Reliable Architectures for Finite Field Multipliers Using Cyclic Codes on FPGA Utilized in Classic and Post-Quantum Cryptography

Alvaro Cintas-Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh

***Abstract*— Fault detection is becoming greatly important in protecting cryptographic designs that can suffer from both natural or malicious faults. Finite fields over GF(2^m) are widely used in such designs, since their data are coded in binary form for practical reasons. Among the different finite field arithmetic, multiplication is the bottleneck operation for many cryptosystems due to its complexity. Therefore, in this work, fault detection schemes based on cyclic codes for finite field multipliers using different fields found in traditional and post-quantum cryptography are derived. Moreover, we implement such schemes by embedding them into the original architectures to perform an exhaustive study, benchmark the different overheads obtained, and prove their suitability for deeply constrained embedded systems. These implementations are performed on advanced micro devices (AMD)/Xilinx field-programmable gate array (FPGA) and provide a very high error coverage with acceptable overhead.**

***Index Terms*— Cyclic codes, fault detection, field-programmable gate array (FPGA), finite field multiplication.**

I. Introduction

Finite field arithmetic and its hardware implementations have gained considerable interest in the literature due to their applicability to cryptography, coding theory, error-correcting codes, and digital signal processing. Most of the runtime of such applications is spent in the computation of multiplications over GF(2^m).

These implementations not only have to be efficient but also secure and free of errors. There are two main types of faults for hardware implementations, i.e., natural faults, which are products of the environment or defects in hardware logic operations, and malicious or intentional faults, which are faults injected by an unauthorized party to generally obtain secret information or even produce a denial of service attack. Even though there are many works on fault detection [1], [2], [3], [4], [5], [6], efficient and reliable hardware implementations of finite field multipliers based on cyclic codes are still lacking in the literature. Moreover, most of the exciting works on fault detection for finite fields are based on either 1-bit parity or multibit parity schemes [7], [8], [9], [10]. The issue with the former one is that if the number of faults is even, they remain undetected, providing an error coverage percentage of 50% at most. Multiparity increases the error coverage, but it is insufficient for intelligent fault injection. This brief proposes fault detection schemes based on cyclic codes, the most commonly used class of linear block codes, to overcome this. Cyclic codes are not only useful in detecting single, double, and even triple errors, but they can also detect burst errors, which are those faults that occur in many consecutive bits rather than happening in bits independent of each other.

Manuscript received 16 August 2022; revised 2 November 2022; accepted 20 November 2022. Date of publication 30 November 2022; date of current version 28 December 2022. This work was supported in part by the Marymount University through the START under Grant 2450100 and in part by the U.S. National Science Foundation (NSF) under Award SaTC-1801488. *(Corresponding author: Mehran Mozaffari Kermani.)*

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

Mehran Mozaffari Kermani is with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: mehran2@usf.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/TVLSI.2022.3224357

TABLE I
CRYPTOGRAPHIC ALGORITHMS AND THEIR RESPECTIVE FIELDS

Algorithm	Field	Irreducible Polynomial
AES	$GF(2^8)$	$x^8+x^4+x^3+x+1$
McEliece Cryptosystem	$GF(2^{13})$	$x^{13}+x^4+x^3+x+1$
WG-16	$GF(2^{16})$	$x^{16}+x^5+x^3+x^2+1$
WG-29	$GF(2^{29})$	$x^{29}+x^2+1$

Binary fields with polynomial basis (PB) are particularly suitable for VLSI implementations due to their simplicity and modularity. In this work, we present an exhaustive study by deriving fault detection schemes based on cyclic codes for finite field multipliers with PB, implementing such proposed schemes for the fields found in Advanced Encryption Standard (AES), WG Welch-Gong-16 (WG-16), WG Welch-Gong-29 (WG-29), and the McEliece cryptosystem, and benchmarking the different overheads on advanced micro devices (AMD)/Xilinx field-programmable gate array (FPGA) family Artix-7 for device xc7a12tcpg238-3.

Our work is outlined as follows. Section II reviews the mathematical properties of PB. In Section III, we discuss the overall finite field multiplier architecture and derive the different fault detection schemes based on cyclic codes for multiplications over GF(2^m) with PB. Next, such fault detection schemes are embedded into the original finite field multipliers in Section IV to benchmark the different overheads. Finally, Section V concludes this brief.

