IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 March 2024, accepted 14 April 2024, date of publication 17 April 2024, date of current version 24 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3390244

RESEARCH ARTICLE

A Universal Single and Double Point Multiplications Architecture for ECDSA Based on Differential Addition Chains

XIANG HE[1], WEIJIANG WANG[1,2], JINGQI ZHANG[1], ZHANTAO ZHANG[1], JIANLEI YANG[3], HUA DANG[1], AND GUIYU WANG[1,2]

[1]School of Integrated Circuits and Electronics, Beijing Institute of Technology (BIT), Beijing 100081, China
[2]BIT Chongqing Institute of Microelectronics and Microsystems, Chongqing 400031, China
[3]Hebei Petroleum University of Technology, Hebei 067000, China

Corresponding author: Guiyu Wang (guiyu.wang@bit.edu.cn)

This work was supported in part by Chongqing Natural Science Foundation under Grant cstc2021jcyj-msxmX1090, and in part by the National Natural Science Foundation of China under Grant 62201039.

ABSTRACT In the 5G and beyond networks, low-latency digital signatures are essential to ensure the security, integrity, and non-repudiation of massive data in communication processes. The binary finite field-based elliptic curve digital signature algorithm (ECDSA) is particularly suitable for achieving low-latency digital signatures due to its carry-free characteristics. This paper proposes a low-latency and universal architecture for point multiplication (PM) and double point multiplication (DPM) based on the differential addition chain (DAC) designed for signing and verification in ECDSA. By employing the DAC, the area-time product of DPM can be decreased, and throughput efficiency can be increased. Besides, the execution pattern of the proposed architecture is uniform to resist simple power analysis and high-order power analysis. Based on the data dependency, two Karatsuba–Ofman multipliers and four non-pipeline squarers are utilized in the architecture to achieve a compact timing schedule without idle cycles for multipliers during the computation process. Consequently, the calculation latency of DPM is minimized to five clock cycles in each loop. The proposed architecture is implemented on Xilinx Virtex-7, performing DPM in 3.584, 5.656, and 7.453 μs with 8135, 13372, and 17898 slices over GF(2^{163}), GF(2^{233}), GF(2^{283}), respectively. In the existing designs that are resistant to high-order analysis, our architecture demonstrates throughput efficiency improvements of 36.7 % over GF(2^{233}) and 9.8% over GF(2^{283}), respectively.

INDEX TERMS Elliptic curve cryptosystems, differential addition chain, point multiplication, double point multiplication, field-programmable gate array.

I. INTRODUCTION

A. BACKGROUND

The 5G and beyond networks achieve ultra-high data transfer rates, ultra-low latency, and super-dense connections [1]. These characteristics make scenarios such as vehicle-to-everything (V2X), edge computing, wireless data centers, etc, possible. In these scenarios, there is a need for end-to-end transmission of massive data involving super-dense devices [2]. Those transmissions between super-dense devices include highly secure and private required data such as personal information, medical data, financial transactions, location data in V2X, etc [3]. Efficient digital signatures are necessary to prevent leakage or malicious tampering of this transmitted data during the communication process. The Digital Signature Algorithm (DSA) is an encryption technique used to ensure the integrity of digital information, authenticate the sender's identity, and resist

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

repudiation [4]. Rivest-Shamir-Adleman (RSA), invented by Rivest and Adleman [5], and elliptic curve cryptosystems (ECC), invented by Koblitz [6] and Miller [7], stand out as prevalent techniques employed in digital signature algorithms. Although RSA and ECC may face potential risks when confronted with quantum searching algorithms in the future post-quantum era, both of them remain a practical and widely adopted solution in contemporary times [8]. Moreover, ECC achieves equivalent security to RSA with shorter key lengths, implying higher efficiency in digital signatures and reduced computational requirements storage overhead [9], which means higher computation speed and less resource consumption. Therefore, the elliptic curve digital signature algorithm (ECDSA) [10] is more suitable for digital signatures.

In ECDSA, there are two steps: signing and verification. Signing involves encrypting the message using the private key to generate a digital signature through point multiplication (PM). Verification consists of performing a double point multiplication (DPM), using the corresponding public key to generate the signature during the signing, and checking if the received signature matches the message [10]. The elliptic curve point multiplication (ECPM) is a core computational step in ECDSA. Currently, ECPM over prime finite fields GF(p) offers better security than over binary finite fields GF(2^m) and exhibits enhanced robustness against side-channel attacks [11], [12]. However, due to the mathematical characteristics of the prime field, the complexity of carry chains in prime-field operations leads to higher latency than the binary field [13]. Therefore, the carry-free feature of the binary field makes binary-field-based ECPM more suitable for high-performance and low-latency application scenarios. Current research on ECPM primarily focuses on two areas: DPM and PM. In the PM field, researchers attempt to achieve acceleration effects by optimizing the multiplier in PM and the scheduling scheme of the multiplier. In the DPM field, researchers use dedicated algorithms to implement DPM. However, either of the two strategies only accelerates hardware in their respective fields. If there is a universal architecture tailored for DPM, capable of computing DPM as well as PM, it would significantly reduce the additional hardware resource overhead and achieve circuit reuse.

B. CONTRIBUTION

In this paper, we propose a universal PM and DPM architecture suitable for ECDSA based on differential addition chain(DAC) over binary finite fields. The main contributions of this paper are as follows:

1) We present an algorithm that compresses the computation of two PMs into the latency of one PM, reducing the computational load of the two PMs. In the PM that is based on the Montgomery algorithm, if we assume that the computation of kP and lQ requires n iterations, calculating $kP + lQ$ requires a total of $2n + 1$ point additions(PAs), $2n$ point doublings(PDs), and $2n + 1$ iterations. In contrast, DPM requires only $2n+1$ PAs and n PDs. Moreover, the final result can be obtained in $n + 1$ iterations. This algorithm not only reduces the overall computational workload but also minimizes the computational latency.
2) The proposed algorithm possesses the capability to resist Simple Power Analysis(SPA) by performing a uniform PA-PD-PA pattern in each iteration without pseudo operations. Since all operations are real, the proposed algorithm also shows resistance to correlation-based high-order power analysis.
3) We propose a universal architecture that performs both PM and DPM. This architecture is not only suitable for both PM and DPM but also accelerates PM by reducing the number of iterations, thereby decreasing the latency of PM. The reuse of the DPM architecture with the PM design has been implemented, avoiding additional hardware resource overhead.
4) We analyzed the data dependency and identified the crucial data path in our architecture. We arranged two PMs and one PA in five clock cycles using two two-stage pipelined multipliers based on the Karatsuba–Ofman algorithm and four non-pipelined squarers. By inserting buffers into the design, we avoided the generation of critical paths. Additionally, there is no idle time for multipliers when calculating DPM, reducing the latency and improving the throughput.

C. STRUCTURE

The remaining sections of this article are organized as follows. In Section II, we introduce the background knowledge of ECPM and DAC. What's more, we introduce relevant works and outlined the motivation behind this paper. Section III presents an algorithm for constructing DAC suitable for PM and DPM. In Section IV, we analyze the data dependency relationships in the addition chain and optimize the timing schedule based on these dependencies. Section V discusses the proposed generic architecture for PM and DPM. Section VI compares our implemented results with existing works. Finally, Section VII concludes our work.

II. BACKGROUND AND MOTIVATION

A. ELLIPTIC CURVE POINT MULTIPLICATION

Fig. 1. illustrates the computational steps involved in ECDSA, along with the key computational units required for these operations.

ECPM is a pivotal step in both the creation and verification of signatures. The signing process is executed through PM, while the verification process is carried out via DPM. By scheduling PA and PD operations, the results of DPM or PM can be achieved. For either PM or DPM, it's essential to derive the formulas for computing PA and PD on a specific elliptic curve. PA and PD are performed over a binary finite field, introducing computational complexity due to the

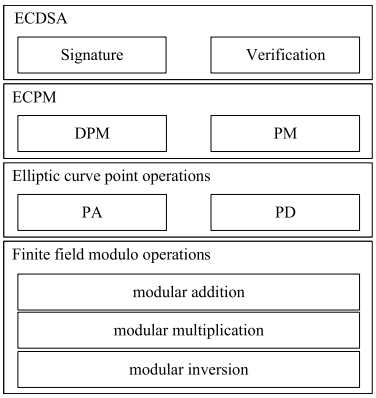

FIGURE 1. The hierarchical structure of ECDSA.

modulus operations of modular addition, multiplication, and inversion involved in the calculation process.

Our design utilizes a curve recommended by NIST [14], [15]. curve over $GF(2^m)$, denoted as E, is defined by Equation (1):

$$E : y^2 + xy = x^3 + ax^2 + b \tag{1}$$

PM involves computing the point kP on an elliptic curve, where $P(x_p, y_p)$ is a base point and k is an integer input. DPM involves computing the point $kP + lQ$ on an elliptic curve, where $P(x_p, y_p)$ and $Q(x_Q, y_Q)$ are two base points and k and l are integer inputs. In almost all algorithms, both PM and DPM require the use of PA and PD operations, which include expensive modular inversion operation when used in affine or mixed coordinates. However, by executing a base conversion from the affine coordinate to the LD projective one through Equation (2), only the final step of the PM or DPM requires the computation of the modular inversion operation, making the process more efficient.

$$(x, y) \rightarrow \{(X, Y, Z)|x = \frac{X}{Z}, y = \frac{Y}{Z}\} \tag{2}$$

During the PA and PD operations, modular multiplication, modular addition, and modular inversion can be simplified because of the carry-free feature of binary finite fields. In binary finite fields, modular addition and modular subtraction essentially involve performing exclusive-OR operations on the coefficients of two polynomials. Therefore, modular addition and modular subtraction can be considered the same operation. Modular multiplication consists of polynomial multiplication and modular reduction. After performing polynomial multiplication, utilizing a NIST-recommended polynomial $a(x)$ for a modular reduction on the result allows obtaining a modular multiplication result of m bits. Modular inversion involves finding $b(x)$ in $a(x) \times b(x)\ mod\ f(x) = 1$ with a given $a(x)$.

