IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 April 2023, accepted 18 May 2023, date of publication 5 June 2023, date of current version 12 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3282781

RESEARCH ARTICLE

EC-Crypto: Highly Efficient Area-Delay Optimized Elliptic Curve Cryptography Processor

KHALID JAVEED[1], (Member, IEEE), ALI EL-MOURSY[1], (Senior Member, IEEE), AND DAVID GREGG[2]

[1]Department of Computer Engineering, College of Computing and Informatics, University of Sharjah, Sharjah, United Arab Emirates
[2]School of Computer Science, Trinity College Dublin, University of Dublin, Dublin, D02 PN40 Ireland

Corresponding author: Khalid Javeed (kjaveed@sharjah.ac.ae)

This work was supported in part by the Science Foundation Ireland, and in part by the University of Sharjah.

ABSTRACT Elliptic Curve Cryptography (ECC) based security protocols require much shorter key space which makes ECC the most suitable option for resource-limited devices as compared to the other public key cryptography (PKC) schemes. This paper presents a highly efficient area-delay optimized ECC crypto processor over the general prime field ($\mathbb{F}_p$). It is structured on a new novel finite field multiplier (FFM) where several optimization techniques have been incorporated to shorten the latency and hardware resource consumption. The proposed FFM architecture is embedded with a finite field adder/subtractor (FFAS) unit which is utilized to perform FFAS operations instead of deploying a dedicated unit. The Common Z (Co-Z) coordinates with the Montgomery ladder method are used to compute point multiplication, a core operation in all ECC-based crypto protocols. The work also proposes an efficient scheduling strategy to execute low-level finite field arithmetic primitives with minimum latency on the employed finite field arithmetic units. Due to these techniques, the proposed ECC processor is optimized for hardware resources, latency, and throughput. It is captured in Verilog-HDL, synthesized, and implemented on Virtex-7, Kintex-7, and Virtex-6 FPGA platforms using Xilinx Vivado and ISE Design Suite tools. On the Virtex-7 FPGA platform, it computes a single 256-bit scalar multiplication primitive in 0.7 *m*s, consumes just 6.2K slices, and delivers a throughput of 1428 operations per second. The implementation results show that it is a highly efficient design outperforming the state-of-the-art by providing a better area-delay product and higher efficiency. Therefore, it has the potential to be deployed in many applications where both latency and resource requirements are critical.

INDEX TERMS Elliptic curve cryptography (ECC), finite field multiplication, field programmable gate array (FPGA), hardware acceleration, finite field arithmetic.

I. INTRODUCTION

Nowadays elliptic curve cryptography (ECC) [1], [2] based cryptographic systems are preferred over Rivest, Shamir, and Adleman (RSA) [3] scheme due to much smaller key lengths which further translate into lower storage, bandwidth, and transmission cost. Different standardization bodies recommended 10-30× smaller key lengths for ECC as compared with RSA [4], [5]. Elliptic curve point multiplication (ECPM) over a well-chosen elliptic curve (EC) is the primary operation and is also the main computational part of almost all ECC-based security protocols. Usually, it is done by combining point doubling (PD) and point addition (PA) group operations which further require low-level finite field (FF) arithmetic primitives such as finite field addition/subtraction (FFAS), finite field multiplication (FFM), and finite field inversion/division (FFID) [6], [7]. Among these, FFID is the most time-critical operation and it is required if EC points are taken in affine coordinates (x, y) representation. However, fortunately, this FFID operation can be eliminated from EC group operations such as PD and PA using the projective coordinates representation at the cost of extra FFM operations. Therefore, in the projective space, FFM is the most

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

time-critical operation which limits the execution performance of the ECC-based cryptographic processor.

A classical method to compute an FFM operation on given numbers a and b over a large prime modulus p is done in two steps: first multiplication ($a \times b$) and then reduction modulo p. This reduction generally needs a long division operation which certainly is not feasible to compute for large operand sizes. Various methods have been proposed to efficiently compute the FFM primitive and can be broadly classified into three categories: using special or standard primes (SP), Montgomery multiplication (MM) [8], and interleaved multiplication (IM) [9]. SP primes have a special structure i.e., $2^a \pm 2^b \pm 2^c \pm 2^d \pm 1$, and are known as Mersenne or pseudo-Mersenne primes [10], [11]. Reduction over this form of prime structure can be achieved by cheap addition and shift operations which can result in high-performance design but turns into very dedicated architecture and lacks generality. MM is a widely deployed method that converts the operands and results into Montgomery and normal domains and performs the reduction using cheap shift and add operations. IM relies on a repetitive add-and-reduce approach where reduction is interleaved in each iteration. Both IM and MM-based designs can work for any general FF with arbitrarily prime modulus p and hence can be used to construct different types of EC cryptosystems using different curve parameters and prime values [12].

Several useful modifications and related architectures have been proposed for all these three approaches [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]. Designs reported in [14], [15], [16], [21], [22], [23], [25], [26], and [27] are based on MM, [19], [20], [24], [28], [29], [30] are based on IM and [17], [18] are using NIST recommended SP primes. Most of these designs are then further utilized in the development of several ECPM architectures [17], [19], [20], [21], [30], [31], [32], [33], [34], [35], [36], [37], [38]. In [31], a 256-bit high-performance EC PM architecture over general prime is developed using a new combined Karatsuba and schoolbook-based FFM architecture to perform low-level finite field operations. A lightweight architecture is proposed in [32] where the scheduling of low-level field operations are optimized to achieve low latency and low resource utilization. ECPM architectures in [19], [20], [21], [30], [33], and [35] are developed by adding modifications to the standard IM algorithm. References [19] and [20] are based on higher-radix (HR) techniques to process multiple bits in a single iteration consequently resulting in fewer iterations and ultimately reducing the required number of clock cycles. References [30], [33], and [35] proposed efficient ECPM implementation using modified radix-2 IM algorithm. Redundant-sign-digit (RSD) arithmetic-based efficient EC PM designs are reported in [21] and [37], where RSD representation is primarily utilized to shorten the long carry propagation delay and delivered a high throughput. The implementation results of these designs are reported for 256-bit after being synthesized on different FPGA platforms. Furthermore, various new types of ECs [12], [39], [40], [41] have been proposed targeting different security levels and speeding up PM operations. Therefore a flexible ECPM design is in demand that can be used with a wide range of curve parameters, allowing for customization and the ability to adjust the security level to fit specific application needs. It is also worth mentioning that ECC-based cryptography protocols can be developed using finite fields with binary characteristics $GF(2^m)$. An interested reader is referred to [42], [43], [44], and [45] for further information about such proposals. This work presents an efficient EC cryptographic processor over a general prime field. It is optimized for both latency and hardware resource occupation, in addition, it supports any modulus and curve parameter values. The main contributions are given as follows:

- This paper first demonstrates a novel and area-time efficient finite field multiplier (FFM) by proposing an efficient parallelism technique that enables the execution of internal critical operations of the IM algorithm concurrently with a significant reduction in employed resources. An efficient hardware architecture is then developed to realize the proposed modifications which can save significant hardware resources without any significant reduction in performance. It can work for any general modulus p, can be reconfigured for any field size up to 521-bit, and delivers high throughput with better area-delay product and efficiency.
- Most of the existing designs deployed multiple copies of the same FFM unit to execute several multiplication instructions simultaneously. In addition, these designs also integrated a separate dedicated FFAS unit to perform modular add/sub operations. However, instead of adding a dedicated FFAS unit, we further modified the FFM design so that it can perform the FFAS operation in addition to its normal multiplication operation which is represented as the FFMA unit. So in our proposed finite field arithmetic core (FFAC), three copies of the FFM and one FFMA unit are deployed.
- Subsequently, an area-delay optimized novel EC cryptographic processor is presented based on the proposed FFAC. A dedicated FFID unit is also designed based on an extended Euclidean method and integrated into the proposed processor to perform the final conversion of point multiplication from projective to affine space. Jacobian coordinates with Common Z (Co-Z) arithmetic is used while the Montgomery ladder algorithm for the computation of EC PM is utilized due to its natural ability to counter timing and simple power analysis attacks. Note that there is good room for parallelism in Co-Z arithmetic so the available multiplier units in FFAC are utilized to execute several instructions concurrently. Therefore, an efficient scheduling mechanism is also presented to ensure maximum utilization of the FFM units while producing the minimum latency simultaneously.

TABLE 1. List of acronyms with full description.

Acronym	Description
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
ECC	Elliptic Curve Cryptography
PKC	Public Key Cryptography
RSA	Rivest Shamir Adleman
ECDLP	Elliptic Curve Discrete Log Problem
NIST	National Institute of Standard and Technology
PD	Point Doublin
PA	Point Addition
SPA	Simple Power Analysis
ZADDU	Common-Z Point Addition
ZADDC	Common-Z Point Addition and Conjugate
ZDBLU	Common-Z point Doubling
ECPM	Elliptic Curve Point Multiplication
IM	Interleaved Multiplication
MM	Montgomery Multiplication
FFM	Finite Field Multiplication
FFAS	Finite Field Addition Subtraction
FFID	Finite Field Inversion Division
FFAC	Finite Field Arithmetic Core
NAF	Non-Adjacent Form
RSD	Redundant Sign Digit
RNS	Residue Number System

- For the functional verification, a software implementation of the proposed EC crypto processor is done by developing a customized C# library. Then, test vectors are generated and subsequently, the design is implemented using Verilog HDL where functional verification is done by providing the test vectors in the Xilinx ISim simulator. Finally, the synthesis and routing of the given ECPM design are done using Xilinx Vivado and ISE Design suit tools targeting different FPGA platforms.
- The proposed ECPM design has been compared with the existing designs where it shows improved performance results in terms of area-delay product with higher throughput and better efficiency. This improved performance of the proposed design demonstrates its viability to be used in many EC-based security protocols for several applications.

The remaining structure of the paper is as follows: In section II, an overview and mathematical foundations of ECC are provided. Section III details the proposed finite field multiplier algorithm with its efficient hardware architecture. In section IV, we present an overall ECPM design with an efficient and compact scheduling strategy. Finally, in section V, FPGA implementation and performance comparison of the proposed design are presented and the paper is concluded in section VI. It is worth elaborating that Table 1 lists the majority of acronyms used in this paper.

