IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 October 2024, accepted 7 November 2024, date of publication 11 November 2024, date of current version 20 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3495995

RESEARCH ARTICLE

Efficient ECC Processor Designs for IoT Using Edwards Curves and Exploiting FPGA Embedded Components

OSAMA AL-KHALEEL[1], SELÇUK BAKTIR[2], (Senior Member, IEEE), MOHAMMAD AL-KHALEEL[3,4], AND ALPTEKİN KÜPÇÜ[5], (Senior Member, IEEE)

[1]Computer Engineering Department, Jordan University of Science and Technology, Irbid 22110, Jordan
[2]College of Engineering and Technology, American University of the Middle East, Egaila 54200, Kuwait
[3]Department of Mathematics, Khalifa University of Science and Technology, Abu Dhabi 127788, United Arab Emirates
[4]Department of Mathematics, Yarmouk University, Irbid 21163, Jordan
[5]Computer Science and Engineering, Koç University, Rumeli Feneri Yolu, Sarıyer 34450, İstanbul, Türkiye

Corresponding authors: Selçuk Baktır (selcuk.baktir@aum.edu.kw) and Mohammad Al-Khaleel (mohammad.alkhaleel@ku.ac.ae)

The work of Osama Al-Khaleel was supported by the AMD-XILINX University Program to Jordan University of Science and Technology (JUST). The work of Alptekin Küpçü was supported by TÜBİTAK, the Scientific and Technological Research Council of Türkiye, under Project 119E088.

ABSTRACT Elliptic Curve Cryptography (ECC) provides high security with shorter key sizes and higher performance. Internet of Things (IoT) is the interconnected network where a vast number of connected devices require high security to ensure data integrity and privacy. Field programmable gate arrays (FPGAs) are configurable hardware fabrics enhancing the performance of applications. We propose two FPGA-based high performance and lightweight ECC processors for IoT devices. One of the ECC processor is serial with less area and the other is parallel with higher speed. The proposed ECC processors use Edwards curves and perform frequency domain multiplication utilizing FPGA-embedded digital signal processors (DSPs) or lookup tables. The proposed ECC processors work over prime characteristic finite fields of the form $GF((2^n - 1)^n)$. Three such prime characteristic finite fields are investigated for each processor: $n = 13,\ 17$ and 19, providing the key lengths of 169, 289 and 361 bits, respectively. Synthesis results, targeting different FPGAs, show that the proposed ECC processors outperform similar existing ECC processors in terms of area and speed. For example, over Virtex5 FPGA, compared to the lowest area ECC implementations in the literature over the same finite fields, our serial ECC processor without DSPs occupies 19.07% to 26.06% fewer slices while being 2.17× to 3.57× faster. Compared to the fastest ECC implementations, our parallel ECC processor without DSPs occupies 20.42% to 27.38% fewer slices while being 1.76× to 2.22× faster, and our parallel ECC processor that use DSPs is 1.93× to 2.50× faster.

INDEX TERMS Elliptic curve cryptography, edwards curve, embedded multiplier, FPGA, discrete Fourier transform, modular multiplication.

I. INTRODUCTION

In recent years, there has been a significant increase in the use of smart Internet of Things (IoT) devices directly connected to the Internet across various applications that involve communication between senders and receivers. These applications span diverse areas such as smart buildings and cities, healthcare, wearable devices, intelligent transportation systems, industrial management, environment, military, agriculture, food supply chains, and energy management (see e.g., [1], [2] and references therein). This has created a crucial need for robust and efficient mechanisms to ensure a high level of security in those applications.

IoT devices usually include real-time embedded systems for security purposes. These embedded systems are used to integrate hardware and software based security solutions,

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

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

utilizing cryptographic mechanisms. For instance, the authenticity and integrity of software updates in IoT devices need to be verified. To this end, the use of a public key cryptographic algorithm, such as elliptic curve cryptography (ECC), is needed to generate public key certificates and digital signatures [3]. The speed and energy efficiency of these mechanisms are therefore critical [4]. However, security solutions often overlook the resource constraints faced by embedded systems in IoT devices, including memory, area, processing power, and energy consumption [5], [6]. To overcome these limitations, lightweight cryptographic mechanisms have been adopted [4], [7]. Achieving efficiency requires striking the right balance between the computational complexity and the security level of the chosen cryptographic algorithm [5]. Among hardware- [8] and software-based realizations [9], hardware realizations outperform software realizations in terms of speed and energy efficiency, making them better-suited for the IoT setting [8].

A. BACKGROUND

The origins of public key cryptosystems can be traced back to 1976 when Whitfield Diffie and Martin Hellman [10] introduced certain techniques for their development. The first practical general-purpose public key cryptographic algorithm, known as RSA, was proposed in 1977 by Rivest et al. [11]. This system's security is based on the integer factorization problem. Another notable realization is elliptic curve cryptography (ECC), which was independently introduced by Victor Miller [12] and Neal Koblitz [13]. ECC has been in practical use since 2000 [14]. The security of ECC-based schemes relies on a different mathematical problem known as the elliptic curve discrete logarithm problem (ECDLP).

The use of ECC schemes has gained widespread popularity and adoption in numerous applications. These schemes have the potential to achieve the desired security level with smaller keys and faster running time, compared to RSA-based alternatives (for more information, refer to the recent work by [15], [16], [17] and the references cited therein). This is primarily due to the inherent difficulty of the underlying mathematical problem, specifically the ECDLP, compared to the hardness of the integer factorization problem for RSA or the finite field discrete logarithm problem for the Diffie-Hellman key-exchange scheme. It is shown that a 160-bit long key in ECC offers security comparable to or exceeding a 1024-bit long key in RSA [18], [19]. For a comprehensive background on finite field arithmetic and elliptic curve cryptography, we refer the reader to various books such as [20] and [21].

Among the various groups of curves specifically introduced for ECC system applications, the Edwards curves stand out. Initially presented in [22], these curves have garnered considerable popularity for their widespread adoption in ECC systems. In their paper [23], Bernstein and Lange expanded on Edwards' earlier work [22] to adapt it for practical cryptographic implementations. The work of Edwards demonstrates that such curves enable a unified formula that replaces the commonly used point operations of addition and doubling. Accordingly, their remarkable advantage lies in enabling a significantly more complete and effective group law. This unified approach is not only more elegant by employing a single formula for all inputs but also offers advantages in terms of side-channel security and simplifying implementations.

Specifically, adopting a balanced approach in implementations, such as side-channel atomicity [24], is known to aid in preventing simple side-channel attacks. To further enhance defense against side-channel attacks, various methods for balanced scalar multiplication algorithms, including the Montgomery ladder and side-channel atomicity, have been proposed. In particular, conducting modular multiplication in the frequency domain proves beneficial. This arises from the fact that the finite field multiplication and squaring operations are carried out identically in the frequency domain, resulting in equivalent complexity, which contributes to better safeguarding against side-channel attacks.

Notably, recent research has extensively explored various ECC architectures, as evidenced by several studies, see for instance, [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35]. However, our current work distinguishes itself by focusing on Edward's curves, renowned for their exceptional properties. Numerous studies [14], [36], [37], [38], [39] have proven that Edwards curves offer a combination of excellent overall security and commendable performance. By employing the complete addition formula, these curves bolster their resilience against attacks and minimize potential vulnerabilities [40]. Moreover, research presented, e.g., in [41], [42] highlight that Edwards curves are remarkably secure against Zero-Value Point (ZVP) and Simple Power Analysis (SPA) attacks. Additionally, our approach involves operating in the frequency domain, a strategy that has demonstrated significant advantages as explained in the sequel. By integrating these two innovations, Edward's curves and frequency domain processing, we aim to advance the frontier of ECC architectures and contribute novel insights to this dynamic field of study.

Around 10% of the publicly available Internet based systems employ cryptographic suits to secure network connections. These cryptographic suits work with the Transport Layer Security (TLS) and Secure Shell (SSH) protocols, which both use different kinds of elliptic curve cryptography algorithms [43]. While different security levels can be attained by employing prime fields or binary fields for elliptic curves, notable advancements have been made in solving the elliptic curve discrete logarithm problem over binary fields [44]. Hence, due to their efficacy and security, the most widely adopted elliptic curves for cryptographic purposes are those defined over prime fields [14], [43].

Field programmable gate arrays (FPGAs) are configurable hardware fabrics that are widely used to implement digital systems and enhance the computational

complexity of a wide range of applications. Examples where FPGAs' implementations are presented can be found in [45], [46], and [47].

B. CONTRIBUTIONS

This work proposes two lightweight elliptic curve cryptography processors for IoT devices. The first ECC processor is a serial ECC processor that is smaller in area while the second one is a parallel ECC processor that is slightly larger. The proposed processors are of high security and resilient against attacks as they use Edwards curves defined over a prime characteristic finite field and perform finite field multiplication within the frequency domain employing modular multiplication using the Discrete Fourier Transform. Moreover, the faster versions of the proposed ECC processors are implemented in FPGAs utilizing FPGA-embedded DSPs to achieve high performance for supporting real-time applications and mitigating the effects of the energy constraints and the limited resources in the IoT devices that serve a wide range of applications including smartphones, smart watches, security cameras, vehicles, industrial machinery, wildlife tracking, traffic monitoring, and natural disaster prediction. The proposed ECC processor architectures are designed to work over prime characteristic finite fields of the form $GF((2^n-1)^n)$. The ECC processors outlined in this work are realized and compared against similar works over the prime characteristic finite fields $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$, and $GF((2^{19}-1)^{19})$, with the key sizes of 169, 289 and 361 bits, respectively. Our synthesis results show that the proposed ECC processors significantly outperform all existing ECC processors over the same finite fields in the literature in terms of both area and speed. For example, over Virtex5 FPGA family:

1) Compared to the lowest area ECC implementations in the literature over the same finite fields, our serial ECC processors without DSP components occupy 19.07%, 26.06% and 17.41% less number of slices while at the same time being 2.17×, 2.92× and 3.57× faster for the key sizes of 169, 289 and 361 bits, respectively.
2) For the same key sizes, compared to the fastest ECC processors in the literature over the same finite fields, our parallel ECC processors without DSP components occupy 20.42%, 27.38% and 22.74% less number of slices while at the same time being 1.76×, 2.01× and 2.22× faster, respectively.
3) Compared to the fastest ECC processors in the literature over the same finite fields, our parallel ECC processors that use FPGA-embedded DSP components are 1.93×, 2.41×, and 2.50× faster for the key sizes of 169-bit, 289-bit, and 361-bit, respectively.

C. ORGANIZATION

Section II provides a mathematical background related to elliptic curve cryptography. Section III summarizes the existing literature. Section IV presents the detailed designs and the FPGA realizations for the proposed ECC processors. The synthesis and performance results are presented and discussed in Section V. Finally, the conclusions are given in Section VI.

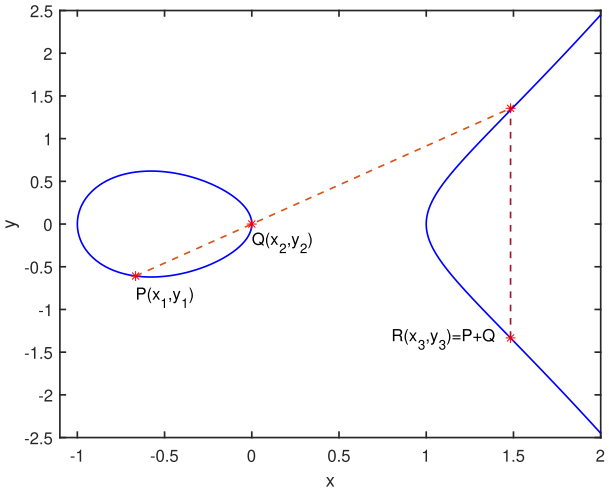


FIGURE 1. Addition: $R = P + Q$.

II. MATHEMATICAL BACKGROUND

A. GROUP OPERATIONS IN ELLIPTIC CURVE CRYPTOGRAPHY

An elliptic curve group is composed of points over an elliptic curve whose coefficients are elements of a finite field, in addition to an artificial point, known as the point at infinity, which serves as the identity element. The group operation in elliptic curve cryptography is called point addition. When an elliptic curve point is added with itself, the point addition operation takes a different form and is called point doubling. Geometric interpretations provide the most comprehensive explanations of point addition and point doubling [20], which we give in the sequel.

