applied sciences

MDPI

Article

A Unified Point Multiplication Architecture of Weierstrass, Edward and Huff Elliptic Curves on FPGA

Muhammad Arif [1,*], Omar S. Sonbul [2], Muhammad Rashid [2,*], Mohsin Murad [2] and Mohammed H. Sinky [2]

[1] Computer Science Department, Umm Al Qura University, Makkah 21955, Saudi Arabia
[2] Computer Engineering Department, Umm Al Qura University, Makkah 21955, Saudi Arabia; ossonbul@uqu.edu.sa (O.S.S.); mmnkhan@uqu.edu.sa (M.M.); mhsinky@uqu.edu.sa (M.H.S.)
* Correspondence: mahamid@uqu.edu.sa (M.A.); mfelahi@uqu.edu.sa (M.R.)

Abstract: This article presents an area-aware unified hardware accelerator of Weierstrass, Edward, and Huff curves over $GF(2^{233})$ for the point multiplication step in elliptic curve cryptography (ECC). The target implementation platform is a field-programmable gate array (FPGA). In order to explore the design space between processing time and various protection levels, this work employs two different point multiplication algorithms. The first is the Montgomery point multiplication algorithm for the Weierstrass and Edward curves. The second is the Double and Add algorithm for the Binary Huff curve. The area complexity is reduced by efficiently replacing storage elements that result in a 1.93 times decrease in the size of the memory needed. An efficient Karatsuba modular multiplier hardware accelerator is implemented to compute polynomial multiplications. We utilized the square arithmetic unit after the Karatsuba multiplier to execute the quad-block variant of a modular inversion, which preserves lower hardware resources and also reduces clock cycles. Finally, to support three different curves, an efficient controller is implemented. Our unified architecture can operate at a maximum of 294 MHz and utilizes 7423 slices on Virtex-7 FPGA. It takes less computation time than most recent state-of-the-art implementations. Thus, combining different security curves (Weierstrass, Edward, and Huff) in a single design is practical for applications that demand different reliability/security levels.

Keywords: hardware accelerator; design; ECC; Weierstrass; Edward; Huff; FPGA

check for updates

Citation: Arif, M.; Sonbul, O.S.; Rashid, M.; Murad, M.; Sinky, M.H. A Unified Point Multiplication Architecture of Weierstrass, Edward and Huff Elliptic Curves on FPGA. *Appl. Sci.* **2023**, *13*, 4194. https://doi.org/10.3390/app13074194

Academic Editor: José António Correia

Received: 13 February 2023
Revised: 16 March 2023
Accepted: 22 March 2023
Published: 25 March 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Elliptic curve cryptography (ECC) [1] and Rivest, Shamir, Adleman (RSA) [2] algorithms are current public-key cryptographic standards. The ECC is generally considered a better choice as compared to RSA. The reason is that the ECC offers identical security to RSA with smaller key sizes. For example, 233 bits of ECC offer equivalent security to a 2048-bit RSA [2]. Many other benefits of ECC include low area utilization, low consumed power, and lower channel bandwidth for transmitting critical information on an unsecured public channel. Hence, ECC is more suitable than other public-key cryptographic algorithms for securing applications that demand low area utilization, such as radio frequency identification networks (RFID) [3,4], wireless sensor nodes (WSNs) [5,6], etc.

The National Institute of Standards and Technology (NIST) has defined elliptic curve parameters in [7] for the implementation of prime, i.e., $GF(P)$, and binary, i.e., $GF(2^m)$, fields. The prime field lengths are optimal for software-based implementations [8]. In contrast, the binary field lengths are more valuable for hardware implementations such as field-programmable gate arrays (FPGA) and application-specific integrated circuits (ASIC) [3,9,10].

The ECC provides two basis representations, polynomial and normal, which are essential for computing finite field modular operations. These finite field modular operations

include addition, multiplication, squaring, and inversion. In particular, a polynomial basis allows efficient modular multiplication operations, as shown in [8,11]. On the other hand, the normal basis is a better choice for frequent squaring operations. Similarly, ECC employs affine and projective coordinate representations for executing essential cryptographic operations. However, the affine representation is relatively expensive from a computational point of view [8,9,12,13]. Therefore, we used projective coordinates to implement binary fields with polynomial basis representations in our computations.

Additionally, ECC offers various models (Weierstrass, Edward, and Huff) to implement the critical point multiplication (PM) operation. Comparatively, the Weierstrass model is subjected to simple power analysis attacks as it involves different point addition (PA) and point double (PD) mathematical operations. Simple power analysis attacks are generally tackled in the Weierstrass ECC curve using the Montgomery ladder PM algorithm, which offers identical modular operations to execute the PA and PD instructions. An efficient PM architecture over the Weierstrass binary field is presented in [9] where a rescheduling of PA and PD instructions is proposed to optimize the throughput and reduce the circuit latency. A digit-serial multiplier is used to minimize hardware resources and results are obtained on Virtex-7 FPGA over $GF(2^{163})$ to $GF(2^{571})$. Recently, in 2020, an area-optimized PM design over the Weierstrass ECC curve was described in [10] where one modular multiplier was utilized to operate the multiplication and squaring operations, reducing hardware resource utilization. The results over $GF(2^{163})$ to $GF(2^{571})$ are demonstrated on Virtex-7 and modern 16 nm ASIC technology.

The Edward and Huff curves offer unified PA and PD operations to implement the PM. Comparatively, the Edward model is preferred for throughput-optimized implementations, while the Huff model is convenient for achieving high security. For the Edward curve, a high-speed, low-area, and simple-power-analysis-resistant PM architecture over $GF(P)$ with $P = 256$ is presented in [14] where computational complexity is reduced by rescheduling the unified PA and PD instructions and design requires 198,715 clock cycles and bears 1.9 ms for one PM computation. Moreover, the design occupies 6543 slices on the Virtex-7 FPGA. Another prime-field-based Edward curve PM architecture is described in [12], where one PM requires 1.48 ms and utilizes 8873 slices on Virtex-7 FPGA. An FPGA-based Edward curve accelerator is presented in [15]. The architecture utilizes less than 1400 slices on Virtex-5 FPGA for a level of security equivalent to 128 bits. A low-complexity PM architecture using the Edward curve over $GF(2^{233})$ is described in [16], where authors have reduced the instruction-level complexity by eradicating numerous operations in a single-instruction format.