II. Preliminaries

The Galois field of order p^m is a finite field with p^m elements, abbreviated as GF(p^m). One of the most common fields used is the Galois field GF(2^m), because they are particularly efficient for implementation in hardware or on a binary computer. Table I shows different traditional and post-quantum cryptographic algorithms and their corresponding fields that they operate with. GF(2^m) is made up of 2^m binary polynomials, that is, polynomials with coefficients of 0 or 1. Each of those polynomials has a degree of no more than $m-1$; thus, the elements may be expressed as m-bit strings. At the same place in the polynomial, each bit in the bit string corresponds to a coefficient. For instance, GF(2^3) has eight elements: 0, 1, x, $x+1$, x^2, x^2+1, x^2+x, and x^2+x+1, and they might be expressed as 000, 001, 010, ..., 111, respectively.

A finite field element A over GF(2^m) with PB $\{1, x, x^2, \ldots, x^{m-1}\}$ can be expressed as follows:

$$A = \sum_{i=0}^{m-1} a_i x^i, \quad a_i \in \{0,\ 1\} \tag{1}$$

where the values of a_i are the coordinates of the input element A.

To perform the different finite field arithmetic operations, e.g., addition, subtraction, multiplication, inversion, and division,

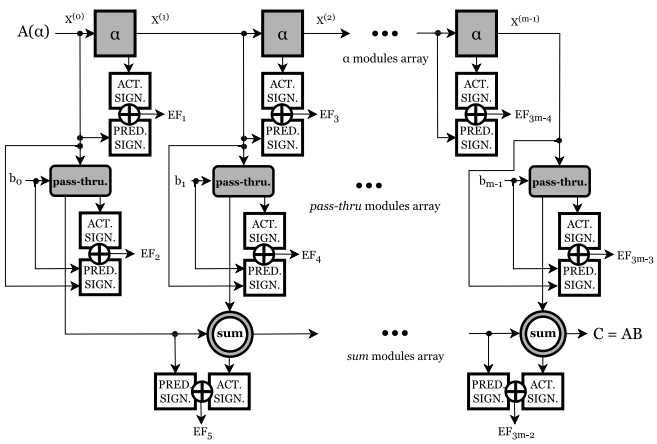

Fig. 1. Finite field multiplier using the proposed error detection blocks based on cyclic codes.

an irreducible polynomial $f(x)$ of degree m over GF(2^m) is needed. This irreducible polynomial limits the number of bits, since it is used to perform the modulo operation. For instance, to multiply two elements A and B over GF(2^m), the output C is computed as follows:

$$C = A \cdot B \bmod f(x).$$

There has been an increasing amount of research on finite field multiplication, since it is used not only to perform the multiplication of two finite field elements, but also to perform finite field squaring, inversion, and division. Multiplication over GF(2^m) is the most time-consuming basic arithmetic operation in many cryptographic algorithms, and its hardware implementation may need thousands of logic gates. Developing large finite field multipliers that always produce error-free outputs is a complex and expensive endeavor.

The multiplication of the elements A and B over GF(2^m) with PB can be further derived as follows:

$$\begin{aligned} C &= \sum_{i=0}^{m-1} b_i \cdot ((Ax^i) \bmod f(x)) \\ &= \sum_{i=0}^{m-1} b_i \cdot X^{(i)} \end{aligned}$$

where the set of b_i values is the B coefficients, $X^{(i)} = \alpha \cdot X^{(i-1)} f(\alpha)$, and $X^{(0)} = A$. To perform such multiplication, we use the architecture from [11], needing three different modules: *Sum*, *pass-thru*, and α modules. The *sum* module is also used to perform finite field addition, using an m-bit XOR gate to add two GF(2^m) elements. The *pass-thru* module multiplies a GF(2^m) element by a GF(2) element. For example, let A (a GF(2^m) element) and b (a GF(2) element) serve as the inputs of the *pass-thru* module, while G (a GF(2^m) element) serves as the output. When $b = 0$, the output of the *pass-thru* module is 0, and when $b = 1$, the output is A. Finally, the α module multiplies an element of GF(2^m) by x, such as

$$A(x) \cdot x = a_{m-1} \cdot x^m + a_{m-2} \cdot x^{m-1} + \cdots + a_0 \cdot x \qquad (2)$$

where

$$x^m \equiv f_{m-1} \cdot x^{m-1} + f_{m-2} \cdot x^{m-2} + \cdots + f_0 \bmod f(x)$$