Consider two distinct points $P(x_1, y_1)$ and $Q(x_2, y_2)$ on an elliptic curve E. Adding P and Q to obtain the result R works as follows: First, consider the line passing through points P and Q. This line intersects the elliptic curve E at a third point. Taking the reflection of this point along the x-axis gives the result R. This process is illustrated in Figure 1, using the elliptic curve $E : y^2 = x^3 - x$. On the other hand, adding P to itself to obtain the double R is defined as follows: Again consider the line that goes through P and P itself: thus, this line is tangent to the elliptic curve E at point P. It then intersects the curve at another point. Then, R is the reflection of this point about the x-axis. This is demonstrated in Figure 2, using the same elliptic curve $E : y^2 = x^3 - x$.

The general form of elliptic curves over a finite field are Weierstrass curves which are given by the formula $y^2 = x^3 + ax + b$. Weierstrass curves are versatile and can represent any elliptic curve. However, point operations on Weierstrass curves are more complex compared with other forms of elliptic curves. Hence, elliptic curve cryptography using the

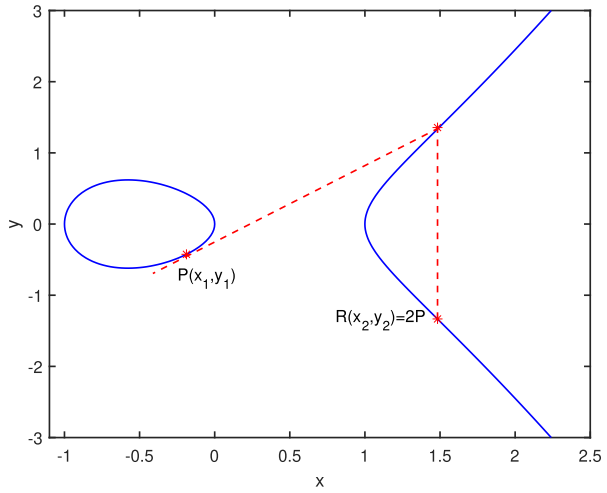


FIGURE 2. Doubling: $R = P + P = 2P$.

Weierstrass form of elliptic curves is slower due to the complexity of arithmetic operations.

Edwards curves are a special form of elliptic curves, named after Harold Edwards who showed that this type of curves allows for a unified formula for point addition and point doubling [22]. Having a unified formula for point arithmetic is not just more stylish, but it naturally provides side-channel protection for elliptic curve point multiplication against simple power analysis [41], eliminating the need for additional algorithms such as Montgomery ladder [48]. An Edwards curve over a prime characteristic finite field is defined by the equation $x^2+y^2 = c^2(1+dx^2y^2)$, where $c, d \neq 0$ and $dc^4 \neq 1$. On this elliptic curve, point addition and doubling operations can be achieved using a unified formula. For the points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, where $P_1 = P_2$ or $P_1 \neq P_2$, the point sum $P_3(x_3, y_3)$ can be computed as follows:

$$x_3 = \frac{x_1y_2 + y_1x_2}{c(1 + dx_1x_2y_1y_2)} \text{ and } y_3 = \frac{y_1y_2 - x_1x_2}{c(1 - dx_1x_2y_1y_2)} \quad (1)$$

The point $\infty = (0, c)$ is the point-at-infinity (identity element) for the elliptic curve group defined over this Edwards curve.

Building upon the work in [22], Bernstein and Lange introduced efficient formulae for point arithmetic on Edwards curves and proposed their use for cryptography [23]. Since Edward curves allow for using the same arithmetic operations for performing both point addition and point doubling, a smaller hardware circuitry, or a smaller software code, would be needed to execute both operations.

Another form of elliptic curves over a finite field are Montgomery curves [49], which are given by the formula $By^2 = x^3 + ax^2 + x$. Montgomery curves also allow for efficient point addition and doubling operations. However, they are more complex to implement compared to Edwards curves, and they do not provide a unified formula for the point addition and doubling operations. Edwards curves allow for faster arithmetic operations for point addition and doubling, compared to the Weierstrass and Montgomery forms. Furthermore, they exhibit strong resistance to side-channel attacks due to their unified formula for point addition and doubling.

1) ELLIPTIC CURVE SCALAR POINT MULTIPLICATION

The main operation that is performed in elliptic curve cryptography is elliptic curve scalar point multiplication. For an elliptic curve point P, the operation kP is known as elliptic curve scalar multiplication and computed by adding P with itself k times. Elliptic curve scalar multiplication is typically computed through a sequence of point addition and doubling operations. Finding the integer k for a given elliptic curve point kP is known as the elliptic curve discrete logarithm problem. For a sufficiently large-sized elliptic curve group, solving the elliptic curve discrete logarithm problem (ECDLP) is considered computationally intractable. ECDLP is the mathematically hard problem that is underlying the security of elliptic curve cryptography.

Algorithm 1 Unified Formula for Elliptic Curve Point Addition and Doubling in Projective Coordinates Using Edwards Curves [23]

Require: $P_1(X_1, Y_1, Z_1)$ and $P_2(X_2, Y_2, Z_2)$, where $P_1 = P_2$ or $P_1 \neq P_2$
Ensure: $P_3(X_3, X_3, X_3) = P_1 + P_2$
1: $R_1 \Leftarrow X_1$
2: $R_2 \Leftarrow Y_1$
3: $R_3 \Leftarrow Z_1$
4: $R_4 \Leftarrow X_2$
5: $R_5 \Leftarrow Y_2$
6: $R_6 \Leftarrow Z_2$
7: $R_3 \Leftarrow R_3 \cdot R_6$
8: $R_7 \Leftarrow R_1 + R_2$
9: $R_8 \Leftarrow R_4 + R_5$
10: $R_1 \Leftarrow R_1 \cdot R_4$
11: $R_2 \Leftarrow R_2 \cdot R_5$
12: $R_7 \Leftarrow R_7 \cdot R_8$
13: $R_7 \Leftarrow R_7 - R_1$
14: $R_7 \Leftarrow R_7 - R_2$
15: $R_7 \Leftarrow R_7 \cdot R_3$
16: $R_8 \Leftarrow R_1 \cdot R_2$
17: $R_8 \Leftarrow d \cdot R_8$
18: $R_2 \Leftarrow R_2 - R_1$
19: $R_2 \Leftarrow R_2 \cdot R_3$
20: $R_3 \Leftarrow R_3^2$
21: $R_1 \Leftarrow R_3 - R_8$
22: $R_3 \Leftarrow R_3 + R_8$
23: $R_2 \Leftarrow R_2 \cdot R_3$
24: $R_3 \Leftarrow R_3 \cdot R_1$
25: $R_1 \Leftarrow R_1 \cdot R_7$
26: $R_3 \Leftarrow c \cdot R_3$
27: $X_3 \Leftarrow R_1$
28: $Y_3 \Leftarrow R_2$
29: $Z_3 \Leftarrow R_3$
30: Return (X_3, Y_3, Z_3)

B. PROJECTIVE COORDINATES

The efficiency of an ECC implementation heavily relies on the efficiency of the underlying finite field arithmetic operations, namely finite field addition, subtraction, multiplication, and inversion, as given in (1). Addition and subtraction in a finite field are fast with only linear complexity in the length of input operands. Multiplication is significantly slower and has quadratic complexity. However, inversion, the slowest of finite field arithmetic operations, is typically orders of magnitude slower than multiplication for operand sizes used in ECC implementations. Therefore, the inversion operation is typically avoided by employing techniques such as using the projective coordinates. The affine point (x, y) is represented in projective coordinates as $(X, Y, Z) = (\lambda x, \lambda y, \lambda)$ for any non-zero λ. Hence, in Edwards curves, the point-at-infinity $\infty = (0, c)$ in affine coordinates becomes $\infty = (0, c, 1)$ in projective coordinates. Another nice feature about point operations over Edwards curve is that no special care is needed to handle operations with the point-at-infinity.

In our work, for efficient and side-channel resilient point arithmetic, we use Edwards curves and associated unified formula for point arithmetic [22]. Furthermore, we avoid performing costly modular inversions by performing unified Edwards curve point arithmetic in projective coordinates using Algorithm 1 [23]. We perform the operations in Algorithm 1 sequentially. For each line of the algorithm, we read the inputs from registers, execute the operation, i.e., finite field addition, subtraction, or multiplication, and then write the result to the memory.

C. FINITE FIELD ARITHMETIC IN ELLIPTIC CURVE CRYPTOGRAPHY

Remarkably, polynomial multiplication, with the computational complexity that grows quadratically with the number of polynomial coefficients, is analogous to convolution. Conversely, convolution exhibits only linear complexity when carried out in the frequency domain. To leverage the benefits of frequency domain arithmetic, one can transform operands to the frequency domain using the discrete Fourier transform (DFT), efficiently carry out multiplication there, and subsequently return the result to the time domain by utilizing the inverse DFT.

The Fast Fourier Transform (FFT) represents an optimized variant of the DFT that can efficiently transform a sequence between the time and the frequency domains [50]. The FFT is employed by prominent integer multiplication algorithms, including the algorithms of Schönhage–Strassen [51] and Fürer [52]. These algorithms leverage the FFT to perform efficient multiplication within the frequency domain.

Nevertheless, the FFT algorithm achieves optimal efficiency when the sequence length is a substantially large value that is also equal to a power of two. Notably, the algorithms developed by Schönhage–Strassen and Fürer demonstrate efficiency solely when dealing with operands that span thousands of bits.

In our ECC implementations over $GF((2^{13} - 1)^{13})$, $GF((2^{17} - 1)^{17})$ and $GF((2^{19} - 1)^{19})$, the operands are polynomials with 13, 17 and 19 coefficients, where the coefficients are of length 13 bits, 17 bits and 19 bits, resulting in the operand lengths of 169 bits, 289 bits and 361 bits, respectively. Consequently, we do not possess substantially large operands, and their number of coefficients is not a power of two. Due to these factors, the efficiency of the FFT algorithm would be diminished in this context.

Notably, the efficiency of an ECC implementation is significantly influenced by the underlying finite field arithmetic. Multiplication over $GF(p^m)$ involves a large number of additions and multiplications in $GF(p)$ [18]. To mitigate this large number of base field multiplications, the DFT modular multiplication algorithm can be employed, enabling multiplication in $GF(p^m)$ within the frequency domain [53]. Importantly, in the context of DFT modular multiplication, it is not necessary for the number of coefficients in an operand to be a power of 2. Moreover, in contrast to common FFT-based algorithms for multiplication, DFT modular multiplication carries out not only multiplication but also modular reduction within the frequency domain. As a result, the need for the expensive conversion back to the time domain after each multiplication is eliminated.

Elements of the finite field $GF(p^n)$ are polynomials of degree $n - 1$, where the polynomial coefficients are $GF(p)$ elements. For $a(x), b(x) \in GF(p^n)$, the product $a(x) \times b(x)$ is computed by first performing polynomial multiplication and then reduction using the irreducible field generating polynomial. Reduction can be achieved with only linear complexity if a simple binomial such as $x^n - 2$, is used as the field generating polynomial. While the classical schoolbook method for polynomial multiplication has quadratic complexity in the usual time domain, the same polynomial multiplication operation has only linear complexity in the frequency domain due to the convolution theorem. However, for repeated multiplications, as they take place in ECC computations, one needs to transfer the polynomial product back to the time domain to perform modular reduction by the field-generating polynomial. While polynomial multiplication has linear complexity in the frequency domain, conversion between the time and the frequency domains is achieved using the DFT which has quadratic complexity. If the operand length n is a power of two, the discrete Fourier transform computation can be performed optimally by using the fast Fourier transform, which still results in the sub-quadratic complexity of $O(n \log n)$.

In our ECC implementations, for finite field multiplication, we use two optimized forms of Algorithm 2, given in Section IV (Algorithms 3 and 4), which perform both polynomial multiplication and modular reduction in the frequency domain. Thus, repeated multiplications can be performed over operands in the frequency domain, without having to do conversion back to the time domain for doing reductions. Furthermore, we perform finite field addition and

Algorithm 2 DFT Modular Multiplication Algorithm for $GF(p^n)$ [54]

Require: $(A) \equiv a(x) \in GF(p^n)$, $(B) \equiv b(x) \in GF(p^n)$, $(P') \equiv p(x)/p(0) \bmod p$.
Ensure: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(n-1)} \in GF(p^n)$
1: **for** $i = 0$ to $2n-1$ **do**
2: $\quad C_i \Leftarrow A_i \cdot B_i$
3: **end for**
4: **for** $i = 0$ to $n-2$ **do**
5: $\quad S \Leftarrow 0$
6: $\quad$ **for** $j = 0$ to $2n-1$ **do**
7: $\quad\quad S \Leftarrow S + C_j \bmod p$
8: $\quad$ **end for**
9: $\quad S \Leftarrow -\frac{S}{2n} \bmod p$
10: $\quad$ **for** $j = 0$ to $2n-1$ **do**
11: $\quad\quad C_j \Leftarrow (C_j + P'_j \cdot S) \cdot X_j^{-1} \bmod p$
12: $\quad$ **end for**
13: **end for**
14: Return (C)

subtraction, as they take place in Algorithm 1, also in the frequency domain.