II. PRELIMINARIES

This section presents background knowledge about elliptic curves (ECs), group operations i.e., PA and PD, different EC points representation systems, common-Z (Co-Z) coordinates, and different techniques to compute ECPM operation. It also elaborates on our chosen EC PM algorithm and coordinates the system along with the number of low-level finite field arithmetic primitives.

Algorithm 1 $ZADDU_{(X,Y)}$

Input: $R_1 = (X_1, Y_1, Z)$ and $R_2 = (X_2, Y_2, Z)$
Output: $(R_3, R_1) = ZADDU_{(X,Y)}(R_1, R_2)$ where $R_3 = R_1 + R_2 = (X_3, Y_3, Z_3)$ and $R_1 = (\lambda^2 X_1, \lambda^3 Y_1, Z_3)$ with $Z_3 = \lambda Z$ for some $\lambda \neq 0$

$B = (X_1 - X_2)^2$;
$E_1 = X_1 U$; $E_2 = X_2 U$; $C = (Y_1 - Y_2)^2$;
$D = Y_1(E_1 - E_2)$; $X_3 = C - E_1 - E_2$;
$Y_3 = (Y_1 - Y_2)(E_1 - X_3) - D$;
$X_1 = E_1$; $Y_1 = D$;
$R_3 = (X_3, Y_3)$, $R_3 = (X_1, Y_1)$;
return (R_3, R_1)

Algorithm 2 $ZADDC_{(X,Y)}$

Input: $R_1 = (X_1, Y_1, Z)$ and $R_2 = (X_2, Y_2, Z)$
Output: $(R_3, \overline{R_3}) = ZADDC_{(X,Y)}(R_1, R_2)$where $R_3 = R_1 + R_2 = (X_3, Y_3, Z_3)$ and $\overline{R_3} = R_1 - R_2 = (\overline{X_3}, \overline{Y_3}, Z_3)$

$B = (X_1 - X_2)^2$;
$E_1 = X_1 U$; $E_2 = X_2 U$; $C = (Y_1 - Y_2)^2$;
$D = Y_1(E_1 - E_2)$; $X_3 = C - V_1 - V_2$;
$Y_3 = (Y_1 - Y_2)(E_1 - X_3) - D$;
$\overline{C} = (Y1 + Y2)^2$; $\overline{X_3} = \overline{C} - E_1 - E_2$;
$\overline{Y_3} = (Y_1 + Y_2)(E_1 - \overline{X_3}) - D$;
return $(R_3, \overline{R_3})$

A. ELLIPTIC CURVE AND GROUP OPERATIONS

An EC representation **E** in simplified Weierstrass form defined over a prime field GF(p) where $p > 3$ is given as

$$\mathbf{E} : y^2 = x^3 + \alpha x + \gamma \tag{1}$$

where $16(4\alpha^3 + 27\gamma^2) \neq 0$. An EC point with coordinates (x, y) is known as an affine representation, and any such point say $R(x, y)$ lies on an EC if it fulfills Eq. 1. Point addition (PA) and point doubling (PD) are the two main group operations which are required to execute the ECPM operation [6], [7]. PA operation of two points in affine coordinates say $R_1(x_1, y_1)$ and $R_2(x_2, y_2)$ generates a third point $R_3(x_3, y_3)$ on the chosen curve. The x_3 and y_3 coordinates of a resultant point are given as

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \quad modulo\ p$$
$$y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1 \quad modulo\ p \tag{2}$$

Similarly, PD operation is an addition of a point $R_1(x_1, y_1)$ with itself, this is given as

$$x_3 = (\frac{3x_1^2 + \alpha}{2y_1})^2 - 2x_1 \ \ modulo\ p$$
$$y_3 = (\frac{3x_1^2 + \alpha}{2y_1})(x_1 - x_3) - y_1 \ \ modulo\ p \qquad (3)$$

EC PA and PD primitves in affine representation are comprised of low-level finite field arithmetic primitives. These primitives include FFAS, FFM, and FFID and among these, FFID is the most time-critical operation which limits the performance of overall ECC-based cryptosystems. To perform a single PA operation in affine coordinates requires six FFAS, two FFM, and one FFID operation while the cost of PD is four FFAS, 2 FFM, and one FFID, hence involved in both PA and PD operations. However, fortunately, there are other coordinates that can facilitate FFID-free PA and PD execution at the cost of more FFM operations. The Jacobian coordinates space is one such system where affine point $R(x, y)$ is represented with triplet $R(XZ^{-2}, YZ^{-3}, Z)$. However, conversions from affine to Jacobian and vice versa are required at the start and completion of an ECPM operation. Meloni et al. [46] further extended the Jacobian coordinates to a new system based on the same Z coordinate. In this, PA and PD operations are calculated only using X, Y coordinates while the Z coordinate of the final point is recovered at the end of the ECPM operation. These techniques are named as common-Z addition ($ZADDU_{(X,Y)}$) and common-Z conjugate $ZADDC_{(X,Y)}$ which are given in algorithms 1 and 2, respectively. It is evident from the algorithms that the computational overhead of $ZADDU_{(X,Y)}$ is 6 FFM + 7 FFAS while $ZADDC_{(X,Y)}$ requires 8 FFM + 11 FFAS low-level finite field arithmetic operations.

B. ELLIPTIC CURVE POINT MULTIPLICATION

ECPM is the core primitive in the forming of any security services using ECC. It is the most computationally intensive operation where a base point R on an EC is multiplied by a scalar d to produce another point T on the curve. Mathematically, it can be represented as $T = dR$. A standard way to compute this primitive is known as double-and-add (DAA). The computational cost of ECPM using the DAA technique is $nPD + \lceil \frac{n}{2} \rceil PA$, where n is the bit length of scalar d. This computational cost can be further reduced to $nPD + \lceil \frac{n}{3} \rceil PA$ by representing scalar d in non-adjacent-form (NAF). However, due to the different computational complexity of PA and PD operations in DAA and NAF, these methods are not resistant to side-channel attacks where the aim is to reveal the scalar d using timing and power consumption information of the algorithm [47].

Another useful method to perform the ECPM operation that can also provide resistance against simple power and timing attacks is known as the Montgomery ladder [48]. In this method, both PA and PD operations are executed in each iteration independent of the scalar bit value. Here, we work with Co-Z coordinates so the Montgomery ladder technique using Co-Z arithmetic is given in algorithm 3. In step 1 of the algorithm, S_0 and S_1 registers are loaded with R and $2R$, respectively. Note that, $2R$ is a PD operation of R using the same Z coordinate known as $DBLU_{(X,Y)}$ which is given as:

$$X_2 = C^2 - 2I$$
$$Y_2 = C(I - X_3) - 8K$$
$$Z_2 = 2Y_1 \qquad (4)$$

where $I = 4X_1Y_1^2$, $C = 3X_1^2 + \alpha$, and $K = Y_1^4$. It is computed only once during the whole PM operation where the computational cost is 6 FFM + 13 FFAS operations. In steps 4 and 5, $ZADDC_{(X,Y)}$ and $ZADDU_{(X,Y)}$ are executed in series for $(n - 2)$ times and dictate the performance of the algorithm. Hence, one iteration of the loop requires 14 FFM and 18 FFAS operations. At the end of the loop, the Z coordinate recovery, and Jacobian to affine conversion are required which are demonstrated by steps 7 to 12 of the algorithm. Note that in steps 8 and 10 $ZADDC_{(X,Y)}$ and $ZADDU_{(X,Y)}$ are required with a combined computational cost of 14 FFM + 13 FAS operations. Whereas the computational costs of steps 9, 11, and 12 are 3 FFM +1 FFAS, 1 FFM, and 4 FFM respectively. It is worth mentioning that one FFID operation is also required in step 11 of the algorithm to perform only a single modular inversion required. Therefore the total cost for Z coordinate recovery and final Jacobian to affine conversion is 22 FFM + 14 FFAS + 1 FFID. However, these are computed only once during the PM operation and there is a good scope of parallelism so several time-critical FFM operations can be executed concurrently to reduce the latency.

Algorithm 3 Montgomery Ladder-Based Co-Z Arithmetic

Input: $R = (x_R, y_R), d = (d_n - 1, \ldots, d_0)$
Output: $T = d \times R$
1 $(S_0, S_1) = DBLU_{(X,Y)}(R)$;
2 **for** $(i = n - 2;\ to\ 1)$ **do**
3 $\quad a = d_i$;
4 $\quad (S_{1-a}, S_a) = ZADDC_{(X,Y)}(S_a, S_{1-a})$;
5 $\quad (S_a, S_{1-a}) = ZADDU_{(X,Y)}(S_{1-a}, S_a)$;
6 **end**
7 $a = d_0$;
8 $(S_{1-a}, S_a) = ZADDC_{(X,Y)}(S_a, S_{1-a})$;
9 $Z_M = x_R Y(S_0)(X(S_0) - X(S_1), \gamma = y_R X(S_a)$;
10 $(S_a, S_{1-a}) = ZADDU_{(X,Y)}(S_{1-a}, S_a)$;
11 $Z(T)^{-1} = \gamma / Z_M$;
12 $T(x_T, y_T) = ((Z(T)^{-1})^2 X(S_0), Z(T)^{-1})^3 Y(S_0))$;
13 **return** T

III. FINITE FIELD ARITHMETIC PRIMITIVES

As FFM is the most frequent and time-critical primitive among low-level finite field arithmetic components in the Jacobian coordinates. This section presents our novel modified IM algorithm by introducing several optimizations to

Algorithm 4 Radix-2 Interleaved Multiplication (R2IM)

Input: $a = \sum_{i=0}^{k-1} a_i \cdot 2^i, b = \sum_{i=0}^{k-1} b_i \cdot 2^i,$
$p = \sum_{i=0}^{k-1} m_i \cdot 2^i$
Output: $z = a \times b \bmod p$

1 $z \leftarrow 0$;
2 **for** $(i = k-1; i \geq 0; i \leftarrow i+1)$ **do**
3 $\quad z \leftarrow 2z \bmod p$;
4 $\quad$ if $b_i = 1$;
5 $\quad z \leftarrow (z + a) \bmod p$;
6 **end**
7 **return** z

reduce the total iteration count and remove redundant operations. Subsequently, an efficient hardware architecture to realize the proposed modified algorithm is presented. Moreover, this section also describes our strategy to perform FFAS operations by modifying the FFM to eliminate the need for a dedicated FFAS unit.