B. PA AND PD BASED ON DIFFERENTIAL ADDITION CHAIN

There are four points C_1, C_2, C_3, and C_4 on the elliptic curve. DAC refers to the existence of a corresponding difference pair for each addition operation $C_3 = C_2 + C_1$ in the addition chain, i.e., $C_4 = C_2 - C_1$ in the difference addition chain. The elimination of the calculation of the Y-coordinate is an inherent problem in ECC since PM and DPM are composed of PA and PD operations. In each iteration, each PM operation involving points $P_1(X_1, Y_1, Z_1), P_2(X_2, Y_2, Z_2)$, $P_3(X_3, Y_3, Z_3)$ in LD coordinate, and $P_{diff}(x_{diff}, y_{diff})$ in affine coordinate, forms a difference chain, and for each addition operation $P_3 = P_2 + P_1$, there exists a known $P_{diff} = P_2 - P_1$. The addition chains present in PM allow the Y-coordinate to be ignored during computation. The simplified point addition equation is shown in Equation (3).

$$\begin{cases} X_3 = x_{diff}(X_1Z_2 + X_2X_1)^2 + X_1^4 + X_1X_2Z_1Z_2 \\ Z_3 = (X_1Z_2 + X_2Z_1)^2 \end{cases} \tag{3}$$

When calculating $P_3 = P_2 + P_1$, if $P_2 = P_1$, then $P_3 = 2P_1$, which is a PD. In the LD coordinate system, when calculating $P_3 = 2P_1$, the Y-coordinate can also be omitted. The simplified point doubling equation is as shown in Equation(4).

$$\begin{cases} X_3 = X_1^4 + bZ_1^4 \\ Z_3 = X_1^2Z_1^2 \end{cases} \tag{4}$$

Due to the properties of the DAC, we can omit the calculation of the Y-coordinate when computing the PM. Obtaining the final PM result always involves PA operations, which allows us to recover the y-coordinate. We can retrieve the LD projective coordinates to the affine coordinate system using Equation (5). Through the LD coordinates of points P_1 and P_2, and the affine coordinates of the difference P_{diff} between P_1 and P_2, we can recover the y-coordinate of point $P_3(x_3, y_3)$.

$$\begin{aligned} y_3 = \frac{(x_{diff} + X_1Z_1)}{x_{diff}Z_1Z_2}\Big[&(x_{diff}^2 + y_{diff})(Z_1Z_2) \\ &+(X_1 + x_{diff}Z_1)(X_2 + x_{diff}Z_2)\Big] + y_{diff} \end{aligned} \tag{5}$$

C. RELATED WORKS

DPM is more complex than PM, and there are three approaches to computing DPM. The first approach involves obtaining DPM using Straus-Shamir's trick [16] and interleaving [17], which cannot resist SPA because its power during the computation process is not uniform.

The second approach to computing DPM is performing two PMs and one PA. Reducing the latency of PM can also achieve the goal of reducing the latency of DPM. In PM, a large number of modular multiplications are required, and designing the modular multiplier can be an effective way to reduce latency. There are currently two main types of multiplier designs: bit-serial and bit-parallel multipliers [18]. Pillutla and Boppana [19] proposed a digit-serial modular multiplier over GF(2^m). Bit-serial multipliers result in many clock cycles, leading to substantial delays. However, serial multipliers can significantly reduce system area, making them applicable in scenarios with limited resources despite

sacrificing some delay. To mitigate latency, almost all current designs opt for bit-parallel multipliers. Sajid et al. [20] proposed a simplified formulation using a single-instruction-with-single arithmetic operation and a 32-bit digit-parallel multiplier that decreases clock cycles, which provides higher throughput. Khan and Benaissa [21] proposed a novel two-stage pipelined full-precision multiplier with scheduling for the combined Montgomery PM algorithm to decrease PM latency highly. Li et al. [22] proposed an architecture that is comprised of two parallel balanced full-precision multipliers to reduce operation latency. Reference [23] proposed a low latency window-based enhanced comb method to decrease the latency in PM. Zhou et al. [24] proposed an efficient implementation of bit-parallel finite field multipliers by analyzing the complexity of the Karatsuba–Ofman algorithm [25]. In addition to optimizing the PM, all of [21] and [26] analyze the data dependency during the process to enhance the performance of PM by increasing parallelism. The above method requires two step-by-step PMs in a single basic PM unit. If there is a structure that can process two PMs in parallel, it would evidently reduce the latency of DPM.

The third approach directly calculates DPM by scanning both k and l simultaneously. Khabbazzian et al. [27] proposed a technique for bandwidth and memory to speed up DPM. Through precomputing, they recoded two scalars in DPM to a suitable integer representation, which made it easy to parallel the process, and they stored some multiples of two points in memory. Adikari et al. [28] use the joint two-dimensional Frobenius expansion, which can improve performance in computing scalar multiplication in Koblitz curves, to decrease the number of PA, which is more complex. They both reduced the number of PA during the computation through algorithms, but the presence of the Y-coordinate still makes its computation a complex challenge.

By employing DAC, the computation of the Y-coordinate can be omitted, which greatly reduces the complexity of the calculation. Bernstein [29] proposed a binary DAC that consists of two PAs and one PD in each iteration. It is a constant-time algorithm that can be used to calculate DPM. Azarderakhsh and Karabina [30], [31] implemented DPM in hardware based on the DJB algorithm [29], the AK algorithm [31] and the JT algorithm [32]. They first used DAC in the hardware architecture to implement DPM. It demonstrated that using the DJB algorithm for DPM results in the minimum latency while employing the AK algorithm, which achieves the smallest area. Due to a more efficient arrangement of timing and less utilization of multiplier resources, redundancy may exist in the multiplier resources. Shahroodi et al. [33] proposed an architecture with a modified DAC that makes decisions based on 3 bits of scalar at one iteration to calculate DPM, reducing the number of PA and PD operations in each round of the DAC iteration process.

D. MOTIVATION

In the works [20], [21], [22], they used the Montgomery ladder to calculate PM, which included one PA and one PD in each iteration. Similarly, when using DAC to compute DPM, there are two PAs and one PD in each iteration [29]. The computational processes present structural similarity between them, and by arranging them properly, we can use one PD and one PA from DAC to construct the PM computation architecture, thus achieving structural reuse. The work [34] points out this aspect, but it does not provide a specific implementation method.

Using DAC not only enables a universal architecture for DPM and PM but also reduces the latency of DPM. The reported works [20], [21], [22], [23], [24], [26] have primarily focused on improving DPM's computation speed by reducing PM latency. These works have reduced the latency of PM by designing high-performance modular multiplication units and increasing the parallelism of modular multiplication by analyzing the data dependency in DPM. If using PM architecture to compute DPM over GF(2^m), it requires $2m+1$ PAs and $2m$ PDs, while in the case of using the DAC, only $2m+1$ PAs and $2m$ PDs are needed, saving on m PDs. Therefore, DAC can effectively reduce the latency of DPM.

The motivation of this paper is to reduce the computational latency of DPM by leveraging the characteristics of DAC. Additionally, a universal architecture for both PM and DPM can be implemented by using DAC, allowing for the reuse of hardware resources.

III. PROPOSED UNIVERSAL ARCHITECTURE FOR PM AND DPM

In the Montgomery algorithm, each iteration involves PA and PD. Now we have points on the elliptic curve, P_1, P_2, P_3, and P_{diff}. For each point addition $P_3 = R_1 + R_2$ in the computation, $P_{diff} = R_1 - R_2$ exists, allowing for the omission of the Y-coordinate during the computation. Based on the Montgomery algorithm, we propose an algorithm with a structure similar to the Montgomery algorithm. The calculation of PM and DPM is achieved through the generation of a two-dimensional DAC.

When executing $kP + lQ$, we can consider (k, l) as the initial value for the first iteration. When generating a two-dimensional DAC, at each iteration, the existence of a pair (k_i, l_i) allows the calculation of (k_i, l_i), (k_i, l_i+1), (k_i+1, l_i), and (k_i+1, l_i+1) for this iteration [29]. The three elements in the current iteration of the DAC, along with one missing element, are obtained from (k_i, l_i), (k_i, l_i+1), (k_i+1, l_i), and (k_i+1, l_i+1). Due to parity, it is evident that the elements can only be obtained from (odd, odd), $(even, even)$, $(odd, even)$, and $(even, odd)$. Moreover, the missing element that is decided by $(k_{i-1}+k_i, l_{i-1}+l_i)$, where $(k_{i-1}, l_{i-1}) = ([k_i/2], [l_i/2])$, is always chosen as either $(even, even)$ or $(odd, even)$.

1) when $(k_{i-1}+k_i, l_{i-1}+l_i) = (odd, odd)$, the choice is same as previous iteration;

2) when $(k_{i-1}+k_i, l_{i-1}+l_i) = (even, even)$, the choice is opposite as previous iteration;
3) when $(k_{i-1}+k_i, l_{i-1}+l_i) = (odd, even)$, the choice is $(even, odd)$.
4) when $(k_{i-1}+k_i, l_{i-1}+l_i) = (even, odd)$, the choice is $(odd, even)$.