Before performing finite field arithmetic in the frequency domain, finite field elements need to be converted into their frequency domain representations. This conversion is done using the discrete Fourier transform. To convert an element of the finite field $GF(p^n)$ to the frequency domain, we first represent it as a $2n$-element sequence in the time domain. We then convert the time domain sequence to the frequency domain by applying the DFT. For instance, $a(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$, an element of $GF(p^n)$, is first represented as the $2n$ element sequence $(a) = (a_0, a_1, a_2, \ldots, a_{n-1}, 0, 0, \ldots, 0)$ by appending n zeroes at the end. The frequency domain sequence $(A) = (A_0, A_1, A_2, \ldots, A_{2n-1})$, corresponding to the time domain sequence (a), is computed by using the DFT as follows:

$$A_j = \sum_{i=0}^{l-1} a_i r^{ij} \bmod p, \quad 0 \le j \le l-1\ , \tag{2}$$

where the sequence length is $l = 2n$ and r is an l^{th} primitive root of unity. In our ECC implementations, we use $r = -2 \in GF(p)$ as the l^{th} primitive root of unity. (A) can be converted back to the time domain by using the inverse DFT computation as follows:

$$a_i = \frac{1}{l} \cdot \sum_{j=0}^{l-1} A_j r^{-ij} \bmod p, \quad 0 \le i \le l-1\ . \tag{3}$$

Algorithm 2 is a DFT-based modular multiplication algorithm that achieves Montgomery multiplication in $GF(p^n)$ in the frequency domain. The inputs to the algorithm are (A) and (B), the frequency domain sequence representations of $a(x), b(x) \in GF(p^n)$, and the output is (C), the frequency domain sequence representation of the Montgomery product $c(x) = a(x) \cdot b(x) \cdot x^{-(n-1)} \in GF(p^n)$. In the algorithm, the parameter (P') is the frequency domain sequence representation of the normalized irreducible polynomial $p'(x) = p(x)/p(0) \bmod p$, where $p(x)$ is the irreducible polynomial generating the finite field $GF(p^n)$.

Algorithms 3 and 4 are optimized serial and parallel versions of Algorithm 2 that are used in our serial and parallel ECC implementations. In these two algorithms, the Mersenne prime field characteristic $p = 2^n - 1$ and a finite field of the form $GF(p^n)$ is used. The frequency domain sequences (A) and (B), with length $l = 2n$, are obtained by applying the DFT and using the l^{th} primitive root of unity $r = -2 \in GF(p)$. In this setting, in line 11 of Algorithm 2, the multiplications with P'_j and X_j^{-1} become multiplications with powers of -2 which can be computed with a simple bit-wise rotation. Thus, Algorithms 3 and 4 perform with only a linear number of $GF(p)$ multiplications in the length of operands, in addition to a quadratic number of simple additions, subtractions and bit-wise rotations.

1) FINITE FIELD ADDITION AND MULTIPLICATION IN THE FREQUENCY DOMAIN

In our ECC implementations, in addition to finite field multiplication, we achieve finite field addition and subtraction also in the frequency domain. We realize addition/subtraction of two finite field elements in the frequency domain through simple pairwise addition/subtraction of their frequency domain sequence coefficients. For instance, for $a(x), b(x) \in GF(p^n)$, we compute $c(x) = a(x) \pm b(x)$ in the frequency domain as follows:

$$C_i = A_i \pm B_i \bmod p, \quad 0 \le i \le 2n-1\ ,$$

where (C), (A) and (B) are the frequency domain sequences for $c(x)$, $a(x)$ and $b(x)$, respectively.

2) SCALABILITY OF DFT MODULAR MULTIPLICATION AND THE PROPOSED ECC ARCHITECTURES

In the proposed ECC implementations, which use the DFT modular multiplication algorithm, we use the finite fields $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$ and $GF((2^{19}-1)^{19})$ with the Mersenne prime field characteristics $2^{13}-1$, $2^{17}-1$ and $2^{19}-1$, respectively. Hence, the proposed ECC implementations support the key lengths of 169, 289, and 361 bits. The DFT modular multiplication algorithm and the proposed ECC architectures using it are scalable in that they can be designed to work over any finite extension field of the form $GF((2^n-1)^n)$ with a Mersenne prime field characteristic. Furthermore, the proposed ECC architecture would work also over a finite extension field with a pseudo-Mersenne prime field characteristic, e.g. $GF((\frac{2^{23}-1}{47})^{23})$, with a minor overhead [54].

III. RELATED WORK

While we compare the performance of our ECC implementations against the most related recent works in Section V, below we provide a more detailed discussion of the literature.

A. PREVIOUS WORKS

Islam et al. [55] presented an FPGA implementation of an ECC processor over a 256-bit prime field using a twisted Edwards curve known as Edwards25519 using Virtex-7. They then proposed an area-time efficient hardware modular multiplier architecture for multiplication over the five National Institute of Standard and Technology (NIST)-recommended prime fields for ECC [56], using a modified radix-2 interleaved modular multiplication algorithm, and implemented it on the Xilinx Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA platforms.

Li et al. [57] used a modified version of the Montgomery Ladder algorithm, a mixed-pipeline technique with two multipliers, and a high-speed modular inversion unit to achieve elliptic curve scalar multiplication over $GF(2^{163})$ and $GF(2^{571})$. They used Xilinx Virtex-5 and Virtex-7.

Lara-Nino et al. [58] implemented an FPGA-based lightweight ECC acceleration engine over a 128-bit binary field using Edward curves using Xilinx Virtex-5.

Mehrabi et al. [59] proposed a residue number system based ECC hardware core that used short Weierstra⌊ and twisted Edwards curves. They performed the implementations specifically for the standard SECP256K1 and ED25519 curves. Their hardware architecture supported the elliptic curve point tripling operation, in addition to the elliptic curve point addition and point doubling operations. They used Xilinx Virtex-7 and Virtex UltraScale+.

Hu et al. [60] proposed an ECC hardware processor that utilizes a three-clock-cycle divide-and-conquer multiplication algorithm to reduce the execution time for multiplication. They designed the operation scheduling of their point addition and point doubling hardware to achieve high throughput by running multiplication in parallel with other operations. Furthermore, they used an improved scalar multiplication algorithm to resist side-channel attacks. They used 55 nm CMOS ASIC as their target platform.

Lee et al. [61] proposed a large field-size ECC processor over GF(2^m), with $m = 409$ and 571, using an optimized Montgomery point multiplication algorithm. They used Xilinx Virtex 7 and Kintex Ultrascale+.

Xie et al. [62] proposed a dual-core high-performance ECC processor that uses a generic Weierstrass curve defined over GF(p). In their implementation, they used an optimized radix-128 Montgomery modular multiplication algorithm, which takes 6 clock cycles to calculate modular multiplication. Their dual-core hardware architecture includes point-doubling and point-addition cores to compute elliptic curve point multiplication. For data transfer between the point-addition and point-doubling cores, they used a 1024-bit register. As their target platform, they used both 90 nm ASIC using the standard cell library and Xilinx Virtex-6.

Chen et al. [64] proposed a modular arithmetic unit that implemented the modular addition, subtraction, and multiplication operations for ECC in one unified module, resulting in low hardware complexity. They used a three-level carry-save-adder (CSA) for the modular addition and subtraction operations. For modular multiplication, they used the three-level CSA, and implemented a fully CSA-based Radix-4 Montgomery multiplication algorithm. They implemented their proposed unified modular arithmetic unit for ECC on Xilinx Virtex-6 and Virtex-7 target platforms.

Ding et al. [65] introduced a fast elliptic curve cryptography processor utilizing NIST primes. The researchers implemented 4-way Toom–Cook multiplication to enhance time efficiency. Additionally, they introduced the nonleast-positive form for product representation to bolster parallelism, effectively disrupting the carry chain within their multiplier design. Pipelining was also employed to mitigate the effects of modular inversion. They used Xilinx Virtex-6 and 180 nm CMOS ASIC as target platforms.

Awaludin et al. [26] devised a unified ECC processor utilizing a Weierstrass curve over a prime field. To achieve an increased level of parallelism with modest complexity, a blend of the schoolbook and Karatsuba multiplication algorithms was employed in designing the elliptic curve point multiplier. Furthermore, an n-bit pipelined multiplier accumulator, leveraging embedded DSP, was employed to build a pipelined n-bit Montgomery multiplier. They used Xilinx Virtex-7 and Kintex-7 FPGAs as their target platform.

Algorithm 3 Serial DFT Based Modular Multiplication [18]

Require: $(A) \equiv a(x) \in GF(p^n), (B) \equiv b(x) \in GF(p^n)$, where $p = 2^n - 1$
Ensure: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(n-1)} \ mod \ f(x)$

$S \Leftarrow 0$
for $i = 0$ *to* $2n - 1$ **do**
  $C_i \Leftarrow A_i \cdot B_i$,
  $S \Leftarrow S + C_i$
end for
for $j = 0$ *to* $n - 2$ **do**
  $S \Leftarrow -S/(2n)$
  $S_{even} \Leftarrow S \gg 1$
  $S_{odd} \Leftarrow (S + (S \gg 1))$
  $S \Leftarrow 0$
  for $i = 0$ *to* $2n - 1$ **do**
    if $i \mod 2 = 0$ **then**
      $C_i \Leftarrow C_i + S_{even}$
    else
      $C_i \Leftarrow -(C_i + S_{odd})$
    end if
    $S_{even} \Leftarrow S_{even} \gg 1$
    $S_{odd} \Leftarrow S_{odd} \gg 1$
    for $k = i + 1$ *to* $2n - 1$ **do**
      $C_k \Leftarrow C_k \gg 1$
    end for
    $S \Leftarrow S + C_i$
  end for
end for
return C

Algorithm 4 Parallel DFT Based Modular Multiplication [63]

Require: $(A) \equiv a(x) \in GF(p^n), (B) \equiv b(x) \in GF(p^n)$, where $p = 2^n - 1$
Ensure: $(C) \equiv a(x) \cdot b(x) \cdot x^{-(n-1)} \ mod \ f(x)$
1: $S \Leftarrow 0$
2: **for** $i = 0$ *to* $n - 1$ **do**
3: $C_{2i} \Leftarrow A_{2i} \cdot B_{2i}, C_{2i+1} \Leftarrow A_{2i+1} \cdot B_{2i+1}$
4: $S \Leftarrow S + C_{2i} + C_{2i+1}$
5: **end for**
6: **for** $j = 0$ *to* $n - 2$ **do**
7: $S \Leftarrow -S/(2n)$
8: $S_{even} \Leftarrow S \gg 1$
9: $S_{odd} \Leftarrow (S \gg 1) + (S \gg 2)$
10: $S \Leftarrow 0$
11: **for** $i = 0$ *to* $n - 1$ **do**
12: $C_{2i} \Leftarrow C_{2i} + S_{even}, C_{2i+1} \Leftarrow -(C_{2i+1} + S_{odd})$
13: $S_{even} \Leftarrow S_{even} \gg 2$
14: $S_{odd} \Leftarrow S_{odd} \gg 2$
15: **for** $k = i + 1$ *to* $n - 1$ **do**
16: $C_{2k} \Leftarrow C_{2k} \gg 2$
17: $C_{2k+1} \Leftarrow C_{2k+1} \gg 2$
18: **end for**
19: $S \Leftarrow S + C_{2i} + C_{2i+1}$
20: **end for**
21: **end for**
22: **return** C

B. MOST NOTABLY

A novel ECC processor, which conducts finite field arithmetic in the frequency domain, was introduced in [18]. This processor was designed utilizing an optimal extension field (OEF) as proposed in previous works [66], [67]. The multiplication operation within the extension field is achieved in the frequency domain through a linear count of base field multiplications coupled with a quadratic count of other operations, e.g. bit-wise rotations and additions of based field elements. This study stands out as the pioneering work in introducing a frequency domain multiplier tailored for ECC, as well as being the first to realize an ECC processor in this domain.

In [63], the authors proposed enhancements to the approach presented in [18]. They achieved greater parallelism by incorporating two base field multipliers and two base field adders into the design of the Discrete Fourier Transform modular multiplier. Additionally, they employed an Edwards curve to enhance security. The implementation targeted the Xilinx Virtex-5 FPGA family.

