Transactions Briefs

CRC-Based Error Detection Constructions for FLT and ITA Finite Field Inversions Over GF(2^m)

Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh

***Abstract*—Binary extension finite fields GF(2^m) have received prominent attention in the literature due to their application in many modern public-key cryptosystems and error-correcting codes. In particular, the inversion over GF(2^m) is crucial for current and postquantum cryptographic applications. Schemes such as Fermat's little theorem (FLT) and the Itoh–Tsujii algorithm (ITA) have been studied to achieve better performance; however, this arithmetic operation is a complex, expensive, and time-consuming task that may require thousands of gates, increasing its vulnerability chance to natural defects. In this work, we propose efficient hardware architectures based on cyclic redundancy check (CRC) as error detection schemes for state-of-the-art finite field inversion over GF(2^m) for a polynomial basis. To verify the derivations of the formulations, software implementations are performed. Likewise, hardware implementations of the original finite field inversions with the proposed error detection schemes are performed over Xilinx field-programmable gate array (FPGA) verifying that the proposed schemes achieve high error coverage with acceptable overhead.**

***Index Terms*—Cyclic redundancy check (CRC), fault detection, field-programmable gate array (FPGA), finite field inversion.**

I. Introduction

Finite fields and their hardware implementations have received prominent attention due to their extensive use in cryptography and error-correcting codes. The computation of inversion is the most time-consuming one. To perform inversion over GF(2^m), many solutions have been proposed to increase the performance of such operation for polynomial basis field element representation. Fermat's little theorem (FLT) and the Itoh–Tsujii algorithm (ITA) are two main approaches to compute finite field inversion over GF(2^m). The latter approach was first intended to be applied over binary extension fields with a normal basis [1]; however, in more recent studies, it has been shown that it can be used for other field element representations [2], [3]. These approaches extensively use multiplication and squaring, requiring thousands of gates. Constructions using such large designs are vulnerable to side-channel attacks, where an adversary exploits system leakages such as power consumption, electromagnetic emissions, or acoustic emanations to reveal secret information to an adversary [4]. It is a complex task to implement such architectures resilient, not only do these structures need low overhead but the error coverage needs to be acceptable. Deteriorated performance can lead to catastrophic results for sensitive applications; accordingly, research has focused on ways to eliminate errors and achieve greater reliability with reasonable overhead [5]–[7].

In this brief, error detection schemes are proposed for both FLT and ITA algorithms, which are used to perform finite field inversion over GF(2^m) with polynomial basis field element representation. These error detection schemes are based on cyclic redundancy check (CRC), providing high error coverage. Even though we work with specific values of m, the proposed schemes for error detection are applicable to different applications no matter how large m is. For verification purposes, software implementations are performed to derive the formulations. Additionally, we benchmark the overhead of the proposed architectures by implementing the original finite field inversion block with our schemes on field-programmable gate array (FPGA), evaluating the error coverage obtained by integrating our error detection schemes into the original architectures. Although the proposed architectures are implemented on FPGA, similar results are expected on application-specific integrated circuit (ASIC) platforms. Moreover, we expect similar results for different FPGA families and ASIC libraries.

Our work is structured as follows. Preliminaries are discussed in Section II, where we introduce both FLT and ITA algorithms. Section III presents finite filed inversion over GF(2^m) for polynomial basis field element representation, CRC schemes, and the derivation of our proposed error detection schemes. In Section IV, we implement the proposed architectures to show the overhead of the derived signatures used with the original architectures. In this section, by implementing our proposed architectures on FPGA, we benchmark our presented work. Finally, our summary of findings is presented in Section V.

II. Preliminaries

In this work, we consider finite fields GF(2^m) with $m > 1$. The elements that the finite field inversion blocks use as inputs can be represented as $A = \sum_{i=0}^{m-1} a_i x^i$, $a_i \in \{0, 1\}$, where a_i's are the coordinates of the input. Each finite field inversion uses an irreducible polynomial or field polynomial denoted by $f(x)$. The inverse of an element $A \neq 0$ in the field GF(2^m) is expressed as $A^{-1} \epsilon \, \mathrm{GF}(2^m)$ where $A \cdot A^{-1} = 1$. To find the inverse of an element A, both FLT and ITA schemes are studied in this work.

