Efficient Low-Latency Multiplication Architecture for NIST Trinomials With RISC-V Integration

José L. Imaña, Luis Piñuel, Yao-Ming Kuo, *Member, IEEE*, Oscar Ruano, and Francisco García-Herrero

***Abstract*—Binary extension field arithmetic is widely used in several important applications such as error-correcting codes, cryptography and digital signal processing. Multiplication is usually considered the most important finite field arithmetic operation. Therefore efficient hardware architectures for multiplication are highly desired. In this brief, a new architecture for multiplication over finite fields generated by irreducible trinomials $f(x) = x^m + x^t + 1$ is presented. The architecture here proposed is based on the use of a polynomial multiplier and a cyclic shift register that can perform the multiplication in $t - 1$ clock cycles. The general architecture is applied to the trinomials recommended by NIST (National Institute of Standards and Technology). Furthermore, a RISC-V instruction set for the proposed multiplier is implemented and validated using VeeR-EL2 on a Nexys A7 FPGA. To the best knowledge of the authors, this is the first work that integrates the multiplication based on NIST trinomials into a RISC-V SoC. Results show an improvement of several orders of magnitude in terms of latency at a cost of less than 50% more of area.**

***Index Terms*—Error-correcting codes, cryptography, finite field arithmetic, multiplication, NIST trinomials, RISC-V.**

I. Introduction

BINARY extension field $GF(2^m)$ arithmetic is widely used in several important applications such as error-correcting codes, cryptography and digital signal processing [1], [2], [3], [4], [5]. These applications often require efficient VLSI implementations of arithmetic operations, especially for multiplication, which is usually considered the most important finite field arithmetic operation. The complexity of the multiplier depends on the irreducible polynomial $f(x)$ selected for the field. For hardware implementations, low Hamming weight irreducible polynomials, such as trinomials and pentanomials, are normally used [6], [7], [8]. Irreducible trinomials $f(x) = x^m + x^n + 1$ are very important because they are abundant and they exhibit the lowest Hamming weight [9]. Specific irreducible trinomials have been recommended by NIST (National Institute of Standards and Technology) for their use in digital signatures [10]. Furthermore, some of the public-key encryption and key-establisment algorithms (such as the code-based Classic McEliece) submitted to the ongoing (Round 4) NIST Post-Quantum Cryptography (PQC) standardization process also use irreducible trinomials in their specifications [4].

Efficient methods and architectures for finite field multiplication have been proposed in the literature [6], [8], [9], [11]. Two-step classic $GF(2^m)$ multiplication requires a multiplication of polynomials followed by a reduction modulo an irreducible polynomial [12]. An efficient multiplication method was proposed by Mastrovito in which a product matrix was introduced to combine the above steps together [13]. Other methods use a divide-and-conquer approach (such as Karatsuba algorithm) for polynomial multiplication [14]. The use of suitable polynomials, such as irreducible trinomials, means that modular polynomial reduction can be efficiently implemented [7], [9], [13], [15].

In this brief, a new architecture for multiplication over finite fields (using the two-step classic method) generated by irreducible trinomials $f(x) = x^m + x^t + 1$ is presented. The architecture here proposed is based on the use of a polynomial multiplier and a cyclic shift register that can perform the multiplication in $t - 1$ clock cycles. The general architecture is applied to the irreducible trinomials $f(x) = x^{409} + x^{87} + 1$ and $f(x) = x^{233} + x^{74} + 1$ recommended by NIST and to the irreducible trinomials $f(x) = x^{193} + x^{15} + 1$ and $f(x) = x^{113} + x^9 + 1$ recommended by SECG (Standards for Efficient Cryptography Group) [16]. Furthermore, a RISC-V instruction set [17] for the proposed multiplier is implemented and validated using VeeR-EL2 on a Nexys A7 FPGA. To the best knowledge of the authors, this is the first work that integrates the multiplication based on NIST trinomials into a RISC-V SoC.

This brief is organized as follows. Section II provides notation and mathematical background. The new multiplier for general irreducible trinomials is introduced in Section III, where an example of multiplication for $f(x) = x^6 + x^4 + 1$ and the description of the new hardware architecture are also given. Section IV presents RISC-V instruction set for the proposed multiplier and experimental results of implemention using VeeR-EL2 on a Nexys A7 FPGA. Finally, conclusions are given in Section V.

II. Background

Any element A in the finite field $GF(2^m)$ can be represented as $A = \sum_{i=0}^{m-1} a_i x^i$, with $a_i \in GF(2) = \{0, 1\}$ and x being a root of an irreducible polynomial $f(y) = \sum_{i=0}^{m} f_i y^i$ over $GF(2)$. Arithmetic operations in $GF(2^m)$ are performed modulo $f(x)$. *Addition* of polynomials is carried out under modulo 2 arithmetic, so the addition of two elements becomes the bitwise XOR of their binary representations.