Algorithm 1 The Two-Dimensional *DAC* Generation Algorithm

Require: two-dimensional input vector (k, l)
Ensure: $V_i^{(1)}, V_i^{(2)}, V_i^{(3)}$
$n = max(\lceil log_2 k \rceil, \lceil log_2 l \rceil)$
$k_n = k, l_n = l$
$D_n = k_n \ mod\ 2$
$V_n^{(1)} = (k_n + (k_n+1) \ mod\ 2, l_n + (l_n+1) \ mod\ 2)$
$V_n^{(2)} = (k_n + k_n \ mod\ 2, l_n + l_n \ mod\ 2)$
$V_n^{(3)} = (k_n + (k_n + D_n) \ mod\ 2, l_n + (l_n + D_n + 1) \ mod\ 2)$
for $i = n-1$ to 0 **do**
  Set $(k_i, l_i) = (\lfloor k_{i+1}/2 \rfloor, \lfloor l_{i+1}/2 \rfloor)$
  if $(k_i + k_{i+1}, l_i + l_{i+1}) \ mod\ 2 = (0,0)$ **then**
    $D_i = D_{i+1}$
  end if
  if $(k_i + k_{i+1}, l_i + l_{i+1}) \ mod\ 2 = (0,1)$ **then**
    $D_i = 0$,
  end if
  if $(k_i + k_{i+1}, l_i + l_{i+1}) \ mod\ 2 = (1,0)$ **then**
    $D_i = 1$,
  end if
  if $(k_i + k_{i+1}, l_i + l_{i+1}) \ mod\ 2 = (1,1)$ **then**
    $D_i = D_{i+1}$,
  end if
  Set $V_i^{(1)} = (k_i + (k_i+1) \ mod\ 2, l_i + (l_i+1) \ mod\ 2)$,
  Set $V_i^{(2)} = (k_i + k_i \ mod\ 2, l_i + l \ mod\ 2)$,
  Set $V_i^{(3)} = (k_i + (k_i + D_i) \ mod\ 2, l_i + (l_i + D_i + 1) \ mod\ 2)$,
end for
Return: $V_i^{(1)}, V_i^{(2)}, V_i^{(3)}$.

We can determine the $V_i^{(1)}$, $V_i^{(2)}$, and $V_i^{(3)}$ elements in DAC for any pair (k, l) through Algorithm 1, What's more, we have obtained the initial elements $V_0^{(1)}$ and $V_0^{(2)}$ which is equal to (1,1) and (0,0), and $V_0^{(3)}$ which is equal to (1,0) or (0,1). In this context, (1,1), (0,0), (1,0), and (0,1) respectively represent the results of $P+Q$, 0, P, and Q, all of which are known or easily obtained. If we can establish the relationship between elements in the i-th and (i-1)-th iteration, then we can obtain the final $V_n^{(1)}$, $V_n^{(2)}$, and $V_n^{(3)}$ through P and Q. Upon observation, we can find that, in each iteration, there is one PD and two PA, and Equation (6) shows the relationship between $\{V_i^{(1)}, V_i^{(2)}, V_i^{(3)}\}$ and $\{V_{i-1}^{(1)}, V_{i-1}^{(2)}, V_{i-1}^{(3)}\}$, where $PD_{i-1} \in \{1,2,3\}$ and $PQ_{i-1} \in \{1,2\}$. As we can see, $V_i^{(1)}$ is always calculated through the point addition of $V_{i-1}^{(1)}$. $V_i^{(2)}$ is calculated through the PD of $V_{i-1}^{(1)}$ and either $V_{i-1}^{(2)}$ or $V_{i-1}^{(3)}$. Finally, $V_i^{(3)}$ is calculated through the point addition of $V_{i-1}^{(3)}$ and one of the other elements, $V_{i-1}^{(1)}$ or $V_{i-1}^{(2)}$.

$$\begin{cases} V_i^{(1)} = V_{i-1}^{(1)} + V_{i-1}^{(2)} \\ V_i^{(2)} = 2V_{i-1}^{(PD_{i-1})} \\ V_i^{(3)} = V_{i-1}^{(3)} + V_{i-1}^{(PA_{i-1})} \end{cases} \tag{6}$$

Additionally, we have observed that by considering the parity of $V_i^{(m)}$ from the previous iteration, we can determine the parity of $V_{i-1}^{(m)}$ in the current iteration. Firstly, we need to clarify that through the equations in Algorithm 1, we can ascertain that $V_i^{(1)}$ is always (odd, odd), $V_i^{(2)}$ is always $(even, even)$, and $V_i^{(3)}$ is always $(even, odd)$ or $(odd, even)$ in each iteration.

Algorithm 2 The Flag Generation Algorithm

Require: $V_i^{(1)}, V_i^{(2)}, V_i^{(3)}, (k, l)$
Ensure: PA_i, PD_i, PQ_i.
$n = max(\lceil log_2 k \rceil, \lceil log_2 l \rceil)$
for $i = n$ to 0 **do**
  if $(V_{i+1}^{(2)}/2) \ mod\ 2 = (1,1)$ **then**
    Set $PDi = 1$
  else if $(V_{i+1}^{(2)}/2) \ mod\ 2 = (0,0)$ **then**
    Set $PDi = 2$
  else
    Set $PDi = 3$
  end if
  if $(V_{i+1}^{(3)} \ mod\ 2 \oplus V_i^{(3)} \ mod\ 2) = (1,1)$ **then**
    $PA_i = 1$
    if $V_i^{(3)} - V_i^{(1)} = (0,1)$ **then**
      $PQ_i = 0$
    else if $V_i^{(3)} - V_i^{(1)} = (1,0)$ **then**
      $PQ_i = 1$
    end if
  else if $(V_{i+1}^{(3)} \ mod\ 2 \oplus V_i^{(3)} \ mod\ 2) = (0,0)$ **then**
    $PA_i = 2$
    if $V_i^{(3)} - V_i^{(2)} = (0,1)$ **then**
      $PQ_i = 0$
    else if $V_i^{(3)} - V_i^{(2)} = (1,0)$ **then**
      $PQ_i = 1$
    end if
  end if
end for
Return: PA_i, PD_i, PQ_i.

Most importantly, for each PA in *DAC*, its P_{diff}, which can be used to omit the calculation of Y-coordinate, can only come from $\{(1,1), (0,1), (1,0), (1,-1)\}$.

Now, we consider the process of obtaining the result $V_i^{(1)}$ as point addition PA_1, the process of obtaining the result $V_i^{(3)}$ as point addition PA_2, and the process of obtaining the result $V_i^{(2)}$ as PD. For PA_1, it is always obtained through $V_{i-1}^{(1)}$ and $V_{i-1}^{(2)}$, and it can be observed that the difference between $V_{i-1}^{(1)}$ and $V_{i-1}^{(2)}$ is always $P+Q$ or $P-Q$ during to the parity of them. For PD, obviously, if $V_i^{(2)} = (even, even)$, it is

obtained through $V_{i-1}^{(2)}$. For PA_2, it is obtained by $V_{i-1}^{(3)}$ and $V_{i-1}^{(1)}$ or $V_{i-1}^{(2)}$. If $V_i^{(3)} = (odd, even)$ and $V_{i-1}^{(3)} = (even, odd)$, we can deduce that $V_i^{(3)} = V_{i-1}^{(3)} + V_{i-1}^{(1)}$. At the same time, we can also determine that $x_{diff} = x_Q$ by calculating the difference between $V_{i-1}^{(3)}$ and $V_{i-1}^{(1)}$. Based on the preceding analysis, in Algorithm 2, we generated PD_i, PQ_i, and PA_i. Here, PD_i is used to indicate which element to choose for PD in the i-th round: if $PD_i = 1$, then $V_i^{(1)}$ is chosen; if $PD_i = 2$, then $V_i^{(2)}$ is chosen; if $PD_i = 3$, then $V_i^{(3)}$ is chosen. PA_i is used to indicate which element to choose for PA_2 with $V_i^{(3)}$: if $PA_i = 1$, then $V_i^{(1)}$ is selected; otherwise, $V_i^{(2)}$ is selected. PQ_i determines whether the difference between $V_i^{(3)}$ and the element used for PA_2 with $V_i^{(3)}$ is P or Q: if $PQ_i = 0$, then the difference is P; otherwise, it is Q. Therefore, by precomputing $P - Q$ and $P + Q$, we can simplify the computation and omit the calculation of the Y-coordinate with existing P and Q.

One more thing to note is that, in order to complete the establishment of the addition chain, it is necessary to determine $V_0^{(3)}$. We find out that all the parity also holds for the initial element $\{V_0^{(1)}, V_0^{(2)}, V_0^{(3)}\}$, which means that $V_0^{(3)}$ can only obtain the form (0,1) or (1,0) that means P or Q.

Algorithm 3 The Double Point Multiplication Algorithm

Require: PA, PD, $V_0^{(3)}$, P, Q.
Ensure: $C = kP + lQ$.

Set $n = max([log_2 k], [log_2 l])$,
Set $C_1 = P + Q$, $C_2 = 0$,
if $V_0^{(3)} = (0, 1)$ **then**
  Set $C_3 = Q$
else if $V_0^{(3)} = (1, 0)$ **then**
  Set $C_3 = P$
end if
for $i = 1$ to n **do**
  $C_1 \leftarrow C_1 + C_2$
  if $PA_i = 0$ **then**
    $C_3 \leftarrow C_1 + C_3$
  else if $PA_i = 1$ **then**
    $C_3 \leftarrow C_2 + C_3$
  end if
  if $PD_i = 0$ **then**
    $C_2 \leftarrow 2C_1$
  else if $PD_i = 1$ **then**
    $C_2 \leftarrow 2C_2$
  else if $PD_i = 2$ **then**
    $C_2 \leftarrow 2C_3$
  end if
end for

Return: $C = kP + lQ$.

In Algorithm 3, the calculated PD_i and PA_i determine the values of n and m in Equation (6). Through PQ_i, the value of x_{diff} in Equation (3) is determined. With $\{PD_i, PA_i, PQ_i\}$ and $\{V_0^1, V_0^2, V_0^3\}$, we can obtain $kP + lQ$ by the value of P, Q, and precomputed $P + Q$, $P - Q$ by Algorithm 4.