Over $GF(2^{233})$, a four-stage pipeline architecture for the Huff curve is described in [17]. The authors revisited the original mathematical formulas of Huff curves to reduce the required area and presented simplified formulas with a 43% area reduction. A two-stage pipeline architecture of Huff curves over $GF(2^{233})$ is presented in [13], where pipelining is utilized to shorten the critical path. Moreover, PA and PD formulas are rescheduled to reduce the cycle counts.

The combined design of Weierstrass and Huff cures for PM computation is implemented in [18], where a Montgomery PM algorithm was utilized for the Weierstrass curve, while the traditional Double and Add PM algorithm was used for the Huff curve. The architecture allows a reasonable compromise between the execution time and various protection levels. Another flexible accelerator architecture is given in [19], where flexibility is achieved by implementing different key lengths of 233, 283, 409, and 571 for the Weierstrass curve.

Several design approaches, such as instruction-level parallelism, pipelining, and innovative modular multiplication techniques have been employed in [9,10,12–19] for the advancement of PM computation. Although several unified ECC designs exist in state-of-the-art applications, the higher area complexity of dedicated [9,10,12–17] and flexible [18,19] ECC architectures reveals that these designs are inappropriate for (several) cryptographic applications such as RFID, WSNs, etc. Therefore, to target different reliability/security

levels, this paper realizes a low-complexity unified hardware accelerator over the ECC's Weierstrass, Edward, and Huff models for PM computation.

There is no prior architecture where several models of ECC have been utilized at the same time. Subsequently, for the first time, we present a unified design of PM using three models of ECC, i.e., Weierstrass, Edward, and Huff. The proposed design allows users to target different security levels according to their demands. In addition to flexibility, our proposed unified design is also reliable because it consumes a fixed number of cycles to compute the PM operation in ECC. Moreover, our contributions can be listed as follows:

- We present a low-complexity unified hardware accelerator of various ECC models (i.e., Weierstrass, Edward, and Huff) over $GF(2^{233})$ for PM computation. We used a Montgomery PM algorithm for the Weierstrass and Edward curves, while for the Huff curve, we employed a Double and Add PM algorithm. Our presented architecture covers the design space between the processing time and different protection levels.
- We realize the area complexity of our unified accelerator architecture by optimizing the required memory size to implement Weierstrass, Edward, and Huff curves. The optimization results in a 1.93 times reduction in the required memory size. Additional details are given in Section 3.
- Moreover, we describe a hardware architecture for the Karatsuba modular multiplier to perform polynomial multiplications. Then, we compute the modular inverse using the hardware resources of the proposed modular multiplier and square arithmetic units. This strategy also preserves the lower hardware resources. The corresponding details are provided in Section 4.3.
- To support three different models of ECC, a finite-state machine (FSM) provides multiple control operations. We present the corresponding implementation of FSM in Section 4.5.

The proposed design is implemented in Verilog language. The implementation results for $GF(2^{233})$ are provided on Xilinx Virtex-6 and Virtex-7 platforms. According to the achieved results, the unified accelerator design in this article obtains a maximum of 247 and 294 MHz frequencies on Virtex-6 and Virtex-7 devices. On the same FPGA devices, the hardware resource utilization of our unified design is 6109 (slices on the Virtex-6 device) and 7423 (slices on the Virtex-7 device). The achieved results validate the applicability of the proposed unified accelerator for all those applications that require different reliability/security levels.

This article is arranged as follows: The relevant information on Weierstrass, Edward, and Huff curves is summarized in Section 2. Before presenting the proposed design, the associated memory optimizations are shown in Section 3. Subsequently, the unified accelerator design is explained in Section 4. Similarly, in Section 5, we elaborate upon and compare the achieved results with respect to state-of-the-art implementations. Section 6 provides concluding remarks about the article.

2. Mathematical Background over $GF(2^m)$

Point Multiplication Concept

It is essential to emphasize that an additive group consists of various points on an elliptic curve. This implies that the addition of two group elements yields another group element. Let us take an example. On an elliptic curve, we have two input points (P and Q). Adding P and Q results in $R = P + Q$. Here, R determines the final produced point. The point R also lies on the curve. The addition of P and Q defines the PA. Furthermore, adding two similar points ($P + P = 2P$ or $Q + Q = 2Q$) on the curve defines the point doubling. Note that the PA and PD operations are an essential part of the PM operation. Hence, adding k copies of PA and PD specifies the PM. It is calculated using Equation (1). The variables P, Q, and k determine the initial point, final point, and scalar multiplier.

$$Q = k.P \tag{1}$$

Weierstrass curve. For the $GF(2^m)$ field, a Lopez Dahab projective form of the Weierstrass ECC curve is shown in Equation (2). The variables X, Y, and Z in Equation (2) specify the projective elements of a given point $P(X : Y : Z)$. Here, the variable $Z \neq 0$, while variables a and b are constants. Moreover, $b \neq 0$.

$$E : Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \tag{2}$$

Edward curve. For the $GF(2^m)$ field, the Edward curves are described in [20]. Let us say the elements over $GF(2^m)$ fields are d_1 and d_2 such that $d_1 \neq 0$ and $d_2 \neq d_1(d_1 + 1)$. Subsequently, the mathematical formulation for BEC is expressed as

$$E_{B,d_1,d_2} : d_1(x + y) + d_2(x + y)^2 = xy(x + 1)(y + 1) \tag{3}$$

Variables d_1 and d_2 in Equation (3) are co-coefficients. Similarly, variables x and y represent the coordinates of the input point P. Finally, the variables d_1 and d_2 are the constants of the corresponding curve.

Huff curve. The Huff model for ECC was initially introduced in 2010 [21] and formally presented in 2011 [22]. However, the vulnerabilities against [22] were highlighted in 2013. Consequently, a new mathematical formulation was constructed and presented in [23]. In 2018, the work in [23] was critically reviewed by [24]. As a result, the work in [24] contributes towards novel formulations of the unified Huff curve. Note that our implementations take the formulations for the Huff curve from [24]. Let us consider a binary field with three points $(X : Y : Z)$. The formalism for Huff curves can be expressed as

$$E : aX(Y^2 + YZ + Z^2) = bY(X^2 + XZ + Z^2) \tag{4}$$

Variables a and b are the parameters of curves over the $GF(2^m)$ field with the condition that $a \neq b$.