reducing the result modulo $f(x)$. For example, if we have a GF(2^8) element going through the α module, the input would be $A(x) = a_7 \cdot x^7 + a_6 \cdot x^6 + \cdots + a_0$, which gets multiplied by x in the α module obtaining $A(x) \cdot x = a_7 \cdot x^8 + a_6 \cdot x^7 + \cdots + a_0 \cdot x$, where $x^8 \equiv x^4 + x^3 + x + 1 \bmod f(x)$ if the irreducible polynomial is $f(x) = x^8 + x^4 + x^3 + x + 1$. Therefore, the output of the α module for this specific case would be $A(x) \cdot x = a_6 \cdot x^7 + a_5 \cdot x^6 + a_4 \cdot x^5 + (a_7 + a_3) \cdot x^4 + (a_7 + a_2) \cdot x^3 + a_1 \cdot x^2 + (a_7 + a_0) \cdot x + a_7$. Fig. 1 shows the entire finite field multiplier with the error detection blocks that are derived in Section III. The order of the error flags EF goes from top to bottom (please note that in the first iteration, there is no α module). Therefore, EF_1 is obtained in the first α module, EF_2 is obtained in the first *sum* module, EF_3 is obtained in the second α module, EF_4 is obtained in the second *sum* module, EF_5 is obtained in the first *pass-thru* module, and so on. Since there are a total of $m-1$ α modules, $m-1$ *sum* modules, and m *pass-thru* modules, the maximum number of error flags is $3m-2$. ACT. SIGN. and PRED. SIGN. stand for actual and predicted signatures, respectively, and we will explain them in Section III.

III. Proposed Fault Detection Architectures

This section presents efficient and overhead-aware error detection schemes for the different finite field multipliers found in some traditional and post-quantum cryptographic algorithms. These schemes are based on cyclic codes, one of the most important subclasses of linear codes, since they possess many algebraic properties that simplify the encoding and decoding implementations. In this work, we will use a (7,4) cyclic code, meaning that for every four data/message bits (k), there are three parity bits ($n-k$) associated to form a codeword. This codeword has the following form:

$$\text{codeword} = [\text{message}, \text{parity}].$$

To encode an (n, k) cyclic code into its systematic form, where the k leftmost digits of each code vector are the message and the $n-k$ rightmost digits are the parity-check digits, the message polynomial $u(x)$, with the form of $u(x) = u_{k-1}x^{k-1} + \cdots + u_1 x + u_0$, is first multiplied by x^{n-k}. For example, if your message is $u(x) = x^3 + x^2 + 1$, it becomes $x^6 + x^5 + x^3$ (leaving three free bits to add the parity bits). $u(x) \cdot x^{n-k}$ is then divided by a generator polynomial $g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \cdots + g_2 x^2 + g_1 x + 1$, which, in this work, is $g(x) = x^3 + x + 1$ to obtain a remainder $b(x)$. $b(x)$ corresponds to the parity bits, and they are added to the shifted message $u(x) \cdot x^{n-k}$ to produce a codeword in the form of $u(x) \cdot x^{n-k} + b(x)$. These steps can be accomplished by the architecture shown in Fig. 2. Let us explain next how this architecture works and how it is embedded in the finite field multiplier.

For example, we have the input message $u(x) \cdot x^{n-k}$, $A(x)$ for short. $A(x)$ over GF(2^4) has the form $A(x) = a_3 x^6 + a_2 x^5 + a_1 x^4 + a_0 x^3$ (leaving three empty bits to append the parity bits). The input $A(x)$ enters 1 bit at a time (noting that the initial content of the registers is 0). The first bit that enters is a_3, and as shown in Fig. 2, it goes to registers FF_1 and FF_2, obtaining a_3 for both parity bits P_1 and P_2 and 0 for the parity bit P_3. The next bit that is fed into the circuit is a_2. Since the content of the registers is shifted to the right, FF_1 now stores only a_2, and FF_2 stores $a_3 + a_2$, since the previous value of FF_1 is XORed with the current value of FF_2, and FF_3 now stores a_3, which is the previous value of FF_2. The rest of the message bits are fed into the circuit in the same manner to obtain the parities shown in Table II.