Algorithm 4 The PA Algorithm

Require: $X_1, Y_1, Z_1, X_2, Y_2, Z_2$.
Ensure: X_3, Z_3, x_p.

$X_1 \leftarrow X_1 Z_2, X_2 \leftarrow X_2 Z_1$
$Y_1 \leftarrow Y_1 Z_2, X_2 \leftarrow Y_2 Z_1$
$X_1 \leftarrow X_1 + X_2$
$Y_1 \leftarrow Y_1 + Y_2$
$Y_1 \leftarrow Y_1 Y_2, Z_1 \leftarrow Z_1 Z_2, X_2 \leftarrow X_1^2$
$Y_1 \leftarrow Y_1 Z_1, Z_3 \leftarrow Z_1 X_2, X_1 \leftarrow X_1 + Z_1$
$X_1 \leftarrow X_1 X_2$
$X_3 \leftarrow Y_1 + X_1$
$x_p \leftarrow X_3 / Z_3$

Return: X_3, Z_3, x_p.

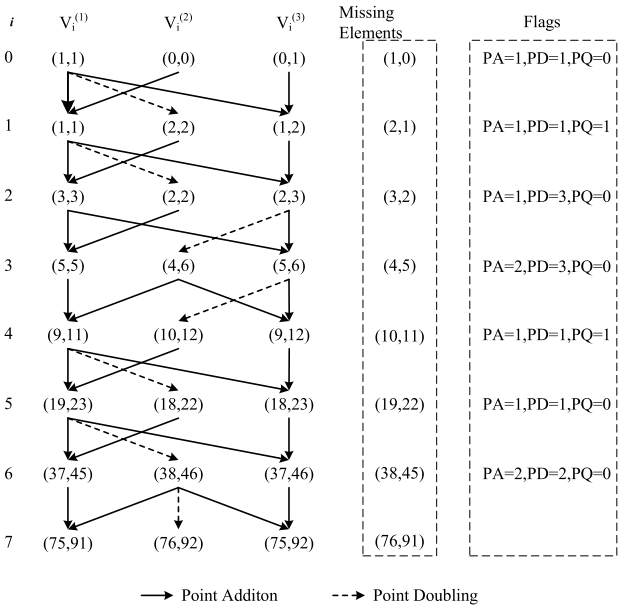

FIGURE 2. Calculating DPM $75P + 91Q$ with the proposed DAC.

When performing precomputation for $P + Q$ and $P - Q$, we use Algorithm 4 to compute the PA of two points (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) in the LD projective coordinate system. We then transform the resulting points to the affine coordinate system to obtain their horizontal coordinate x_P for PA. As shown in Algorithm 3, we ultimately obtain the complete pathway for computing DPM. All of the above algorithms also hold for PM. When calculating kP, we need to calculate $hP + eP$, where $k = h + e$.

It appears that Fig. 2. provides an example of DPM using a two-dimensional DAC to calculate $75P + 91Q$. The left side of the figure shows the elements in the DAC, while the right side shows the computed flags. By using these flags, the final result can be obtained by following the computation sequence as shown in the diagram.

It seems that when computing $91P$, it can be considered as calculating $91P + 0P$, which makes the construction of a two-dimensional DAC more feasible. Furthermore, based on

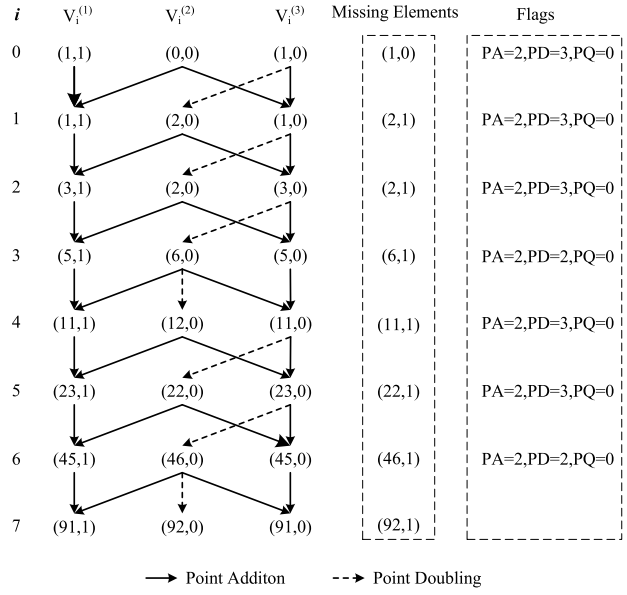

FIGURE 3. Calculating PM 91*P* with the proposed DAC.

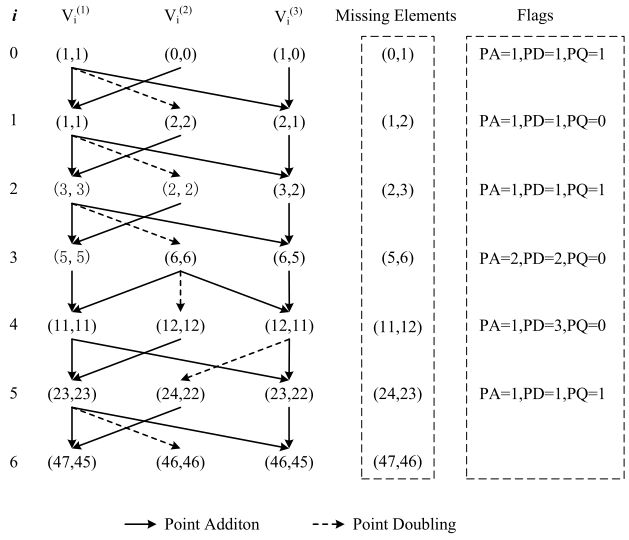

FIGURE 4. Calculating PM 45*P* + 46*P* with less iterations.

Algorithm 2, if $l = 0$, each iteration of PA_2 can be obtained from the computation of $V_i^{(2)}$ and $V_i^{(3)}$, and only $V_i^{(2)}$ and $V_i^{(3)}$ are involved in the calculation of PD. In this scenario, it is only necessary to determine, based on PD_i, whether $V_i^{(2)}$ or $V_i^{(3)}$ is involved in PD for each iteration, and the computation of PA_1 can be skipped. We can also express $91P$ as $46P+45P$, which allows us to reduce one iteration, making full use of the two PA operations in the design, which is shown in Fig. 4.

IV. DATA DEPENDENCY AND TIME SCHEDULE IN THE PROPOSED GENERAL ARCHITECTURE

A. DATA DEPENDENCY

It seems that in the previous context, we demonstrated that by transforming the affine coordinate into the LD coordinate,

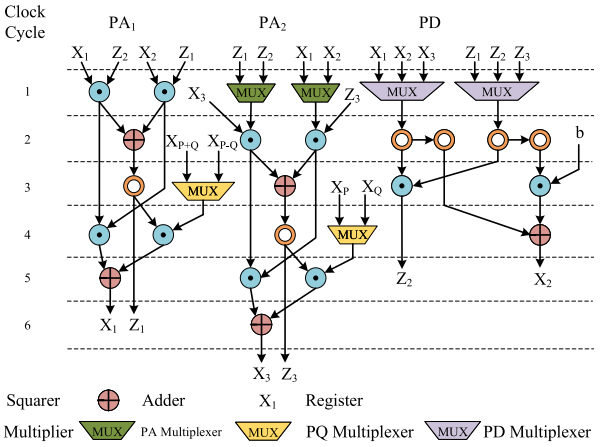

FIGURE 5. Data Dependency of proposed universal architecture.

and with the help of DAC, it is possible to omit the calculation of the Y-coordinate in PA and PD. This allows us to obtain the data dependency relationships for each iteration for PA_1, PA_2, and PD. As shown in Fig. 5., there are four modular multiplications, one modular squaring, and two modular additions in each iteration of PA_1 and PA_2. In PD, there are two modular multiplications, four modular squaring, and one modular addition. Additionally, in PA_2, there is a PA multiplexer and a PQ multiplexer, which are selected based on the PA_i flag and PQ_i flag generated by Algorithm 2. In PD, a PD multiplexer is selected based on the PD_i flag generated by Algorithm 2. Observing the data paths in the architecture with two multipliers takes five clock cycles to complete PA. However, not every cycle uses the multipliers. For PA, multiplications are used in two clock cycles, while for PD, multiplications are used in one clock cycle. In the architecture with two multipliers, this consumption of clock cycles in multiplications happens to be completed in five clock cycles. Therefore, strategically arranging the multiplication computation order can accomplish two PAs and one PD within five clock cycles.

In the above analysis, we used two KOMs with a latency of two clock cycles each to increase the system frequency. As shown in Table 1, for the two-cycle multiplier MUL_0, X_2 and Z_1 are inputted in the first clock cycle, X_1 and Z_2 in the second clock cycle, and the resulting output X_2Z_1 is written to a register. The result X_2Z_1 from the multiplier is utilized in the third cycle. This approach minimizes the clock cycles consumed in each iteration, ultimately increasing the system frequency. In clock cycle 1, we need to use PA_i to determine whether Z_1 and X_1 or Z_2 and X_3 are inputted to register Z_k and X_k, and we need to use PD_i to determine Z_i and X_i from Z_1 and X_1, Z_2 and X_2, or Z_3 and X_3. In clock cycle 4, we need to use PQ_i to determine x_{diff} from x_P or x_Q.

V. HARDWARE ARCHITECTURE

A. OVERALL ARCHITECTURE

The proposed DPM and PM universal architecture based on two-dimensional DAC on FPGA appears in Fig. 6.