Manuscript received 12 February 2024; accepted 20 February 2024. Date of publication 23 February 2024; date of current version 31 July 2024. The work of José L. Imaña and Luis Piñuel was supported in part by MCIN/AEI/10.13039/501100011033 and in part by "ERDF A Way of Making Europe" under Grant PID2021-123041OB-I00. This brief was recommended by Associate Editor Z. Yu. *(Corresponding author: José L. Imaña.)*

José L. Imaña, Luis Piñuel, Oscar Ruano, and Francisco García-Herrero are with the Department of Computer Architecture and Automation, Complutense University, 28040 Madrid, Spain (e-mail: jluimana@ucm.es; lpinuel@ucm.es; oruano@ucm.es; francg18@ucm.es).

Yao-Ming Kuo is with the Digital Design Engineering, Monolithic Power Systems, 08029 Barcelona, Spain (e-mail: ykuo@ieee.org).

Digital Object Identifier 10.1109/TCSII.2024.3369103

Multiplication in $GF(2^m)$ of two elements $C = A \cdot B \bmod f(x)$ is usually considered the most important and complex operation. The rest of the operations in $GF(2^m)$ such as inversion or exponentiation are derived from multiplication. Two-step classic multiplication in $GF(2^m)$ [12] requires a polynomial multiplication followed by a reduction modulo the irreducible polynomial $f(x)$. *Polynomial multiplication* $D = A \cdot B$ is a polynomial with maximum degree $2m-2$ where its coefficients are determined by the expressions [12]:

$$d_j = \begin{cases} \sum_{i=0}^{j} a_i b_{j-i}, & j = 0 \ldots m-1 \\ \sum_{i=j}^{2m-2} a_{j-i+(m-1)} b_{i-(m-1)}, & j = m \ldots 2m-2 \end{cases} \quad (1)$$

It can be observed in (1) that $d_0, \ldots, d_{m-1}$ are the coefficients of $x^0, \ldots, x^{m-1}$ of the polynomial D, respectively, that must not be reduced. However, the powers $x^m, \ldots, x^{2m-2}$ of D (with coefficients $d_m, \ldots, d_{2m-2}$, respectively) must be reduced modulo $f(x)$.

After the polynomial multiplication $D = A \cdot B$ given in (1), a *reduction modulo the irreducible polynomial* $f(x)$ must be performed. In modular reduction $C = D \bmod f(x)$, the degree $2m-2$ polynomial D is reduced by the degree m irreducible polynomial $f(x)$, resulting in a polynomial C with maximum degree $m-1$. The product C can be represented in matrix notation as $C = \mathbf{M} \cdot D$, where $\mathbf{M} = (\mathbf{I}|\mathbf{R})$ is a $(m \times 2m-2)$ matrix that can be decomposed in a $(m \times m)$ identity matrix $\mathbf{I}$ and a $(m \times m-1)$ *reduction matrix* $\mathbf{R}$, only dependent on the irreducible polynomial $f(x)$. Therefore the reduction modulo $f(x)$ can be given as follows [12]:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & r_0^0 & \cdots & r_0^{m-2} \\ 0 & 1 & \cdots & 0 & r_1^0 & \cdots & r_1^{m-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & r_{m-1}^0 & \cdots & r_{m-1}^{m-2} \end{pmatrix} \cdot \begin{pmatrix} d_0 \\ \vdots \\ d_{m-1} \\ d_m \\ \vdots \\ d_{2m-2} \end{pmatrix} \quad (2)$$

The coefficients $r_j^i \in GF(2)$ given in (2) can be computed as follows [12]:

$$r_j^i = \begin{cases} f_j, & j = 0 \ldots m-1, i = 0 \\ r_{j-1}^{i-1} + r_{m-1}^{i-1} r_j^0, & j = 0 \ldots m-1, i = 1 \ldots m-2 \end{cases} \quad (3)$$

where $r_{j-1}^{i-1} = 0$ if $j = 0$. Finally, the product $C = A \cdot B \bmod f(x) = D \bmod f(x)$ can be computed using (2), where the coefficients of the reduction matrix $\mathbf{R}$ are given in (3).

III. New Multiplier Architecture

For hardware implementation of multipliers over $GF(2^m)$, low Hamming weight irreducible polynomials are normally used [7], [8], [9]. *Irreducible trinomials* $f(x) = x^m + x^n + 1$ are very important because they are abundant and they exhibit the lowest Hamming weight. Furthermore, trinomials $f(x) = x^{409} + x^{87} + 1$ and $f(x) = x^{233} + x^{74} + 1$ have been recommended by NIST for digital signatures [10] and trinomials $f(x) = x^{193} + x^{15} + 1$ and $f(x) = x^{113} + x^9 + 1$ have been also recommended by SECG [16].