Algorithms for Weierstrass, Edward and Huff models of ECC. For the implementation of Equation (1), we have numerous state-of-the-art PM algorithms. Some typical examples are the Double and Add algorithm, the Lopez Dahab algorithm, and the Montgomery algorithm. In contrast, the literature recommends the use of the Double and Add algorithm for the PM execution of Huff curves, as these require more instructions to implement unified operations. For performance improvement through instruction-level parallelism, a Lopez Dahab PM algorithm is more practical. However, the Montgomery PM algorithm results in the simple-power-analysis-protected implementation of ECC. We analyzed a variety of PM algorithms of ECC in [25]. Therefore, our implementation uses the Montgomery PM algorithm for the Weierstrass and Edward models of ECC. At the same time, we utilize the Double and Add algorithm for the Huff curve. The corresponding algorithms are presented in Algorithms 1–3. Here, in this article, we are not investigating simple power analysis (SPA) attacks in our design. Instead, we simply utilize the SPA-protected PM algorithm.

Algorithms 1–3 start with the initialization phase, where input points must transform into another coordinate system to obtain more benefits during implementation. Algorithms 1 and 3 compute PM in (Lopez Dahab) projective coordinates while our implemented Algorithm 2 executes the PM in a differential coordinate system. After transforming the coordinates, the PM is implemented (using the statements inside the for loop of Algorithms 1–3). A k_i in Algorithms 1–3 is the bit stream of the scalar multiplier, which determines the execution of the corresponding PA and PD operations. The corresponding PA and PD formulations of Weierstrass, Edward, and Huff curves for PM computations are listed in Table 1. The column provides the number of instructions. Column two presents the PA and PD instructions for the Weierstrass model of ECC. Similarly, the unified addition instructions for Edward ($UADD_{Edw}$) and Huff ($UADD_{Huff}$) curves are shown in columns three and four of Table 1.

Algorithm 1: Montgomery PM algorithm for Weierstrass curve [9,11].

Input: $k = (k_{n-1}, \ldots, k_1, k_0)$ with $k_{n-1} = 1$, $P = (x_p, y_p) \in GF(2^m)$
Output: $Q = (x_q, y_q) = k \cdot P$
Affine to projective conversions: $X_1 = x_p$, $Z_1 = 1$, $Z_2 = x_p^2$ and $X_2 = x_p^4 + b$
PM in projective coordinates:
for *(i from m-2 down to 0)* **do**
  if $(k_i = 1)$ **then**
    $PA(X_1, Z_1) = (X_1, Z_1, X_2, Z_2)$ and $PD(X_2, Z_2) = (X_2, Z_2)$
  else
    $PA(X_2, Z_2) = (X_2, Z_2, X_1, Z_1)$ and $PD(X_1, Z_1) = (X_1, Z_1)$
    end if
  end
  end for
end
Reconversion from projective to affine: $x_q = \frac{X_1}{Z_1}$, $y_q = x_p + (\frac{X_1}{Z_1})[(X_1 + x_p \times Z_1)(X_2 + x_p \times Z_2) + (x_p^2 + y_p)(Z_1 \times Z_2)](x_p \times Z_1 \times Z_2)^{-1} + y_p$

Algorithm 2: Montgomery PM algorithm for Edward curve [16].

Input: $E_{B,d1,d2}/GF(2^m) : d1(x+y) + d2(x+y)^2 = xy(x+1)(y+1)$, $P \in E_{B,d1,d2}$ and $k = (k_{m-1}, \ldots, k_1, k_0)$
Output: $Q = w(k.P)$
Affine to ω conversions: $W_1 = 0$, $Z_1 = 1$, $W_2 = w(P)$ and $Z_2 = w(P)$
PM in ω coordinates:
for *(i from m-2 down to 0)* **do**
  if $(k_i = 0)$ **then**
    $((W_1 : Z_1), (W_2 : Z_2)) := UADD_{Edw}((W_0 : Z_0), (W_1 : Z_1), (W_2 : Z_2))$
  else
    $((W_2 : Z_2), (W_1 : Z_1)) := UADD_{Edw}((W_0 : Z_0), (W_2 : Z_2), (W_1 : Z_1))$
    end if
  end
  end for
end
Return : $(W_1 : Z_1), (W_2 : Z_2)$

Algorithm 3: Double and Add PM algorithm for Huff curve [17].

Input: $k = (k_{n-1}, \ldots, k_1, k_0)$ with $k_{n-1} = 1$, $P = (x, y) \in GF(2^m)$
Output: $Q = \omega(k.P)$
Affine to projective conversions: $X_1 = x_p$, $Y_1 = Y_2 = y_p$, $Z_1 = 1$, $X_2 = x_p^4 + b$ and $Z_2 = x_p^2$
PM in projective coordinates:
for *(i from m-2 down to 0)* **do**
  $Q = UADD_{Huff}(Q, Q)$
  if $(k_i = 1)$ **then**
    $Q = UADD_{Huff}(P, Q)$
  end
  end for
end
Reconversion from projective to affine: $Q(x_q, y_q) = (\frac{X_2}{Z_2}, \frac{Y_2}{Z_2^2})$

Table 1. Original PA and PD formulations for Weierstrass, Edward and Huff curves over $GF(2^m)$.