A. FINITE FIELD MULTIPLICATION

A standard radix-2 IM method for FFMUL is given in algorithm 4 [9]. It multiplies, reduces, and accumulates by repeatedly shifting (or doubling) and adding. The intermediate product is reduced by a modulus p after each doubling and addition step. It starts operating from the most significant bit (MSB) and computes double-reduce and then add-reduce primitives serially. The intermediate values are kept below the modulus p by reducing them. It requires n iterations for an n-bit FFM operation. If one iteration of this algorithm is completed in a single clock cycle then one n-bit FFM primitive is completed in total n clock cycles. Step 3 (modular doubling) is done by a single-bit left-shift ($2z$) followed by a n-bit subtraction ($2z-p$) primitive. Step 5 (modular addition) requires two n-bit adders with some 2:1 multiplexers. The first adder computes $z+a$. Then, if $z+a > p$, the second adder subtracts p to perform the modular reduction. Therefore, a radix-2 IM hardware architecture consists of three n-bit adders in addition to some multiplexers. These operations are executed serially so the critical path $T_{cp} = 3\ add + 5\ mux$.

B. PROPOSED PARALLEL IM ALGORITHM

Several pertinent modifications and associated hardware architectures [19], [20], [24], [30], [33], [35] have been proposed for the R2IM algorithm. References [19], [30], and [33] are executing single iteration in one clock cycle while [19], [24], and [35] are based on radix-4, where two consecutive bits of a multiplier are executed in a single clock cycle. Moreover, [20] and [24] reduced the data dependency among critical operations and executed them in parallel. However, these designs employed multiple processing units for the generation, reduction, and addition of possible partial products. A proposed novel modification in the standard R2IM algorithm is presented in algorithm 5. Two modifications based

Algorithm 5 Proposed Parallel FFM Algorithm

Input: $a = \sum_{i=0}^{k-1} a_i \cdot 2^i, b = \sum_{i=0}^{k-1} b_i \cdot 2^i,$
$p = \sum_{i=0}^{k-1} m_i \cdot 2^i$
Output: $z = a \times b \bmod p$

1 $z \leftarrow 0\ S_1 \leftarrow a, S_1 \leftarrow 2a \bmod p$
// Pre-computed value //

$$\mathrm{M} = \begin{cases} k+3, & \text{if } k \bmod 2 = 0, \text{ append two 0} \\ k+2, & \text{if } k \bmod 2 = 1, \text{ append single 0} \end{cases}$$

2 $M \leftarrow M + 1$ // append 0 to right of LSB of b //
3 **for** $(i = 0; i \leq M-2; i \leftarrow i+2)$ **do**
4 $\quad$ **switch** $(b_{(i+2:i)})$ **do**
5 $\quad\quad$ **when** $000 \mid 111 \Longrightarrow v \leftarrow 0$
6 $\quad\quad$ **when** $001 \mid 010 \mid 101 \mid 110 \Longrightarrow v \leftarrow S_1$
7 $\quad\quad$ **else** $\Longrightarrow v \leftarrow S_2$
8 $\quad$ **end**
$\quad$ `// Steps 9 and 10 are independent of step 11`
9 $\quad S_1 \leftarrow 2 \times S_2$ modulo p
10 $\quad S_2 \leftarrow 2 \times S_1$ modulo p
11 $\quad z \leftarrow z \pm v$ modulo p
12 **end**
13 **return** z

on Montgomery laddering (ML) [49] and Booth encoding (BE) [50] in combination with radix-4 are proposed. Note that our modifications to the algorithm involve constructing detailed dataflow graphs to examine the relationship between critical operations and eliminate redundant operations, thus introducing parallelism at the expense of lower hardware cost. The ML eliminates data dependency among critical operations whereas BE and radix-4 reduced the design space complexity and total iteration count respectively. The proposed parallel FFM algorithm scans a multiplier from LSB to MSB in contrast to R2IM which iterates from MSB to LSB. Overall, algorithm 5 is comprised of several steps where the main computation are performed in steps 9, 10, and 11. In step 1, registers c and S_1 are initialized with values 0 and a multiplicand a respectively, whereas a pre-computed value $2a$ mod p is loaded in the register S_2. Note that this is computed once before the start of an n-bit $\mathbb{F}_p$ MUL operation. In a simple radix-4 technique, two bits of a multiplier are executed in each iteration with possible partial products $\{0, 1, 2, 3\} \times a \bmod p$. In [9] and [20] four processing units (PUs) are deployed to ensure the readily availability of all possible values of the partial product in each iteration. Using BE we can save one PU because the possible partial products are $\{0, \pm 1, \pm 2\} \bmod p$. We further observed from the dataflow graphs that the value in a register S_2 in step 1 is always doubling of S_1 which is also evident from steps 9 and 10 of Algorithm 5. This enables us to save one more PU so in total we can save two PUs. The main computational steps 9 and 10 can be executed concurrently with step 11 and it is

TABLE 2. Design space complexity analysis of different FFMs.

Design	Design space	CP_{delay}	# clock cycles	Computational Time (CT)
Proposed FFM	4 n-bit add + 4 2-to-1 mux+ 3 n-bit reg	$2d_{add} + 2d_{mux}$	$(\lceil n/2 \rceil + 1)$	$(\lceil n/2 \rceil + 1) \times CP_{delay}$
Version I [24]	6 n-bit add+ 17 2-to-1 mux+ 4 n-bit reg	$3d_{add} + 4d_{mux}$	$(\lceil n/2 \rceil + 1)$	$(\lceil n/2 \rceil + 1) \times CP_{delay}$
Version II [24]	14 n-bit add + 28 2-to-1 mux+ 6 n-bit reg	$3d_{add} + 4d_{mux}$	$(\lceil n/3 \rceil + 4)$	$(\lceil n/3 \rceil + 4) \times CP_{delay}$
[9]	3 n-bit add + 2 2-to-1 mux+ 1 n-bitreg	$3d_{add} + 2d_{mux}$	$(n + 1)$	$(n + 1) \times CP_{delay}$
[28]	3 n-bit add + 2 2-to-1 mux+ 4 n-bit reg	$2d_{add} + 2d_{mux}$	$(n + 1)$	$(n + 1) \times CP_{delay}$
[33]	3 n-bit add + 3 2-to-1 mux+ 2 n-bit reg	$2d_{add} + 2d_{mux}$	$(n + 1)$	$(n + 1) \times CP_{delay}$
[29]	3 n-bit add + 5 2-to-1 mux+ 2 n-bit reg	$2d_{add} + 2d_{mux}$	$(n + 1)$	$(n + 1) \times CP_{delay}$
[19]	4 n-bit add + 4 2-to-1 mux+ 2 n-bit reg	$4d_{add} + 5d_{mux}$	$(\lceil n/2 \rceil + 4)$	$(\lceil n/2 \rceil + 4) \times CP_{delay}$
[30], [36]	3 n-bit add + 4 2-to-1 mux+ 3 n-bit reg	$3d_{add} + 4d_{mux}$	$(n + 1)$	$(n + 1) \times CP_{delay}$
[37]	8 n-bit add + 4 n-bit mux+ 5 n-bit reg	$3d_{add} + 3d_{mux}$	$(\lceil n/2 \rceil + 3)$	$(n/2 \rceil + 3) \times CP_{delay}$

worth mentioning that by adopting BE logic, FFAS operation is required in step 11 as compared to finite field addition operation as mentioned in step 5 of algorithm 4. Instead of scanning a single bit of the multiplier in the R2IM algorithm from left to right, algorithm 5 completes three bits (including one overlapping bit) of the multiplier from right to left and computes steps 9, and 10 in parallel with step 11 iteratively. The total number of rounds in algorithm 5 is $\lceil \frac{n}{2} \rceil$ where n is the bit size of a modulus p.

1) HARDWARE ARCHITECTURE

This section discusses a hardware realization of the proposed parallel FFM algorithm. It is shown in Fig. 1 where it is comprised of two processing units ($PU_{1,2}$), three n-bit registers (S_1, S_2, z) with some multiplexing logic, and a control unit. PU_1 is quadrupling modulo p which is further comprised of two identical double modulo p ($DBP_{1,2}$) units cascaded serially. PU_2 is $\mathbb{F}_p$ add/sub which can perform $\mathbb{F}_p$ add or $\mathbb{F}_p$ sub-operation based on the control signal generated by a BE logic unit. The PU_1 as shown in Fig.1(a) is a quadrupling mod p unit which is comprised of two identical doubling units DBP_1 and DBP_2. The internal structure of these doubling units is shown in Fig. 1(b) which consists of a left-shift of an input followed by a reduction modulo p. Then, these intermediate results are multiplexed, and the result ($2z \bmod p$) is available at the output. Note that the DBP_1 unit executes step 9 while step 10 of the algorithm is executed by DBP_2. The internal structure of PU_2 is shown in Fig. 1(c) and is responsible to execute step 11 which is the FFAS operation based on the output of BE unit. The architecture consists of two n-bit adders in addition to some multiplexing logic. In the case of the FF addition operation, the first adder performs the addition of operands followed by subtraction of a modulus p. Whereas, a subtraction of operands is performed in the first adder followed by the modulus addition for the FF subtraction operation. These intermediate values are multiplexed and available at the output after a single clock cycle.

2) DESIGN SPACE COMPLEXITY

This section presents the design space complexity analysis of the proposed parallel FFM architecture along with its comparison to other state-of-the-art IM-based designs. This type of analysis is very useful because it actually demonstrates platform-independent performance evaluation.