As shown in Section II, the product $C = A \cdot B \bmod f(x)$ can be computed for general irreducible polynomials $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + 1$ using equations (1)-(3). For trinomials, only one non-null coefficient f_t is given for $f(x) = x^m + f_t x^t + 1$, with $f_t = 1$, so the products $f_k \cdot f_l$, with $k \neq l$, in (3) are null. Furthermore, for a given column i, the expressions for the computation of $\mathbf{R}$ in (3) are:

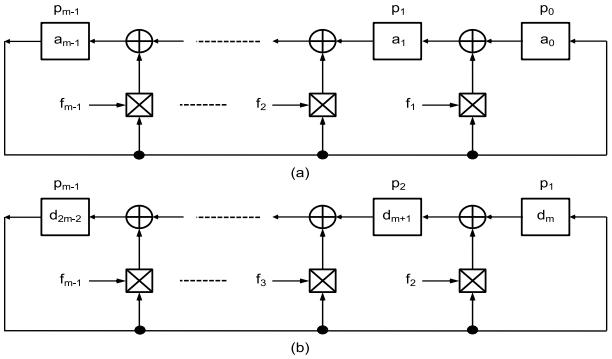

Fig. 1. Implementations (a) $A \cdot x \bmod f(x)$, (b) $D_H \cdot x^t \bmod f(x)$.

$$r_j^i = \begin{cases} r_{m-1}^{i-1}, & j = 0 \\ r_{j-1}^{i-1} + r_{m-1}^{i-1} f_j, & j = 1 \ldots m-1 \end{cases} \quad (4)$$

These expressions are similar to those obtained for the computation of the product $P = A \cdot x \bmod f(x)$, with x being a root of the irreducible polynomial $f(y) = y^m + f_{m-1}y^{m-1} + \ldots + f_1 y + 1$ and $A \in GF(2^m)$ [12]. The product $A \cdot x = a_{m-1}x^m + \ldots + a_0 x$ can be computed modulo $f(x)$ using the fact that $x^m = f_{m-1}x^{m-1} + \ldots + f_1 x + 1$ and replacing x^m in the previous expression $A \cdot x$, in such a way that $P = A \cdot x \bmod f(x) = p_{m-1}x^{m-1} + \ldots + p_0$, with [12]

$$p_j = \begin{cases} a_{m-1}, & j = 0 \\ a_{j-1} + a_{m-1} f_j, & j = 1 \ldots m-1 \end{cases} \quad (5)$$

It can be observed the similarity among equations (5) and (4). The operation given in (5) can be implemented using a cyclic shift register as shown in Fig. 1(a), where $\oplus$ refers to an XOR, $\boxtimes$ stands for an AND gate and $\square$ refers to a 1-bit register. The registers are initially loaded with the coordinates of A, and the coefficients f_i, $i = 1 \ldots m-1$, of the irreducible polynomial are connected to the AND gates together with the output of the last 1-bit register of the cyclic shift register. It can be observed that after one clock cycle, the registers contents will be the coefficients $(p_0, p_1, \ldots, p_{m-1})$ of the product $P = A \cdot x \bmod f(x)$.

The use of the cyclic shift register given in Fig. 1(a) for the implementation of $P = A \cdot x \bmod f(x)$ let us consider a new architecture for the implementation of the product $C = A \cdot B \bmod f(x) = D \bmod f(x)$ using irreducible trinomials.

As given in Section II, the terms $x^0, \ldots, x^{m-1}$ of D (with coefficients $d_0, \ldots, d_{m-1}$, respectively) do not need to be reduced. However, the powers $x^m, \ldots, x^{2m-2}$ of D (with coefficients $d_m, \ldots, d_{2m-2}$, respectively) must be reduced modulo $f(x)$. The polynomial $D = d_{2m-2}x^{2m-2} + \ldots + d_m x^m + d_{m-1}x^{m-1} + \ldots + d_0$ can be rewritten as follows:

$$\begin{aligned} D &= D_H x^m + D_L \\ &= \left(d_{2m-2}x^{m-2} + \ldots + d_m\right)x^m + d_{m-1}x^{m-1} + \ldots + d_0 \end{aligned} \quad (6)$$

with $D_L \bmod f(x) = D_L$ and $D_H \bmod f(x) = D_H$. Using modulo properties, $D \bmod f(x) = (D_H x^m + D_L) \bmod f(x) = D_H x^m \bmod f(x) + D_L$, where $D_H x^m \bmod f(x)$ is:

$$\begin{aligned} &\left((D_H \bmod f(x)) \cdot \left(x^m \bmod f(x)\right)\right) \bmod f(x) \\ &= \left(D_H \cdot \left(x^m \bmod f(x)\right)\right) \bmod f(x). \end{aligned} \quad (7)$$