Inst_i	Weierstrass Curve	Edward Curve	Huff Curve
I_1	$Z_1 = X_2 \times Z_1$	$A = W_1 \times Z_1$	$m_1 = X_1 \times X_2$
I_2	$X_1 = X_1 \times Z_2$	$B = W_1 \times W_2$	$m_2 = Y_1 \times Y_2$
I_3	$T_1 = X_1 + Z_1$	$C = Z_1 \times Z_2$	$m_3 = Z_1 \times Z_2$
I_4	$X_1 = X_1 \times Z_1$	$W_d = A \times A$	$m_4 = (X_1 + Z_1)(X_2 + Z_2)$
I_5	$Z_1 = T_1^2$	$Z_d = ((e_1 \times W_1 + Z_1)^4)$	$m_5 = (Y_1 + Z_1)(Y_2 + Z_2)$
I_6	$T_1 = x_p \times Z_1$	$Z_a = ((e_2 \times B + C)^2)$	$m_6 = m_1 \times m_3$
I_7	$X_1 = X_1 + T_1$	$W_a = ((B \times C + w \times Z_a)^2)$	$m_7 = m_2 \times m_3$
I_8	$Z_2 = Z_2^2$	–	$m_8 = m_1 \times m_2 + m_3^2$
I_9	$T_1 = Z_2^2$	–	$m_9 = m_6(m_2 + m_3)^2$
I_{10}	$T_1 = b \times T_1$	–	$m_{10} = m_7(m_1 + m_3)^2$
I_{11}	$X_2 = X_2^2$	–	$m_{11} = m_8(m_2 + m_3)$
I_{12}	$Z_2 = X_2 \times Z_2$	–	$m_{12} = m_8(m_1 + m_3)$
I_{13}	$X_2 = X_2^2$	–	$X_3 = \alpha \times m_9 + (m_4 + m_{11})m_{11} + m_{11}^2 + Z_3$
I_{14}	$X_2 = X_2 + T_1$	–	$Y_3 = \beta \times m_{10} + (m_5 + m_{12})m_{12} + m_{12}^2 + Z_3$
I_{15}	–	–	$Z_3 = m_{11}(m_1 + m_3)$

In Table 1, variables X_1, X_2, Y_1, Y_2, Z_1, Z_2, W_1, W_2, Z_a, Z_d,W_a and W_d depict the initial and final points in a projective coordinate system. The remaining variables, such as m_1 to m_{12}, A, B, C, and T_1, represent the storage elements/items. These storage items are required to compute the PM operation on the respective ECC model (Weierstrass, Edward, and Huff). In column two of Table 1, the x_p is the base point in affine coordinates while b is the constant for the Weierstrass curve. Similarly, in column three, the values e_1, e_2 and e can be calculated as $\sqrt[4]{e}$, $\sqrt{e}$ and $d_1^4 + d_1^3 + d_1^2 d_2$. Moreover, the variable ω is a rational function for the corresponding elliptic curve. It can be calculated on point P as $w(P) = \frac{x+y}{d_1(x+y+1)}$ for $P = (x, y)$ in $E_{B,d1,d2}$. The $w - coordinate$ differential addition and doubling implies calculating $w(2P_1)$ and $w(P_1 + P_2)$ from the given values $w(P_1)$, $w(P_2)$ and $w(P_1$-$P_2)$, where P_1 and P_2 are the points on E over $GF(2^m)$. The variables α and β in column four of Table 1 are constants. These curve constants are calculated as $\alpha = \frac{a+b}{b}$ and $\beta = \frac{a+b}{a}$. To obtain maximum throughput, our work uses pre-computed values of e_1, e_2, e, α, and β. For more detailed descriptions, we refer readers to [8] (for the Weierstrass curve) [20] (for the Edward curve), and [21–24] (for the Huff curve).

3. Memory Optimizations

As we mentioned in Section 2, the variables in Table 1 are X_1, X_2, X_3, Y_1, Y_2, Y_3, Z_1, Z_2, Z_3, W_1, W_2, Z_a, Z_d, W_a and W_d. These variables hold the initial and final points in a projective coordinate system. Similarly, the remaining variables m_1 to m_{12}, A, B, C, and T_1 are the storage elements required to compute the PM operation on the respective ECC model (Weierstrass, Edward, and Huff). If we count these variables, this becomes 31. We need a memory unit of $31 \times m$ to implement Weierstrass, Edward, and Huff curves simultaneously, where m is the implemented field length. Moreover, columns three and four of Table 1 show that the Edward and Huff curves contain complex mathematical formulations, meaning that some instructions require more than one modular operation to compute. The I_5, i.e., $((e_1 \times W_1 + Z_1)^4)$, can be observed in column three of Table 1. It depends on three modular operations: (i) the multiplication of $e_1 \times W_1$, (ii) $e_1 \times W_1 + Z_1$, and (iii) $((e_1 \times W_1 + Z_1)^4)$. Such complex instructions demand specialized hardware to execute them. Hence, we need to simplify the mathematical formulations of columns three and four of Table 1 to implement them in the proposed unified design.

Consequently, in Table 2, we use the following replacement strategy to obtain simplified mathematical instructions. Moreover, we consider one modular operator (adder, multiplier, and square) to show the simplified instructions.

- ***For Weierstrass curve.*** We replaced only T_1 with m_1. Differences are given in column two of Tables 1 and 2.
- ***For Edward curve.*** We replaced W_1, W_2, A, B, C, W_d, Z_d, Z_a and W_a with m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, and m_9. Moreover, m_{10} to m_{12} are used to simplify the sub-operations of instructions five to seven of column three of Table 1.
- ***For Huff curve.*** We replaced X_3 and Z_3 with X_2 and Z_2. Moreover, we preserved Y_1 and Y_2 in m_{11} and m_{12}. Then, Y_3 is also replaced with m_{12}. The m_9 and m_{10} are used to simplify the sub-operations of instructions four, five, and eight to fifteen of column four of Table 1.

Table 2. Simplified form of a PA and PD formulations for Weierstrass, Edward and Huff curves over $GF(2^m)$.