In [15] and [16], the authors significantly enhanced the works of [18] and [63]. Notably, they designed a more advanced arithmetic unit for achieving base field arithmetic. Their implementation also covered the finite prime fields $GF((2^{13} - 1)^{13})$, $GF((2^{17} - 1)^{17})$ and $GF((2^{19} - 1)^{19})$. Our work improves upon the ECC processor designs in [15] and [16] by utilizing improved multiplier designs and FPGA embedded components. We provide our performance comparison against these works.

IV. THE PROPOSED ECC PROCESSORS

This work proposes two ECC processors that perform finite field multiplication in the frequency domain using discrete Fourier transform modular multipliers (DFTMMs). These DFTMMs utilize the embedded multipliers on the FPGA to perform the finite field multiplication operation that is required for ECC. Our first proposed ECC processor is called the serial ECC processor, and the second is called the parallel ECC processor. The DFTMM in our serial ECC processor is designed according to Algorithm 3 and referred to as the serial DFTMM, or as $DFTMM_S$, throughout this work. Whereas, the DFTMM in our parallel ECC processor is designed according to Algorithm 4 and referred to as the parallel DFTMM, or as $DFTMM_P$. In Algorithms 3 and 4, multiplication in the finite field $GF((2^n - 1)^n)$ is achieved in the frequency domain. The size of the used finite field determines the size of the used elliptic curve group, and thus the key length for ECC. For an elliptic curve defined over $GF((2^n - 1)^n)$, the elliptic curve group has size n^2. In this work, we choose n as 13, 17 and 19, and implement ECC processors working over the finite fields $GF((2^{13} - 1)^{13})$, $GF((2^{17} - 1)^{17})$ and $GF((2^{19} - 1)^{19})$, which have the key sizes of 169-bit, 289-bit and 361-bit, respectively.

Our serial ECC processor uses one base field adder and one DFTMM that is designed using only one base field multiplier. On the other hand, our parallel ECC processor uses two base field adders and one DFTMM that is designed using two base field multipliers. Therefore, the parallel ECC processor performs point multiplication in almost half the number of cycles required by the serial ECC processor with some extra hardware.

The top-level schematic for our proposed parallel ECC processor with the 169-bit key, which is generated using AMD-Xilinx tools, is shown in Figure 3. As illustrated, the proposed parallel ECC processor consists of three main components: the arithmetic unit, the control unit, and the memory unit, labeled A, B, and C, respectively.

The arithmetic unit is responsible for performing the main modular arithmetic operations that are the core of elliptic curve point multiplication. These main modular arithmetic operations are modular addition, modular subtraction, and modular multiplication. The top-level schematic for the arithmetic unit of our parallel ECC processor with the 169-bit key is illustrated in Figure 4. It consists of two base field adders and one DFTMM (with two base field multipliers).

The finite field multiplication operation is performed by the DFTMM in the frequency domain. The DFTMM is designed according to Algorithm 4, and it mainly consists of two base field adders, two base field multipliers, and registers. On the other hand, the finite field addition and subtraction operations are performed using the base field adders.

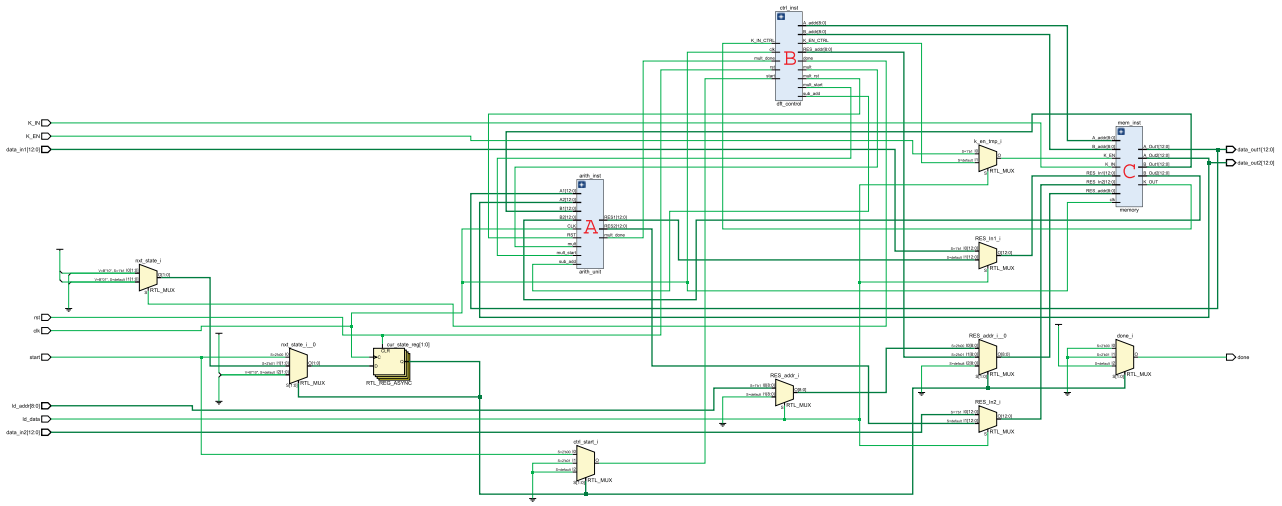


FIGURE 3. Top level design of the proposed parallel ECC processor with the 169-bit key.

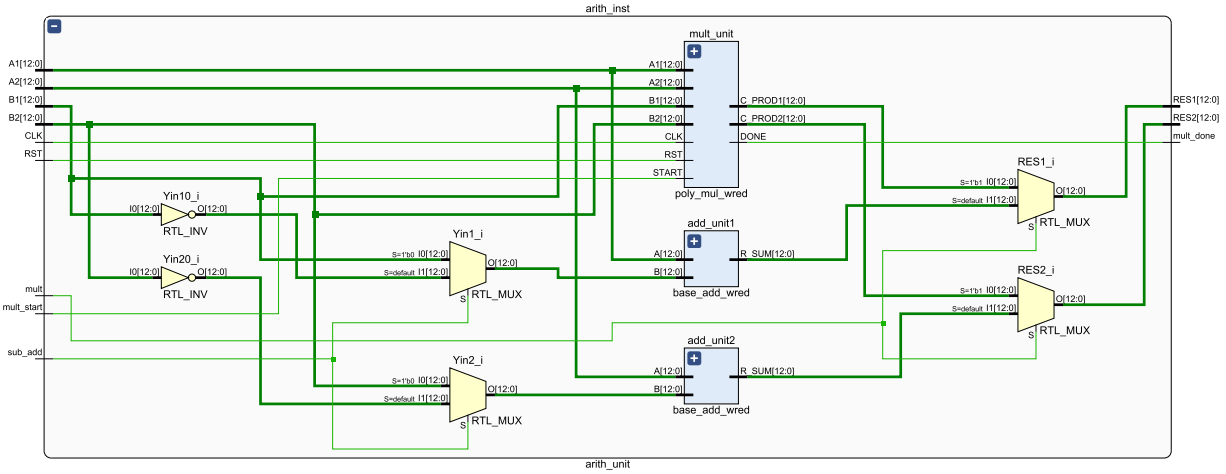


FIGURE 4. Top level design of the arithmetic unit used in the proposed parallel ECC processor with the 169-bit key.

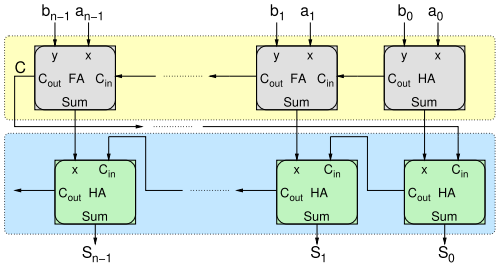


FIGURE 5. Base field adder.

A. BASE FIELD ADDER AND MULTIPLIER

The base field adder that is adopted in this work is a standard base field adder. It is a binary adder followed by a modular reduction circuit. The base field adder circuit is depicted in Figure 5.

The base field multiplier is illustrated in Figure 6. The n-bit operands are multiplied using a binary multiplier that produces a $2n$-bit result. This $2n$-bit product is reduced using a base field adder, which adds the least and most significant n bits of the product.

This work utilizes the embedded multipliers on the FPGA to perform the binary multiplication operation in the base field multiplier. The embedded multipliers on the AMD-Xilinx FPGA are 18 × 18 multipliers, therefore the largest operand size for multiplication using these embedded multipliers is 18-bit. AMD-Xilinx IP core generator is used to configure and generate these embedded multipliers. If the operand size is less than or equal to 18-bit, one embedded multiplier is needed to multiply the two operands. If the size of the operands is greater than 18-bit, then more than one of the 18 × 18 FPGA embedded multipliers or additional logic is required to build a larger-sized multiplier.

In this work, the operands for base field arithmetic are of size 13-bit, 17-bit, and 19-bit, because our ECC processors are designed for the finite fields $GF((2^n - 1)^n)$ where the

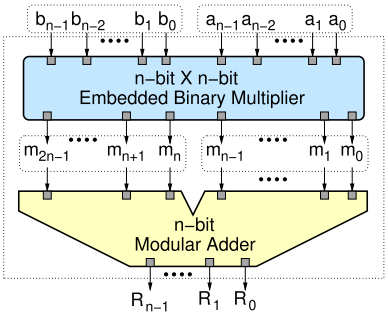


FIGURE 6. Base field multiplier using the FPGA embedded multiplier.

corresponding values of n are 13, 17 and 19, respectively. Therefore, it should be expected that there would be extra hardware overhead for the case when $n = 19$ which is larger than the size of the DSP embedded multiplier on the FPGA. This is experimentally demonstrated in Section V.

Xilinx IP core generator has the option of generating a multiplier either by using the DSP embedded components or by using the look-up tables (LUTs) on the FPGA. Both cases are investigated in this work. Our base field multiplier, designed using the DSP-embedded components or the LUTs, is depicted in Figure 6. The following symbols are used to distinguish the type of the used Base Field Multiplier (BFM):

1) BFM_{DSP}
In this BFM, the binary multiplier is generated using the DSP-embedded components on the FPGA.

2) BFM_{LUT}
In this BFM, the binary multiplier is generated using the LUT components on the FPGA.

Since one of the above two base field multiplier designs can be used in the design of the serial and the parallel versions of the DFTMM, the following four different configurations of the DFTMM can be designed.

3) $DFTMM_{S_DSP}$
The serial DFTMM that is designed according to Algorithm 3 where the base field multiplier is designed with the FPGA embedded multiplier that is generated using the DSP embedded components on the FPGA.

4) $DFTMM_{S_LUT}$
The serial DFTMM that is designed according to Algorithm 3 where the base field multiplier is designed with the FPGA embedded multiplier that is generated using the LUT components on the FPGA.

5) $DFTMM_{P_DSP}$
The parallel DFTMM that is designed according to Algorithm 4 where the base field multiplier is designed with the FPGA embedded multiplier that is generated using the DSP embedded components on the FPGA.

TABLE 1. Versions of the proposed ECC processors.

ECC Processor	Description
ECC_{S_DSP}	Serial ECC processor designed using the $DFTMM_{S_DSP}$
ECC_{S_LUT}	Serial ECC processor designed using the $DFTMM_{S_LUT}$
ECC_{P_DSP}	Parallel ECC processor designed using the $DFTMM_{P_DSP}$
ECC_{P_LUT}	Parallel ECC processor designed using the $DFTMM_{P_LUT}$

6) $DFTMM_{P_LUT}$
The parallel DFTMM that is designed according to Algorithm 4 where the base field multiplier is designed with the FPGA embedded multiplier that is generated using the LUT components on the FPGA.

Each of these four DFTMM configurations along with the modular adder are used in building two versions of the proposed serial ECC processors and two versions of the proposed parallel ECC processor. These are summarized in Table 1.

V. EXPERIMENTAL RESULTS AND DISCUSSION
VHDL language is used to describe all of the designs presented in this work. These include two BFM configurations, four DFTMM configurations, two configurations of the proposed serial ECC processor, and two configurations of the proposed parallel ECC processor. The functionality of each design is verified using the ISE14.2 simulator from AMD-Xilinx. The designs are synthesized using the ISE14.2 synthesis tool from AMD-Xilinx with Virtex-5 as the target FPGA. Three finite prime fields are investigated, which are $GF((2^{13}-1)^{13})$, $GF((2^{17}-1)^{17})$, and $GF((2^{19}-1)^{19})$. The comparisons are done against [15], [18], [63], and [16], as up to the knowledge of the authors, these are the only existing works that are similar to this work in that they all use the same finite fields and operate in the frequency domain. Further, the proposed ECC processors along with the works in [15], [18], [63], and [16] are implemented using Virtex-7, Artex-7, and Kintex-7 FPGA families and the implantation results are reported and discussed.