For irreducible trinomials $f(x) = x^m + x^t + 1$, with $t = 1 \ldots m-1$, we have that $x^m = x^t + 1 \bmod f(x)$, therefore using again

modulo properties we have $D_H x^m \bmod f(x) = (D_H \cdot (x^t + 1)) \bmod f(x) = (D_H x^t + D_H) \bmod f(x) = D_H x^t \bmod f(x) + D_H$. Finally, we obtain the following expression (8) for the computation of the product $C = A \cdot B \bmod f(x)$:

$$\begin{aligned} C = A \cdot B \bmod f(x) &= D \bmod f(x) \\ &= D_H x^t \bmod f(x) + D_H + D_L \end{aligned} \tag{8}$$

where the addition $D_H + D_L$ is given as follows

$$\begin{aligned} D_H + D_L = d_{m-1}x^{m-1} + (d_{m-2} + d_{2m-2})x^{m-2} + \ldots \\ +(d_2 + d_{m+2})x^2 + (d_1 + d_{m+1})x + (d_0 + d_m) \end{aligned} \tag{9}$$

Therefore, the implementation of $D_H + D_L$ can be done with the bitwise XOR of the coefficients as given in equation (9).

The cyclic shift register given in Fig. 1(a) for the implementation of $P = A \cdot x \bmod f(x)$ could be used to compute $D_H x^t \bmod f(x)$. It can be observed that the product $D_H x^t$ (corresponding with the irreducible trinomial $f(x) = x^m + x^t + 1$) can be written as the t products $(\ldots(((D_H x)x)\ldots x)$, so using modulo properties we have:

$$\begin{aligned} D_H x^t \bmod f(x) &= (\ldots(((D_H x)x)\ldots x) \bmod f(x) \\ &= (\ldots(((D_H x \bmod f(x))x \bmod f(x))\ldots) \bmod f(x). \end{aligned} \tag{10}$$

As shown in equation (6), D_H only have $m-1$ coefficients, i.e., $D_H = d_{2m-2}x^{m-2} + \ldots + d_m$, so the product $D_H x \bmod f(x) = d_{2m-2}x^{m-1} + \ldots + d_m x$ does not need to be reduced modulo $f(x)$. The lowest trinomial we could use would be $f(x) = x^m + x + 1$ ($t = 1$), so for this trinomial no reduction would be needed and, therefore, no shifts (i.e., 0 clock cycles) would be needed in the structure given in Fig. 1(a) to get the reduction. In this way, the computation of $D_H x^t \bmod f(x)$, would need $t-1$ clock cycles to be completed.

Fig. 1(a) could be modified to implement the above behaviour as follows: the $m-1$ coefficients of D_H are stored in the $m-1$ most-significant registers of the shift register, while for the least-significant register p_0, the AND gate with f_1 and the XOR gate feeding register p_1 are removed. The output of the most-significant register p_{m-1} is adjusted to feed 1-bit register p_1. Fig. 1(b) shows the final architecture needed to compute $D_H x^t \bmod f(x)$ for $f(x) = x^m + x^t + 1$ in $t-1$ clock cycles, where the 1-bit registers $p_{m-1}, p_{m-2}, \ldots, p_2, p_1$ are initially loaded with the coordinates of $D_H = (d_{2m-2}, \ldots, d_{m+1}, d_m)$, respectively.

It can be observed that for $t = 1$, the product $D_H x \bmod f(x) = d_{2m-2}x^{m-1} + \ldots + d_{m+1}x^2 + d_m x$ does not need to be reduced modulo $f(x)$ and the result (stored in registers $p_{m-1}, \ldots, p_2, p_1$, respectively) does not include a coefficient for x^0, as explained before in this section.