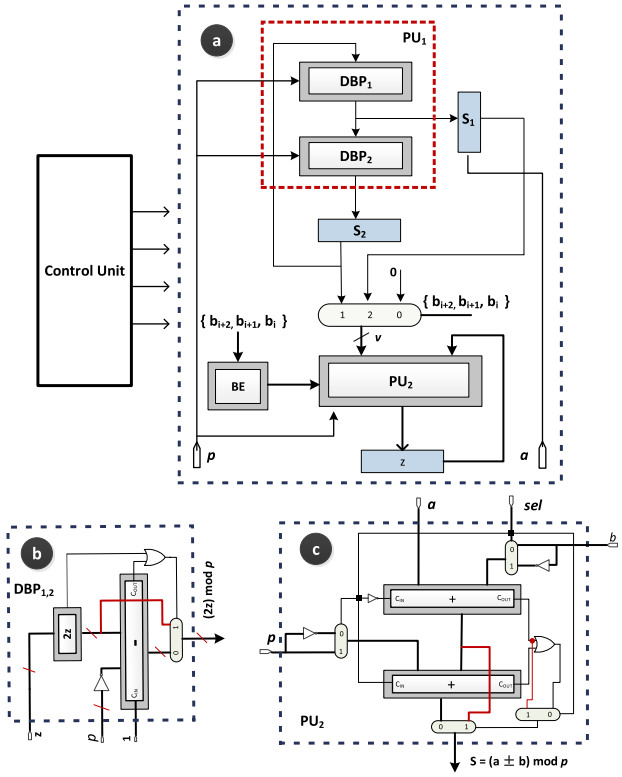

FIGURE 1. Proposed $\mathbb{F}_p$ multiplier architecture.

Moreover, this evaluation acts as a fair performance comparison tool because the same design when implemented on different platforms produces different results. We demonstrate this analysis on the basis of design space (resource consumption), critical path delay (CP_{delay}), number of clock cycles (cc) consumed, and the computational time (CT) taken by architecture to complete an n-bit finite field multiplication primitive. We evaluate several IM-based FFM designs on the basis of these design metrics as demonstrated in Table 2. In the table, adder, multiplexer, and register are represented with add, mux, and reg respectively. Moreover, combinational delays of an n-bit adder and a 2-to-1 multiplexer are denoted by d_{add} and d_{mux}, respectively. The proposed parallel FFM design has a design space complexity of 4 n-bit add + 4 2-to-1 mux+ 3 n-bit reg with a CP_{delay} of $2d_{add} + 2d_{mux}$.

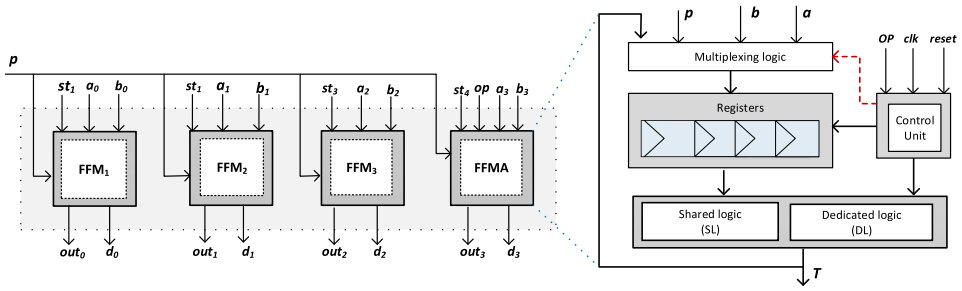

FIGURE 2. Proposed FFAC architecture.

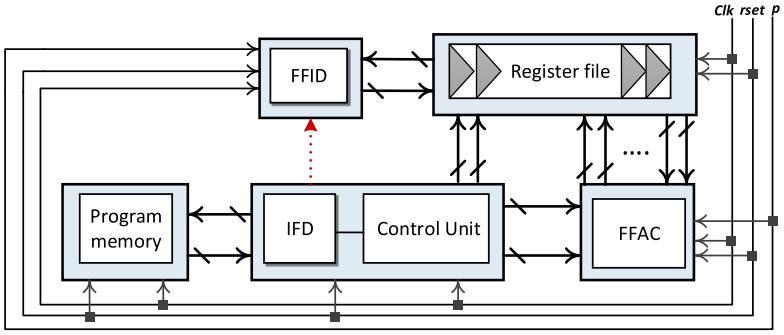

FIGURE 3. Proposed ECPM architecture.

It requires $(\lceil n/2 \rceil + 1)$ cc to compute an $n-$bit modular multiplication primitive with a total CT of $(\lceil n/2 \rceil + 1) \times \text{CP}_{delay}$. Note that the proposed design outperforms the other listed designs in terms of design space complexity and CP_{delay}. Designs reported in [9], [28], [30], and [33] require one fewer n-bit adder with a similar CP_{delay}, however, these take almost two times more clock cycles as compared to the proposed design. This is reflected in the actual FPGA implementation results demonstrated in section V.

IV. ELLIPTIC CURVE POINT MULTIPLIER

As the ECPM operation is composed of EC group operations that can be executed through a pre-determined sequence of low-level FF arithmetic operations. This section outlines our approach to constructing an FF Arithmetic Core (FFAC) capable of performing basic field operations like FFM and FFAS. We then demonstrate the design of a comprehensive ECPM hardware architecture using the proposed FFAC. Finally, this section also covers our chosen efficient scheduling approach to carry out these low-level field operations on the proposed FFAC

A. FINITE FIELD ARITHMETIC CORE

It is evident that hardware implementation of ECPM operation requires FF arithmetic primitives. For example, a single iteration of the adopted PM method (algorithm 3) requires 14 FFM + 18 FFAS operations because of serial execution of $ZADDC_{(X,Y)}$ and $ZADDU_{(X,Y)}$ operations given in algorithms 1 and 2, respectively. These 14 FFM and 18 FFAS operations have a good scope of parallelism so their execution can be speed up by employing multiple processing units. The majority of the available designs deployed dedicated units for these primitives which is not an efficient strategy because FFAS operations are much simpler as compared to FFM. Therefore, to further optimize the resource consumption, the internal FFAS unit (PU_2 in Fig. 1) in the proposed FFM architecture can be configured to perform FFAS operation. This unified FF unit is represented as FF multiplication and addition (FFMA). It is able to perform FFM or FFAS operations and is shown in Fig. 2. Note that at a time, only one of these operations can be executed by this architecture. It consists of dedicated logic (DL), shared logic (SL), register file, control unit, and multiplexing logic. DL consists of PU_1 and BE units which are only used when FFM operation is executed. Whereas, SL is comprised of PU_2 which is the FFAS unit and shared among all three operations. Two bits opcode (op) is used to execute a required task among three available operations (multiplication, addition, subtraction) and the register file is used to store the input operands and intermediate results. In the case of FFAS operation, the result is available at the output port (T) after a single clock cycle while FFM operation is done in $\lceil \frac{n}{2} \rceil$ clock cycles.

Thus, in the proposed FFAC, we deployed three copies of the FFM and a single copy of a unified FFMA unit. Therefore, it can process four FFM instructions or three FFM and one FFAS instruction, concurrently. It accepts eight input operands, a modulus p, and generates 4 independent results.

TABLE 3. Scheduling of ZADDC, ZADDU, and DBLU operations on the proposed FFAC.

Stage	#cc	FFMA	FFM_1	FFM_2	FFM_3
		ZADDC			
$stg_{1<1-(\lceil \frac{n}{2} \rceil+3)>}$	1	$r_1 = Y_1 - Y_2$	$r_2 = X_1 \times C$	$r_3 = X_2 \times C$	-
	2	$r_4 = Y_1 + Y_2$			$r_5 = r_1 \times r_1$
	3	$r_6 = r_4 \times r_4$			
$stg_{2<(\lceil \frac{n}{2} \rceil+4)-(n+10)>}$	$(\lceil \frac{n}{2} \rceil + 4)$	$r_7 = r_2 - r_3$	$r_{91} = Y_1 \times r_7$		
	$(\lceil \frac{n}{2} \rceil + 5)$	$r_8 = r_2 + r_3$			
	$(\lceil \frac{n}{2} \rceil + 6)$	$r_9^{(X3)} = r_8 - r_5$			
	$(\lceil \frac{n}{2} \rceil + 7)$	$r_{10} = r_2 - r_9$			
	$(\lceil \frac{n}{2} \rceil + 8)$	$r_{11}^{(X3)} = r_6 - r_8$		$r_{12} = r_1 \times r_{10}$	
	$(\lceil \frac{n}{2} \rceil + 9)$	$r_{14} = r_9 - r_{11}$			$r_{13} = r_{11} \times r_4$
	$(\lceil \frac{n}{2} \rceil + 10)$	$r_{15} = r_{14} \times r_{14}$			
$stg_{3<(n+11)-(n+11)>}$	$(n+11)$	$r_{16}^{(Y3)} = r_{91} + r_{12}$			
	$(n+12)$	$r_{17}^{(Y3)} = r_{13} + r_{91}$			
		ZADDU			
$stg_{1<1-(\lceil \frac{n}{2} \rceil+2)>}$	1	$r_1 = Y_1 - Y_2$	$r_2 = X_1 \times C$	$r_3 = X_2 \times C$	-
	2				$r_4 = r_1 \times r_1$
$stg_{2<(\lceil \frac{n}{2} \rceil+3)-(n+7)>}$	$(\lceil \frac{n}{2} \rceil + 3$	$r_5 = r_2 - r_3$	$r_7 = Y_1 \times r_5$		
	$(\lceil \frac{n}{2} \rceil + 4)$	$r_6 = r_2 + r_3$			
	$(\lceil \frac{n}{2} \rceil + 5)$	$r_8^{(X3)} = r_4 - r_6$			
	$(\lceil \frac{n}{2} \rceil + 6)$	$r_9 = r_2 - r_8$			
	$(\lceil \frac{n}{2} \rceil + 7)$			$r_{10} = r_9 \times r_9$	$r_{11} = r_9 \times r_1$
$stg3_{<(n+8)-(n+9)>}$	$(n+8)$	$r_{12}^{(Y3)} = r_{11} - r_7$			
		ZDBLU			
$stg1_{<1-(\lceil \frac{n}{2} \rceil+1)>}$	1		$r_1 = X_1 \times X_1$	$r_2 = Y_1 \times Y_1$	-
$stg_{2<(\lceil \frac{n}{2} \rceil+2)-(n+6)>}$	$(\lceil \frac{n}{2} \rceil + 2$	$r_3 = X_1 + r_2$	$r_4 = r_2 \times r_2$		
	$(\lceil \frac{n}{2} \rceil + 3)$	$r_5 = r_1 + r_1$		$r_6 = r_3 \times r_3$	
	$(\lceil \frac{n}{2} \rceil + 4)$	$r_7 = r_5 + r_1$			
	$(\lceil \frac{n}{2} \rceil + 5)$	$r_8 = \alpha + r_7$			
	$(\lceil \frac{n}{2} \rceil + 6)$				$r_9 = r_8 \times r_8$
$stg_{3<(n+7)-(3\lceil \frac{n}{2} \rceil+15)>}$	$(n+7)$	$r_{10} = r_4 - r_1$			
	$(n+8)$	$r_{11} = r_{10} - r_6$			
	$(n+9)$	$r_{12} = r_{11} + r_{11}$			
	$(n+10)$	$r_{13}^{(X3)} = r_9 - r_{12}$			
	$(n+11)$	$r_{14} = r_{13} - r_{11}$			
	$(n+12)$		$r_{15} = r_{14} \times r_{14}$	$r_{16} = r_{14} \times r_8$	-
	$(n+13)$	$r_{17} = r_4 + r_4$			
	$(n+14)$	$r_{18} = r_{17} + r_{17}$			
	$(n+15)$	$r_{19} = r_{18} + r_{18}$			
$stg_{4<(3\lceil \frac{n}{2} \rceil+16)-(3\lceil \frac{n}{2} \rceil+17)>}$	$3(\lceil \frac{n}{2} \rceil + 16)$	$r_{20}^{(Y3)} = r_{16} - r_{17}$			