To provide error detection to the finite field multipliers, each of its modules (α, *sum*, and *pass-thru* modules) will need to produce actual parities and predicted parities that will be compared with each other to detect any faults that have been introduced (fault analysis attack) or product of the environment (natural faults). Therefore, the circuit to encode the (7,4) cyclic code needs to be placed twice on each module, as shown in Fig. 1. We note that we refer to the combination

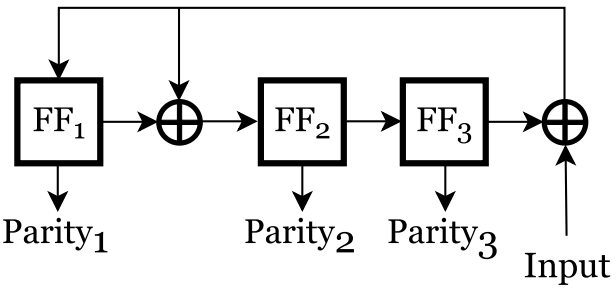

Fig. 2. Architecture embedded in the original finite field multipliers to produce the different signatures.

TABLE II
DERIVATION OF THE ACTUAL SIGNATURES USING THE CIRCUIT FROM FIG. 2

Input	Parity 1	Parity 2	Parity 3
-	0	0	0
a_3	a_3	a_3	0
a_2	a_2	$a_3 + a_2$	a_3
a_1	$a_3 + a_1$	$a_2 + a_1$	$a_3 + a_2$
a_0	$a_3 + a_2 + a_0$	$a_3 + a_1 + a_0$	$a_2 + a_1$

of *Parity 1*, *Parity 2*, and *Parity 3* as signatures. The actual signatures of each module correspond to the combination of parities shown in the last row of Table II. However, the predicted signatures for the α module are calculated differently, using different inputs, since the α module multiplies $A(x)$ with x, as shown in (2). For example, if we have a GF(2^4) element going through the α module, the input would be $A(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$, which gets multiplied by x to obtain $A(x) \cdot x = a_3 \cdot x^4 + a_2 \cdot x^3 + a_1 \cdot x^2 + a_0 \cdot x$, where $x^4 \equiv x^2 + 1 \mod f(x)$ if the irreducible polynomial is $f(x) = x^4 + x^2 + 1$. Thus, the output of the α module is $A(x) \cdot x = a_2 x^3 + (a_3 + a_1)x^2 + a_0 x + a_3$. This means that the first bit entering the circuit from Fig. 2 is a_2, then $(a_3 + a_1)$, and so on. Table III shows the derivation of predicted signatures for the α module, and as it can be seen, the inputs are different, since the coefficients of $A(x) \cdot x$ are different from $A(x)$. The predicted signatures are then XORed with the actual signatures ($\text{EF}_1 = \text{ActualParity}_1 \oplus \text{PredictedParity}_1$, $\text{EF}_2 = \text{ActualParity}_2 \oplus \text{PredictedParity}_2$, and $\text{EF}_3 = \text{ActualParity}_3 \oplus \text{PredictedParity}_3$), as shown in Fig. 1, to detect if any fault has occurred. For example, if an error flag EF signals a "1," the system has detected a fault in that particular module.

On the other hand, the predicted and actual signatures for the *sum* and the *pass-thru* modules are easier to derive than those for the α module, since the order of the coefficients does not change. For the *sum* module, which adds elements $A(x)$ and $B(x)$ over GF(2^m), both signatures are similar as those from Table II, but since $B(x)$ is added, its coefficients would be added as well, obtaining $p_1 = a_3 + b_3 + a_2 + b_2$, $p_2 = a_2 + b_2 + a_1 + b_1 + a_0 + b_0$, and $p_3 = a_3 + b_3 + a_2 + b_2 + a_1 + b_1$. Finally, for the *pass-thru* module, which adds element $A(x)$ over GF(2^m) with a GF(2) element b, both parities are also similar as those from Table II, but since b is multiplied, the parities from Table II are multiplied as well with b to obtain $p_1 = b \cdot (a_3 + a_2)$, $p_2 = b \cdot (a_2 + a_1 + a_0)$, and $p_3 = b \cdot (a_3 + a_2 + a_1)$.