FLT specifies that the inverse of an element A can be derived as follows: $A^{2^m-1} \equiv 1 \bmod f(x) \Longrightarrow A^{2^m-2} \cdot A \equiv 1 \bmod f(x) \Longrightarrow A^{2^m-2} \cdot A \cdot A^{-1} \equiv A^{-1} \bmod f(x) \Longrightarrow A^{2^m-2} \equiv A^{-1} \bmod f(x)$. For hardware implementations, this theorem leads to a total of $2^m - 2$ finite field multiplications, and it may require additional memory to store the precomputed values. Approaches to reduce the complexity of inversions have been studied, e.g., square-and-multiply algorithm [8], Kaliski inversion [9], and ITA algorithm [1]–[3].

The method introduced by Itoh and Tsujii achieves less amount of multiplications by an efficient use of addition chains. The inverse can be rewritten as $A^{-1} = [\beta_{m-1}(A)]^2$, where $\beta_k(A) = A^{2^k-1} \in \mathrm{GF}(2^m)$ and $k \in \mathbb{N}$. To calculate an addition chain $C = \{c_1, c_2, \ldots, c_t\}$ with a field polynomial $f(x)$ of m degree, we have $c_1 = 1$ and $c_t = m - 1$. If c_i is even, $c_{i-1} = c_i/2$ and if c_i is odd, $c_{i-1} = c_i - 1$. Moreover, the Multiplicative Inversion Addition-Chain ITA is shown in Algorithm 1.

III. Proposed Fault Detection Architectures

Practical and low-complexity error detection approaches are required to provide subblocks of inversion constructions with accept-

Manuscript received October 18, 2020; revised January 7, 2021 and February 1, 2021; accepted February 21, 2021. Date of publication March 10, 2021; date of current version April 28, 2021. This work was supported by the U.S. National Science Foundation (NSF) under Award SaTC-1801488. *(Corresponding author: Mehran Mozaffari Kermani.)*

Alvaro Cintas Canto and Mehran Mozaffari Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: alvarocintas@usf.edu; 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.2021.3061987

Algorithm 1 Multiplicative Inversion Addition-Chain ITA

```
$\beta_0 = A(x)$
for $i$ from 1 to $t$ do
    $\beta_i = [\beta_{i_1}]^{2^{e_{i_2}}} \cdot \beta_{i_2} \pmod{f(x)}$
return $((\beta_t)^2 \pmod{f(x)})$
```

able remedies against faults. The main operations to perform finite field inversion of an element A in the field GF(2^m) are GF(2^m) multiplication, GF(2^m) squaring, and GF(2^m) addition. To perform GF(2^m) multiplication, three modules are used: *sum*, α, and *pass-thru* modules, which are described in [10]. The *sum* module adds two elements in GF(2^m); the α module multiplies an element of GF(2^m) by α and it reduces the result modulo $f(x)$; and the *pass-thru* module multiplies a GF(2^m) element by a GF(2) element.

To perform GF(2^m) addition, the *sum* module is used, where two elements in GF(2^m) are added by utilizing m XOR gates. For finite field squaring, we utilize an architecture that only uses two modules, i.e., the α^2 and the *sum* modules. In α^2 module, an element A is multiplied by α^2 to achieve

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

where $x^{m+1} \equiv f_{m-1} \cdot x^m + f_{m-2} \cdot x^{m-1} + \cdots + f_0 \cdot x \bmod f(x)$ and $x^m \equiv f_{m-1} \cdot x^{m-1} + f_{m-2} \cdot x^{m-2} + \cdots + f_0 \bmod f(x)$. Moreover, the output X coordinates are expressed as

$$x_i = \begin{cases} a_{m-1} \cdot f_{i-1} + (a_{m-1} f_{m-1} + a_{m-2}) \cdot f_i + a_{i-2} \\ \qquad 2 \leq i \leq m-1 \\ a_{m-1} + (a_{m-1} \cdot f_{m-1} + a_{m-2}) \cdot f_1, \quad i = 1 \\ a_{m-1} \cdot f_{m-1} + a_{m-2}, \quad i = 0. \end{cases}$$