$-, +, \times$ represent FFA, FFS, and FFM operations, respectively, r_i holds the operation output where i denotes the respective operation count. Note that, the ZDBLU operation is executed only once while ZADDC and ZADDU are executed $(n-1)$ times. This table shows only a single iteration of a loop (step 2) in algorithm 3.

B. ARCHITECTURE DESCRIPTION

Our proposed hardware architecture to compute the ECPM operation is shown in Fig. 3. It is comprised of FFAC, finite field inversion/division (FFID) unit, program memory, registers file, and a control unit. As one can eliminate the requirement of FFID operations from the EC group operations (PA and PD) by adopting projective coordinate systems. However, most of the existing ECC-based security protocols are developed for affine coordinates so we need to convert the final point from projective to affine space. This conversion requires FFID operation and here in our case, it costs 2 FFID + 4 FFM operations. This work adopted a binary version of the extended Euclidean algorithm (EEA) in the design of the FFID unit. A detailed hardware architecture and implementation guidelines of this method are provided in [51]. Our implementation of this method is able to compute a single modular inversion or division operation in $2n$ clock cycles, where n is the number of bits in the operands.

The proposed architecture is flexible to support any curve parameter and can work for general prime value $p \leq 521$-bit. Note that most of the existing designs support $p \leq 256$-bit. Another advantage of our design is to resist simple power and timing attacks [47] because of constant time low-level finite field arithmetic primitives and by the adoption of the Montgomery ladder technique. The FFAC can compute four FFM operations concurrently or three FFM in parallel to one FFAS operation while the FFID unit is utilized for the final conversion of EC PM from projective to affine space. The register file (RF) is comprised of several registers which are further categorized into general-purpose and dedicated register sets. Input point coordinates, prime modulus p, curve parameters, and scalar d are loaded in

the dedicated registers while intermediate values are stored and retrieved through general-purpose registers. As in algorithm 3, ZADDC (step 4) and ZADDU (step 5) are the main operations that are repeated $(n-1)$ times while DBLU (step 1) is required only once. Note that ZADDC and ZADDU are executed in series and in total these operations require 14 FFM and 18 FFAS lower field operations.

C. SCHEDULING AND LATENCY

The presented FFAC can perform either four FFM instructions or three FFM and a single FFAS instruction concurrently. Our efficient strategy to schedule these operations on the FFAC is demonstrated in Table 3. The execution flow of the *ZADDC* operation is further subdivided into three stages (stg_{1-3}) where in each stage multiple low-level finite field arithmetic instructions are scheduled on the proposed FFAC. In stg_1, two FFAS and four FFM instructions are scheduled on their respective execution units. At the first clock cycle, one FFAS and two FFM instructions are executed while the remaining one FFAS and FFM instructions are executed at the second and third clock cycles respectively. The result of the last FFM instruction (r_6) is available at $(\lceil \frac{n}{2} \rceil + 3)$ clock cycle and the execution units in FFAC are free to execute instructions in stg_2. The stg_2 is comprised of 6 FFAS and 4 FFM instructions out of these, r_7, r_8, and r_9 are scheduled at $(\lceil \frac{n}{2} \rceil + 4)$ and $(\lceil \frac{n}{2} \rceil + 5)$, respectively. Subsequently, the rest of the instructions in this stage from r_9 to r_{15} are scheduled on their respective execution units. The last instruction r_{15} is a FFM instruction which starts at $(\lceil \frac{n}{2} \rceil + 5)$ and result is available after $(n - 11)$ clock cycles. In the last stage (stg_3), only two FFAS instructions are scheduled hence this is completed in just two clock cycles. The result of the ZADDC operation is available after $(n - 12)$ clock cycles and now the FFAC is free to accommodate ZADDU operation. Similarly, we scheduled 6 FFAS and 7 FFAS operations in ZADDU in three stages (stg_{1-3}). In the stg_1, only a single FFAS and three FFM instructions are executed on their respective units while four FFAS, three FFM, and single FFAS instructions are part of stg_2 and stg_3, respectively. The result of the ZADDU operation is available after $(n + 8)$ clock cycles. As these operations are executed in series so a single loop iteration of the Montgomery ladder is completed in $(2n + 20)$ clock cycles. It is worth mentioning that the DBLU operation in step 1 of the algorithm is executed only once and it requires 7 FFM and 13 FFAS instructions. Our scheduling strategy in Table 2 shows that this can be completed in $3(\lceil \frac{n}{2} \rceil)+16$ clock cycles using four stages (stg_{1-4}). The two coordinates X, Y of the resultant point T in Jacobian space are generated by the *ZADDU* operation while the third coordinate of point $P(Z)$ is obtained using the output of the *ZADDU* operation. This Z coordinate recovery and final Jacobian to affine conversion require 22 FFM, 14 FFAS, and a single FFID operation. *ZADDC* and *ZADDU* operations in steps 8 and 10 of algorithm 3 are scheduled on the proposed FFAC in the same way as illustrated in Table 3. Then, 8 FFM operations are mapped to the four FFM multipliers in two stages and finally a single n-bit FFID operation is executed on the dedicated FFID unit in $2n$ clock cycles. Note that few FFM operations in the Z coordinate recovery and final conversion steps can be scheduled in parallel to the last iteration of the loop on the proposed FFAC unit. This is because in each stage at most three FFM units are occupied. Hence, the latency of some of these operations is hidden by *ZADDU* latency, however, $2n$ clock cycles are taken for the FFID instruction. After the Z coordinate recovery, the resultant point in Jacobian coordinates $T(X, Y, Z)$ needs to be transferred back to affine space $T(x, y)$. Let's say the latencies of $ZDBLU, ZADDC, ZADDU$, Z coordinate recovery and Jacobian to affine conversion are represented as $l_{ZDBLU}, l_{ZADDC}, l_{ZADDU}, l_Z, l_{J2A}$ respectively, then the overall latency L_{total} of the proposed design is given as follows:

$$\begin{aligned} L_{total} &= l_{ZDBLU} + l_{ZADDC} + l_{ZADDU} + l_Z + l_{J2A} \\ L_{total} &= 3\lceil \frac{n}{2} \rceil + 17 + (n-1)(2n+20) + 2n + 5n \\ L_{total} &= 2n^2 + 25n + 3\lceil \frac{n}{2} \rceil + 37 \end{aligned} \tag{5}$$

V. IMPLEMENTATION AND RESULTS

The FPGA implementation results of the given ECPM design are discussed in this section along with a performance comparison to the state of the art based on various design metrics. As FFM is the core computational unit so the implementation results of the given FFM design are presented first followed by the implementation, analysis, and comparison of the overall ECPM design. The proposed FFM and ECPM designs are written in Verilog-HDL, synthesized, routed, and placed using Xilinx Vivado and ISE Design Suite tools targeting Xilinx Virtex-7 (xc7vx690t), Kintex-7, and Virtex-6 (Xc6vlx760) FPGA platforms. A customized C# library is developed for functional verification and test vector generation. The simulation and verification steps were performed using Modelsim and ISim simulators. It is important to note that the ECPM architecture is programmable and flexible for varying operand sizes and security levels, with the ability to work for any prime modulus p value. The design is entirely based on Look-Up Tables (LUTs) where FPGA on-chip embedded blocks such as digital signal processing (DSP) and block RAMs (BRAMs) are not utilized. Thus, the implementation results are not dependent on the FPGA technology and associated tools, allowing it to be translated to any FPGA device or even to different ASIC nodes.

A. FFM IMPLEMENTATION RESULTS

FFM is the fundamental unit in the proposed ECPM architecture so it is also the main performance bottleneck. Thus the implementation results and the performance comparison of the proposed FFM architecture to the state of the art is very important and useful. Most of the existing modular multiplier designs report their results for 256-bit operand size targeting the Virtex-6 FPGA platform. So to have a fair performance evaluation, Table 3 demonstrates a 256-bit Virtex-6 FPGA implementation results of the proposed FFM along with

TABLE 4. Performance comparison of several FFMs on Virtex-6 FPGA platform.