For $t = 2$, the product $D_H x^2 \bmod f(x) = (D_H x)x \bmod f(x) = d_{2m-2}x^m + d_{2m-3}x^{m-1} + \ldots + d_{m+1}x^3 + d_m x^2 = d_{2m-3}x^{m-1} + \ldots + d_{m+1}x^3 + (d_{2m-2} + d_m)x^2 + d_{2m-2}$ and therefore the result (also stored in registers $p_{m-1}, \ldots, p_2, p_1$, respectively) does not include a coefficient for x^1. In general, the product $D_H x^t \bmod f(x)$ given in the 1-bit registers does not include a coefficient for the term x^{t-1}. Furthermore, it can be proven that the product $D_H x^t \bmod f(x)$ computed by the architecture shown in Figure 1(b) is given by:

$$\begin{aligned} D_H x^t \bmod f(x) = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \ldots + p_t x^t \\ +p_{t-1}x^{t-2} + p_{t-2}x^{t-3} + \ldots + p_2 x + p_1 \end{aligned} \tag{11}$$

TABLE I
COMPUTATION OF $D_H \cdot x^4 \bmod (x^6 + x^4 + 1)$

Cycle	p_5	p_4	p_3	p_2	p_1
Init.	d_{10}	d_9	d_8	d_7	d_6
t_1	d_9	$d_{10} + d_8$	d_7	d_6	d_{10}
t_2	$d_{10} + d_8$	$d_9 + d_7$	d_6	d_{10}	d_9
t_3	$d_9 + d_7$	$d_{10} + d_8 + d_6$	d_{10}	d_9	$d_{10} + d_8$

where p_i, with $i = 1 \ldots m-1$, represent the coefficients stored in the corresponding registers after $t-1$ clock cycles.

A. Example: Multiplication for $f(x) = x^6 + x^4 + 1$

For the specific case $f(x) = x^6 + x^4 + 1$, the product $C = A \cdot B \bmod f(x)$ is computed using expression (8), in such a way that $C = D \bmod f(x) = D_H x^4 \bmod f(x) + D_H + D_L$.

The coefficients of the polynomial multiplication $D = (d_{10}, \ldots, d_0)$ are computed using the expression (1), where $D_H = (d_{10}, \ldots, d_6)$ and $D_H = (d_5, \ldots, d_0)$. The addition of $D_H + D_L$ is determined using expression (9) as follows:

$$\begin{aligned} D_H + D_L = d_5 x^5 + (d_4 + d_{10})x^4 + (d_3 + d_9)x^3 \\ +(d_2 + d_8)x^2 + (d_1 + d_7)x + (d_0 + d_6) \end{aligned} \tag{12}$$

The computation of $D_H x^4 \bmod f(x)$ is done in 3 clock cycles using the cyclic shift register given in Fig. 1(b). Table I shows the contents of the 1-bit registers in the 3 clock cycles, where *Init.* represents the initial contents of the registers. Following (11), the polynomial $D_H x^4 \bmod f(x)$ is given as:

$$\begin{aligned} D_H x^4 \bmod f(x) = (d_9 + d_7)x^5 + (d_{10} + d_8 + d_6)x^4 \\ +d_{10}x^2 + d_9 x + (d_{10} + d_8) \end{aligned} \tag{13}$$

Finally, the product $C = A \cdot B \bmod f(x)$ is computed by the addition of (12) and (13) as follows:

$$\begin{aligned} C = A \cdot B \bmod f(x) = (d_9 + d_7 + d_5)x^5 \\ +(d_8 + d_6 + d_4)x^4 + (d_9 + d_3)x^3 + (d_{10} + d_8 + d_2)x^2 \\ +(d_9 + d_7 + d_1)x + (d_{10} + d_8 + d_6 + d_0) \end{aligned} \tag{14}$$

where in x^4, the addition $d_{10} + d_{10} = 0$. It can be proven that the result given in (14) matches the one obtained by direct application of equations (2) and (3).

B. Hardware Architecture of the Multiplier

Based on the above considerations, the hardware architecture of the new multiplier proposed for the computation of the product $C = A \cdot B \bmod f(x) = D \bmod f(x) = D_H x^t \bmod f(x) + D_H + D_L$ using irreducible trinomials $f(x) = x^m + x^t + 1$ is shown in Fig. 2.

The *polynomial multiplier* module in the upper part of Fig. 2 computes the product of polynomials $D = A \cdot B$ as given in equation (1). It must be noted that this polynomial multiplier can be implemented using the commonly integrated multiplier found in most coprocessors. The lower part in Fig. 2 efficiently computes the product based on irreducible trinomials, completing the computation in $t-1$ clock cycles. The $2m-1$ outputs of the multiplier are stored in $2m-1$ 1-bit registers as follows: $D_L = (d_{m-1}, \ldots, d_0)$ is stored in $m-1$ registers (at the right in Fig. 2) and $D_H = (d_{2m-2}, \ldots, d_m)$ is stored in the $m-2$ registers of the cyclic shift register given in Fig. 1(b). The addition of $D_H + D_L$ is performed in Fig. 2 by the $m-1$ XOR gates (within a dotted box) below the 1-bit registers storing D_L. The computation of $D_H x^t \bmod f(x)$ is performed, after $t-1$ clock cycles, by the cyclic shift