The implementation results for the BFM designs of this work and the BFM designs of [15], [18], [63], and [16] for the finite fields $GF((2^n-1)^n)$, where $n = 13$, 17 and 19, are presented in Table 2. In all cases, both of the BFM designs proposed in this work achieve better delay and area results compared to the BFM designs of the existing works. For example, for $n = 13$, the proposed base field multiplier BFM_{DSP} is 2.9 times faster than the base field multiplier BFM_{RCA} of [16]. BFM_{DSP} has the area of 33 LUTs + 1 DSP compared to an area of 388 LUTs for the base field multiplier BFM_{RCA} of [16]. The proposed base field multiplier BFM_{LUT} is 1.52 times faster than the base field multiplier BFM_{FCC} of [16] with an area saving of 53.8%. For the case where $n = 19$, as expected, the proposed BFM_{DSP} requires 2

TABLE 2. Synthesis and performance results (area and delay) of the designed base field multipliers for $GF(2^n - 1)$ where $n = 13$, 17 and 19.

	$n = 13$		$n = 17$		$n = 19$	
Base Field Multiplier	**Delay (ns)**	**Area (LUTs)**	**Delay (ns)**	**Area (LUTs)**	**Delay (ns)**	**Area (LUTs)**
BFM of [18], [64]	46.611	256	78.951	437	97.473	546
BFM_{RCA} of [16]	36.108	388	58.839	677	71.48	852
BFM_{FCC} of [16]	22.141	481	29.388	833	32.678	1046
BFM_{DSP}	12.445	33+1 DSP	13.886	44+1 DSP	15.932	50+2 DSPs
BFM_{LUT}	14.500	222	17.152	364	17.825	445

TABLE 3. Synthesis and performance results of different DFT modular multipliers for multiplication in $GF((2^n - 1)^n)$ with $n = 13$.

DFT Modular Multiplier	# Slice LUTs	# Slices	Freq. (MHz)
$DFTMM$ of [18]	1124	420	40.1
$DFTMM$ of [64]	1445	751	32.5
$DFTMM_1$ of [15], [16]	1750	699	43.6
$DFTMM_2$ of [15], [16]	1809	821	41.8
$DFTMM_3$ of [15], [16]	1792	694	41.9
$DFTMM_4$ of [15], [16]	1681	774	41.8
$DFTMM_5$ of [15], [16]	1753	705	45.7
$DFTMM_6$ of [15], [16]	1838	785	45.6
$DFTMM_7$ of [15], [16]	1875	847	45.6
$DFTMM_8$ of [15], [16]	1798	762	47.7
$DFTMM_{S_DSP}$	715+1 DSP	320	107.4
$DFTMM_{S_LUT}$	915	402	91.9
$DFTMM_{P_DSP}$	761+2 DSPs	413	86.1
$DFTMM_{P_LUT}$	1134	537	80.1

TABLE 4. Synthesis and performance results of different DFT modular multipliers for multiplication in $GF((2^n - 1)^n)$ with $n = 17$.

DFT Modular Multiplier	# Slice LUTs	# Slices	Freq. (MHz)
$DFTMM$ of [18]	1917	908	22.45
$DFTMM$ of [64]	2788	1027	24.75
$DFTMM_1$ of [16]	2888	1242	25.94
$DFTMM_2$ of [16]	3010	1128	25.41
$DFTMM_3$ of [16]	3002	1239	26.41
$DFTMM_4$ of [16]	2883	1102	26.59
$DFTMM_5$ of [16]	2994	1184	31.36
$DFTMM_6$ of [16]	3064	1140	32.74
$DFTMM_7$ of [16]	3167	1260	32.90
$DFTMM_8$ of [16]	2983	1057	33.25
$DFTMM_{S_DSP}$	1126+1 DSP	597	98.7
$DFTMM_{S_LUT}$	1465	654	74.5
$DFTMM_{P_DSP}$	1223+2 DSPs	644	78.3
$DFTMM_{P_LUT}$	1844	832	63.0

DSPs since the largest size for embedded multipliers on the FPGA is 18 × 18. Hence, 2 DSPs are needed to build a 19 × 19 multiplier.

The implementation results for the proposed DFTMMs for multiplication in $GF((2^n - 1)^n)$ are listed and compared against existing DFTMMs in Tables 3, 4, and 5, for $n = 13$ (key size = 169-bit), $n = 17$ (key size = 289-bit), and $n = 19$ (key size = 361-bit), respectively.

TABLE 5. Synthesis and performance results of different DFT modular multipliers for multiplication in $GF((2^n - 1)^n)$ with $n = 19$.

DFT Modular Multiplier	# Slice LUTs	# Slices	Freq. (MHz)
$DFTMM$ of [18]	2707	1001	20.298
$DFTMM$ of [64]	3429	1359	18.67
$DFTMM_1$ of [16]	3564	1432	21.64
$DFTMM_2$ of [16]	3739	1130	21.21
$DFTMM_3$ of [16]	3647	1352	22.46
$DFTMM_4$ of [16]	3550	1398	22.19
$DFTMM_5$ of [16]	3597	1401	28.72
$DFTMM_6$ of [16]	3693	1321	28.49
$DFTMM_7$ of [16]	3746	1282	27.25
$DFTMM_8$ of [16]	3664	1350	25.99
$DFTMM_{S_DSP}$	1379+2 DSPs	466	81.7
$DFTMM_{S_LUT}$	1739	833	65.3
$DFTMM_{P_DSP}$	1453+4 DSPs	722	71.7
$DFTMM_{P_LUT}$	2271	934	62.1

Clearly, in all cases, our proposed DFTMMs significantly outperform the existing DFTMMs in terms of both area and frequency. For example, for $n = 13$, it can be seen in Table 3 that the proposed $DFTMM_{P_LUT}$ gains 99.7%, 146.4% and 67.9% improvement in frequency over those of [18], [63] and the $DFTMM_8$ of [16], respectively. Most importantly, this frequency improvement is achieved with only 27.8% extra slices compared to the DFFMM of [18], 28.4% fewer slices compared to the DFFMM of [63], and 29.5% fewer slices compared to the $DFTMM_8$ of [16].

For the case when $n = 17$, the proposed $DFTMM_{P_LUT}$ gains 180.6%, 154.5% and 89.4% improvement in frequency over the DFTMM of [18], the DFTMM of [63] and the $DFTMM_8$ of [16], respectively. In addition to the significant frequency gains, area reduction is also achieved in the proposed designs. In the proposed $DFTMM_{P_LUT}$, 8.3%, 18.9% and 21.2% area reduction is achieved in terms of the number of slices, compared to the DFFMM of [18], the DFFMM of [63], the $DFTMM_8$ of [16], respectively.

For $n = 19$ as listed in Table 5, the proposed $DFTMM_{P_LUT}$ gains 205.9%, 232.6%, and 116.2% improvement in frequency over those of [18], [63], and the $DFTMM_5$ of [16], respectively. On top of its significant frequency gains, its area is still smaller. The area of the proposed $DFTMM_{P_LUT}$ in terms of number of slices is only 934, compared to 1001, 1359 and 1401 slices for the DFFMM of [18], the DFFMM of [63] and the $DFTMM_8$ of [16], respectively.

TABLE 6. Synthesis and performance results of the designed ECC processors for $n = 13$ (key length of 169 bits).

FPGA Family	ECC Processor	# of DSP48Es	Cycles Count	# of Slice Registers	# of Slice LUTs	# of Slices	Cycle time (ns)	Frequency (MHz)	Time (ms)	Power (W)
Virtex5	ECC_{S_DSP}	1	1136836	772	1280	691	10.599	94.349	12.049	1.470
	ECC_{S_LUT}	0	1136836	772	1442	641	12.539	79.751	14.255	1.466
	ECC_{P_DSP}	2	594437	993	1688	785	12.258	81.579	7.287	1.500
	ECC_{P_LUT}	0	594437	993	2065	912	13.464	74.272	8.003	1.536
Virtex7	ECC_{S_DSP}	1	1136836	727	1039	429	9.116	109.697	10.363	0.192
	ECC_{S_LUT}	0	1136836	728	1230	538	9.348	106.975	10.627	0.209
	ECC_{P_DSP}	2	594437	955	1471	730	10.889	91.836	6.473	0.213
	ECC_{P_LUT}	0	594437	972	1867	810	11.546	86.610	6.863	0.230
Artex7	ECC_{S_DSP}	1	1136836	727	1025	433	10.992	90.975	12.496	0.088
	ECC_{S_LUT}	0	1136836	730	1234	504	12.352	80.959	14.042	0.104
	ECC_{P_DSP}	2	594437	948	1481	670	13.796	72.485	8.201	0.104
	ECC_{P_LUT}	0	594437	955	1837	725	14.595	68.517	8.676	0.121
Kintex7	ECC_{S_DSP}	1	1136836	727	985	464	7.870	127.065	8.947	0.089
	ECC_{S_LUT}	0	1136836	728	1199	510	8.462	118.175	9.620	0.104
	ECC_{P_DSP}	2	594437	948	1457	564	9.935	100.654	5.906	0.104
	ECC_{P_LUT}	0	594437	950	1820	817	10.216	97.886	6.073	0.124

TABLE 7. Synthesis and performance results of the designed ECC processors for $n = 17$ (key length of 289 bits).

FPGA Family	ECC Processor	# of DSP48Es	Cycles Count	# of Slice Registers	# of Slice LUTs	# of Slices	Cycle time (ns)	Frequency (MHz)	Time (ms)	Power (W)
Virtex5	ECC_{S_DSP}	1	3200006	1104	1732	844	11.293	88.550	36.138	1.496
	ECC_{S_LUT}	0	3200006	1104	2179	976	14.051	71.169	44.963	1.543
	ECC_{P_DSP}	2	1654162	1393	2377	1111	13.239	75.534	21.899	1.558
	ECC_{P_LUT}	0	1654162	1393	3002	1167	15.876	62.988	26.261	1.601
Virtex7	ECC_{S_DSP}	1	3200006	1058	1433	646	9.597	104.199	30.710	0.219
	ECC_{S_LUT}	0	3200006	1075	1852	883	10.924	91.542	34.957	0.255
	ECC_{P_DSP}	2	1654162	1351	2010	1028	12.560	79.618	20.776	0.243
	ECC_{P_LUT}	0	1654162	1367	2570	1066	12.960	77.160	21.438	0.256
Artex7	ECC_{S_DSP}	1	3200006	1058	1384	525	11.665	85.727	37.328	0.105
	ECC_{S_LUT}	0	3200006	1063	1733	732	14.256	70.146	45.619	0.131
	ECC_{P_DSP}	2	1654162	1348	1963	808	14.671	68.162	24.268	0.126
	ECC_{P_LUT}	0	1654162	1352	2664	1173	17.504	57.130	28.954	0.155
Kintex7	ECC_{S_DSP}	1	3200006	1058	1369	450	8.876	112.663	28.403	0.102
	ECC_{S_LUT}	0	3200006	1063	1685	774	10.161	98.416	32.515	0.134
	ECC_{P_DSP}	2	1654162	1348	1963	725	10.461	95.593	17.304	0.130
	ECC_{P_LUT}	0	1654162	1351	2621	939	11.959	83.619	19.782	0.150

Detailed implementation results, using different FPGA families, for the proposed ECC processors are listed in Table 6, Table 7 and Table 8, for $n = 13, 17$ and 19, respectively. These results include the number of used DSPs, the number of required clock cycles to finish a point multiplication (N), the clock cycle time (T_c), and the area in terms of the number of LUTs and in terms of the number of occupied slices. The frequency F at which the ECC processor is running and the time (T) needed to finish one point multiplication operation are calculated using Equations 4 and 5, respectively.

$$F = 1/T_c \tag{4}$$

$$T = N \times T_c \tag{5}$$

It is notable that, for any key size, the serial ECC processors occupied less area than the parallel ECC processors. This is due to the fact that the DFTMM in the serial ECC processor uses only one base field multiplier and one base field adder, compared to the two base field adders and the two base field multipliers in the parallel ECC processor. This smaller area results in a shorter cycle time for the serial ECC processor and hence a higher frequency. However, the number of cycles needed to perform a point multiplication in the parallel ECC processor is almost half those needed by the serial ECC processor. Hence, the time for a point multiplication over the parallel ECC processor is shorter than that over the serial ECC processor.