Design	size	Freq. (MHz)	LUTs	DSPs	NLUT	Time (ns)	TP(Mbps)	E (TP/NLUT)	Method
Our	256	210	2910	0	2910	610	420	0.145	IM
	384	170	3871	0	3871	1120	342	0.088	IM
	521	139	4315	0	4315	1840	283	0.065	IM
[13]	256	143	1104	0	1104	1790	143	0.129	MM
[14]	256	441	22403	16	32371	133	1948	0.06	MM
[15]	256	81.98	6700	120	78340	353	725.21	0.009	MM
[16]	256	40.06	24000	256	176832	1248	205.13	0.001	MM
[17]	256	78.2	1680	0	1680	3270	78.29	0.047	SP
[18]	256	160	19750	0	19750	1325	193.21	0.01	SP
[19]	256	85.5	4606	0	4606	1500	170.6	0.037	IM
[20]	256	166	6300	0	6300	790	324.05	0.051	IM
[21]	256	219	13800	0	13800	598	428.09	0.031	MM
[22]	256	68.2	700	33	20401	350	731.43	0.036	MM
[23]	256	206	16900	108	81376	133	1954.2	0.024	MM
[24]	256	171	4231	0	4231	754	339	0.081	IM
[25]	256	97	1780	57	35809	680	376.47	0.011	MM
[26]	256	124	8400	42	33474	467	548.18	0.016	MM
[27]	256	205.7	22500	108	86976	142	1802.8	0.021	MM
[28]	256	161	1551	0	1551	1600	160	0.103	IM
[29]	256	174	3207	0	3207	1480	172.9	0.054	IM
[30]	256	128	1210	0	1210	1990	128	0.105	IM

Normalize LUTs (NLUT), Throughput (TP), Mega-bits per second (Mbps), Efficiency (E), Interleaved Multiplication (IM), Montgomery Multiplication (MM), Standard Prime (SP)

several other existing designs. Note that the proposed design is reconfigurable for any value and length of the modulus p. However, we only list the implementation results for three common key sizes (256, 384, 521). Most of the existing modular multiplier designs are reported for only 256-bit modulus length. Therefore, we compare our 256-bit FFM design to the other listed designs in Table 4. The proposed design takes 610 ns to perform a 256-bit modular multiplication operation, occupies 2910 FPGA look-up tables (LUTs), and attains a maximum frequency of 210 MHz.

Various serial and parallel IM and MM-based FFM architectures have been proposed with their FPGA implementations. For these designs, design objectives may be varied depending on the targeted applications. Note that some of the designs utilized DSPs blocks in addition to LUTs so we used a normalized LUTs (NLUTs) parameter. To calculate NLUTs, we add the number of LUTs to the DSP blocks multiplied by 623 LUTs. This DSP block to LUTs equivalent is taken by forcing the synthesis tool to use LUTs for a small multiplier as given in [23]. We compare our design to the others listed designs in Table 3 on the basis of NLUTs, computational time is taken in (ns), throughput (TP) in megabits per second (Mbps), and efficiency (E) terms. Note that, E is calculated by TP/NLUTs which provides a fair comparison tool because it accounts for both the employed NLUTs and computation time. The proposed FFM design completes a single 256-bit FFM operation in 610 ns, consumes 2901 LUTs, runs at a maximum frequency of 210 MHz, and produces a throughput of 420 Mbps. Designs reported in [19], [20], [24], [28], and [29] are based on IM, [17], [18] is based on SP, and the rest of the listed designs are based on the MM method. In terms of computational time, [13], [14], [15], [21], [22], [23], [26], [27] are 2.93×, 4.5×, 1.72 ×, 1.02×, 1.74×, 4.58×, 1.3×, 4.3× times better but consumes 2.6×, 9.17×, 22×, 3.9×, 5.77×, 23×, 9.5×, and 24.6× more NLUTs, respectively.

If we consider E, a more global and fair performance metric, then the presented design outperforms all the listed designs in Table 3. It has 1.12×, 1.23×, 1.3× times better E with 2.93×, 2.6× and 4.7× higher TP as compared with [13], [28], and [30](designs with the best E). Therefore, our proposal shows the best efficiency by optimizing both employed resources and computational time and produces higher TP, which ultimately resulted in the best area-delay optimized design. Note that, the FFM design has an integrated FFAS unit that is configured to perform FFAS operation in the FFMA unit to avoid the hardware cost of the dedicated FFAS unit in the proposed FFAC. This has further reduced the hardware resource requirements of the overall EC crypto processor design as evident by Table 5.

B. ECPM IMPLEMENTATION RESULTS

Table 5 elaborates on the implementation results of the proposed EC crypto processor over different FPGA platforms for different field lengths from 256 to 521 bits. We present the implementation results after the proposed design is synthesized, mapped, placed, and routed on Xilinx Vivado 2017 targeting Virtex-7 and Kintex-7 FPGA platforms. Note that, to demonstrate a fair performance evaluation, the design is also synthesized and implemented on the Virtex-6 FPGA platform using Xilinx ISE Design Suite. On the Virtex-7 FPGA platform, it completes a 256-bit PM operation in 0.7 ms by using 6.2K slices, takes 137K clock cycles, runs at 195 MHz frequency, and delivers a TP of 1428.6 operations per second (ops). Several ECC FPGA implementation proposals exist for various field sizes and different curves. Note that

TABLE 5. Performance comparison of several EC PMs on FPGA platforms.

Design	Platform	Size	Slices	LUTs	F. (MHz)	cc	Time (ms)	ADP	TP(ops)	E	Remarks
our	Virtex-7	256	6.2K	18.1K	195	137K	0.7	4.3	1428.6	0.231	Supports general prime $p \leq$521-bit, parallel IM based FFM, SPA resistant, Montgomery ladder with Co-Z coordinates
		384	7.6	24.8K	157	305K	1.94	14.7	515.5	0.067	
		521	8.5K	31.7K	131	556K	4.24	36	235.8	0.028	
	Kintex-7	256	6.3K	18.4K	197	137K	0.69	4.4	1440.6	0.221	
		384	7.7	25.1K	161	305K	1.89	14.5	529.5	0.068	
		521	8.7K	32K	134	556K	4.14	36	241.8	0.027	
	Virtex-6	256	6.8K	20.1K	182	137K	0.75	5.1	1333.8	0.196	
		384	7.9	26.9K	149	305K	2.05	16.2	487.8	0.061	
		521	8.8K	32.6K	123	556K	4.52	39.8	221	0.025	
[31]	Virtex-7	256	33K	108K	232	32.3K	0.158	5.2	6329	0.191	$p \leq$256-bit, MM based FFM, non-SPA resistant, Montgomery ladder
[32]	Virtex-7	256	6.4K	-	158	270K	1.70	10.9	588	0.091	$p \leq$256-bit, pre-calculation based FFM, SPA resistant, Montgomery ladder
[33]	Virtex-7	256	8.9K	-	177	2.62K	1.48	13.2	675.7	0.076	$p \leq$256-bit, serial IM-based FFM, non-SPA resistant, double and add
[34]	Virtex-7	256	24.2K	-	73	2.16K	2.96	71.6	337.8	0.014	$p \leq$256-bit, RNS-based FFM, non-SPA resistant, NAF
[21]	Virtex-6	256	65.6K	-	327	153K	0.47	30.8	2127.7	0.032	$p \leq$256-bit, MM-based FFM, non-SPA resistant, double-and-add
[20]	Virtex-6	256	10.1K	32.4K	144	208K	1.43	14.4	699.3	0.069	$p \leq$256-bit, serial IM-based FFM, non-SPA resistant, double-and-add
[19]	Virtex-6	256	7.9K	23.5K	95	22K	2.3	18.2	434.8	0.055	$p \leq$256-bit, serial radix-4 IM-based FFM, non-SPA resistant, double-and-add
[35]	Virtex-7	256	6.5K	-	104	199K	1.9	12.4	526.3	0.081	$p \leq$256-bit, twisted Edwards curve, radix-4 IM-based FFM, SPA resistant
[17]	Kintex-7	256	11.3K	-	121.5	397K	3.27	37	305.8	0.027	$p =$ 256-bit NIST curve, non-SPA resistant, double-and-add
[30]	Virtex-6	256	6.6K	-	76.3	300K	2.83	18.7	353.4	0.054	$p \leq$ 256-bit, radix-2 IM based FFM, non-SPA resistant, double-and-add
[36]	Virtex-7	256	6.4K	-	124	464K	3.73	23.9	268.1	0.042	$p \leq$ 256-bit, radix-4 IM based FFM, non-SPA resistant, double-and-add
[37]	Virtex-6	256	15.8K	44.3K	221	144.5K	0.65	10.3	1538.5	0.097	$p \leq$ 521-bit, RSD-MM based FFM, non-SPA resistant, NAF
		384	21.9	66.4K	221	315K	1.43	31.3	699.3	0.032	
		521	26.3K	90.4K	219	570K	2.63	68.4	384.4	0.015	

look-up-tables (LUTs), Throughput (TP), operation per second (ops), Efficiency (E), area-delay product (ADP), Frequency (F), Residue number system (RNS), Redundant-Signed-Digit (RSD), Non-Adjacent Form (NAF)

implementation results for a few other listed designs in the same table are available only for the Virtex-6 FPGA platform. Hence, to evaluate the efficiency of the proposed design fairly, we also provide the Virtex-6 FPGA implementation results of the proposed design. Moreover, implementation results for most of the listed designs are available for 256-bit prime modulus size. Therefore, we compare the performance of our 256-bit design with all the listed designs in Table 5.