In this work, we propose efficient fault detection schemes based on CRC. CRC is based on the theory of cyclic error-correcting codes. A generator polynomial $g(\alpha)$ is required to implement CRC, which becomes the divisor in a long division of polynomials. The message becomes the dividend, the quotient is discarded, and the result is generated by the remainder. A fixed number of check bits is appended to the data and these check bits are checked when the output is obtained to detect any errors. The selection of the CRC scheme in our proposed constructions has been done to leverage overhead-aware architectures, providing a high error detection coverage. In this work, the National Institute of Standards and Technology (NIST) field GF(2^{163}) [11] is used with CRC-10; however, the proposed fault detection schemes are applicable to any field size and CRC signature. Additionally, the field polynomial used is $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. The choice of the utilized CRC relies on the constraints of each system. In other words, for applications such as game consoles in which performance is critical (and power consumption is not because these are plugged in), one can increase the size of CRC. However, for deeply embedded systems such as implantable and wearable medical devices, smaller CRC is preferred.

CRC signatures in the *sum* and *pass-thru* modules do not require as much derivations as the ones needed for α and α^2 modules. For the *sum* module, the predicted CRC-1 signature $\hat{p}_X$ is equal to the sum of the parity bits of the input elements A and B in GF(2^m), $\hat{p}_X = p_A + p_B$. Moreover, for the *pass-thru* module, $\hat{p}_X = b \cdot p_A$, where b is an element in GF(2). For any CRC-X scheme, instead of summing all the parity bits (which is done in CRC-1), it checks X bits at a time in the *sum* and *pass-thru* modules. In the following, the NIST field GF(2^{163}) is used with CRC-10; however, the proposed fault detection schemes are applicable to any field size and CRC signature.

A. α Module: Case Study for CRC-10

For $m = 163$ with CRC-10, the generator polynomial used is $g(x) = x^{10} + x^9 + x^5 + x^4 + x + 1$. To find its signatures, $g(x)$ is used as follows:

$$\begin{aligned} x^{10} &\equiv x^9 + x^5 + x^4 + x + 1 \bmod g(x) \\ x^{11} &\equiv x^9 + x^6 + x^4 + x^2 + 1 \bmod g(x) \\ x^{12} &\equiv x^9 + x^7 + x^4 + x^3 + 1 \bmod g(x) \\ &\vdots \\ x^{160} &\equiv x^9 + x^8 + x^7 + x^6 + x^5 + x + 1 \bmod g(x) \\ x^{161} &\equiv x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1 \bmod g(x) \\ x^{162} &\equiv x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x \bmod g(x). \end{aligned}$$

In the α module, the multiplication of any element in GF(2^{163}) by x gives

$$A(x) \cdot x = a_{162} \cdot x^{163} + a_{161} \cdot x^{162} + \cdots + a_1 \cdot x^2 + a_0 \cdot x$$

where $x^{163} = f_{162}x^{162} + f_{161}x^{161} + \cdots + f_1 x + f_0 \bmod f(x)$.

The irreducible polynomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ is applied to obtain

$$\begin{aligned} A(x) \cdot x \equiv\ & a_{162}x^7 + a_{162}x^6 + a_{162}x^3 + a_{162} + a_{161}x^{162} \\ & + a_{160}x^{161} + a_{159}x^{160} + \cdots + a_3x^4 + a_2x^3 + a_1x^2 \\ & + a_0x \bmod f(x). \end{aligned}$$

Then, the generator polynomial $g(x)$ is applied to calculate the predicted CRC-10 for GF(2^{163}) in the α module (PCRC10_{163})

$$\begin{aligned} A(x) \cdot x \equiv\ & a_{162}(x^7 + x^6 + x^3 + 1) + a_{161} \\ & \times (x^9 + x^8 + x^7 + x^6 + x^5 + x^3 + x) + a_{160} \\ & \times (x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1) \\ & + a_{159}(x^9 + x^8 + x^7 + x^6 + x^5 + x + 1) \\ & + \cdots + a_3x^4 + a_2x^3 + a_1x^2 + a_0x \bmod g(x) \end{aligned}$$