We note that the example shown is for GF(2^4), whose inputs have precisely 4 bits. However, if one is working with more extensive fields, the input can be split into blocks of 4 bits, and if m is not divisible by 4, e.g., GF(2^{13}) and GF(2^{29}), 0's can be appended to fill all the blocks of 4 bits. In this work, we have used a (7,4) cyclic code for simplicity of the examples, to produce more error flags per multiplication, and to not append many 0's at the end of each last block, e.g., if the field is GF(2^{12}), the system does not need to append "0" using a (7,4) cyclic code, but it needs to append "0000" to each last block if using a (12,8) cyclic code. In addition, in the previous example, using a (7,4) cyclic code obtains nine parity bits per operation, while the (12,8) obtains eight. Employing smaller cyclic codes provides more parity bits; however, they can use more computational resources than bigger cyclic codes, especially for larger fields. The choice of the utilized cyclic codes can be tailored based on the reliability requirements and the overhead to be tolerated.

TABLE III
DERIVATION OF THE PREDICTED SIGNATURES USING THE CIRCUIT FROM FIG. 2

Input	Parity 1	Parity 2	Parity 3
-	0	0	0
a_2	a_2	a_2	0
$a_3 + a_1$	$a_3 + a_1$	$a_3 + a_2 + a_1$	a_2
a_0	$a_2 + a_0$	$a_3 + a_1 + a_0$	$a_3 + a_2 + a_1$
a_3	$a_2 + a_1$	$a_3 + a_2 + a_0$	$a_3 + a_1 + a_0$

TABLE IV
CALCULATION OF THE ERROR COVERAGE PERCENTAGE FOR EACH MULTIPLIER

Multiplier	Number of Signatures ($sign.$)	Error coverage %
$GF(2^8)$	$7_\alpha + 7_{sum} + 8_{pass}$	$100 \cdot (1 - (\frac{1}{2})^{22})$
$GF(2^{13})$	$12_\alpha + 12_{sum} + 13_{pass}$	$100 \cdot (1 - (\frac{1}{2})^{37})$
$GF(2^{16})$	$15_\alpha + 15_{sum} + 16_{pass}$	$100 \cdot (1 - (\frac{1}{2})^{46})$
$GF(2^{29})$	$28_\alpha + 28_{sum} + 29_{pass}$	$100 \cdot (1 - (\frac{1}{2})^{85})$
General	$m - 1_\alpha + m - 1_{sum} + m_{pass}$	$100 \cdot (1 - (\frac{1}{2})^{sign.})$

IV. ERROR COVERAGE AND FPGA IMPLEMENTATIONS

As mentioned earlier, the proposed error detection schemes are intended to detect natural faults product of the environment and intentional faults, e.g., fault analysis attacks. Fault analysis attacks are active attacks where the adversary seeks to interfere with the cryptographic process handling sensitive data, causing inaccurate outputs, which can reveal sensitive data. Bit-fault injection at the desired place and at the desired cycle would be the perfect attack to carry out in order to gather a wealth of data. However, this is generally not practical, as costly equipment is required due to the shrinking geometry size of integrated chips. Our fault model covers single as well as multiple stuck-at faults (stuck-at 0 and stuck-at 1), since technological limitations may make it difficult for an attacker to flip precisely 1 bit. In this work, we assume that the comparators are hardened, i.e., the comparators are fault free and not compromised, and that the inputs are not compromised prior to the execution of the multiplier units.

The total number of signatures needs to be determined to calculate the error coverage provided by the various fault detection strategies proposed in this work. The formula used to calculate the error coverage is $100 \cdot (1 - (1/2)^s)\%$, where s is the total amount of signatures. The percentage of not detecting errors using the signatures is independent, and thus, this is the percent of detecting errors for randomly distributed faults. Each multiplier needs a total of $m - 1$ α modules, $m - 1$ *sum* modules, and m *pass-thru* modules, and each of them has a signature (as mentioned earlier, we note that the term signature through this brief refers to the combination of the parities, and one actual signature with one predicted signature is equivalent to just one signature in the previous formula). Therefore, each cryptosystem has a different number of signatures and a different error detection coverage, as it is shown in Table IV.