In [31], a high-speed ECC architecture is proposed over a general prime field and the results are shown for up to 256-bit field size over different FPGA platforms. It proposed a new combined schoolbook and Kartsuba-based algorithm to achieve higher parallelism with low latency. On the Virtex-7 FPGA platform, it consumes 7281 slices (22,736 LUTs), 136 DSPs, and 15 BRAMs. As the proposed design only utilizes the LUTs so we calculated the NLUTs for [31]. On the same implementation platform, our design occupies 5.96 times lower LUTs, with lower ADP and higher efficiency. The other advantage of the proposed design over [31] is to support higher security levels up to 521-bit. Hoe et al. in [32] presented a lightweight ECPM architecture over the general prime field. On the low-level finite arithmetic, a pre-calculation strategy is adopted to optimize the critical path delay and resource consumption in the development of FFM and divider units. On the top level, the Montgomery ladder with Jacobian coordinates is adopted. The Implementation results of up to 256-bit are presented for different Xilinx FPGA platforms. On Virtex-7, it completes a single 256-bit ECPM operation in 1.70 *ms* in 270K clock cycles at 158 MHz frequency. It occupies 6.4K FPGA slices and delivers a TP of 588 *ops*. Our design is 2.42× faster, consumes almost similar resources, has 2.42× lower ADP, 2.53× higher efficiency, and delivers 2.42× higher TP as compared to [32]. An area-efficient high-speed ECC implementation over

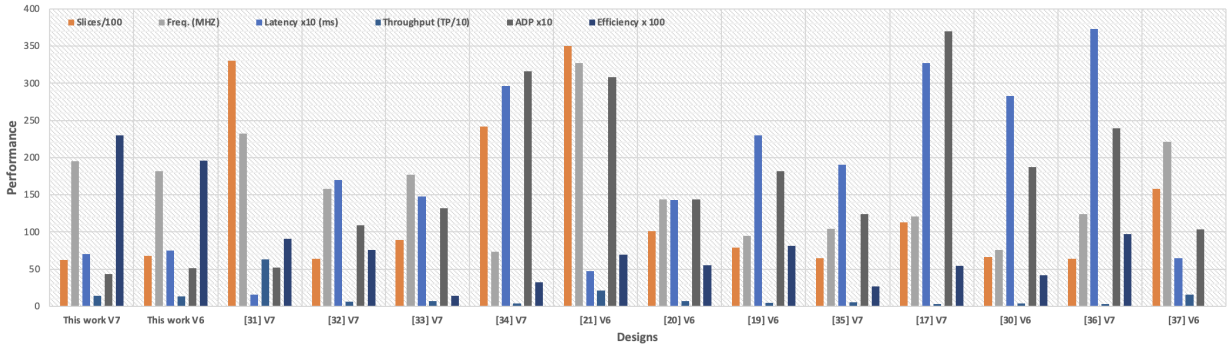

FIGURE 4. Performance evaluation of FPGA implementation of different 256-bit EC PM designs.

FPGA is presented in [33]. This is developed over a general prime field where it can support any value of a prime modulus p. However, implementation results are only demonstrated up to 256-bit after being implemented on the Virtex-7 FPGA platform. It is based on a simple radix-2 IM algorithm where it takes n clock cycles to perform an n-bit FFM operation and the total latency of the design is $4n^2 + 2n - 6$. It completes one 256-bit ECPM operation in 1.48 *ms*, delivering a TP of 675.7 *ops* and consumes 8.9K FPGA slices. The proposed design outperforms it in all aspects: it is 2.11× faster, consumes 1.43× lower FPGA slices, produces 2.11× higher TP, 3.06× lower ADP, and 3× higher efficiency. A similar 256-bit ECPM design is reported in [35] where twisted Edwards curves using unified point addition techniques are utilized. A modified radix-4 IM-based FFM unit is developed to perform low-level finite field multiplication operations while projective coordinates are adopted at the system level. To complete a single ECPM operation on the Xilinx Virtex-7 FPGA platform, it takes 1.9 *ms* running at 104 MHz, 199K clock cycles and requires 6.5K slices. It is 2.7× slower with 1.04× more FPGA slices as compared to the proposed design. Moreover, our design delivers 2.7× higher TP, 2.8× lower ADP, and 2.9× higher E. Designs reported in [19] and [20] are based on serial and parallel radix-4 IM algorithm. Both these designs are developed using projective coordinates and implementation results are reported for Xilinx Virtex-6 FPGA for field sizes up to 256-bit. Our design is 3.3× faster, consumes 1.16× lower FPGA slices, produces 3.07× higher TP, 3.56× lower ADP, and 4.2× higher E than [20]. Whereas as compared to [19], it is 1.9× faster, consumes 1.48× lower FPGA slices, produces 1.9× higher TP, 2.82× lower ADP, and 2.84× higher E. In [17], a high-performance ECC implementation is reported over a specific prime curve of 256-bit proposed by NIST. This design lacks flexibility and despite the generic nature of our design, it outperforms [17] in latency, slice consumption, TP, ADP, and E by 4.67×, 1.8×, 4.7×, 8.6×, and 8.5×, respectively. Kudithi et al. in [30] and [36] proposed efficient EC cryptography processors for IoT security applications over the general prime field. For low-level field multiplication, a finite field multiplier is developed using the IM algorithm in [30] whereas the MM technique is adopted in [36]. [30] is 4× slower, consumes 1.1× more slices, delivers 3.8× lower TP, 3.7× higher ADP with 3.7× lower E whereas, [36] is 5× slower with almost similar slice consumption, 5.3× lower TP, 2.4× lower ADP and 2.38× lower E as compared to the proposed design.

We summarise the performance evaluation of all the listed designs in Table 4 for 256-bit operand sizes and it is shown in Fig. 4, where normalized values are used. Note that these designs are implemented on different Virtex (V) FPGA platforms so the underlying implementation platform is also mentioned in the figure. It is also worth mentioning that a design having a lower value for ADP and a higher E value is considered to be better optimized for latency, area consumption, and TP. The given design outperforms all the listed designs in terms of ADP and E which means that it is better optimized for latency, resource consumption, and TP. Only [21] and [31] have lower latency than the proposed design, however, these have higher ADP and lower E values as compared to the proposed design. Hence, our design delivers the best ADP and E values as evident from the figure. It is worth noticing that most of the listed designs in Table 5 are not resilient to timing and simple power analysis attacks (SPA) [47]. As the given design adopted the Montgomery ladder method for ECPM which provides inherent resistance against these attacks. This is due to the computation of PD and PA operations at each iteration irrespective of the scalar bit d_i In addition to algorithmic level countermeasures, at the circuit level, the proposed low-level ECC arithmetic modules such as the FMM, FFMA, and FFID are developed using a balanced implementation and produce outputs in a constant time. Furthermore, the presented scheduling strategy is organized in a fixed number of stages resulting in constant time execution of *ZADDU*, *ZADDC*, and *ZDBLU* operations as evident from Table 3. Therefore, the proposed ECPM design is robust against timing and SPA attacks which is missing in most of the listed designs.

Lastly, the power consumption estimates of the proposed 256-bit ECC architecture are generated by utilizing XPower, which is a customized power estimation tool developed by

Xilinx. The power consumption of the given design on the Virtex-6 FPGA platform while attaining at a maximum frequency of 182 MHz is estimated at 190 *m*W. It is worth mentioning that these power figures are obtained by considering default values for temperature and voltage parameters i.e., the temperature is 25°C and different voltages such as Vccint and VCCAUX at 1 V while VCCO is set to 2.5 V. It is worth noticing that similar power consumption values are not available for all the existing designs mentioned in Table 5 and figure 4. Therefore, it is best optimized for latency, hardware resource consumption, and TP and it can be suited for ECC-based cryptosystems in many resource-limited environments.

VI. CONCLUSION

This paper introduced a novel area-delay optimized finite field multiplier where hardware resource consumption is reduced by eliminating redundant operations and the latency is minimized through a novel parallelism approach. The proposed multiplier is then utilized to develop an area-time optimized elliptic curve cryptographic processor using the Montgomery ladder technique with common-Z coordinates at the system level. Moreover, an efficient scheduling strategy to schedule low-level finite field arithmetic primitives is presented. The cryptographic processor is synthesized, placed, and routed on various FPGA platforms for operand lengths up to 521 bits. The implementation results showed that it is a highly efficient design in terms of latency, area-delay product, throughput, and efficiency. Additionally, the design is robust against simple power analysis attacks, enabling it as suitable option for use in various elliptic curve cryptography-based security protocols for applications where both latency and resource requirements are critical.

A. FUTURE WORKS

REFERENCES

[1] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1985, pp. 417–426.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[4] N. Smart, M. Abdalla, E. Bjørstad, C. Cid, and B. Gierlichs, "Algorithms, key size and protocols report," ECRYPT CSA, U.K., Tech. Rep. H2020-ICT-2014–Project 645421, 2018.

[5] E. Barker and W. C. Barker, "Recommendation for key management—Part 2: Best practices for key management organization," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 800-500, 2018.

[6] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York, NY, USA: Springer, 2006.

[7] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Des., Codes Cryptogr.*, vol. 19, nos. 2–3, pp. 173–193, Mar. 2000.

[8] P. Montgomery, "Modular multiplication without trial division," *Math. Comput.*, vol. 44, no. 170, pp. 519–521, 1985.

[9] G. R. Blakely, "A computer algorithm for calculating the product AB modulo M," *IEEE Trans. Comput.*, vol. C-32, no. 5, pp. 497–500, May 1983.

[10] E. Barker and Q. Dang, "NIST special publication 800–57. Part 1, revision 4," NIST, Gaithersburg, MD, USA, Tech. Rep. 800-500, 2016.

[11] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Boca Raton, FL, USA: CRC Press, 2005.

[12] D. J. Bernstein and T. Lange. (2013). *Safecurves: Choosing Safe Curves for Elliptic-Curve Cryptography*. [Online]. Available: https://safecurves.cr.yp.to

[13] A. A. H. Abd-Elkader, M. Rashdan, E.-S. A. M. Hasaneen, and H. F. A. Hamed, "Efficient implementation of Montgomery modular multiplier on FPGA," *Comput., Electr. Eng.*, vol. 97, Jan. 2022, Art. no. 107585.

[14] F. Pajuelo-Holguera, J. M. Granado-Criado, and J. A. Gómez-Pulido, "Fast Montgomery modular multiplier using FPGAs," *IEEE Embedded Syst. Lett.*, vol. 14, no. 1, pp. 19–22, Mar. 2022.