$\mathbf{Inst_i}$	Weierstrass Curve	Edward Curve	Huff Curve
I_1	$Z_1 = X_2 \times Z_1$	$m_3 = m_1 \times Z_1$	$m_1 = X_1 \times X_2$
I_2	$X_1 = X_1 \times Z_2$	$m_4 = m_1 \times m_2$	$m_2 = m_{11} \times m_{12}$
I_3	$m_1 = X_1 + Z_1$	$m_5 = Z_1 \times Z_2$	$m_3 = Z_1 \times Z_2$
I_4	$X_1 = X_1 \times Z_1$	$m_6 = m_3 \times m_3$	$m_9 = (X_1 + Z_1)$
I_5	$Z_1 = m_1^2$	$m_{10} = e_1 \times W_1$	$m_{10} = (X_2 + Z_2)$
I_6	$m_1 = x_p \times Z_1$	$m_{11} = m_{10} + Z_1$	$m_4 = m_9 \times m_{10}$
I_7	$X_1 = X_1 + m_1$	$m_7 = m_{11}^4$	$m_9 = m_{11} + Z_1$
I_8	$Z_2 = Z_2^2$	$m_{10} = e_2 \times m_4$	$m_{10} = m_{12} + Z_2$
I_9	$m_1 = Z_2^2$	$m_{11} = m_{10} + m_5$	$m_5 = m_9 \times m_{10}$
I_{10}	$m_1 = b \times m_1$	$m_8 = m_{11}^2$	$m_6 = m_1 \times m_3$
I_{11}	$X_2 = X_2^2$	$m_{10} = m_4 \times m_5$	$m_7 = m_2 \times m_3$
I_{12}	$Z_2 = X_2 \times Z_2$	$m_{11} = m_{10} + \omega$	$m_9 = m_3^2$
I_{13}	$X_2 = X_2^2$	$m_{12} = m_{11} \times m_8$	$m_{10} = m_1 \times m_2$
I_{14}	$X_2 = X_2 + m_1$	$m_9 = m_{12}^2$	$m_8 = m_9 + m_{10}$
I_{15}	–	–	$m_9 = m_2 + m_3$
I_{16}	–	–	$m_{10} = m_9^2$
I_{17}	–	–	$m_2 = m_6 \times m_{10}$
I_{18}	–	–	$m_{10} = m_1 + m_3$
I_{19}	–	–	$m_1 = m_{10}^2$
I_{20}	–	–	$m_3 = m_7 \times m_1$
I_{21}	–	–	$m_6 = m_8 \times m_9$
I_{22}	–	–	$m_7 = m_8 \times m_{10}$
I_{23}	–	–	$Z_2 = m_6 \times m_{10}$
I_{24}	–	–	$m_9 = m_6 + m_4$
I_{25}	–	–	$m_{10} = m_9 \times m_6$
I_{26}	–	–	$m_9 = m_6^2$
I_{27}	–	–	$m_1 = m_9 + m_{10}$
I_{28}	–	–	$m_9 = \alpha \times m_2$
I_{29}	–	–	$m_{10} = m_9 + m_1$
I_{30}	–	–	$X_2 = m_{10} + Z_2$
I_{31}	–	–	$m_9 = m_5 + m_7$
I_{32}	–	–	$m_{10} = m_9 \times m_7$
I_{33}	–	–	$m_1 = m_7^2$
I_{34}	–	–	$m_9 = m_{10} + m_1$
I_{35}	–	–	$m_{10} = \beta \times m_3$
I_{36}	–	–	$m_1 = m_9 + m_{10}$
I_{37}	–	–	$Y_2 = m_1 + Z_2$

As shown in Table 2, the Weierstrass curve requires fourteen instructions after simplification. Similarly, columns three and four of Table 2 reveal that the Edward and Huff curves need fourteen and thirty-seven instructions to execute the $UADD_{Edw}$ and $UADD_{Huff}$ formulas of Algorithms 2 and 3. The number of instructions increases with the decrease in storage elements and modular operators. In other words, we show the simplified instructions using the 16 storage elements (X_1, X_2, Z_1, Z_2, and m_1 to m_{12}) and one modular

adder, square, and multiplier operator for Weierstrass, Edward, and Huff curves in Table 2. Consequently, using the above-mentioned strategy, the replacement reduces memory size by 1.93 (ratio of 31 with 16) times compared to the original required memory size of $31 \times m$.

4. Proposed Crypto Processor Architecture

We present the unified architecture of our proposed PM design in Figure 1. It includes five units: (i) a curve parameter block, (ii) a memory block, (iii) an arithmetic and logic unit, (iv) routing networks, and (v) a control block. The curve parameter block bears the initial inputs to execute the PM operation on the selected Weierstrass, Edward, and Huff curves. After the computation, the intermediate and final results are kept on the memory block. The arithmetic and logic unit comprises modular operations (addition, square, multiplication, and inversion). The dedicated control unit produces the related/required control signals. We provide more precise particulars in the subsequent sections.

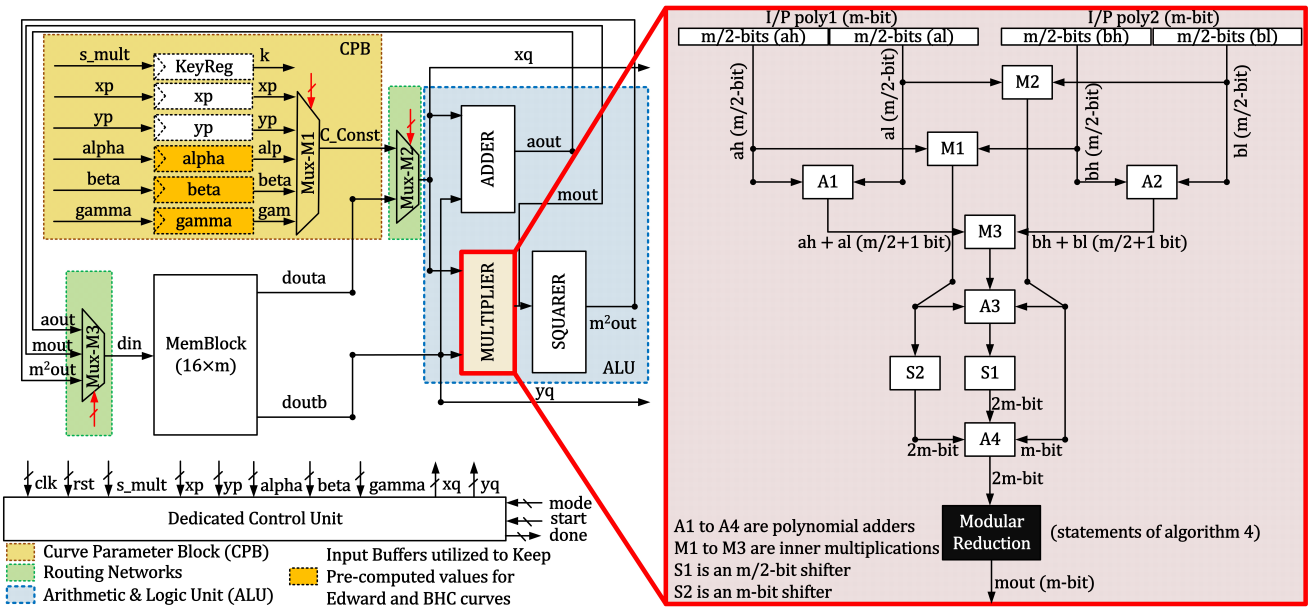


Figure 1. FPGA comparison with most recent PM architectures of the Weierstrass, Edward, and Huff curves.