TABLE 1. Timing schedule of proposed universal architecture.

Clock	MUL_0 In	MUL_0 out	MUL_1 In	MUL_1 out	SQR_0 In	SQR_0 Out	SQR_1 In	SQR_1 Out	SQR_2 In	SQR_2 Out	SQR_3 In	SQR_4 Out
1	X_2, Z_1	–	X_1, Z_2	–	–	–	–	–	–	–	–	–
2	X_3, Z_k	X_2Z_1	X_k, Z_3	X_1Z_2	X_i	X_i^2	X_i^2	X_i^4	Z_i	Z_i^2	Z_i^2	Z_i^4
3	Z_i^4, b, X_i^4	X_3Z_k	Z_i^2, X_i^2	X_kZ_3	X_2Z_1, X_1Z_2	$Z_1' = (X_2Z_1 + X_1Z_2)^2$	–	Z_i^2, X_i^2	–	–	–	
4	X_1Z_2, X_2Z_1	$bZ_i^4 + X_i^4$	X_{P+Q}, Z_1'	$Z_2' = Z_i^2X_i^2$	X_3Z_k, X_kZ_3	$Z_3' = (X_3Z_k + X_kZ_3)^2$	–	–	–	–	–	
5	X_3Z_k, X_kZ_3	$X_1Z_2X_2Z_1$	X_{diff}, Z_3'	$X_1' = X_{P+Q}Z_1' + X_1Z_2X_2Z_1$	–	–	–	–	–	–	–	–
6	X_2, Z_1	$X_3Z_kX_kZ_3$	X_1, Z_2	$X_3' = X_{\text{diff}}Z_3' + X_3Z_kX_kZ_3$	–	–	–	–	–	–	–	–

* In the table, (X_1, Z_1), (X_2, Z_2), (X_3, Z_3) represent the coordinates of $V_{i-1}^{(1)}, V_{i-1}^{(2)}, V_{i-1}^{(3)}$ in the *LD* coordinate system, and $(X_1', Z_1'), (X_2', Z_2'), (X_3', Z_3')$ represent the coordinates of $V_i^{(1)}, V_i^{(2)}, V_i^{(3)}$ in the *LD* coordinate system. $Z_k \in \{Z_1, Z_2\}, X_k \in \{X_1, X_2\}, Z_i \in \{Z_1, Z_2, Z_3\}, X_i \in \{X_1, X_2, X_3\}$.

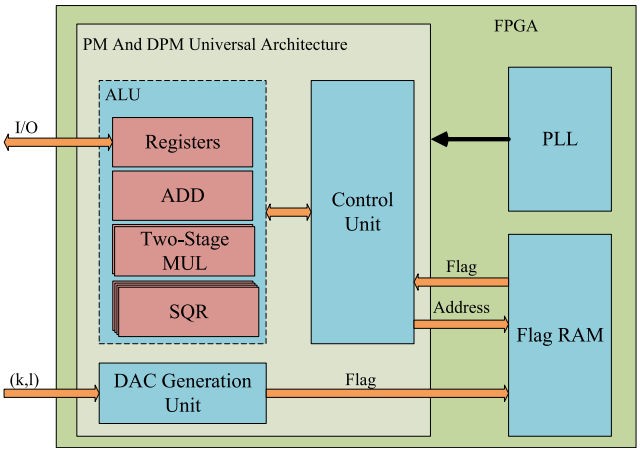

FIGURE 6. Proposed PM and DPM Universal architecture.

The overall architecture includes an arithmetic logical unit (ALU), a DAC generation unit, a control unit, a flag RAM, and a built-in phase-locking loop (PLL) that provides the global clock. The ALU is responsible for PA, PD, modular inversion, and precomputes the initial elements $P + Q$ of DAC chains. The DAC generation unit is used to generate the flags in the DAC. The flag ram stores the control signals of multiplexers in the ALU generated by the DAC generation unit. The control unit completes the iterative calculation steps in Algorithm 3 by reading the flags stored in the flag storage unit.

B. ALU ARCHITECTURE

Based on the timing schedule proposed in Table 1, an ALU architecture has been proposed, shown in Fig. 7. The ALU consists of registers, adders, multipliers, and squares. Registers store pre-computed values of $P + Q$, $P - Q$, input values P, Q, and inputs and outputs of multipliers and squaring. By reading the DAC generation unit generates PA_i, PD_i, and PQ_i through the input scalar pairs (k, l), the control unit performs different operations.

As shown in Fig. 7. the ALU includes two Karatsuba–Ofman multiplier (KOM) [25], four squares, and several multiplexers. The control unit controls different multiplexers by controlling the MUX_s signals in each clock cycle and stores the results in specific registers. For example, when we need to compute X_1X_2 in Table 1, we need to control two multiplexers corresponding to MUL_0 to select X_2 and Z_1 and store the result in the X_1Z_2 register. The same applies to other calculations. It is important to note that registers with the same name in the figure represent the same register.

Through the corresponding algorithm 3 and DAC generation unit, it can be observed that the PQ flag determines whether the X_{diff} register selects X_P or X_Q in each iteration; the PA flag determines whether the Z_k and X_k registers select Z_1 and X_1 or Z_2 and X_2 in each iteration; and the PD flag determines whether the Z_i and X_i registers select Z_1 and X_1, Z_2 and X_2, or Z_3 and X_3 in each iteration.

C. DAC GENERATION UNIT ARCHITECTURE

As illustrated in Fig. 8, the entire *DAC* generation unit is composed of modules for generating $V_i^{(1)}$, $V_i^{(2)}$, $V_i^{(3)}$, and a flag generation module. The entire system comprises four types of m-bit registers. Registers ($C1\_REG0$, $C1\_REG1$), ($C2\_REG0$, $C2\_REG1$) and ($C2\_REG0$, $C2\_REG1$) are designated for storing $V_i^{(1)}$, $V_i^{(2)}$ and $V_i^{(3)}$ respectively. Another type of register, as depicted in the diagram, is utilized for storing the (k_i, l_i) pairs of the current iteration. Once the values for the registers ($C1\_REG0$, $C1\_REG1$), ($C2\_REG0$, $C2\_REG1$) and ($C2\_REG0$, $C2\_REG1$) have been obtained, the corresponding flag signals can be generated by utilizing the values of their 0th or 1st bits.

If we need to calculate $(V_{i+1}^{(3)} \ mod \ 2) \oplus (V_i^{(3)} \ mod \ 2)$ in Algorithm 2, we can obtain the result of $(V_i^{(3)} \ mod \ 2)$ by getting the 0-th bit of ($C3\_REG0$, $C3\_REG1$), and the result of $(V_{i-1}^{(3)} \ mod \ 2)$ by getting the 0-th bit of ($C3\_REG0'$, $C3\_REG1'$). In this way, we can obtain PA_i using only (K_i, l_i). In the same way, we can obtain PQ_i and PD_i.

VI. IMPLEMENTATION RESULTS AND COMPARISON

In this section, we first discuss the evaluation metrics. Then, we give a brief security analysis of our proposed work. Finally, we conduct comparisons with existing works. The FPGA implementation results of our design and existing closely related designs in recent years are listed in Table 2.

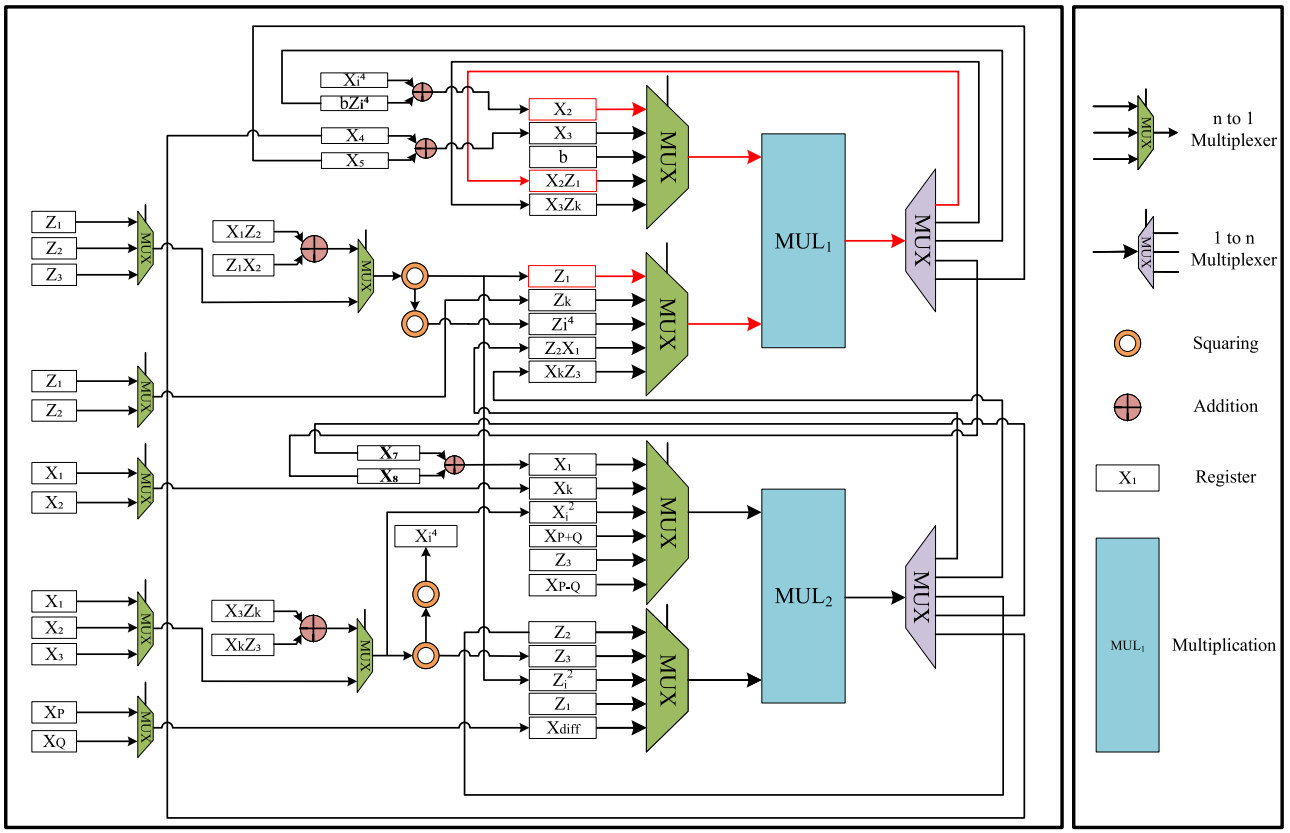

FIGURE 7. Proposed ALU architecture with two KOM and four squares.