[15] S. Khan, K. Javeed, and Y. A. Shah, "High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications," *Microprocessors Microsyst.*, vol. 62, pp. 91–101, Oct. 2018.

[16] Y. Yang, C. Wu, Z. Li, and J. Yang, "Efficient FPGA implementation of modular multiplication based on Montgomery algorithm," *Microprocessors Microsyst.*, vol. 47, pp. 209–215, Nov. 2016.

[17] M. S. Hossain, Y. Kong, E. Saeedi, and N. C. Vayalil, "High-performance elliptic curve cryptography processor over NIST prime fields," *IET Comput., Digit. Techn.*, vol. 11, no. 1, pp. 33–42, Jan. 2017.

[18] H. Marzouqi, M. Al-Qutayri, K. Salah, D. Schinianakis, and T. Stouraitis, "A high-speed FPGA implementation of an RSD-based ECC processor," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 151–164, Jan. 2016.

[19] K. Javeed, X. Wang, and M. Scott, "High performance hardware support for elliptic curve cryptography over general prime field," *Microprocessors Microsyst.*, vol. 51, pp. 331–342, Jun. 2017.

[20] K. Javeed and X. Wang, "Low latency flexible FPGA implementation of point multiplication on elliptic curves over GF*p*," *Int. J. Circuit Theory Appl.*, vol. 45, no. 2, pp. 214–228, Feb. 2017.

[21] Y. A. Shah, K. Javeed, S. Azmat, and X. Wang, "A high-speed RSD-based flexible ECC processor for arbitrary curves over general prime field," *Int. J. Circuit Theory Appl.*, vol. 46, no. 10, pp. 1858–1878, Oct. 2018.

[22] M. Morales-Sandoval and A. Díaz-Pérez, "Scalable GF*p* Montgomery multiplier based on a digit–digit computation approach," *IET Comput., Digit. Techn.*, vol. 10, no. 3, pp. 102–109, May 2016.

[23] J. Ding and S. Li, "Broken-Karatsuba multiplication and its application to Montgomery modular multiplication," in *Proc. IEEE 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–4.

[24] K. Javeed, K. Saeed, and D. Gregg, "High-speed parallel reconfigurable F*p* multipliers for elliptic curve cryptography applications," *Int. J. Circuit Theory Appl.*, vol. 50, no. 4, pp. 1160–1173, Apr. 2022.

[25] A. Mrabet, N. El-Mrabet, R. Lashermes, J. B. Rigaud, B. Bouallegue, S. Mesnager, and M. Machhout, "A systolic hardware architectures of Montgomery modular multiplication for public key cryptosystems," Cryptol. ePrint Arch., Tech. Rep., 2016, vol. 487.

[26] K. Bigou and A. Tisserand, "Single base modular multiplication for efficient hardware RNS implementations of ECC," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Springer, Sep. 2015, pp. 123–140.

[27] X. Yan, G. Wu, D. Wu, F. Zheng, and X. Xie, "An implementation of Montgomery modular multiplication on FPGAs," in *Proc. IEEE Int. Conf. Inf. Sci. Cloud Comput.*, Dec. 2013, pp. 32–38.

[28] M. Islam, S. Hossain, Shahjalal, K. Hasan, and Y. M. Jang, "Area-time efficient hardware implementation of modular multiplication for elliptic curve cryptography," *IEEE Access*, vol. 8, pp. 73898–73906, 2020.

[29] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Secure dual-core cryptoprocessor for pairings over barreto-naehrig curves on FPGA platform," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 3, pp. 434–442, Mar. 2013.

[30] T. Kudithi and R. Sakthivel, "An efficient hardware implementation of the elliptic curve cryptographic processor over prime field," *Int. J. Circuit Theory Appl.*, vol. 48, no. 8, pp. 1256–1273, Aug. 2020.

[31] A. M. Awaludin, H. T. Larasati, and H. Kim, "High-speed and unified ECC processor for generic Weierstrass curves over GF*p* on FPGA," *Sensors*, vol. 21, no. 4, p. 1451, Feb. 2021.

[32] Y. Hao, S. Zhong, M. Ma, R. Jiang, S. Huang, J. Zhang, and W. Wang, "Lightweight architecture for elliptic curve scalar multiplication over prime field," *Electronics*, vol. 11, no. 14, p. 2234, Jul. 2022.

[33] Md. M. Islam, M. S. Hossain, M. K. Hasan, M. Shahjalal, and Y. M. Jang, "FPGA implementation of high-speed area-efficient processor for elliptic curve point multiplication over prime field," *IEEE Access*, vol. 7, pp. 178811–178826, 2019.

[34] S. Asif, M. S. Hossain, and Y. Kong, "High-throughput multi-key elliptic curve cryptosystem based on residue number system," *IET Comput., Digit. Techn.*, vol. 11, no. 5, pp. 165–172, Sep. 2017.
[35] M. M. Islam, M. S. Hossain, M. K. Hasan, M. Shahjalal, and Y. M. Jang, "Design and implementation of high-performance ECC processor with unified point addition on twisted Edwards curve," *Sensors*, vol. 20, no. 18, p. 5148, Sep. 2020.
[36] T. Kudithi and R. Sakthivel, "High-performance ECC processor architecture design for IoT security applications," *J. Supercomput.*, vol. 75, no. 1, pp. 447–474, Jan. 2019.
[37] Y. A. Shah, K. Javeed, S. Azmat, and X. Wang, "Redundant-signed-digit-based high speed elliptic curve cryptographic processor," *J. Circuits, Syst. Comput.*, vol. 28, no. 5, May 2019, Art. no. 1950081.
[38] Y.-M. Kuo, F. Garcia-Herrero, O. Ruano, and J. A. Maestro, "Flexible and area-efficient Galois field arithmetic logic unit for soft-core processors," *Comput. Electr. Eng.*, vol. 99, Apr. 2022, Art. no. 107759.
[39] H. Edwards, "A normal form for elliptic curves," *Bull. Amer. Math. Soc.*, vol. 44, no. 3, pp. 393–422, Jul. 2007.
[40] C. Costello and B. Smith, "Montgomery curves and their arithmetic," *J. Cryptograph. Eng.*, vol. 8, no. 3, pp. 227–240, Sep. 2018.
[41] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters, "Twisted Edwards curves," in *Proc. Int. Conf. Cryptol. Afr.* Berlin, Germany: Springer, 2008, pp. 389–405.
[42] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Elliptic curve lightweight cryptography: A survey," *IEEE Access*, vol. 6, pp. 72514–72550, 2018.
[43] B. Rashidi, "A survey on hardware implementations of elliptic curve cryptosystems," 2017, *arXiv:1710.08336*.
[44] Z. U. A. Khan and M. Benaissa, "High-speed and low-latency ECC processor implementation over GF(2^m) on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 1, pp. 165–176, Jan. 2016.
[45] M. Arif, O. S. Sonbul, M. Rashid, M. Murad, and M. H. Sinky, "A unified point multiplication architecture of weierstrass, Edward and huff elliptic curves on FPGA," *Appl. Sci.*, vol. 13, no. 7, p. 4194, Mar. 2023.
[46] N. Meloni, "New point addition formulae for ECC applications," in *Arithmetic of Finite Fields*. Madrid, Spain: Springer, Jun. 2007, pp. 189–201.
[47] P. C. Kocher, "Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems," in *Proc. Annu. Int. Cryptol. Conf.* Berlin, Germany: Springer, 1996, pp. 104–113.
[48] R. R. Goundar, M. Joye, A. Miyaji, M. Rivain, and A. Venelli, "Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic," *J. Cryptograph. Eng.*, vol. 1, no. 2, pp. 161–176, Aug. 2011.
[49] M. Joye and S.-M. Yen, "The Montgomery powering ladder," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Springer, 2002, pp. 291–302.
[50] A. D. Booth, "A signed binary multiplication technique," *Quart. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236–240, 1951.
[51] K. Javeed, "Efficient hardware architecture for scalar multiplications on elliptic curves over prime field," Ph.D. thesis, School Electron. Eng., Dublin City Univ., Dublin, Ireland, 2016.

KHALID JAVEED (Member, IEEE) received the B.S. degree from the COMSATS Institute of Information Technology, Abbottabad, Pakistan, the M.S. degree from Linkoping University, Sweden, in 2010, and the Ph.D. degree from the School of Electronic Engineering, Dublin City University, Dublin, Ireland, in 2016. He has been a Postdoctoral Researcher with the School of Computer Science, Trinity College Dublin, Ireland. Currently, he is an Assistant Professor with the Computer Engineering Department, University of Sharjah, United Arab Emirates. His research interests include computer architecture, finite field arithmetic, and VLSI architectures for cryptographic primitives.

ALI EL-MOURSY (Senior Member, IEEE) received the Ph.D. degree in high-performance computer architecture from the University of Rochester, Rochester, NY, USA, in 2005. He was with the Software Solution Group, Intel Corporation, CA, USA, until 2007. In 2007, he joined the Electronics Research Institute, Giza, Egypt. He has also participated with the IBM Cairo Technology Development Center, Egypt, as a Visiting Research Scientist, from February 2007 to January 2010. In September 2010, he joined as an Assistant Professor with the ECE Department, University of Sharjah, Sharjah, United Arab Emirates. From January 2017 and January 2023, he was promoted to an associate professor and a full professor. His research interests include high-performance computer architecture, multi-core multi-threaded micro-architecture, power-aware micro-architecture, simulation and the modeling of architecture performance and power, workload profiling and characterization, parallel programming, high-performance computing, parallel computing, the IoT architecture, and cloud computing.

DAVID GREGG received the Ph.D. degree in computer science from Technical University (TU) Wien, in 2001. He is currently a Professor in computer science with Trinity College Dublin. He was elected as a fellow of Trinity College Dublin, in 2007. His research interests include compiler optimization, algorithm design, computer arithmetic, and the challenges of adapting deep neural network technologies to resource-constrained embedded systems.