or denoting XOR operations with sum symbol for brevity

$$\begin{aligned} &\text{PCRC}10_{163} \\ &= \Bigg(a_{161} + \sum_{i=158}^{159} a_i + \sum_{i=153}^{154} a_i + \sum_{i=148}^{151} a_i \\ &\quad + \sum_{i=143}^{145} a_i + \sum_{i=138}^{141} a_i + \sum_{i=129}^{136} a_i + \sum_{i=125}^{127} a_i + \sum_{i=121}^{123} a_i \\ &\quad + \sum_{i=117}^{118} a_i + \sum_{i=113}^{115} a_i + a_{109} + a_{107} + a_{105} + a_{102} \\ &\quad + \sum_{i=97}^{99} a_i + \sum_{i=91}^{93} a_i + \sum_{i=88}^{89} a_i + a_{86} + a_{84} + \sum_{i=80}^{82} a_i \\ &\quad + a_{77} + \sum_{i=73}^{74} a_i + \sum_{i=66}^{67} a_i + \sum_{i=59}^{61} a_i + a_{57} + a_{54} + \sum_{i=49}^{50} a_i \\ &\quad + a_{47} + \sum_{i=44}^{45} a_i + \sum_{i=38}^{42} a_i + \sum_{i=35}^{36} a_i + \sum_{i=31}^{32} a_i + a_{27} \\ &\quad + \sum_{i=23}^{25} a_i + a_{17} + \sum_{i=8}^{12} a_i \Bigg) x^9 \\ &\quad + \cdots + \Bigg(a_{162} + \sum_{i=159}^{160} a_i + \sum_{i=154}^{155} a_i + \sum_{i=149}^{152} a_i + \sum_{i=144}^{146} a_i \end{aligned}$$

$$
\begin{aligned}
&+\sum_{i=139}^{142} a_i+\sum_{i=130}^{137} a_i+\sum_{i=126}^{128} a_i+\sum_{i=122}^{124} a_i\\
&+\sum_{i=118}^{119} a_i+\sum_{i=114}^{116} a_i+a_{110}+a_{108}+a_{106}+a_{103}\\
&+\sum_{i=98}^{100} a_i+\sum_{i=92}^{94} a_i+\sum_{i=89}^{90} a_i+a_{87}+a_{85}\\
&+\sum_{i=81}^{83} a_i+a_{78}+\sum_{i=74}^{75} a_i+\sum_{i=67}^{68} a_i+\sum_{i=60}^{62} a_i+a_{58}\\
&+a_{55}+\sum_{i=50}^{51} a_i+a_{48}+\sum_{i=45}^{46} a_i+\sum_{i=39}^{43} a_i+\sum_{i=36}^{37} a_i\\
&+\left.\sum_{i=32}^{33} a_i+a_{28}+\sum_{i=24}^{26} a_i+a_{18}+\sum_{i=9}^{13} a_i\right).
\end{aligned}
$$

We rename the coefficients to calculate the actual CRC-10 for GF(2^{163}) in the α module (ACRC10_{163}): a_{161} as $\gamma_{162}, \ldots, a_0$ as γ_1

$$
\begin{aligned}
A(x)\cdot x \equiv\ &\gamma_{162}x^{162}+\gamma_{161}x^{161}+\gamma_{160}x^{160}\\
&+\cdots+\gamma_4x^4+\gamma_3x^3+\gamma_2x^2+\gamma_1x^1+\gamma_0 \bmod g(x)
\end{aligned}
$$

and the generator polynomial is applied as follows:

$$
\begin{aligned}
A(x)\cdot x \equiv\ &\gamma_{162}(x^9+x^8+x^7+x^6+x^5+x^3+x)\\
&+\gamma_{161}(x^8+x^7+x^6+x^5+x^4+x^2+1)+\gamma_{160}\\
&\times(x^9+x^8+x^7+x^6+x^5+x+1)+\cdots+\gamma_4x^4\\
&+\gamma_3x^3+\gamma_2x^2+\gamma_1x^1+\gamma_0 \bmod g(x)
\end{aligned}
$$