4.1. Curve Parameter Block (CPB)

As illustrated in Figure 1, the curve parameter block comprises six m-bit buffers (KeyReg, xp, yp, alpha, beta, and gamma). Moreover, it includes one 5×1 multiplexer $M1$. The x and y coordinates of the initial point P are stored in the corresponding buffers (xp and yp). The size of the alpha, beta, and gamma buffers are m-bit, where m is the field size (233 in this work). In our architecture of Figure 1, we used these three buffers to load the pre-computed values for the Edward and Huff curves. More precisely, for the Edward curve, alpha, beta, and gamma buffers keep the values of e_1, e_2, and e during the PM implementation. Similarly, the alpha and beta buffers are needed for the Huff curve to load the pre-computed values of α and β, while we use the gamma buffer to load the y coordinate of the initial point. In the case of the Weierstrass curve, we need only to load parameter b. Hence, we utilized only the alpha buffer to load the curve constant b for the PM computation on the Weierstrass curve. A *KeyReg* buffer holds an m-bit scalar multiplier value to execute the if and *else* pieces of Algorithms 1–3. In short, xp, yp, alpha, beta, gamma, and KeyReg contain the input parameters that need to load from outside to our unified architecture. These buffers are preserved during the entire PM computation. We select the initial values as input for our processor for computation from the standardized document by NIST [7].

4.2. Memory Block

Section 3 describes the memory optimizations to implement the Weierstrass, Edward, and Huff curves at the same time. Consequently, we use a $16 \times m$ array size for a register file (marked with MemBlock in Figure 1). It takes an m-bit vector as input (din) and results in two m-bit vectors as output (douta and doutb). The objective of MemBlock is to keep initial, intermediate, and final results. For read/write operations, it contains two 16×1 multiplexers and one 1×16 demultiplexer (not shown in Figure 1). The multiplexers and demultiplexers provide a mechanism to read two m-bit vectors. The control unit provides the corresponding control signals for two multiplexers and a demultiplexer.

4.3. Arithmetic Unit

Adder and square units. The arithmetic and logic unit of our architecture, shown in Figure 1, contains an adder, a square, and a multiplier. We implemented the adder circuit by executing a bitwise exclusive (OR) operation. It is essential to emphasize that adding two polynomials in the binary field is carry-free. Therefore, for two m-bit polynomials, only the m-exclusive (OR) gates are needed to add polynomials. We have implemented the square circuit by inserting a '0' bit after each successive data bit, as implemented in [10,11,13].

Karatsuba multiplier and polynomial reduction. As shown in Figure 1, we have employed a Karatsuba modular multiplier to perform polynomial multiplication over $GF(2^{233})$. The Karatsuba multiplier bears two m-bit polynomials a and b as input and splits them into two $\frac{m}{2}$ parts. The divided portions of polynomial a are ah and al. Likewise, bh and bl are the pieces of polynomial b. According to [26], we need three multipliers to perform the internal polynomial multiplications. Therefore, in our Karatsuba design of Figure 1, the used multipliers are M1, M2, and M3. The purpose of multiplier M1 is to perform multiplication of ah with bh, whereas M2 multiplies al with bl. The M3 multiplier is liable to perform multiplication over the outputs generated after adders, i.e., A1 and A2. Instead of the multipliers, M1, M2, and M3, four adders (A1, A2, A3, and A4) are also required. Adder A1 adds ah with al. Adder A2 adds bh with bl. Adder A3 accumulates the multiplication results produced by multipliers M1, M2, and M3. Adder A4 yields the final polynomial with $2 \times m$ bit size and drives the output to the reduction block as input to produce an m-bit final output. Despite the multipliers and adders, we also need two shifters (S1 and S2) to shift polynomials by $\frac{m}{2}$ and m bit towards the right. As shown in Figure 1, the multiplier circuit results in a $2m$ bit. Therefore, an essential inversion operation is necessary to implement after the modular square and multiplier circuits. To operate polynomial reduction over $GF(2^{233})$, we implemented the NIST-recommended algorithm, as described in Algorithm 4. We implemented a reduction algorithm using combinational logic, as it requires only shift and exclusive (OR) operations.

Algorithm 4: Polynomial reduction algorithm over $GF(2^{233})$ [8].

Input: Polynomial, $B(x)$ with $2 \times m - 1$-bit length
Output: Polynomial, $B(x)$ with m-bit length

1. *for* (i *from* 15 *down to* 8) *do*
   - 1.1 $T_{emp} \longleftarrow B[i]$
   - 1.2 $B[i-8] \longleftarrow B[i-8] \oplus (T_{emp} \ll 23)$
   - 1.3 $B[i-7] \longleftarrow B[i-7] \oplus (T_{emp} \gg 9)$
   - 1.4 $B[i-5] \longleftarrow B[i-5] \oplus (T_{emp} \ll 1)$
   - 1.5 $B[i-4] \longleftarrow B[i-4] \oplus (T_{emp} \gg 31)$
2. $T_{emp} \longleftarrow B[7] \gg 9$
3. $B[0] \longleftarrow B[0] \oplus T_{emp}$
4. $B[2] \longleftarrow B[2] \oplus (T_{emp} \ll 10)$
5. $B[3] \longleftarrow B[3] \oplus (T_{emp} \gg 22)$
6. $B[7] \longleftarrow B[7]$ & 0x1FF
7. *Return* ($B[7]$, $B[6]$, $B[5]$, $B[4]$, $B[3]$, $B[2]$, $B[1]$, $B[0]$)