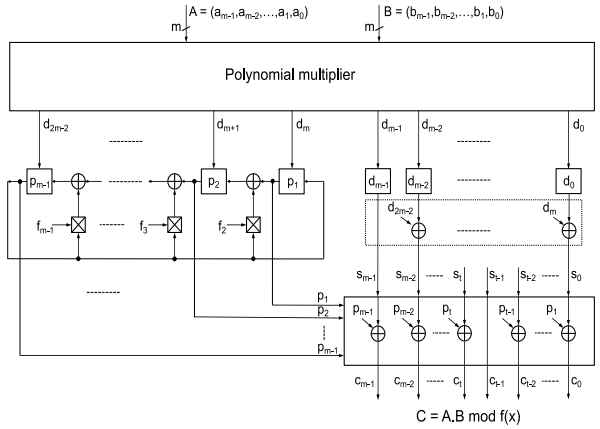

Fig. 2. Architecture of the multiplier $C = A \cdot B \bmod (x^m + x^t + 1)$.

register at the left in Fig. 2. The final addition is performed by the $m - 1$ XOR gates at the bottom of Fig. 2 as given in (11). It can be observed that the general architecture given in the lower part (specific for trinomials) in Fig. 2 has a delay $T_X + max\{T_X, T_A\}$, where T_X and T_A stand for the delay of 2-input XOR and AND gates, respectively.

IV. Implementation and Hardware Results

To address the challenge of resource sharing and computational speed, the polynomial multiplier in the upper part of Fig. 2 is implemented using the existing general-purpose multiplier integrated in most coprocessors. Our attention is centered on NIST-recommended trinomials, so to ensure the architecture's generality, multiplexors are added to provide support for any of these trinomials. The entire design incorporates pipeline registers at the input and output of each module and divides the polynomial multiplier into 14 pipeline stages to maintain a lower critical path compared to the RISC-V CPU. This enables operations to be performed at maximum speed. Additionally, this approach achieves significant area savings as the bits stored in the shift register also act as pipeline registers, eliminating the need for duplication. This architecture not only facilitates resource sharing but also provides a substantial speedup compared to a CPU execution using optimized software code. Other fully combinational architectures and designs with higher degrees of parallelism were explored; however, their integration with the RISC-V CPU proved challenging due to the substantial increase in hardware resources required for working with large-order polynomials (e.g., trinomials of order 409).

A. Control and Final Architecture

To integrate these operations, new instructions are added to the compiler. Given that the operands can be up to 409-bit words, three different instructions have been introduced for each operand (A and B). *ffloadas* (*ffloadbs*) begins loading the operand from the general-purpose registers, while *ffloada* (*ffloadb*) loads the information into the internal pipeline register for inputs A and B (Fig. 2). Finally, *ffloadae* (*ffloadbe*) indicates the completion of operand loading. Note that different trinomial orders will require a varying number of *ffloada* (*ffloadb*) instructions. However, the control mechanism is simplified and completely generic, thanks to the start and end instructions. Fig. 3 provides a summary of these instructions, following the standard format for RISC-V processors.

ffloadas	0000000	LOAD_H	LOAD_L	000	RESERVED	0001011
ffloada	0000001	LOAD_H	LOAD_L	000	RESERVED	0001011
ffloadae	0000010	LOAD_H	LOAD_L	000	RESERVED	0001011
ffloadbs	0000000	LOAD_H	LOAD_L	001	RESERVED	0001011
ffloadb	0000001	LOAD_H	LOAD_L	001	RESERVED	0001011
ffloadbe	0000010	LOAD_H	LOAD_L	001	RESERVED	0001011
ffmul	IDX	RESERVED	RESERVED	010	RESULT	0001011

Fig. 3. New instruction set.

TABLE II
Area for the VeeR-EL2 and the Modified Version

	LUT	Reg.	F7 Mux	F8 Mux	Slices	DSP
EL2	19547	9549	400	76	5991	4
EL2 mod.	28333	12187	464	108	8356	4
Increase	1,45	1,28	1,16	1,42	1,39	1,00

Furthermore, the integration of the multiplication module follows a similar approach to the division module of the VeeR-EL2 architecture [18], with the distinction that hardware resources are shared with the already integrated multiplier and the Galois Field extension given in [19].