or

$$
\begin{aligned}
&\mathrm{ACRC10}_{163}\\
&=\left(\gamma_{162}+\sum_{j=159}^{160}\gamma_j+\sum_{j=154}^{155}\gamma_j+\sum_{j=149}^{152}\gamma_j\right.\\
&\quad+\sum_{j=144}^{146}\gamma_j+\sum_{j=139}^{142}\gamma_j+\sum_{j=130}^{137}\gamma_j+\sum_{j=126}^{128}\gamma_j+\sum_{j=122}^{124}\gamma_j\\
&\quad+\sum_{j=118}^{119}\gamma_j+\sum_{j=114}^{116}\gamma_j+\gamma_{110}+\gamma_{108}+\gamma_{106}+\gamma_{103}\\
&\quad+\sum_{j=98}^{100}\gamma_j+\sum_{j=92}^{94}\gamma_j+\sum_{j=89}^{90}\gamma_j+\gamma_{87}+\gamma_{85}+\sum_{j=81}^{83}\gamma_j\\
&\quad+\gamma_{78}+\sum_{j=74}^{75}\gamma_j+\sum_{j=67}^{68}\gamma_j+\sum_{j=60}^{62}\gamma_j+\gamma_{58}+\gamma_{55}\\
&\quad+\sum_{j=150}^{151}\gamma_j+\gamma_{48}+\sum_{j=45}^{46}\gamma_j+\sum_{j=39}^{43}\gamma_j+\sum_{j=36}^{37}\gamma_j\\
&\quad+\left.\sum_{j=32}^{33}\gamma_j+\gamma_{28}+\sum_{j=24}^{26}\gamma_j+\gamma_{18}+\sum_{j=9}^{13}\gamma_j\right)x^9\\
&+\cdots+\left(\sum_{j=160}^{161}\gamma_j+\sum_{j=155}^{156}\gamma_j+\sum_{j=150}^{153}\gamma_j+\sum_{j=145}^{147}\gamma_j+\sum_{j=140}^{143}\gamma_j\right.\\
&\qquad+\sum_{j=131}^{138}\gamma_j+\sum_{j=127}^{129}\gamma_j+\sum_{j=123}^{125}\gamma_j+\sum_{j=119}^{120}\gamma_j
\end{aligned}
$$

$$
\begin{aligned}
&+\sum_{j=115}^{117}\gamma_j+\gamma_{111}+\gamma_{109}+\gamma_{107}+\gamma_{104}+\sum_{j=99}^{101}\gamma_j\\
&+\sum_{j=93}^{95}\gamma_j+\sum_{j=90}^{91}\gamma_j+\gamma_{88}+\gamma_{86}+\sum_{j=82}^{84}\gamma_j+\gamma_{79}\\
&+\sum_{j=75}^{76}\gamma_j+\sum_{j=68}^{69}\gamma_j+\sum_{j=61}^{63}\gamma_j+\gamma_{59}+\gamma_{56}\\
&+\sum_{j=51}^{52}\gamma_j+\gamma_{49}+\sum_{j=46}^{47}\gamma_j+\sum_{j=40}^{44}\gamma_j+\sum_{j=37}^{38}\gamma_j\\
&+\left.\sum_{j=33}^{34}\gamma_j+\gamma_{29}+\sum_{j=25}^{27}\gamma_j+\gamma_{19}+\sum_{j=10}^{14}\gamma_j+\gamma_0\right).
\end{aligned}
$$

The entire finite field multiplier with our error detection schemes is presented in Fig. 1, where Actual Cyclic Redundancy Check (ACRC) and Predicted CRC (PCRC) stand for actual CRC signatures and predicted CRC signatures, respectively. In Fig. 1, only one error flag (EF) is shown for clarity; however, for CRC-10, which is the case study proposed in this brief, ten EFs are computed on each module. Next, for α^2 module, the NIST field GF(2^{163}) is used with CRC-10; however, the proposed fault detection schemes are applicable to any field size and CRC signature.

B. α^2 Module: Case Study for CRC-10

In the α^2 module, the multiplication of any element in GF(2^{163}) by x gives

$$
A(x)\cdot x^2 = a_{162}\cdot x^{164}+a_{161}\cdot x^{163}+\cdots+a_1\cdot x^3+a_0\cdot x^2
$$

where $x^{164} = f_{162}x^{163}+f_{161}x^{162}+\cdots+f_1x^2+f_0x \bmod f(x)$ and $x^{163} = f_{162}x^{162}+f_{161}x^{161}+\cdots+f_1x+f_0 \bmod f(x)$.