Normal parity and multibit parity schemes are usually fast and do not require too many extra resources to implement. However, normal parity has only an error coverage of up to 50% (it can only detect an

TABLE V
OVERHEADS OBTAINED WHEN IMPLEMENTING THE PROPOSED ERROR DETECTION SCHEMES ON TOP OF THE ORIGINAL ARCHITECTURES USING AMD/XILINX FPGA FAMILY ARTIX-7 DEVICE XC7A12TCPG238-3

Architecture	Area (occupied slices)	Delay (ns)	Power (mW) @50 MHz	Throughput (Gbps)
$GF(2^8)$ Multiplier	35	2.825	0.064	2.83
$GF(2^8)$ Multiplier with Error Detection	35 (~0%)	3.245 (14.87%)	0.064 (~0%)	2.47 (-12.72%)
$GF(2^{13})$ Multiplier	91	3.669	0.067	3.54
$GF(2^{13})$ Multiplier with Error Detection	97 (6.59%)	3.763 (2.56%)	0.067 (~0%)	3.45 (-2.54%)
$GF(2^{16})$ Multiplier	131	3.561	0.069	4.49
$GF(2^{16})$ Multiplier with Error Detection	137 (4.58%)	3.934 (10.47%)	0.069 (~0%)	4.07 (-9.35%)
$GF(2^{29})$ Multiplier	402	5.088	0.078	5.70
$GF(2^{29})$ Multiplier with Error Detection	529 (31.59%)	6.412 (26.02%)	0.079 (1.28%)	4.52 (-20.70%)

odd number of faults), and multibit parity is vulnerable to intelligent fault injection. While normal parity only produces a single parity bit, multibit parity schemes group several bits obtaining multiple parity bits. For example, if your system has GF(2^{10}) elements, you could have five 2-bit groups, giving five error flags or parity bits. However, faults usually occur in clumps, since it is very costly to inject a single fault, and multiparity bit schemes would not detect some clumps of injected faults, e.g., four consecutive faults. Cyclic codes outperform normal and multibit parities by increasing the complexity of the arithmetic operations employed. As it can be observed in Table III, the only combination of injected faults that the system would not detect faults in the α module is when they are injected at a_3 and a_1, since both are located in parity bits 2 and 3 but not in parity bit 1. Note that producing two single faults in the same cycle is extremely challenging and costly. Using the (7,4) cyclic code presented in this work, there are four different places where single fault can be injected, six different combinations where two faults can be injected, four different combinations where triple faults can be injected, and only one combination where four faults can be injected, which makes a total of 15 combinations. Out of those 15 cases, our schemes would detect faults in 14 of them, or in other words, in 93.33% of the cases. For the *sum* and *pass-thru* modules, the error coverage is very similar. For the *sum* module, since the coefficients of the GF(2^m) element $B(x)$ are added to the parities, the only case where the scheme does not detect faults is if there is a double fault at b_3 and b_1 or at a_3 and a_1 (253 out of 255 cases would be covered: 99.2% error coverage). Finally, for the *pass-thru* module, since b is a GF(2) element and gets multiplied on each parity bit, the only case where the scheme would not detect faults is if there is a double fault at a_3 and a_1 (30 out of 31 cases would be covered: 96.77% error coverage).

We performed a total of 2^{16} fault simulations on FPGA for GF(2^8) multipliers using normal parity schemes and cyclic codes. Of those simulations, 2^{14} were with single fault injected, 2^{14} were with double faults injected, 2^{14} were with three faults injected, and 2^{14} were with four faults injected. The GF(2^8) multiplier with normal parity had an error coverage percentage of 50%, since it detected all odd amounts of faults but not any even amounts of faults. On the other hand, the GF(2^8) multiplier with a (7,4) cyclic code detected >99.9% of the injected faults. More information regarding the implementation results of these two schemes can be found at the end of this section.

In addition, we have implemented the proposed error detection schemes on the AMD/Xilinx FPGA family Artix-7 device xc7a12tcpg238-3 to benchmark the different overheads obtained when cyclic codes are applied to the original architectures. We use the Vivado tool and Verilog as the hardware design entry to analyze the area (occupied slices), delay (ns), power (mW) with the clock frequency of 50 MHz, and throughput (Gb/s). The different overheads obtained are shown in Table V, and as one can see, the schemes added a worst case area overhead of 31.59% and a worst case delay overhead of 26.02% when they were added to the WG-29. Regarding the power, the overheads obtained when we apply the error detections schemes are negligible due to the size of the designs, obtaining a worst case power overhead of 1.28% when cyclic codes were applied to the original GF(2^{29}) multiplier.