A. LATENCY AND PERFORMANCE ANALYSIS

Our design proposes a universal architecture applicable to DPM and PM. The resource consumption of our design is calculated under the premise of implementing DPM. Therefore, compared with other works, the comparison is not made under the PM condition. To ensure a fair comparison, when calculating the latency of other works, we assume that they require two PMs, simply multiplying the clock cycles by 2. This method does not consider the latency of the final PA, so the actual latency of other works is bigger than the values listed in Table 2.

In our design, the total latency includes the DAC generation unit, PA unit, ALU, and modular inversion unit. When performing DPM, it is necessary to construct the DAC and precompute $P+Q$ and $P-Q$. However, it is worth noting that the construction of the DAC and the calculation of $P+Q$ and $P-Q$ do not consume the same hardware resources. Therefore, it is possible to perform DAC calculation, $P+Q$, and $P-Q$ calculation simultaneously. For $GF(2^m)$, the construction of the DAC chain requires m clock cycles, while the calculation of $P+Q$ and $P-Q$ using the proposed Algorithm 4 requires clock cycles within m. The total latency can be calculated by Equation (7).

$$T = (C_{DAC} + C_{ALU} + C_{INV}) \times T_{CLK} \tag{7}$$

In our design, there are two multipliers and four squares. When utilizing Itoh and Tsujii's [45] and Rashidi et al.'s [46], [47] proposed modular inversion algorithm, the calculation of the modular inverse can be completed within $\frac{m+1}{2}$ cycles for GF(2^m). The ALU consumes $5 \times m + 1$ clock cycle over GF(2^m) and one additional clock cycle to wait for the final multiplication result.

$$C_{Tot} = m + 5 \times m + \frac{m+1}{2} + 1 \tag{8}$$

The prevailing trend in most existing works involves utilizing area-time product (ATP) as a performance metric to assess the balance between hardware consumption and latency.

$$ATP = Slice \times T \tag{9}$$

In practical scenarios involving ECDSA, throughput is a crucial metric that determines whether a design can handle a significant number of ECDSA operations within a unit of time.

$$Throughput = \frac{Bit\ Width}{T}\ bps \tag{10}$$

Due to differences in the area of different designs, we used Equation (11) to compare the throughput efficiency between different designs.

$$Efficiency = \frac{Throughput}{Silce} \times 10^3 \tag{11}$$

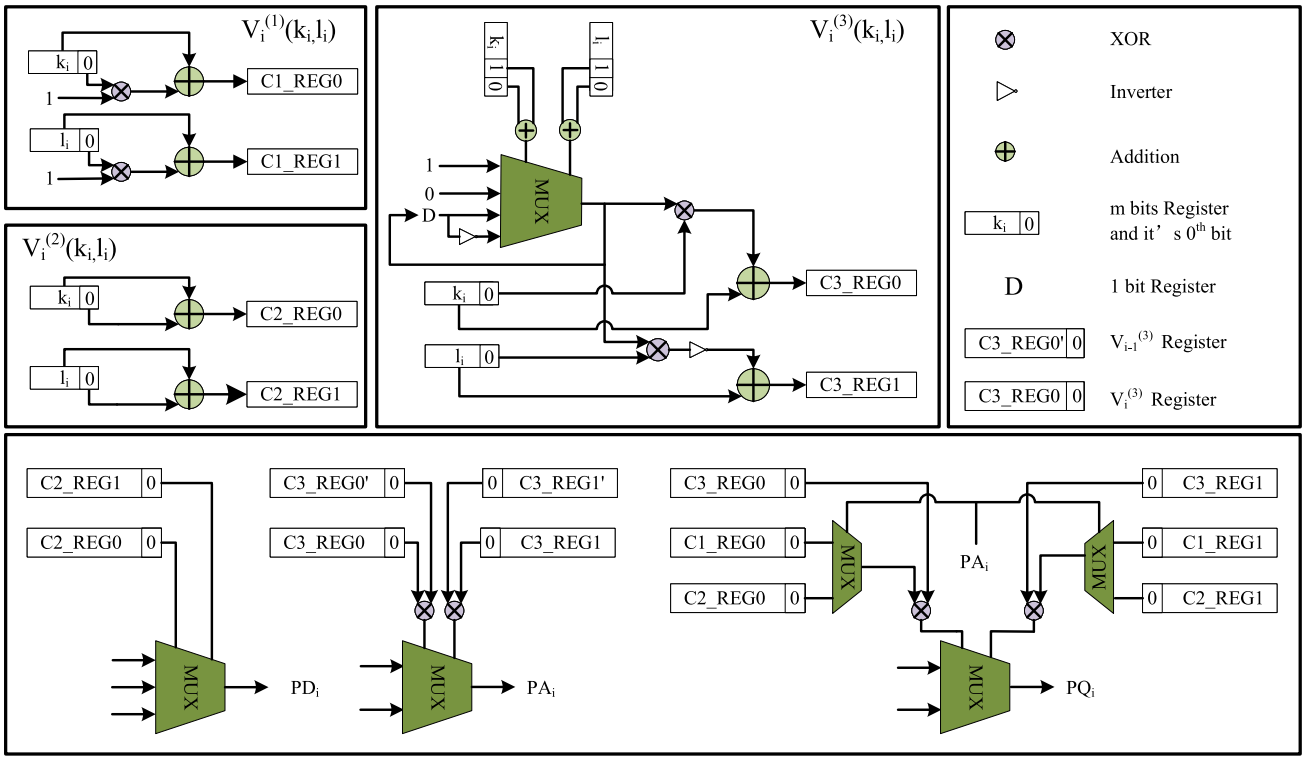

FIGURE 8. DAC generation unit generate $\{V_i^{(1)}, V_i^{(2)}, V_i^{(3)}\}$ and $\{PA_i, PQ_i, PD_i\}$.

TABLE 2. FPGA implementation results over Xilinx Vertex-7.

Design	m	Clock Cycle	Freq (MHz)	LUT	Slice	Latency (μs)	ATP	Efficiency	$\delta_L(\%)$	$\delta_A(\%)$	$\delta_E(\%)$
Our Work	163	1061	296	28997	8135	3.584	29160	5.590	0	0	0
[35]	163	795 (1590)	264	9429	2435	6.022	14665	11.115	-40.5	–†	-49.7
[21]	163	450 (900)	159	41090	11657	5.660	65983	2.470	-36.7	-55.8	126.3
[22]	163	547 (1094)	320.5	28911	8460	3.413	28878	5.0	5.6	0.98	-0.97
[36]	163	780 (1560)	223	27105	8736	6.995	61113	2.7	-48.7	-52.3	109.5
[37]	163	4168 (8236)	397	4271	1476	20.997	30992	5.259	-82.9	-5.9	6.3
[38]	163	3960 (7920)	369	9965	2207	21.463	47370	3.441	-83.3	-3.8	62.5
[39]	163	3960 (7920)	351	10955	3107	22.564	70107	2.325	-84.1	-5.8	140.4
[40]	163	13000 (26000)	320.8	6169	2201	81.047	178385	0.914	-95.2	-83.6	511.7
[41]	163	52012 (104024)	800	–	4665	130.03	606590	0.268	-97.2	-95.2	19.8
[42]	163	3426 (6852)	135	10128	3657	50.076	185613	0.878	-92.9	-84.3	536.5
[33]	163	1353	496	–	2856	2.727	7791	20.922	31.4	–†	-73.3
Our Work	233	1516	268	47014	13372	5.656	75642	3.080	0	0	0
[43]	233	2440 (4880)	497	–	10528	9.819	103374	2.253	-42.4	-26.8	36.7
[35]	233	1112 (2224)	266	14861	3832	8.361	32039	7.272	-32.3	–†	-57.6
[20]	233	3344 (6688)	179.81	24533	2660	37.195	912500	2.353	-84.8	-46.9	88.4
[33]	233	1929	542	–	4307	3.559	15329	15.200	72.2	–†	-81.2
Our Work	283	1841	247	63752	20126	7.453	133402	2.153	0	0	0
[44]	283	856 (1712)	274.1	80970	23116	6.246	144380	1.960	17.6	-9.0	9.8
[35]	283	1359 (2718)	226	20620	5417	12.0265	65148	4.344	-38.9	–†	-50.4
[37]	283	6850 (13700)	337	11593	5207	40.653	211679	1.337	-81.9	-37.9	61.0
[33]	283	2341	496	–	5810	4.719	27422	10.320	55.6	–†	-79.1

* To facilitate comparison with other work, we introduce three parameters, δ_L, δ_A, and δ_E. $\delta_L = \frac{Our_{Latency} - Ref_{Latency}}{Ref_{Latency}}$, $\delta_A = \frac{Our_{ATP} - Ref_{ATP}}{Ref_{ATP}}$, and $\delta_E = \frac{Our_{Efficiency} - Ref_{Efficiency}}{Ref_{Efficiency}}$

–†: With lower power analysis resistance.