Modular inversion. Algorithms 1 and 3 require an inversion operation during the projective to affine conversions. In state-of-the-art, several inversion algorithms exist to operate the polynomial inversion. Therefore, we bear a quad-block version of the Itoh–Tsujii algorithm [27] to implement polynomial inversion. For the $GF(2^{233})$ field, the Itoh–Tsujii algorithm needs $m-1$ squares followed by 10 modular multiplications, where m specifies the length of the polynomial [11,13]. As shown in Figure 1, we connected a square block after the modular multiplier. This strategy permits us to implement the quad-block variant of an Itoh–Tsujii algorithm by executing the first square using a multiplier circuit and then the second square using the square block in the same cycle after the multiplication. This implies that the Itoh–Tsujii inversion algorithm is executed utilizing the resources of the multiplier and square circuits. It is essential to mention that the adder, square, and multiplier circuits in the ALU of our Figure 1 require one clock cycle for implementation because we implemented these circuits using combinational logic.

4.4. Multiplexers M2 Furthermore, M3

The utilized multiplexers (M2 and M3) in Figure 1 decide the routing networks. A multiplexer M2 bears an operand as input from a curve parameter block. It takes the second operand as input from the MemBlock. The output of the multiplexer M2 connects as an input to the ALU for the execution of the modular arithmetic operations. Similarly, multiplexer M3 takes three inputs incoming from an adder, a multiplier, and a square connected after the multiplier unit. The output of M3 ties as input to MemBlock to modify the data contents. Additionally, during each state of the FSM, generating the corresponding control signals is the control unit's responsibility.

4.5. Control Unit and Clock Cycle Calculation

In the following text, we first describe the FSM-based dedicated controller, and then we show the overall cycle counts required to implement Algorithms 1–3.

FSM control unit. The FSM of our accelerator architecture comprises a total of 104 states. Descriptions of these states are as follows:

- *Idle state (State 1).* State one is an idle state. The processor remains in this state until a one-bit start signal becomes 1.
- *Affine to projective conversions (State 2 to State 6).* As the processor receives the start signal, it switches from state one to state two to implement the affine to projective conversions of Algorithms 1–3.
- *Mode checker (State 7).* This is a conditional state. It checks a two-bit mode signal to implement the PM on the respective curve (Weierstrass, Edward, and Huff). The encoding of a two-bit mode signal is as follows: (i) 00 means not to do anything, (ii) 01 determines the execution of PM computation for the Weierstrass curve using Algorithm 1, (iii) 10 means the computation of the PM operation for the selected Edward curve using Algorithm 2, and (iv) 11 shows the implementation of the PM operation for the Huff curve using Algorithm 3.
- *PM computation in projective coordinates (States 8 to 73).* The PM states of the Weierstrass curve are from states 8 to 21, which are liable to execute the formulations of column two of Table 2. Similarly, the PM states of the Edward curve are from states 22 to 35 and are responsible for the execution of the formulations of column three of Table 2. Finally, the Huff curve's PM states are from 36 to 72, and these are responsible for operating the execution of the formulations of column four of Table 2. State 73 acts as a conditional state which checks the inspected key bit value (k_i in Algorithms 1–3). Moreover, the processor switches from state 73 to either 8, 22, or 36 based on a two-bit mode signal and a value of $m-2$. Whenever the value of $m-2$ becomes 0, the processor switches to state 74. It is essential to state that we set the initial value of m to 233.

- *Projective to affine conversions (States 74 to 103).* As shown in Algorithms 1 and 3, the projective to affine conversion needs an inversion operation and some additional arithmetic operations to generate the resultant point on the respective curve (Weierstrass, Edward, and Huff). Therefore, states 74 to 92 execute the modular inversion using the quad-block variant of an Itoh–Tsujii algorithm of [27]. Nine additional states from 93 to 101 generate the control signals to implement the remaining projective to affine conversion instructions of Algorithm 1. Similarly, two states from 102 to 103 generate the control signals to implement the remaining projective to affine conversion instructions of Algorithm 3.
- *Finish (State 104).* This state is responsible for generating the finished signal after completing the PM operation either on the Weierstrass, Edward, or Huff curve, depending on a two-bit mode signal. In the case of the Weierstrass curve, the finish state comes after state 101. Similarly, for Edward curve implementation, the finish state comes after state 73. Finally, for Huff curve implementation, the finish state comes after state 103.

Clock cycle calculations. One clock cycle is needed for the idle state. Moreover, five cycles are required to implement the affine to projective conversions during states 2 to 6. Furthermore, one cycle is needed for the mode checker state of the FSM. As we can see, for loops of Algorithms 1–3 require modular operations to be executed. Moreover, columns two to four of Table 2 reveal that the Weierstrass, Edward, and Huff curves require 14, 14, and 37 instructions to implement the PM operation. Consequently, the complexity of the for loop of Algorithm 1 in terms of required computations is $15 \times (m-2)$ cycles. Here, m specifies the targeted key length (233). Similarly, the cycle count of the for loop of Algorithm 1 is 3465. Furthermore, the execution cost of the for loop of Algorithm 2 is $15 \times (m-2)$ cycles, and the cycle count is 3465. The implementation cost of the for loop of Algorithm 3 is $38 \times (m-2)$ cycles, and the cycle count is 8778. It is important to emphasize that our unified architecture takes m extra cycles to implement the iterative for loops of Algorithms 1–3. The reason is a conditional state to check the inspected key bit value based on the mode signal.

As we stated earlier, the projective to affine conversions in Algorithms 1 and 2 require an inversion operation. As mentioned, the inversion requires $m-1$ squares followed by 10 modular multiplications. In our unified architecture of Figure 1, a modular square and a multiplier take one clock cycle for execution. Using the quad-block variant of an Itoh–Tsujii algorithm, $m-1$ squares can be executed in $\frac{m-1}{2}$ cycles. Thus, the cycle count for one inversion over $GF(2^{233})$ is 126 (116 cycles for $m-1$ square computations and 10 cycles for multiplications). Additionally, 9 and 2 cycles are required to compute the remaining instructions of the projective to affine conversions of Algorithms 1 and 3.

