accelerator. The proposed schemes are compared with other fault detection works to demonstrate that the techniques proposed in this work have a reasonable overhead.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. IEEE Annu. Symp. Foundations Comput. Sci.*, 1994, pp. 124–134.
[2] D. Moody, "Post-quantum cryptography: NIST's plan for the future," in *Proc. 7th Int. Conf. Post-Quntum Cryptography*, Feb. 2016.
[3] W. Wang, J. Szefer, and R. Niederhagen, "FPGA-based niederreiter cryptosystem using binary Goppa codes," in *Proc. Int. Conf. Post-Quantum Cryptography*, 2018, pp. 77–98.
[4] J. Howe, T. Oder, M. Krausz, and T. Güneysu, "Standard lattice-based key encapsulation on embedded devices," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2018, pp. 372–393, 2018.
[5] K. Gaj, "Challenges and rewards of implementing and benchmarking post-quantum cryptography in hardware," in *Proc. Great Lakes Symp. VLSI*, 2018, pp. 359–364.
[6] Y. Xing and S. Li, "A compact hardware implementation of CCA-secure key exchange mechanism CRYSTALS-KYBER on FPGA," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2021, no. 2, pp. 328–356, 2021.

[7] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 4, pp. 443–466, 2020.

[8] J. W. Bos, M. Ofner, J. Renes, T. Schneider, and C. V. Vredendaal, "The matrix reloaded: Multiplication strategies in FrodoKEM," in *Proc. Int. Conf. Cryptol. Netw. Secur.*, 2021, Art. no. 72–91.

[9] P. A. Fouque, P. Kirchner, T. Pornin, and Y. Yu, "BAT: Small and fast KEM over NTRU lattices," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2022, no. 2, pp. 240–265, 2022.

[10] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," in *Proc. IEEE Int. Conf. Field-Programmable Technol.*, 2019, pp. 206–214.

[11] F. Farahmand, V. B. Dang, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking seven round 2 lattice-based key encapsulation mechanisms using a software/hardware codesign approach," in *Proc. Second PQC Standardization Conf.*, 2019, pp. 22–24.

[12] A. Esser, A. May, J. Verbel, and W. Wen, "Partial key exposure attacks on BIKE, rainbow and NTRU," in *Proc. Annu. Int. Cryptology Conf.*, 2021, pp. 346–375.

[13] T. Espitau, P. A. Fouque, B. Gerard, and M. Tibouchi, "Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, Art. no. 1857.

[14] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2017, Art. no. 513.

[15] G. Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2018, Art. no. 21–43.

[16] A. Kamal and A. M. Youssef, "Fault analysis of the NTRUEncrypt cryptosystem," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E94A, 2011, Art. no. 1156.

[17] A. Kamal and A. Youssef, "Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks," *J. Cryptographic Eng.*, vol. 3, no. 4, 2013, Art. no. 227.

[18] M. Mozaffari-Kermani and R. Azarderakhsh, "Reliable architecture-oblivious error detection schemes for secure cryptographic GCM structures," *IEEE Trans. Rel.*, vol. 68, no. 4, pp. 1347–1355, Dec. 2019.

[19] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh, "Reliable architectures for composite-field-oriented constructions of McEliece post-quantum cryptography on FPGA," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 40, no. 5, pp. 999–1003, May 2021.

[20] J. Howe, A. Khalid, M. Martinoli, F. Regazzoni, and E. Oswald, "Fault attack countermeasures for error samplers in lattice-based cryptography," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2019, pp. 1–5.

[21] M. Naehrig et al., "FrodoKEM: Practical quantum-secure key encapsulation from generic lattices," NIST submissions, 2019.

[22] J. P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "SABER: Mod-LWR based KEM," NIST submissions, 2019.

[23] A. HÃêlsing, J. Rijneveld, J. M. Schanck, and P. Schwabe, "NTRU-HRSS-KEM: Algorithm specifications and supporting documentation," NIST submissions, 2019.

[24] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh, "Reliable CRC-based error detection constructions for finite field multipliers with applications in cryptography," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 1, pp. 232–236, Jan. 2021.