In general, it can be noticed that the ECC processors where DSPs are utilized in the design of the BFM are faster than the ECC processors where LUTs are utilized in the design of the BFM.

When DSPs are utilized, the number of DSPs required by the serial ECC processor is 1 DSP while it is 2 DSPs for the parallel ECC processor. This is valid for the cases

TABLE 8. Synthesis and performance results of the designed ECC processors for $n = 19$ (key length of 361 bits).

FPGA Family	ECC Processor	# of DSP48Es	Cycles Count	# of Slice Registers	# of Slice LUTs	# of Slices	Cycle time (ns)	Frequency (MHz)	Time (ms)	Power (W)
Virtex5	ECC_{S_DSP}	2	4961248	1294	2245	801	13.443	74.388	66.694	1.526
	ECC_{S_LUT}	0	4961248	1294	2650	1124	14.755	67.774	73.203	1.570
	ECC_{P_DSP}	4	2554822	1617	2799	1297	14.993	66.698	38.304	1.607
	ECC_{P_LUT}	0	2554822	1617	3576	1529	16.871	59.273	43.102	1.673
Virtex7	ECC_{S_DSP}	2	4961248	1248	1812	751	10.842	92.234	53.790	0.237
	ECC_{S_LUT}	0	4961248	1266	2332	937	11.904	84.005	59.059	0.268
	ECC_{P_DSP}	4	2554822	1578	2318	1172	13.642	73.303	34.853	0.274
	ECC_{P_LUT}	0	2554822	1600	3072	1380	13.255	75.443	33.864	0.287
Artex7	ECC_{S_DSP}	2	4961248	1248	1852	830	12.845	77.851	63.727	0.133
	ECC_{S_LUT}	0	4961248	1249	2290	931	15.489	64.562	76.845	0.162
	ECC_{P_DSP}	4	2554822	1572	2279	1218	15.826	63.187	40.433	0.155
	ECC_{P_LUT}	0	2554822	1582	3069	1280	17.689	56.532	45.192	0.183
Kintex7	ECC_{S_DSP}	2	4961248	1248	1810	813	9.690	103.199	48.074	0.130
	ECC_{S_LUT}	0	4961248	1249	2198	725	10.763	92.911	53.398	0.144
	ECC_{P_DSP}	4	2554822	1572	2276	1171	12.842	77.869	32.809	0.164
	ECC_{P_LUT}	0	2554822	1579	3082	1190	12.274	81.473	31.358	0.177

when $n = 13$ or 17. However, when $n = 19$ the number of DSPs is 2 for the serial ECC processor and 4 for the parallel DSP processor. This due to the fact that the operand size for the embedded DSPs on the FPGA is 18-bit. To perform a 19×19 multiplication, a larger DSP component is built using two of the available DSP components on the FPGA.

Comparisons of the proposed ECC processors for $n = 13$, 17 and 19 against existing similar works are provided in Tables 9, 10, 11 and 12, over Virtex5, Virtex7, Artix7, and Kintex7 FPGA families, respectively. The comparisons are made based on the number of LUTs, the number of Slices, the frequency, the execution time, and the power dissipation. It should be pointed out that similar works in the literature report results over Virtex5 only. Therefore, for fair comparisons, we implement these works targeting Virtex7, Artix7, and Kintex7 FPGA families. Almost in all cases, the proposed ECC processors outperform all of the other similar ECC processors in the literature in all aspects.

For example, over Virtex5, compared to the existing lowest area ECC implementation over the same finite fields in the literature (*ECC* of [18]), our serial ECC processors without DSPs (ECC_{S_LUT}) occupy 19.07%, 26.06% and 17.41% less number of slices while at the same time being 2.17×, 2.92× and 3.57× faster for the key sizes of 169, 289 and 361 bits, respectively. Compared to the existing fastest ECC processors in the literature (ECC_8 of [15], [16] for the 169-bit key size, ECC_6 of [16] for the 289-bit key size, and ECC_8 of [16] for the 361-bit key size), our parallel ECC processors without DSPs (ECC_{P_LUT}) occupy 20.42%, 27.38% and 22.74% less number of slices while at the same time being 1.76×, 2.01× and 2.22× faster, respectively. Finally, again compared to the same existing fastest ECC processors in the literature, our parallel ECC processors that use DSPs are 1.93×, 2.41×, and 2.50× faster for the key sizes of 169, 289, and 361 bits, respectively. In terms of power, the proposed ECC_{S_LUT} consumes 8.09%, 12.68%, and 16.00% less power than the *ECC* of [18], for the key sizes of 169, 289 and 361 bits, respectively.

Similarly, over Virtex7 as shown by Table 10, the proposed ECC processors outperform all of the other ECC processors in all metrics. For example, for 17-bit (key = 289-bit), the proposed ECC_{S_LUT} achieves {reduction in the number of slices, speed up, reduction in power consumption} of {56.99%, 1.42, 49.90%}, {63.10%, 1.45, 51.24%}, {58.64%, 1.53, 50.96%}, {58.80%, 1.53, 53.80%}, {41.87%, 1.21, 27.56%}, {44.71%, 1.28, 29.36%}, {38.81%, 1.33, 28.57%}, {43.79%, 1.24, 28.17%}, {27.80%, 2.74, 40.70%}, and {49.25%, 1.49, 48.80%} comparing to ECC_1 of [16], ECC_2 of [16], ECC_3 of [16], ECC_4 of [16], ECC_5 of [16], ECC_6 of [16], ECC_7 of [16], ECC_8 of [16], *ECC* of [18], and *ECC* of [63], respectively. On the other hand, for the 19-bit (key = 361-bit), the proposed ECC_{P_LUT} gains {reduction in number of slices, speed up, reduction in power consumption} of {45.63%, 2.62, 55.09%}, {49.62%, 2.63, 54.01%}, {41.62%, 2.96, 57.92%}, {44.69%, 2.75, 55.91%}, {19.72%, 2.23, 29.31%}, {23.76%, 2.25, 29.48%}, {16.62%, 2.27, 29.48%}, {21.05%, 2.23, 28.61%}, {21.32%, 6.26, 51.93%}, and {34.66%, 2.80, 48.75%}, comparing to ECC_1 of [16], ECC_2 of [16], ECC_3 of [16], ECC_4 of [16], ECC_5 of [16], ECC_6 of [16], ECC_7 of [16], ECC_8 of [16], *ECC* of [18], and *ECC* of [63], respectively.

For 13-bit (key = 169-bit) over Artix7 FPGA family, it is found from Table 11 that comparing to *ECC* of [18] which scores the best area, the proposed ECC_{P_DSP} gains 14.43% reduction in number of slices, 3.77 speed up, and 47.47% reduction in power dissipation. It is also found that the proposed ECC_{P_DSP} gains 33.27% reduction in number of slices, 1.81 speed up, and 39.53% reduction in power dissipation comparing ECC_5 of [15] and [16] which is the fastest among all other works from the literature. Compared

TABLE 9. Comparison of the proposed ECC processors against existing similar works for $n = 13$, 17, and 19 over Virtex5 FPGA family.

	$n = 13$ (key size = 169-bit)					$n = 17$ (key size = 289-bit)					$n = 19$ (key size = 361-bit)				
Work	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)
ECC_1 [16], [15]*	2779	1237	38.5	15.3	1401	4324	1666	23.3	71.0	1907	5189	1727	18.0	141.6	2075
ECC_2 [16], [15]*	2917	1330	38.5	15.3	1404	4508	1431	24.0	69.0	1927	5352	2086	20.1	127.3	2100
ECC_3 [16], [15]*	2817	1039	38.6	15.3	1397	4487	1433	23.7	69.8	1966	5219	2018	21.7	117.5	1975
ECC_4 [16], [15]*	2745	1113	38.6	15.3	1396	4259	1758	24.9	66.4	1885	5101	1685	21.1	121.2	2086
ECC_5 [16], [15]*	2683	1249	40.1	14.7	1386	4075	1426	28.6	57.8	1735	4933	1799	25.7	99.2	1822
ECC_6 [16], [15]*	2801	1166	40.1	14.7	1388	4295	1607	31.3	52.9	1726	5145	1979	23.8	107.5	1870
ECC_7 [16], [15]*	2893	1320	41.9	14.1	1393	4432	1616	29.7	55.7	1788	5318	1780	23.6	108.1	1873
ECC_8 [16], [15]*	2764	1146	41.9	14.1	1388	4257	1657	28.4	58.3	1770	5239	1979	26.7	95.7	1827
ECC of [18]	1736	792	36.6	31.1	1595	3145	1320	24.4	131.4	1767	3751	1361	19.0	261.3	1869
ECC of [64]	2336	1001	32.0	18.4	1385	3521	1460	18.3	90.4	1842	4478	1506	15.5	164.9	2005
ECC_{S_DSP}	1280+ 1 DSP	691	94.3	12.0	1470	1732+ 1 DSP	844	88.6	36.1	1496	2245+ 2 DSPs	801	74.4	66.7	1526
ECC_{S_LUT}	1442	641	79.8	14.3	1466	2179	976	71.2	45.0	1543	2650	1124	67.8	73.2	1570
ECC_{P_DSP}	1688+ 2 DSPs	785	81.6	7.3	1500	2377+ 2 DSPs	1111	75.5	21.9	1558	2799+ 4 DSPs	1297	66.7	38.3	1607
ECC_{P_LUT}	2065	912	74.3	8.0	1536	3002	1167	63.0	26.3	1601	3576	1529	59.3	43.1	1673

* Only for $n = 13$

TABLE 10. Comparison of the proposed ECC processors against existing similar works for $n = 13$, 17, and 19 over Virtex7 FPGA family.

	$n = 13$ (key size = 169-bit)					$n = 17$ (key size = 289-bit)					$n = 19$ (key size = 361-bit)				
Work	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)
ECC_1 [16], [15]*	3323	1293	45.5	13.1	332	5742	2053	33.2	49.8	509	7119	2538	28.8	88.7	639
ECC_2 [16], [15]*	3352	1280	43.3	13.7	364	5884	2393	32.5	50.9	523	7202	2739	28.7	89.0	624
ECC_3 [16], [15]*	3531	1256	45.3	13.1	353	5561	2135	30.8	53.7	520	7485	2364	25.4	100.4	682
ECC_4 [16], [15]*	3582	1378	44.4	13.4	361	5936	2143	31.0	53.4	552	7460	2495	27.4	93.3	651
ECC_5 [16], [15]*	2570	1089	49.4	12.0	283	4018	1519	39.0	42.4	352	4926	1719	33.8	75.7	406
ECC_6 [16], [15]*	2530	1076	46.9	12.7	282	3985	1597	36.8	44.9	361	4665	1810	33.5	76.3	407
ECC_7 [16], [15]*	2683	1075	51.3	11.6	285	4121	1443	35.7	46.4	357	4843	1655	33.2	77.1	407
ECC_8 [16], [15]*	2549	1019	51.3	11.6	275	4073	1571	38.1	43.4	355	4711	1748	33.9	75.4	402
ECC [18]	2073	814	46.6	24.4	311	3315	1223	33.3	96.0	430	4829	1754	23.4	212.3	597
ECC [64]	3095	1210	45.6	13.0	328	5016	1740	31.8	52.1	498	5721	2112	26.9	95.0	560
ECC_S_DSP	1039+ 1 DSP	429	109.7	10.4	192	1433+ 1 DSP	646	104.2	30.7	219	1812+ 2 DSPs	751	92.2	53.8	237
ECC_S_LUT	1230	538	107.0	10.6	209	1852	883	91.5	35.0	255	2332	937	84.0	59.1	268
ECC_P_DSP	1471+ 2 DSPs	730	91.8	6.5	213	2010+ 2 DSPs	1028	79.6	20.8	243	2318+ 4 DSPs	1172	73.3	34.9	274
ECC_P_LUT	1867	810	86.6	6.9	230	2570	1066	77.2	21.4	256	3072	1380	75.4	33.9	287

* Only for $n = 13$

to best work from the literature in terms of power, which is ECC_6 of [15] and [16], the proposed ECC_{P_DSP} achieves 21.64% reduction in number of slices, 1.91 speed up, and 37.72% reduction in power dissipation.

On the other hand, for 17-bit (key = 289-bit), compared to the smallest ECC processor from the literature, which is the ECC of [18], the proposed ECC_{P_DSP} gains 30.52% reduction in number of slices, 4.65 speed up, and 60.38% reduction in power dissipation. Further, the proposed ECC_{P_DSP} achieves 46.38% reduction in number of slices, 2.15 speed up, and 47.93% reduction in power dissipation when it is compared against ECC_7 of [16], which is the fastest among other ECC processors from literature.