B. Hardware Results and Comparisons

Table II presents a comparison between the standard RISC-V-based SoC VeeR-EL2 and our proposed solution, which incorporates to the VeeR-EL2 the multiplication previously described for the trinomials $x^{409}+x^{87}+1$ (NIST, SEGC), $x^{233}+x^{74}+1$ (NIST, SEGC), $x^{193}+x^{15}+1$ (SECG), and $x^{113}+x^9+1$ (SECG). The modified EL2 SoC can be found in the repository from [20], which allows readers to replicate all the experiments on the Nexys A7-100T board, which features an Artix7 FPGA and the C codes including all the compilation details. The architecture has been described using System Verilog and functionally verified using Maple as a golden model. As shown in Table II, there is approximately a 45% increase in LUTs and a 28% increase in registers. However, when comparing with Table III, the number of clock cycles is reduced by 2246 to 389 times, with the best case corresponding to the highest-order trinomial. In other words, the latency is reduced by 99.74% to 99.96% compared to the same SoC executing the same operations with optimized baremetal code.

To provide a fairer comparison, a figure of merit considering area and timing can be computed. For the worst-case scenario with the trinomial of order 113, an improvement of 389/1.45 = 268 is achieved, while for the best-case scenario with the trinomial of order 409, an improvement of 2246/1.45 = 1549 is obtained. In other words, with less than 50% of the area, the latency for computing multiplications with these trinomials is reduced from 3.3ms to 8.64μs in the worst case and from 0.79s to 14.12μs in the best case, demonstrating the efficiency of the proposed solution.

Compared to previous works from the authors on finite field arithmetic [19] and to the recently proposed RISC-V cryptography extensions [21] the improvement in terms of speedup is kept as the simplifications introduced in Section III reduce timing complexity from m to t thanks to the special properties of the trinomials exploited in this brief and not considered in [19] or [21].

Comparing in Table IV the multiplier that has been integrated into the proposal with the solutions available in the

TABLE III
TIMING RESULTS FOR THE STANDARD VEER-EL2 AND THE MODIFIED VERSION. BOTH DESIGNS HAVE THE SAME CRITICAL PATH, AS THE OPERATOR IS DEFINED FOR NOT BEING THE LIMITING MODULE

Polynomial order	113	193	233	409
RISC-V IMAC	83967	212854	309466	792862
Custom	216	260	277	353
Speedup	389	819	1117	2246

TABLE IV
ASIC SYNTHESIS RESULTS FOR A MULTIPLIER WITH A POLYNOMIAL OF ORDER 409

Work	Latency	Critical path	Area (Kgates)
[22]	26	$T_A + T_X$	620.2
[23]	11	$T_A + 5T_X$	39.85
[24]	64896	$T_A + 2T_X$	45.5
[25]	718	$T_A + T_X$	48.8
[26]	818	$T_A + T_X$	12.9
[27]	10816	$T_A + T_X$	9.2
[15]	818	$T_A + 2T_X$	2.6
This work	270	T_A+max(T_X, T_A)	24.3 (CLMUL) + 5.1

state-of-the-art it can be concluded that thorough examination of critical parameters, such as the critical path of each design, maximum latency in terms of clock cycles, and the total logical gates for the most restrictive polynomial (order 409), only two multipliers in this comparison reduce the number of clock cycles when compared to our proposal. Additionally, one multiplier fails to meet the critical path of the RISC-V core [23]. Furthermore, another multiplier, while achieving a performance improvement, comes at a cost of 620k gates, which is equivalent to 21 times the size of our accelerator [22]. Compared to the efficient solution from [15], our proposal has three times less latency in terms of clock cycles and shorter critical path, and the overhead in gates, compared to a processor that integrates the CLMUL module is only 1.9 times larger, as it only requires 5.1 additional Kgates. Also, the proposal described in our paper is 100% modular, so it can be applied to polynomials of different orders without adding extra area, while other solutions require a particular implementation for each polynomial.

V. CONCLUSION

To the best knowledge of the authors, this is the first work that integrates the multiplication based on NIST and SEGC trinomials into a RISC-V SoC. All the previous works have been based on the implementation of full-custom accelerators and co-processors which are isolated from the pipeline of the processor. This brief introduces a new approach to implement this multiplication taking into account some mathematical properties that allow the hardware resource sharing with other functional units from the processor, the reduction of latency with a moderate increase of area and the generalization for future cases due to the structure of the designed architecture and the definition of the ISA.

REFERENCES

[1] J. Lin, J. Sha, Z. Wang, and L. Li, "Efficient decoder design for nonbinary quasicyclic LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 5, pp. 1071–1082, May 2010.

[2] L.C. Washington, *Elliptic Curves: Number Theory and Cryptography*. Boca Raton, FL, USA: CRC Press, 2008.

[3] S. Heron, "Advanced encryption standard (AES)," *Netw. Secur.*, vol. 12, pp. 8–12, Dec. 2009.

[4] "Classic MeEliece, NIST PQC round 4 submission." Accessed: Sep. 7, 2023. [Online]. Available: https://classic.mceliece.org/mceliece-spec-20221023.pdf

[5] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *Proc. 3rd Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2005, pp. 164–175.