The irreducible polynomial $f(x) = x^{163}+x^7+x^6+x^3+1$ is applied to obtain

$$
\begin{aligned}
A(x)\cdot x^2 \equiv\ &a_{162}x^8+a_{162}x^7+a_{162}x^4+a_{162}x+a_{161}x^7\\
&\times a_{161}x^6+a_{161}x^3+a_{161}+a_{160}x^{162}+a_{159}x^{161}\\
&+\cdots+a_3x^5+a_2x^4+a_1x^3+a_1x^2 \bmod f(x).
\end{aligned}
$$

Then, the generator polynomial $g(x)$ is applied to calculate the predicted CRC-10 for GF(2^{163}) in the α^2 module (PCRC10_{163})

$$
\begin{aligned}
A(x)\cdot x^2 \equiv\ &a_{162}(x^8+x^7+x^4+x)+a_{161}(x^7+x^6+x^3+1)\\
&+a_{160}(x^9+x^8+x^7+x^6+x^5+x^3+x)\\
&+a_{159}(x^8+x^7+x^6+x^5+x^4+x^2+1)\\
&+\cdots+a_3x^5+a_2x^4+a_1x^3+a_0x^2 \bmod g(x)
\end{aligned}
$$

or

$$
\begin{aligned}
&\mathrm{PCRC10}_{163}\\
&=\left(a_{160}+\sum_{i=157}^{158} a_i+\sum_{i=152}^{153} a_i+\sum_{i=147}^{150} a_i\right.\\
&\quad+\sum_{i=142}^{144} a_i+\sum_{i=137}^{140} a_i+\sum_{i=128}^{135} a_i+\sum_{i=124}^{126} a_i+\sum_{i=120}^{122} a_i\\
&\quad+\sum_{i=116}^{117} a_i+\sum_{i=112}^{114} a_i+a_{108}+a_{106}+a_{104}+a_{101}\\
&\quad+\sum_{i=96}^{98} a_i+\sum_{i=90}^{92} a_i+\sum_{i=87}^{88} a_i+a_{85}+a_{83}+\sum_{i=79}^{81} a_i
\end{aligned}
$$

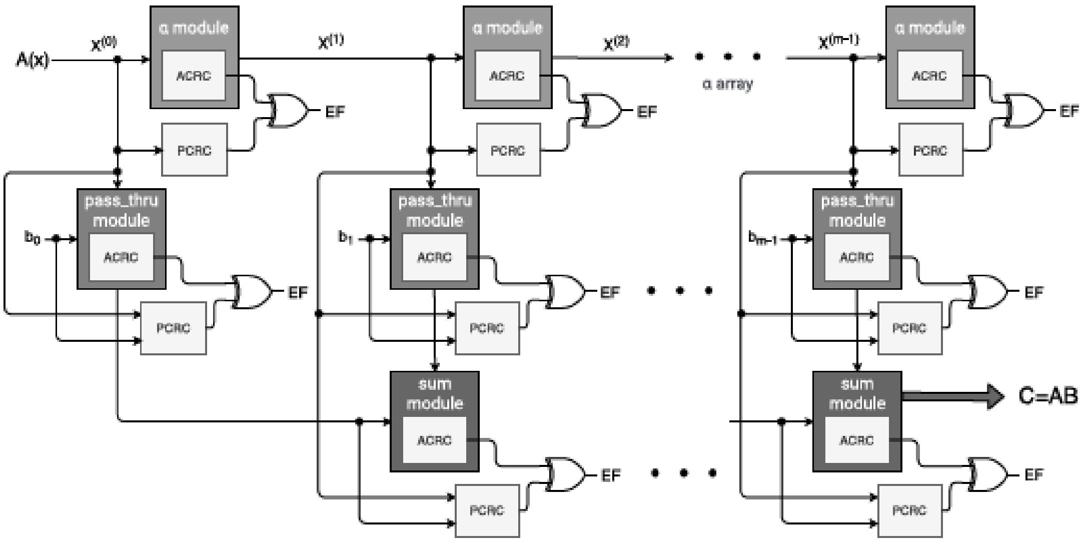

Fig. 1. Finite field multiplier with the proposed error detection schemes based on CRC.