The results for 19-bit (key = 361-bit) over Artix7 also show a similar trend. The proposed ECC_{P_DSP} outperforms the best ECC processor from the literature in terms of area, the best ECC processor from the literature in terms of speed, and the best ECC processor from the literature in terms of power.

TABLE 11. Comparison of the proposed ECC processors against existing similar works for n = 13, 17, and 19 over Artix7 FPGA family.

	$n = 13$ (key size = 169-bit)					$n = 17$ (key size = 289-bit)					$n = 19$ (key size = 361-bit)				
Work	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)
ECC_1 [16], [15]*	3377	1116	37.7	15.8	224	5638	1984	25.2	65.7	425	7304	3203	22.6	112.9	532
ECC_2 [16], [15]*	3379	1241	34.5	17.2	239	5667	1955	25.7	64.5	423	7177	2456	21.8	117.3	568
ECC_3 [16], [15]*	3520	1264	35.2	16.9	236	5591	1935	24.9	66.4	410	7687	3338	21.6	118.5	529
ECC_4 [16], [15]*	3372	1211	35.8	16.6	245	6060	1985	25.2	65.7	448	7764	3424	21.9	116.8	520
ECC_5 [16], [15]*	2589	1004	40.1	14.8	172	4023	1343	30.3	54.5	232	4879	1741	27.1	94.2	288
ECC_6 [16], [15]*	2515	855	37.9	15.7	167	3854	1239	26.1	63.3	252	4659	1592	25.6	99.9	285
ECC_7 [16], [15]*	2671	890	39.4	15.1	176	4138	1507	31.8	52.1	242	4813	1617	25.7	99.4	290
ECC_8 [16], [15]*	2534	1013	38.9	15.3	170	3954	1349	27.2	60.8	251	4698	1677	25.7	99.3	278
ECC of [18]	2072	783	36.8	30.9	198	3284	1163	28.3	112.9	318	4709	1549	21.8	227.8	457
ECC of [64]	3085	1087	36.0	16.5	206	4910	1654	24.6	67.3	388	5755	2004	19.2	132.8	489
ECC_{S_DSP}	1025+ 1 DSP	433	91.0	12.5	88	1384+ 1 DSP	525	85.7	37.3	105	1852+ 2 DSPs	830	77.9	63.7	133
ECC_{S_LUT}	1234	504	81.0	14.0	104	1733	732	70.1	45.6	131	2290	931	64.6	76.8	162
ECC_{P_DSP}	1481+ 2 DSPs	670	72.5	8.2	104	1963+ 2 DSPs	808	68.2	24.3	126	2279+ 4 DSPs	1218	63.2	40.4	155
ECC_{P_LUT}	1837	725	68.5	8.7	121	2664	1173	57.1	29.0	155	3069	1280	56.5	45.2	183

* Only for $n = 13$

TABLE 12. Comparison of the proposed ECC processors against existing similar works for n = 13, 17, and 19 over Kintex7 FPGA family.

	$n = 13$ (key size = 169-bit)					$n = 17$ (key size = 289-bit)					$n = 19$ (key size = 361-bit)				
Work	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)	# LUTs	# Slices	Freq. (MHz)	Time (ms)	Power (mW)
ECC_1 [16], [15]*	2654	1056	53.5	11.1	196	4672	1649	34.9	47.3	351	5213	1979	28.1	90.8	419
ECC_2 [16], [15]*	2696	1168	53.4	11.1	196	4261	1722	32.9	50.3	346	5327	1899	27.8	91.8	434
ECC_3 [16], [15]*	2741	1005	52.8	11.2	201	4233	1674	33.4	49.5	323	4962	1903	29.0	88.2	382
ECC_4 [16], [15]*	2629	933	54.1	11.0	190	4253	1465	33.4	49.5	324	4889	1870	28.9	88.4	360
ECC_5 [16], [15]*	2561	934	53.8	11.0	168	3917	1363	39.6	41.8	240	4939	1814	35.8	71.4	302
ECC_6 [16], [15]*	2512	931	54.5	10.9	169	3991	1386	39.6	41.8	250	4610	1537	31.7	80.6	301
ECC_7 [16], [15]*	2636	870	53.4	11.1	174	4104	1356	38.8	42.7	256	4867	1729	36.3	70.3	292
ECC_8 [16], [15]*	2536	988	55.3	10.7	168	3795	1427	38.9	42.6	241	4532	1634	33.8	75.6	268
ECC of [18]	1509	712	40.4	28.2	164	2507	811	29.6	108.2	259	3077	968	24.3	204.0	274
ECC of [64]	2157	911	39.5	15.1	174	3244	1316	25.5	64.8	270	3897	1296	20.0	127.8	338
ECC_{S_DSP}	985+ 1 DSP	464	127.1	8.9	89	1369+ 1 DSP	450	112.7	28.4	102	1810+ 2 DSPs	813	103.2	48.1	130
ECC_{S_LUT}	1199	510	118.2	9.6	104	1685	774	98.4	32.5	134	2198	725	92.9	53.4	144
ECC_{P_DSP}	1457+ 2 DSPs	564	100.7	5.9	104	1963+ 2 DSPs	725	95.6	17.3	130	2276+ 4 DSPs	1171	77.9	32.8	164
ECC_{P_LUT}	1820	817	97.9	6.1	124	2621	939	83.6	19.8	150	3082	1190	81.5	31.4	177

* Only for $n = 13$

These are *ECC* of [18], ECC_5 of [16], and ECC_8 of [16], respectively. Comparing to these three ECC processors, the proposed ECC_{P_DSP} scores {reduction in number of slices, speed up, and % reduction in power dissipation} of {21.37%, 5.63, 66.08%}, {30.04%, 2.33, 46.18%}, and {27.37%, 2.46, 44.24%}, respectively.

Over Kintex7 FPGA family, the proposed ECC processors also show comparable results in terms of area, speed, and power dissipation for key lengths of 169, 289, and 361 bits. For 13-bit (key = 169-bit), the proposed ECC_{P_LUT} achieves 4.62 speed up with 24.39% reduction in power with only 14.74% extra slices comparing to *ECC* of [18]. Additionally, it scores 17.31% fewer slices, 1.75 speed up, and 26.19% reduction in power compared to the ECC_8 of [15] and [16]. For 17-bit (key = 289-bit), the proposed ECC_{P_LUT} consumes 15.78% extra slices with a speedup of 5.47 and 42.08% reduction in power compared to *ECC* of [18]. It also gains 2.11 speed up, 40% reduction in power, and 32.25% fewer

slices compared to ECC_6 of [16]. While, it achieves 31.11% reduction in slices, 37.5% reduction in power, and 2.11 speed up compared to ECC_5 of [16]. Similar findings are observed for the 19-bit (key = 361-bit).

Generally, it can be observed that the proposed ECC_{S_DSP} is faster than the proposed ECC_{S_LUT}. Similarly, the proposed ECC_{P_DSP} is faster than the proposed ECC_{P_LUT}. Therefore, for an application where the speed is critical, it is recommended to use the ECC_{P_DSP} if DSP components are available. If DSP components are not available on the FPGA, or if they are utilized for other tasks in the application, then it is recommended to use the proposed ECC_{P_LUT}. Moreover, if DSP components are available on the FPGA, it is recommended to use the proposed ECC_{S_DSP} for any application that requires a small area. Otherwise, the proposed ECC_{S_LUT} is recommended.

VI. CONCLUSION

We designed and implemented two ECC processors, one serial and one parallel, which can be utilized in IoT devices due to their secure, lightweight, and high-performance properties. We designed our ECC processors over prime characteristic finite fields using Edwards curves that enable a unified formula for the point addition and doubling operations. The adoption of Edwards curves, with the unified formula for point addition and doubling, made our ECC implementations both simpler and more secure against side-channel cryptanalysis. In our ECC implementations, we used projective coordinates to avoid the costly finite field inversion operation. We designed our finite field multiplier unit to carry our multiplication in the frequency domain utilizing a DFT-based modular multiplication algorithm. We investigated the use of the two different types of embedded multipliers on the FPGA: LUT-based multipliers and DSP-based multipliers. We described, verified, and synthesized our ECC processors using VHDL and the AMD-Xilinx FPGA tools. We synthesized and tested on multiple FPGA families. While our ECC processor designs that utilize FPGA-embedded DSP components for the multiplication operation run faster than those that utilize FPGA-embedded LUTs, all versions of our ECC processor designs excel in terms of both speed and area, compared to existing similar ECC processors in the literature. The proposed ECC processors can be deployed in a wide range of IoT applications such as smartphones, smart watches, security cameras, vehicles, industrial machinery, wildlife tracking, traffic monitoring, and natural disaster prediction.

REFERENCES

[1] S. R. J. Ramson, S. Vishnu, and M. Shanmugam, "Applications of Internet of Things (IoT)—An overview," in *Proc. 5th Int. Conf. Devices, Circuits Syst. (ICDCS)*, Mar. 2020, pp. 92–95.

[2] M. Morales-Sandoval, L. A. R. Flores, R. Cumplido, J. J. Garcia-Hernandez, C. Feregrino, and I. Algredo, "A compact FPGA-based accelerator for curve-based cryptography in wireless sensor networks," *J. Sensors*, vol. 2021, no. 1, Jan. 2021, Art. no. 8860413.

[3] *Cyber Security for Consumer Internet of Things: Baseline*, Standard ETSI TS 103 645, Eur. Telecommun. Standards Inst., 2024.

[4] P. K. Dhillon and S. Kalra, "Elliptic curve cryptography for real time embedded systems in IoT networks," in *Proc. 5th Int. Conf. Wireless Netw. Embedded Syst. (WECON)*, Oct. 2016, pp. 1–6.

[5] J. Joglekar, S. Bhutani, N. Patel, and P. Soman, "Lightweight elliptical curve cryptography (ECC) for data integrity and user authentication in smart transportation IoT system," in *Sustainable Communication Networks and Application*. Cham, Switzerland: Springer, 2020, pp. 270–278.

[6] Z. Liu and H. Seo, "IoT-NUMS: Evaluating NUMS elliptic curve cryptography for IoT platforms," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 720–729, Mar. 2019.

[7] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos, "Lightweight cryptography for embedded systems—A comparative analysis," in *Data Privacy Management and Autonomous Spontaneous Security*. Berlin, Germany: Springer, 2014, pp. 333–349.

[8] L.-Y. Yeh, P.-J. Chen, C.-C. Pai, and T.-T. Liu, "An energy-efficient dual-field elliptic curve cryptography processor for Internet of Things applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1614–1618, Sep. 2020.

[9] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya, "ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash," in *Proc. USENIX Secur. Symp.*, 2010, p. 14. [Online]. Available: https://api.semanticscholar.org/CorpusID:2514955

[10] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, Jan. 1983.

[12] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. Conf. Theory Appl. Cryptograph. Techn.*, vol. 218. Berlin, Germany: Springer, 1986, pp. 417–426.

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

[14] J. W. Bos, C. Costello, P. Longa, and M. Naehrig, "Selecting elliptic curves for cryptography: An efficiency and security analysis," *J. Cryptograph. Eng.*, vol. 6, no. 4, pp. 259–286, Nov. 2016.

[15] O. Al-Khaleel, S. Baktır, and A. Küpçü, "FPGA implementation of an ECC processor using Edwards curves and DFT modular multiplication," in *Proc. 12th Int. Conf. Inf. Commun. Syst. (ICICS)*, 2021, pp. 344–351.

[16] O. Al-Khaleel, S. Baktir, and A. Küpçü, "An ECC processor for IoT using Edwards curves and DFT modular multiplication," *Cluster Comput.*, vol. 26, no. 2, pp. 1063–1075, Apr. 2023.

[17] S. Ullah, J. Zheng, N. Din, M. T. Hussain, F. Ullah, and M. Yousaf, "Elliptic curve cryptography; applications, challenges, recent advances, and future trends: A comprehensive survey," *Comput. Sci. Rev.*, vol. 47, Feb. 2023, Art. no. 100530.

[18] S. Baktır, S. Kumar, C. Paar, and B. Sunar, "A state-of-the-art elliptic curve cryptographic processor operating in the frequency domain," *Mobile Netw. Appl.*, vol. 12, no. 4, pp. 259–270, Oct. 2007.

[19] J. Bos, M. Kaihara, T. Kleinjung, A. Lenstra, and P. Montgomery, "On the security of 1024-bit rsa and 160-bit elliptic curve cryptography," Cryptol. ePrint Arch., IACR, USA, Tech. Rep. 2009/389, 2009.

[20] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, 1st ed. New York, NY, USA: Springer, 2004.