To the best of our knowledge, there has not been any prior work done on error detection based on general cyclic codes for finite field multipliers with PB elements. Let us review several case studies on error detection in GF(2^m) arithmetic hardware for qualitative comparison to ensure that the overheads incurred are reasonable. Fault detection techniques for the key generator of code-based post-quantum cryptosystems on FPGA are implemented in [12], obtaining between ≈5% (best case) and ≈49% (worst case) area overheads. In [10], fault detection capability on binary extension fields based on parity is proposed for elliptic curve cryptography (ECC), obtaining a little over 10% area overhead. In [13], cyclic redundancy checks (CRC), which are a type of cyclic codes, are proposed for the inverse of an element with PB in GF(2^{163}), yielding an area overhead of around 26%. In addition, CRC schemes are implemented on the key generator of the Niederreiter cryptosystem in [14], obtaining the efficiency degradations of at most 7.72%. For the sake of comparison, we have also compared our schemes with a GF(2^8) finite field multiplier using error detection based on normal parity. The latter obtained worse results in terms of area, with an overhead of 2.86% (36 occupied slices), but a lower delay overhead of 13.55% (3.208 ns). It is also worth mentioning that normal parity error detection does not detect an even amount of faults. These and other comparable research on error detection demonstrate the acceptability of the overheads obtained in this work.

V. CONCLUSION

This brief presents error detection techniques for finite field multipliers with PB elements based on cyclic codes. Moreover, we have also added the suggested fault detection schemes to the original finite field multipliers of algorithms, such as AES, WG, and the McEliece cryptosystem in the AMD/Xilinx FPGA family Artix-7 device xc7a12tcpg238-3. In addition, we calculate the different overheads added by such schemes and compare them with related works to prove that the overheads from the proposed schemes are acceptable. The schemes added a worst case area overhead of 31.59% and a worst case delay overhead of 26.02% when the schemes were added to the WG-29 cryptosystem, which is suitable due to the high error coverage achieved of close to 100%.

REFERENCES

[1] A. C. 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.

[2] D. Heinz and T. Poppelmann, "Combined fault and DPA protection for lattice-based cryptography," *IEEE Trans. Comput.*, early access, Aug. 8, 2022, doi: 10.1109/TC.2022.3197073.

[3] V. Arribas, F. Wegener, A. Moradi, and S. Nikova, "Cryptographic fault diagnosis using VerFI," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, Dec. 2020, pp. 229–240.

[4] J. Kaur, M. Mozaffari-Kermani, and R. Azarderakhsh, "Hardware constructions for error detection in lightweight authenticated cipher ASCON benchmarked on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 4, pp. 2276–2280, Apr. 2022.

[5] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Reliable hardware architectures for the third-round SHA-3 finalist grostl benchmarked on FPGA platform," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, Oct. 2011, pp. 325–331.

[6] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A high-performance fault diagnosis approach for the AES SubBytes utilizing mixed bases," in *Proc. Workshop Fault Diagnosis Tolerance Cryptography*, Sep. 2011, pp. 80–87.

[7] C.-Y. Lee, P. K. Meher, and J. C. Patra, "Concurrent error detection in bit-serial normal basis multiplication over GF(2^m) using multiple parity prediction schemes," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 8, pp. 1234–1238, Aug. 2010.

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

[9] A. Reyhani-Masoleh and M. A. Hasan, "Fault detection architectures for field multiplication using polynomial bases," *IEEE Trans. Comput.*, vol. 55, no. 9, pp. 1089–1103, Sep. 2006.

[10] C. Fei, F. Zhou, N. Wu, F. Ge, J. Wen, and P. Qin, "A scalable bit-parallel word-serial multiplier with fault detection on GF(2^m)," in *Proc. IEEE 20th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2020, pp. 1660–1664.

[11] A. Reyhani-Masoleh and M. A. Hasan, "Error detection in polynomial basis multipliers over binary extension fields," in *Proc. CHES*, 2002, pp. 515–528.

[12] A. C. Canto, M. M. Kermani, and R. Azarderakhsh, "Reliable constructions for the key generator of code-based post-quantum cryptosystems on FPGA," *ACM J. Emerg. Technol. Comput. Syst.*, Jun. 2022, pp. 1–10.

[13] A. Cintas-Canto, M. Mozaffari-Kermani, and R. Azarderakhsh, "CRC-based error detection constructions for FLT and ITA finite field inversions over $GF(2^m)$," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 5, pp. 1033–1037, May 2021.

[14] A. Cintas-Canto, M. Mozaffari-Kermani, R. Azarderakhsh, and K. Gaj, "CRC-oriented error detection architectures of post-quantum cryptography niederreiter key generator on FPGA," in *Proc. IEEE Nordic Circuits Syst. Conf. (NorCAS)*, Oct. 2022, pp. 1–7.