$$
\begin{aligned}
&+a_{76}+\sum_{i=72}^{73} a_i+\sum_{i=65}^{66} a_i+\sum_{i=58}^{60} a_i+a_{56}+a_{53}\\
&+\sum_{i=48}^{49} a_i+a_{46}+\sum_{i=43}^{44} a_i+\sum_{i=37}^{41} a_i+\sum_{i=34}^{35} a_i\\
&+\sum_{i=30}^{31} a_i+a_{26}+\sum_{i=22}^{24} a_i+a_{16}+\sum_{i=7}^{11} a_i\Bigg) x^9\\
&+\cdots+\Bigg(a_{161}+\sum_{i=158}^{159} a_i+\sum_{i=153}^{154} a_i+\sum_{i=148}^{151} a_i+\sum_{i=143}^{145} a_i\\
&\quad+\sum_{i=138}^{141} a_i+\sum_{i=129}^{136} a_i+\sum_{i=125}^{127} a_i+\sum_{i=121}^{123} a_i+\sum_{i=117}^{118} a_i\\
&\quad+\sum_{i=113}^{115} a_i+a_{109}+a_{107}+a_{105}+a_{102}+\sum_{i=97}^{99} a_i\\
&\quad+\sum_{i=91}^{93} a_i+\sum_{i=88}^{89} a_i+a_{86}+a_{84}+\sum_{i=80}^{82} a_i+a_{77}\\
&\quad+\sum_{i=73}^{74} a_i+\sum_{i=66}^{67} a_i+\sum_{i=59}^{61} a_i+a_{57}+a_{54}+\sum_{i=49}^{50} a_i\\
&\quad+a_{47}+\sum_{i=44}^{45} a_i+\sum_{i=35}^{42} a_i+\sum_{i=31}^{32} a_i+a_{27}+\sum_{i=23}^{25} a_i\\
&\quad+a_{17}+\sum_{i=8}^{12} a_i\Bigg).
\end{aligned}
$$

We rename the coefficients to calculate the actual CRC-10 for GF(2^{163}) in the α^2 module (ACRC10_{163}), obtaining the same formulations as for the α module, not presented for the sake of brevity.

IV. Error Coverage and FPGA Implementations

To calculate the error coverage provided by the different error detection schemes presented in this brief, the total number of operations need to be taken into account. For the finite field GF(2^{163}), the addition chain C obtained is $C = \{1, 2, 4, 5, 10, 20, 40, 80, 81, 162\}$. The computational steps to calculate the inverse of $A \in$ GF(2^{163}) using such an addition chain are illustrated in Table I, where V_i's are the integers in the addition chain, $V_j = V_{i-1}$, and $U_k = V_i - V_j$.

TABLE I
Steps Needed to Perform the Inverse of $A \in$ GF(2^{163}) Using Addition Chain

Step	$\beta_{V_i}(x)$	$\beta_{V_j+U_k}(x)$	Exponentiation
1	$\beta_1(x)$	$-$	A
2	$\beta_2(x)$	$\beta_{1+1}(x)$	$(\beta_1)^{2^1}\beta_1 = A^{2^2-1}$
3	$\beta_4(x)$	$\beta_{2+2}(x)$	$(\beta_2)^{2^2}\beta_2 = A^{2^4-1}$
4	$\beta_5(x)$	$\beta_{4+1}(x)$	$(\beta_4)^{2^1}\beta_1 = A^{2^5-1}$
5	$\beta_{10}(x)$	$\beta_{5+5}(x)$	$(\beta_5)^{2^5}\beta_5 = A^{2^{10}-1}$
6	$\beta_{20}(x)$	$\beta_{10+10}(x)$	$(\beta_{10})^{2^{10}}\beta_{10} = A^{2^{20}-1}$
7	$\beta_{40}(x)$	$\beta_{20+20}(x)$	$(\beta_{20})^{2^{20}}\beta_{20} = A^{2^{40}-1}$
8	$\beta_{80}(x)$	$\beta_{40+40}(x)$	$(\beta_{40})^{2^{40}}\beta_{40} = A^{2^{80}-1}$
9	$\beta_{81}(x)$	$\beta_{80+1}(x)$	$(\beta_{80})^{2^1}\beta_1 = A^{2^{81}-1}$
10	$\beta_{162}(x)$	$\beta_{81+81}(x)$	$(\beta_{81})^{2^{81}}\beta_{81} = A^{2^{162}-1}$