[6] A. Halbutogullari and Ç.K. Koç, "Mastrovito multiplier for general irreducible polynomials," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 503–518, May 2000.

[7] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$," *IEEE Trans. Comput.*, vol. 53, no. 8, pp. 945–959, Aug. 2004.

[8] J. L. Imaña, "Fast bit-parallel binary multipliers based on type-I pentanomials," *IEEE Trans. Comput.*, vol. 67, no. 6, pp. 898–904, Jun. 2018.

[9] J. L. Imaña, "LFSR-based bit-serial $GF(2^m)$ multipliers using irreducible trinomials," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 156–162, Jan. 2021.

[10] "Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters," Dept. Commer., Natl. Inst. Stand. Technol., Gaithersburg, MD, USA, Rep. NIST SP 800-186, 2023.

[11] H. Wu, "Bit-parallel finite field multiplier and squarer using polynomial basis," *IEEE Trans. Comput.*, vol. 51, no. 7, pp. 750–758, Jul. 2002.

[12] J. P. Deschamps, J. L. Imaña, and G. D. Sutter, *Hardware Implementation of Finite-Field Arithmetic*. New York, NY, USA: McGraw-Hill, 2009.

[13] B. Sunar and Ç.K. Koç, "Mastrovito multiplier for all trinomials," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 522–527, May 1999.

[14] Y. Li, S. Sharma, Y. Zhang, X. Ma, and C. Qi, "On the complexity of hybrid n-term Karatsuba multiplier for trinomials," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 3, pp. 852–865, Mar. 2020.

[15] F. Gebali and A. Ibrahim, "Efficient scalable serial multiplier over $GF(2^m)$ based on trinomial," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 10, pp. 2322–2326, Oct. 2015.

[16] *SEC 2: Recommended Elliptic Curve Domain Parameters: Standards for Efficient Cryptography Group Version 1.0*, Certicom Corp., Mississauga, ON, Canada, 2000.

[17] A. S. Waterman, *Design of the RISC-V Instruction Set Architecture*. Berkeley, CA, USA: Univ. California, 2016.

[18] "VeeR EL2 RISC-V core." Chips Alliance, Accessed: Jul. 7, 2023. [Online]. Available: https://github.com/ chipsalliance/Cores-VeeR-EL2

[19] Y.-M. Kuo, F. García-Herrero, O. Ruano, and J. A. Maestro, "RISC-V galois field ISA extension for non-binary error-correction codes and classical and post-quantum cryptography," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 682–692, Mar. 2023.

[20] "RISC-V GF ISA extension for trinomials." Accessed: Sep. 7, 2023. [Online]. Available: https://github.com/kuoyaoming93/Cores-SweRV-EL2#customizing-risc-v-gnu-toolchain-with-trinomial-instructions

[21] "RISC-V cryptography extensions volume II." Accessed: Sep. 7, 2023. [Online]. Available: https://github.com/riscv/riscv-crypto/releases/download/v20230823/riscv-crypto-spec-vector.pdf

[22] J. Xie, P. K. Meher, and J. He, "Low-latency area-delay-efficient systolic multiplier over GF(2m) for a wider class of trinomials using parallel register sharing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2012, pp. 89–92.

[23] J.-S. Pan, C.-Y. Lee, and P. K. Meher, "Low-latency digit-serial and digit-parallel systolic multipliers for large binary extension fields," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 12, pp. 3195–3204, Dec. 2013.

[24] Y. Y. Hua, C. W. Chiou, J.-M. Lin, C.-Y. Lee, and Y. H. Liu, "Low space-complexity digit-serial dual basis systolic multiplier over Galois field GF(2^m) using Hankel matrix and Karatsuba algorithm," *IET Inf. Secur.*, vol. 7, no. 2, pp. 75–86, Jun. 2013.

[25] C.-C. Chen, C.-Y. Lee, and E.-H. Lu, "Scalable and systolic montgomery multipliers over GF(2^m)," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E91-A, no. 7, pp. 1763–1771, 2008.

[26] G. Orlando and C. Paar, "A super-serial Galois fields multiplier for FPGAs and its application to public-key algorithms," in *Proc. 7th Annu. IEEE Symp. Field-Programm. Custom Comput.*, 1999, pp. 232–239.

[27] S. Bayat-Sarmadi, M. M. Kermani, R. Azarderakhsh, and C.-Y. Lee, "Dual-basis superserial multipliers for secure applications and lightweight cryptographic architectures," *IEEE Trans. Circuits Syst. II Exp. Briefs*, vol. 61, no. 2, pp. 125–129, Feb. 2014.