[21] L. C. Washington, *Elliptic Curves Number Theory and Cryptography*, 2nd ed., London, U.K.: Chapman & Hall, 2008.

[22] H. Edwards, "A normal form for elliptic curves," *Bull. Amer. Math. Soc.*, vol. 44, pp. 393–422, Jul. 2007.

[23] D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *Advances in Cryptology—ASIACRYPT*, K. Kurosawa, Ed., Berlin, Heidelberg: Springer, 2007, pp. 29–50.

[24] B. Chevallier-Mames, M. Ciet, and M. Joye, "Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity," *IEEE Trans. Comput.*, vol. 53, no. 6, pp. 760–768, Jun. 2004.

[25] K. Javeed, A. El-Moursy, and D. Gregg, "EC-crypto: Highly efficient area-delay optimized elliptic curve cryptography processor," *IEEE Access*, vol. 11, pp. 56649–56662, 2023.

[26] 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.

[27] K. Javeed, A. El-Moursy, and D. Gregg, "E^2CSM: Efficient FPGA implementation of elliptic curve scalar multiplication over generic prime field GF(p)," *J. Supercomput.*, vol. 80, no. 1, pp. 50–74, 2023.

[28] 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.

[29] K. Javeed and A. El-Moursy, "Area-time efficient point multiplication architecture on twisted Edwards curve over general prime field GF(p)," *Int. J. Circuit Theory Appl.*, vol. 51, no. 12, pp. 5962–5979, Dec. 2023.

[30] A. M. Awaludin, J. Park, R. W. Wardhani, and H. Kim, "A high-performance ECC processor over curve 448 based on a novel variant of the Karatsuba formula for asymmetric digit multiplier," *IEEE Access*, vol. 10, pp. 67470–67481, 2022.

[31] Y. A. Shah, K. Javeed, M. I. Shehzad, and S. Azmat, "LUT-based high-speed point multiplier for goldilocks-curve 448," *IET Comput. Digit. Techn.*, vol. 14, no. 4, pp. 149–157, Jul. 2020.

[32] K. Javeed, "FPGA implementation of area-time aware ECC scalar multiplication core," in *Proc. 30th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, 2023, pp. 1–4.

[33] M. Kaur, A. A. AlZubi, T. S. Walia, V. Yadav, N. Kumar, D. Singh, and H.-N. Lee, "EGCrypto: A low-complexity elliptic Galois cryptography model for secure data transmission in IoT," *IEEE Access*, vol. 11, pp. 90739–90748, 2023.

[34] A. R. Alharbi, M. M. Hazzazi, S. S. Jamal, A. Aljaedi, A. Aljuhni, and D. J. Alanazi, "DCryp-Unit: Crypto hardware accelerator unit design for elliptic curve point multiplication," *IEEE Access*, vol. 12, pp. 17823–17835, 2024.

[35] K. Javeed and D. Gregg, "Point multiplication accelerator for arbitrary Montgomery curves," *IEEE Embedded Syst. Lett.*, early access, May 9, 2024, doi: 10.1109/LES.2024.3399071.

[36] R. Abarzúa, S. Martínez, V. Mendoza, and N. Thériault, "Same value analysis on Edwards curves," *J. Cryptograph. Eng.*, vol. 10, no. 1, pp. 27–48, Apr. 2020.

[37] D. J. Bernstein, "Curve25519: New Diffie–Hellman speed records," in *Proc. Int. Workshop Public Key Cryptogr.*, 2006, pp. 207–228.

[38] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 967–980.

[39] M. Hamburg, "Ed448-Goldilocks, a new elliptic curve," IACR Cryptol. ePrint Arch., USA, Tech. Rep. 2015/625, 2015.

[40] J. Renes, C. Costello, and L. Batina, "Complete addition formulas for prime order elliptic curves," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2016, pp. 403–428.

[41] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 1999, pp. 388–397.

[42] S. Martínez, D. Sadornil, J. Tena, R. Tomàs, and M. Valls, "On Edwards curves and ZVP-attacks," *Applicable Algebra Eng., Commun. Comput.*, vol. 24, pp. 507–517, Dec. 2013.

[43] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, "Elliptic curve cryptography in practice," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2014, pp. 157–175.

[44] J.-C. Faugère, L. Perret, C. Petit, and G. Renault, "Improving the complexity of index calculus algorithms in elliptic curves over binary fields," in *Proc. EUROCRYPT*, 2012, pp. 27–44.

[45] O. Al-Khaleel, Z. Al-Qudah, M. Al-Khaleel, and C. Papachristou, "High performance FPGA-based decimal-to-binary conversion schemes for decimal arithmetic," *Microprocessors Microsyst.*, vol. 37, no. 3, pp. 287–298, May 2013.

[46] O. Al-Khaleel, Z. Al-Qudah, M. Al-Khaleel, R. Bani-Hani, C. Papachristou, and F. Wolff, "Efficient hardware implementations of binary-to-BCD conversion schemes for decimal multiplication," *J. Circuits, Syst. Comput.*, vol. 24, no. 2, Feb. 2015, Art. no. 1550019.

[47] A. A.-R. Al-Nounou, O. Al-Khaleel, F. Obeidat, and M. Al-Khaleel, "FPGA implementation of fast binary multiplication based on customized basic cells," *J. Universal Comput. Sci. (JUCS)*, vol. 28, no. 10, pp. 1030–1057, Oct. 2022.

[48] M. Joye and S.-M. Yen, "The Montgomery powering ladder," in *Cryptographic Hardware and Embedded Systems—CHES 2002* (Lecture Notes in Computer Science), B. S. Kaliski, ç. K. Koç, and C. Paar, Eds., Berlin, Germany: Springer, 2003, pp. 291–302.

[49] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, p. 243, Jan. 1987.

[50] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965.

[51] A. Schönhage and V. Strassen, "Schnelle multiplikation großer zahlen," *Computing*, vol. 7, nos. 3–4, pp. 281–292, Sep. 1971.

[52] M. Fürer, "Faster integer multiplication," *SIAM J. Comput.*, vol. 39, no. 3, pp. 979–1005, Jan. 2009.

[53] S. Baktır and B. Sunar, "Finite field polynomial multiplication in the frequency domain with application to elliptic curve cryptography," in *Proc. Int. Symp. Comput. Inf. Sci. (ISCIS)*, 2006, pp. 991–1001.

[54] S. Baktır, "Frequency domain finite field arithmetic for elliptic curve cryptography," PhD dissertation, Worcester Polytech. Inst., 2008.

[55] M. 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.

[56] M. M. Islam, M. S. Hossain, M. Shahjalal, M. 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.

[57] J. Li, S. Zhong, Z. Li, S. Cao, J. Zhang, and W. Wang, "Speed-oriented architecture for binary field point multiplication on elliptic curves," *IEEE Access*, vol. 7, pp. 32048–32060, 2019.

[58] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Lightweight elliptic curve cryptography accelerator for Internet of Things applications," *Ad Hoc Netw.*, vol. 103, Jun. 2020, Art. no. 102159.

[59] M. A. Mehrabi, C. Doche, and A. Jolfaei, "Elliptic curve cryptography point multiplication core for hardware security module," *IEEE Trans. Comput.*, vol. 69, no. 11, pp. 1707–1718, Nov. 2020.

[60] X. Hu, X. Li, X. Zheng, Y. Liu, and X. Xiong, "A high speed processor for elliptic curve cryptography over NIST prime field," *IET Circuits, Devices Syst.*, vol. 16, no. 4, pp. 350–359, Jul. 2022.

[61] C.-Y. Lee, M. Zeghid, A. Sghaier, H. Y. Ahmed, and J. Xie, "Efficient hardware implementation of large field-size elliptic curve cryptographic processor," *IEEE Access*, vol. 10, pp. 7926–7936, 2022.

[62] Y. Xie, Y. Liu, X. Zheng, W. Zhu, J. Li, J. Li, S. Cai, and X. Xiong, "A dual-core high-performance processor for elliptic curve cryptography in GF(p) over generic weierstrass curves," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 11, pp. 4523–4527, Nov. 2022.

[63] N. Mentens, L. Batina, and S. Baktır, "An elliptic curve cryptographic processor using Edwards curves and the number theoretic transform," in *Proc. Int. Conf. Cryptogr. Inf. Secur. Balkans*, 2014, pp. 94–102.

[64] H.-Y. Chen, K.-Y. Peng, and K.-J. Lee, "A novel unified modular arithmetic unit for elliptic curve cryptography," in *Proc. Int. VLSI Symp. Technol., Syst. Appl. (VLSI-TSA/VLSI-DAT)*, Apr. 2023, pp. 1–4.

[65] J. Ding, S. Li, and Z. Gu, "High-speed ECC processor over NIST prime fields applied with Toom–Cook multiplication," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 3, pp. 1003–1016, Mar. 2019.

[66] D. V. Bailey and C. Paar, "Optimal extension fields for fast arithmetic in public-key algorithms," in *Proc. Annu. Int. Cryptol. Conf.*, pp. 472–485, 1998.

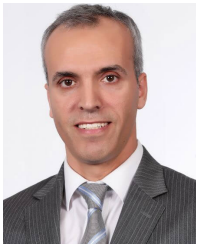
[67] D. V. Bailey and C. Paar, "Efficient arithmetic in finite field extensions with application in elliptic curve cryptography," *J. Cryptol.*, vol. 14, no. 3, pp. 153–176, Jun. 2001.



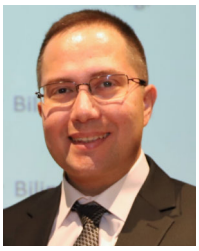
OSAMA AL-KHALEEL received the B.S. degree in electrical engineering from Jordan University of Science and Technology, Irbid, Jordan, in 1999, and the M.Sc. and Ph.D. degrees in computer engineering from Case Western Reserve University, Cleveland, OH, USA, in 2003 and 2006, respectively. He is currently an Associate Professor of computer engineering with the Department of Computer Engineering, Jordan University of Science and Technology. His main research interests include embedded systems design, reconfigurable computing, computer arithmetic, cryptography, hardware accelerators, and logic design.



SELÇUK BAKTIR (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from Bilkent University, Ankara, Türkiye, in 2001, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from Worcester Polytechnic Institute, MA, USA, in 2003 and 2008, respectively. He worked for companies including IBM T. J. Watson Research Center and Intel Corporation. He was a Visiting Researcher with the Chair for Embedded Security, Ruhr University Bochum, Germany. In 2013, he founded the M.Sc. Program in cybersecurity at Bahçeşehir University, İstanbul, Türkiye. He is currently a Faculty Member with the Computer Engineering Department, American University of the Middle East, Kuwait. His research interests include applied cryptography and computer security. He received the IBM Research Pat Goldberg Memorial Best Paper Award, in 2007, the European Union FP7 Marie Curie IRG Award, in 2010, and the TUBITAK Career Award, in 2016.



MOHAMMAD AL-KHALEEL received the M.Sc. and Ph.D. degrees in applied mathematics (numerical analysis and scientific computing) from McGill University, Montreal, QC, Canada, in 2003 and 2007, respectively. In 2007, he became an Assistant Professor of mathematics with the Department of Mathematics, Yarmouk University, Jordan, and afterwards was promoted to an Associate Professor. He is currently an Associate Professor with the Department of Mathematics, Khalifa University, United Arab Emirates. His research is multidisciplinary but the bulk of it has revolved around developing and implementing efficient and parallel numerical methods and analytical methods for solving differential equations (ordinary, partial, and fractional) that appear in many applications in different fields, including seismic waves, nanofluid and microfluid dynamics, and circuit simulations. He has also research interest in other areas of mathematics including topics in metric fixed point theory, fractional calculus, numerical parameter optimization, inverse and control problems, and the mathematical background of computer arithmetic circuits.



ALPTEKİN KÜPÇÜ (Senior Member, IEEE) received the Ph.D. degree from the Computer Science Department, Brown University, in 2010. Since then, he has been a Faculty Member with Koç University, leading the Cryptography, Cyber Security and Privacy Research Group that he founded. His research mainly focuses on provably-secure applied cryptography, with applications to cloud security, privacy-preserving and adversarial machine learning, peer-to-peer networks, blockchains, and game theory and mechanism design. He has various accomplishments, including eight international patents granted, 15 funded research projects, two European Union COST Action Management Committee Memberships, four Outstanding Teaching Awards, four Outstanding Young Scientist Awards, ACM and IEEE Senior Member Awards, and the Royal Society of U.K. Newton Advanced Fellowship. He is also a Co-Founder of Xtinge Technology Inc.