B. SECURITY ANALYSIS

Currently, the main attack methods against cryptographic chips include side-channel attacks [48], fault injection, and so on.

1) SPA and high-order power analysis can be utilized to analyze the power consumption curve of an FPGA [49]. This approach allows us to discern the operations a certain design executes at different time intervals [50].

In our algorithm, during each iteration, two PAs and one PD are performed, forming a structure similar to the Montgomery Ladder [51]. This ensures that the power trace of our structure exhibits a uniform pattern, making it resistant to SPA. Many works use an extra PA with the point at infinity, such as [35], to defend against SPA. However, during the PD-PA-PD sequence, the extra PA with the point at infinity performed during the first PD does not change the internal value. This leads to a correlation between the first PD and the second PD, making this method ineffective against high-order power analysis [23]. However, our design is resistant to high-order power analysis because every PA is valid, eliminating any such correlation.

2) The execution time of the proposed algorithm is fixed for a given field $GF(2^m)$, as the number of iteration loops depends solely on the length of k. Consequently, our proposed architecture is secure against timing attacks.
3) The proposed algorithm does not include any dummy operations, and all components in the architecture are utilized. Therefore, any fault injection will lead to an incorrect computation result, indicating that our proposed architecture is secure against fault injection attacks.

C. IMPLEMENTATION RESULTS COMPARISON

Table 2 presented in this paper shows the implementation results of Vertex-7 Series over GF(2^m). Our work has latency of 3.584, 5.656, and 7.453 μs, using 29160, 75642, and 133402 ATP, with throughput efficiency of 5.590, 3.080, and 2.153 over GF(2^{163}), GF(2^{233}), and GF(2^{283}), respectively.

1) Security: Both work [33] and work [35] require the introduction of dummy PAs in specific steps of every iteration to resist SPA. The analysis presented in Section VI-B suggests that the dummy PAs operation of [33] and [35] may be vulnerable to high-order power analysis, posing a potential security risk. Our work and other works don't employ dummy PA, which enables them to resist high-order power analysis.
2) Latency: The work [33] demonstrates the lowest latency among existing work. Compared to our work, it reduces latency by 31.4%, 72.2%, and 55.6% over GF(2^{163}), GF(2^{233}), and GF(2^{283}), respectively. Another work [22] has a latency that is 5% lower than ours over GF(2^{163}), and work [44] has a latency that is 17.6% lower than ours over GF(2^{283}). Aside from those, our work demonstrates superiority in terms of latency, which reduces the latency by 36.7%, 82.9%, and 83.3% compared to the works [21], [37], and [38] over GF(2^{163}). Our work reduces the latency by 32.3%, 42.4%, and 84.8 % compared to the works [20], [35], and [43] over GF(2^{233}) and reduces the latency by 38.9% and 81.9% compared to the works [35] and [37] over GF(2^{283}), respectively.
3) Area Time Product: Among all existing works, work [33] and [35] show advantages in terms of low ATP. Aside from these two designs, the ATP of our work is 0.98% higher than work [22], 5.9%, 3.8%, 5.8%, and 52.3% lower than work [36], [37], [38], and [39] over GF(2^{163}). The ATP of our work is 26.8% and 46.9% lower than works [20] and [43] over GF(2^{233}), and 9% and 37.9% lower than works [37] and [44] over GF(2^{283}).
4) Throughput Efficiency: Among all existing works, [33] has the highest throughput efficiency, which is 73.3%, 81.2%, and 79.1% higher than ours over GF(2^{163}), GF(2^{233}), and GF(2^{283}), respectively. Work [35] has the second highest throughput efficiency, which is 49.7%, 57.6%, and 50.4% higher than ours over GF(2^{163}), GF(2^{233}), and GF(2^{283}), respectively. Aside from those, the throughput efficiency of our work is 0.97% lower than work [22], 6.3%, 19.8%, and 62.5% higher than works [37], [38], and [41] over GF(2^{163}). The throughput efficiency of our work is 9.8% and 61.0% higher works [44] and [37].

Work [33] and Work [35] currently demonstrate the best performance, boasting the lowest latency and ATP, and highest throughput for both PM and DPM. While other works may pale in comparison, in ECDSA, prioritizing security over performance underscores the value of sacrificing some efficiency for the sake of safety.

With resistance to higher power analysis, our design outperforms the best designs [20] over GF(2^{233}) with increasing throughput efficiency by 36.7 % and reducing ATP by 26.8 % and the best design [44] over GF(2^{283}) with increasing throughput efficiency by 9.8 % and reducing ATP by 9.0 %. The implementation results of the reported works indicate that the proposed architecture based on DAC has advantages in ATP and throughput efficiency when computing DPM, which makes our work more suitable for ECDSA. Some lightweight PM architectures proposed in previous works [20], [37] may find applications in resource-constrained scenarios. However, with an acceptable increase in hardware resources, our work reduces the computation burden of PD through DAC, leading to higher throughput efficiency, less ATP, and lower latency in our work.

In addition, these advantages also exist over GF(2^{163}). Two works over GF(2^{163}) perform similarly to our work. Compared to [36], our work shows a 6.3% improvement in throughput efficiency and a 5.9% reduction in ATP, and compared to work [36] with 20.997 μs, our work, 3.584 μs, significantly reduces 82.9% latency, making it more suitable for scenarios with strict latency requirements. Work [22] outperforms our work by 0.98 % in throughput efficiency and reduces ATP by 0.97 % compared to our work. However, it is important to note that our work is a universal architecture for DPM and PM, while work [22] is a PM

TABLE 3. Symbol and notation table.

Symbol	Notation
Abbreviation	
SPA	Simple Power Analysis
ECDSA	Elliptic curve digital signature algorithm
ECPM	Elliptic curve point multiplication
PA	Point addition
PD	Point doubling
PM	Point multiplication
DPM	Double point multiplication
DAC	Differential addition chain
ALU	Arithmetic logical unit
Variable	
x, y	Affine coordinates
X, Y, Z	LD projective coordinates
$a(x), b(x), f(x)$	Represents the polynomial of x.
$P_1, P_2, P_3, P_3, P_{diff}, Q$	Points on an Elliptic Curve
k, l, h, e	Scalar for point multiplication
m	Degree of field extension.
PA_i	Index determines which element to calculate point addition.
PD_i	Index determines which element to calculate point doubling.
PQ_i	Index determines the difference in point addition.
$V_i^{(1)}, V_i^{(2)}, V_i^{(3)}$	(k_i, l_i) pair in differential addition chain
Mathematical symbols	
GF(p)	Elliptic Curve Prime Field
GF(2^m)	Elliptic Curve Binary Field
$\oplus$	Exclusive OR
$\times$	Multiplication

architecture. Additionally, it should be mentioned that we assume two calls to PM without considering the latency of PA. Finally, [22] does not mention whether its design has the functionality for PA, suggesting that additional hardware may be needed for PA when performing DPM. This could incur additional hardware resource overhead and decrease throughput efficiency, far from the claimed 0.98 %. Our work is faster than other works [21], [36], [37], [38], [39], [40], [41], [42], and with lower ATP and higher throughput efficiency.

VII. CONCLUSION

The article introduces a DAC-based algorithm suitable for DPM and PM, enabling high throughput and low latency for ECDSA in embedded scenarios. A detailed timing schedule is provided through data dependency analysis. A unified architecture for DPM and PM, including a DAC generation unit, an ALU, and a control unit, is proposed in this article. This architecture exhibits high throughput, low latency, and resistance to attacks. Tailored for resource-constrained embedded devices, it achieves resource reuse for DPM and PM, addressing the challenge of additional PM modules required for DPM architecture. Compared to existing works with the same level of security, our design slightly improves ATP and throughput efficiency but excels in terms of versatility. Currently, the setup for DAC still requires precomputation, and the PM calculation cannot be completed during the bit-by-bit scanning of the scalar, as is done with the traditional Montgomery algorithm. In future work, we will seek improvements in this area and further exploit DAC-based algorithms over $GF(p)$ and try to implement hardware architectures on application-specific integrated circuits.

APPENDIX A
SYMBOLS AND NOTATION

In Table 3, we have explained the symbols that appear in the article.

REFERENCES

[1] C.-X. Wang, X. You, X. Gao, X. Zhu, Z. Li, C. Zhang, H. Wang, Y. Huang, Y. Chen, H. Haas, J. S. Thompson, E. G. Larsson, M. D. Renzo, W. Tong, P. Zhu, X. Shen, H. V. Poor, and L. Hanzo, "On the road to 6G: Visions, requirements, key technologies and testbeds," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 905–974, 2023.

[2] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The road towards 6G: A comprehensive survey," *IEEE Open J. Commun. Soc.*, vol. 2, pp. 334–366, 2021.

[3] P. Meena, M. B. Pal, P. K. Jain, and R. Pamula, "6G communication networks: Introduction, vision, challenges, and future directions," *Wireless Pers. Commun.*, vol. 125, no. 2, pp. 1097–1123, Jul. 2022.

[4] C. Nist, "The digital signature standard," *Commun. ACM*, vol. 35, no. 7, pp. 36–40, Jul. 1992.

[5] G. R. Blakley and I. Borosh, "Rivest-Shamir–Adleman public key cryptosystems do not always conceal messages," *Comput. Math. with Appl.*, vol. 5, no. 3, pp. 169–178, 1979.

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

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

[8] V. Mavroeidis, K. Vishi, M. D. Zych, and A. Jøsang, "The impact of quantum computing on present cryptography," 2018, *arXiv:1804.00200*.

[9] F. Mallouli, A. Hellal, N. S. Saeed, and F. A. Alzahrani, "A survey on cryptography: Comparative study between RSA vs ECC algorithms, and RSA vs El-Gamal algorithms," in *Proc. 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud) 5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2019, pp. 173–176.