As is shown in Table I, 9 finite field multiplications and 162 finite field squarings are required. Each multiplication in GF(2^{163}) uses 162 *sum* modules, 162 α modules, and 163 *pass-thru* modules; on the other hand, each squaring in GF(2^{163}) uses 162 *sum* modules and 162 α^2 modules. Therefore, the total number of operations and signatures is $9 \cdot (162 + 162 + 163) + 162 \cdot (162 + 162)$ or close to 5.7×10^4. The error coverage percentage is calculated by performing $100 \cdot (1-(1/2)^{\text{sign}})$%, where $sign$ denotes the number of signatures. For the case of GF(2^{163}), the error coverage percentage is $100(1-(1/2)^{5.7\times10^4})$% or very close to 100%. The proposed error detection schemes target embedded systems where low-complexity realizations are highly important. Therefore, we have implemented our error detection schemes for the entire inversion architecture of the NIST field GF(2^{163}) with CRC-10 for Xilinx FPGA family Kintex Ultrascale+ device xcku15p-ffve1760-1LV-i using the Vivado tool and Verilog as the hardware design entry. The proposed schemes in this brief have an area overhead of 25.51% in terms of configurable logic blocks (CLB) look-up tables (LUTs) (198 402 CLB LUTs for the inversion architecture without any error detection schemes and 248 807 when CRC-10 is applied to the original inversion block).

The choice of the utilized signature relies on the constraints of each system. For applications where performance is critical, the signature size can be increased while for deeply embedded systems, smaller signatures are preferred. The overhead achieved is acceptable taking into account the high error coverage obtained.

No previous research has been performed on this type of scheme for detecting errors in finite field inversions using FLT and ITA architectures to the best of our knowledge. Reyhani and Hasan [10] derived formulations for parity signatures in GF(2^m) for multiplication (not inversion), providing one EF on each module. The major drawback of parity signatures is that their error coverage is approximately 50%, i.e., if the number of faults is even, the approach would not be able to detect the faults. This highly predictable countermeasure can be circumvented by intelligent fault injection. With the CRC signatures derived in this brief, each module outputs ten EFs, making each module practicable immune to fault analysis attacks.

V. Conclusion

Finite field inversion is a complex, expensive, and time-consuming task that may require thousands of gates. In this brief, error detection schemes are proposed for both FLT and ITA algorithms, which are used to perform finite field inversion over GF(2^m) with polynomial basis field element representation. Such error detection schemes are based on CRC signatures and they can be used in any application that utilizes finite field inversions. We have derived closed formulations for CRC-10 signatures over GF(2^{163}) and implemented these signatures on FPGA to benchmark the overhead and show their suitability for constrained embedded systems. The proposed schemes in this brief have an area overhead of 25.51% in terms of CLB LUTs (198 402 CLB LUTs for the inversion architecture without any error detection schemes and 248 807 when CRC-10 is applied to the original inversion block). As the results show, the proposed error detection architectures achieve very high error coverage at the cost of acceptable overhead. We would also like to note that the proposed approaches are oblivious of the hardware platform and indifferent in error coverage with respect to permanent, transient, and long transient faults, making them suitable for different applications ranging from classical/postquantum cryptography to error detecting codes.

References

[1] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, Sep. 1988.

[2] J. Guajardo and C. Paar, "Itoh-Tsujii inversion in standard basis and its application in cryptography and codes," *Des., Codes Cryptogr.*, vol. 25, pp. 207–216, Dec. 2002.

[3] F. Rodriguez-Henriquez, N. A. Saqib, and N. Cruz-Cortes, "A fast implementation of multiplicative inversion over GF(2^m)," in *Proc. Int. Symp. Inf. Technol.*, pp. 574–579, Apr. 2005.

[4] J. S. Coron, A. Roy, and S. Vivek, "Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures," in *Proc. CHES*, 2014, pp. 170–187.

[5] S. Subramanian, M. Mozaffari Kermani, R. Azarderakhsh, and M. Nojoumian, "Reliable hardware architectures for cryptographic block ciphers LED and HIGHT," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 10, pp. 1750–1758, Oct. 2017.

[6] 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. (DFT)*, Vancouver, BC, Canada, Oct. 2011, pp. 325–331.

[7] M. Mozaffari Kermani and R. Azarderakhsh, "Reliable hash trees for post-quantum stateless cryptographic hash-based signatures," in *Proc. DFT*, Oct. 2015, pp. 103–108.

[8] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1997.

[9] B. S. Kaliski, "The Montgomery inverse and its applications," *IEEE Trans. Comput.*, vol. 44, no. 8, pp. 1064–1065, Aug. 1995.

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

[11] D. Hankerson and A. Menezes, "NIST elliptic curves," in *Encyclopedia of Cryptography and Security*. Boston, MA, USA: Springer, 2011, pp. 843–844.