[10] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, Aug. 2001.

[11] D. J. Bernstein and T. Lange. (2014). *Safecurves: Choosing Safe Curves for Elliptic-Curve Cryptography*. Accessed: Jan. 12, 2024. [Online]. Available: https://safecurves.cr.yp.to

[12] M. M. Islam, M. S. Hossain, Moh. 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.

[13] Y. A. Shah, K. Javeed, S. Azmat, and X. Wang, "Redundant-signed-digit-based high speed elliptic curve cryptographic processor," *J. Circuits, Syst. Comput.*, vol. 28, no. 5, May 2019, Art. no. 1950081.

[14] 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.

[15] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography* (Springer Professional Computing). New York, NY, USA: Springer, 2006. [Online]. Available: https://books.google.com.hk/books?id=V5oACAAAQBAJ

[16] M. Alfred and V. Scott, *Handbook of Applied Cryptography*, 1997.

[17] B. Möller, "Algorithms for multi-exponentiation," in *International Workshop on Selected Areas in Cryptography*. Springer, 2001, pp. 165–180.

[18] M. Imran and M. Rashid, "Architectural review of polynomial bases finite field multipliers over GF(2^m)," in *Proc. Int. Conf. Commun., Comput. Digit. Syst. (C-CODE)*, Mar. 2017, pp. 331–336.

[19] S. R. Pillutla and L. Boppana, "A high-throughput fully digit-serial polynomial basis finite field GF(2^m) multiplier for IoT applications," in *Proc. IEEE Region 10 Conf. (TENCON)*, Oct. 2019, pp. 920–924.

[20] A. Sajid, M. Rashid, M. Imran, and A. R. Jafri, "A low-complexity edward-curve point multiplication architecture," *Electronics*, vol. 10, no. 9, p. 1080, May 2021.
[21] Z. U. A. Khan and M. Benaissa, "High-speed and low-latency ECC processor implementation over GF(2^m) on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 1, pp. 165–176, Jan. 2017.
[22] 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.
[23] J. Zhang, Z. Chen, M. Ma, R. Jiang, H. Li, and W. Wang, "High-performance ECC scalar multiplication architecture based on comb method and low-latency window recoding algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 2, pp. 382–395, Feb. 2024.
[24] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity analysis and efficient implementations of bit parallel finite field multipliers based on karatsuba-ofman algorithm on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 7, pp. 1057–1066, Jul. 2010.
[25] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Sov. Phys. Doklady*, vol. 7, no. 7, pp. 595–596, Jan. 1963.
[26] H. Mahdizadeh and M. Masoumi, "Novel architecture for efficient FPGA implementation of elliptic curve cryptographic processor over GF(2^{163})," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 12, pp. 2330–2333, Dec. 2013.
[27] M. Khabbazian, T. A. Gulliver, and V. K. Bhargava, "A new technique for improving the speed of double point multiplication," in *Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process.*, Aug. 2005, pp. 653–656.
[28] J. Adikari, V. S. Dimitrov, and R. J. Cintra, "A new algorithm for double scalar multiplication over Koblitz curves," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2011, pp. 709–712.
[29] D. J. Bernstein. (2006). *Differential Addition Chains*. [Online]. Available: http://cr.yp.to/ecdh/diffchain-20060219.pdf
[30] R. Azarderakhsh and K. Karabina, "Efficient algorithms and architectures for double point multiplication on elliptic curves," in *Proc. 3rd Workshop Cryptography Secur. Comput. Syst.*, Jan. 2016, pp. 25–30.
[31] R. Azarderakhsh and K. Karabina, "A new double point multiplication algorithm and its application to binary elliptic curves with endomorphisms," *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2614–2619, Oct. 2014.
[32] M. Joye and M. Tunstall, "Exponent recoding and regular exponentiation algorithms," in *Progress in Cryptology—AFRICACRYPT*, Gammarth, Tunisia. Springer, 2009, pp. 334–349.
[33] T. Shahroodi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Low-latency double point multiplication architecture using differential addition chain over GF(2^m)," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1465–1473, 2019.
[34] J. Zhang, Z. Chen, H. Li, and W. Wang, "A high-performance universal FPGA implementation for PM and DPM based on differential addition chain," in *Proc. 8th Int. Conf. Signal Image Process. (ICSIP)*, 2023, pp. 789–793.
[35] R. Salarifard, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "A low-latency and low-complexity point-multiplication in ECC," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 9, pp. 2869–2877, Sep. 2018.
[36] Z. U. A. Khan and M. Benaissa, "High speed ECC implementation on FPGA over GF(2^m)," in *Proc. 25th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2015, pp. 1–6.
[37] Z.-U.-A. Khan and M. Benaissa, "Throughput/area-efficient ECC processor using Montgomery point multiplication on FPGA," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 11, pp. 1078–1082, Nov. 2015.
[38] M. Imran, M. Rashid, A. R. Jafri, and M. Kashif, "Throughput/area optimised pipelined architecture for elliptic curve crypto processor," *IET Comput. Digit. Techn.*, vol. 13, no. 5, pp. 361–368, Sep. 2019.
[39] M. Imran, M. Rashid, A. R. Jafri, and M. Najam-Ul-Islam, "ACrypproc: Flexible asymmetric crypto processor for point multiplication," *IEEE Access*, vol. 6, pp. 22778–22793, 2018.
[40] S. Harb and M. Jarrah, "FPGA implementation of the ECC over GF(2^m) for small embedded applications," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 2, pp. 1–19, Mar. 2019.
[41] T. T. Nguyen and H. Lee, "Efficient algorithm and architecture for elliptic curve cryptographic processor," *JSTS, J. Semicond. Technol. Sci.*, vol. 16, no. 1, pp. 118–125, Feb. 2016.
[42] M. Imran, I. Shafi, A. R. Jafri, and M. Rashid, "Hardware design and implementation of ECC based crypto processor for low-area-applications on FPGA," in *Proc. Int. Conf. Open Source Syst. Technol. (ICOSST)*, Dec. 2017, pp. 54–59.
[43] B. Rashidi, S. M. Sayedi, and R. R. Farashahi, "High-speed hardware architecture of scalar multiplication for binary elliptic curve cryptosystems," *Microelectron. J.*, vol. 52, pp. 49–65, Jun. 2016.
[44] J. Li, W. Wang, J. Zhang, Y. Luo, and S. Ren, "Innovative dual-binary-field architecture for point multiplication of elliptic curve cryptography," *IEEE Access*, vol. 9, pp. 12405–12419, 2021.
[45] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, Sep. 1988.
[46] B. Rashidi, R. Rezaeian Farashahi, and S. M. Sayedi, "High-performance and high-speed implementation of polynomial basis Itoh–Tsujii inversion algorithm over GF(2^m)," *IET Inf. Secur.*, vol. 11, no. 2, pp. 66–77, Mar. 2017.
[47] B. Rashidi, "High-speed hardware implementation of Gaussian normal basis inversion algorithm overF2m," *Microelectron. J.*, vol. 63, pp. 138–147, May 2017.
[48] M. Randolph and W. Diehl, "Power side-channel attack analysis: A review of 20 years of study for the layman," *Cryptography*, vol. 4, no. 2, p. 15, May 2020.
[49] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure Integrated Circuits and Systems*, 2010, pp. 27–42.
[50] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smartcards," in *Cryptographic Hardware and Embedded Systems*, Worcester, MA, USA. Springer, 1999, pp. 144–157.
[51] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Math. Comput.*, vol. 48, no. 177, pp. 243–264, Jan. 1987.

XIANG HE received the B.Sc. degree in electronic science and technology from Beijing Institute of Technology, Beijing, China, in 2018 and 2022, respectively, where he is currently pursuing the M.Sc. degree with the School of Integrated Circuits and Electronics. His research interests include digital circuit design, hardware implementation for cryptography, and encrypting network transmission.

WEIJIANG WANG received the B.Eng. and Ph.D. degrees in communication and information systems from Beijing Institute of Technology, Beijing, China, in 1999 and 2004, respectively. In 2004, he joined the School of Information and Electronics, Beijing Institute of Technology. His research interests include image processing, array signal processing, and hardware implementation for cryptography.

JINGQI ZHANG received the B.Sc. and M.Sc. degrees in electronic engineering from Beijing Institute of Technology, Beijing, China, in 2017 and 2020, respectively, where he is currently pursuing the D.Eng. degree with the School of Integrated Circuits and Electronics. His research interests include digital circuit design, VLSI implementation of cryptosystems, and high-speed data processing.

ZHANTAO ZHANG received the B.Sc. degree in electronic science and technology from Beijing Institute of Technology, Beijing, China, in 2019 and 2023, respectively, where he is currently pursuing the M.Sc. degree with the School of Integrated Circuits and Electronics. His research interests include digital circuit design and the hardware implementation of prime field elliptic encryption algorithms.

JIANLEI YANG, photograph and biography not available at the time of publication.

HUA DANG received the Ph.D. degree from the School of Information and Electronics, Beijing Institute of Technology, Beijing, China, in 2013. He is currently an Assistant Professor with the School of Information and Electronics, Beijing Institute of Technology. His current research interests include integrated circuit design, microsystem integration, wireless communication, and digital signal processing.

GUIYU WANG was born in Sichuan, China, in January 1991. He received the B.Eng. degree in information countermeasures and the Ph.D. degree in information and communication engineering from Beijing Institute of Technology (BIT), Beijing, China, in 2013 and 2021, respectively. He is currently a Postdoctoral Research Fellow with the School of Integrated Circuits and Electronics, BIT, and the BIT Chongqing Institute of Microelectronics and Microsystems. His research interests include sparse array processing and direction-of-arrival estimation and the hardware realization of the array processing technique.