Consequently, if we accumulate the clock cycles to calculate the total counts, the Weierstrass, Edward, and Huff curves require 3607, 3472, and 8913 cycles to implement the corresponding Algorithms 1–3.

5. Implementation Results and Comparison

We implement our proposed unified processor architecture in Verilog HDL using the Vivado IDE tool over $GF(2^{233})$. This tool supports the implementation of the PM over the Weierstrass, Edward, and Huff curves at the same time. For the selected binary field length of 233, our unified architecture takes inputs from the NIST specifications [7]. Consequently, the achieved results and their significance as compared to existing designs are shown in Sections 5.1 and 5.2, respectively.

5.1. Implementation Results

Table 3 shows the implementation results of our proposed unified architecture. The first column shows the implemented ECC curves (Weierstrass, Edward, and Huff). The second column represents the implementation platform (the name of a particular device used in the experiments). The consumed area information is generally provided in three

different notations. These notations are slices (column three), look-up tables (represented as LUTs and shown in column four), and flip-flops (represented as FFs and shown in column five). The circuit frequency (Freq), total cycle counts, information about latency, and the achieved throughput (represented as Thrpt) results are available in columns six to nine of Table 3. To evaluate the performance of our unified architecture, we built a figure of merit (FoM); related values are available in the last column. Note that the slices, LUTs, and FFs values were taken directly from the Vivado synthesis tool. We already described the total cycle counts for different ECC curves (Weierstrass, Edward, and Huff) in Section 4.5. Latency specifies the time needed to execute one PM operation, and it can be calculated using Equation (5). The throughput values can be calculated using Equation (6). Finally, dividing the achieved throughput by the total number of slices provides the value of FoM (Equation (7)).

$$Latency\ (\mu s) = \frac{Clock\ cycles}{Clock\ frequency\ (MHz)} \tag{5}$$

$$Throughput\ (Thrpt) = \frac{1}{Latency\ (\mu s)} = \frac{10^6}{Latency} \tag{6}$$

$$FoM = \frac{Throughput\ (Thrpt)}{Area\ (Slices)} \tag{7}$$

Table 3. Implementation results of our unified accelerator architecture over $GF(2^{233})$ on FPGA.

ECC Curve	Technology	Slices	LUTs	FFs	Freq (MHz)	Cycles	Lat (µs)	Thrpt	FoM
Weierstrass						3607	14.60	68.49 kbps	11.21
Edward	Virtex-6 (40 nm)	6109	19,869	3421	246.738 ≈ 247	3472	14.05	71.17 kbps	11.65
Huff						8913	36.08	27.71 kbps	4.53
Weierstrass						3607	12.26	81.56 kbps	10.98
Edward	Virtex-7 (28 nm)	7423	21,917	3824	293.671 ≈ 294	3472	11.80	84.74 kbps	11.41
Huff						8913	30.31	32.99 kbps	4.44

As can be seen in Table 3, the area utilization of our proposed unified accelerator architecture increases when driving from Virtex-6 to Virtex-7 FPGA devices. The reason is the technology advancement, as Virtex-6 is produced on 40 nm process technology, while Virtex-7 is built on 28 nm technology. Moreover, column six of Table 3 reveals that the circuit frequency also increases when we change the implementation device from Virtex-6 to modern Virtex-7 FPGA.

As for the clock cycles used for comparison, the Edward curve requires the lower cycle count of 3472, compared with the Weierstrass (3607) and Huff (8913) curves. This is normally due to the number of PA and PD computation instructions. As shown in columns two to four of Table 2, the instructions for PA and PD computations for Weierstrass, Edward, and Huff curves are fourteen, fourteen, and thirty-seven. Similarly, the execution cost (latency) of Weierstrass, Edward, and Huff curves is obtained using Equation (5). Subsequently, the calculated latency values for Weierstrass, Edward, and Huff curves on the Virtex-6 device are 14.60, 14.05, and 36.08 µs. The maximum circuit frequency achievement for Virtex-7 FPGA ensures a shorter computation time. Hence, on Virtex-7 FPGA, the calculated latency values for Weierstrass, Edward, and Huff curves are 12.26, 11.80, and 30.31 µs. Our Virtex-7 implementation of the unified design (for implementing Weierstrass, Edward, and Huff curves) is 1.19× faster than the Virtex-6 implementation. Similarly, our Virtex-7 implementation of the unified accelerator architecture for implementing Weierstrass, Edward, and Huff curves results in a 0.83× improvement compared with the Virtex-6 implementation.

If we consider the throughput and hardware area at the same time for comparison, i.e., FoM from Equation (7), it can be observed that the unified design in this article performs better on Virtex-6 FPGA than the Virtex-7 implementation. It is essential to mention that the higher value of FoM guarantees the higher performance of the architecture. As we can see in Equation (7), the defined FoM metric is a ratio of throughput with slices, so the lower hardware resources (area) achieved for Virtex-6 result in higher FoM.

5.2. Comparison with the State of the Art

We compare the implementation of our unified accelerator architecture with state-of-the-art accelerators in Table 4. Column one provides the reference of the implemented design (Ref). The implemented curve and the corresponding PM algorithm are presented in columns two to three. Column four shows the type (either dedicated or flexible/unified) of the implemented design. The field size m is shown in column five. The implementation device for logic synthesis and place-and-route steps is illustrated in column six. The area information is provided in columns seven to eight. We show the timing results, frequency of the circuit, and information about the latency in the last three columns (nine to eleven) of Table 4.

Table 4. FPGA comparison with the most recent PM architectures of the Weierstrass, Edward, and Huff curves.

Ref	Implemented Curves	PM Algorithm	Design	m	Device	Slices	LUTs	Cycles	Freq MHz	Lat µs	Thrpt kbps
[9]	Weierstrass	Montgomery	Dedicated	233	Virtex-7	2647	7895	–	370	16.01	62.46
[10]	Weierstrass	Montgomery	Dedicated	233	Virtex-7	2048	6407	–	379	14.25	70.15
[19]	Weierstrass	Montgomery	Flexible	233	Virtex-6	–	116,241	2609	135	19.33	51.73
[14]	Edward	Double and Add	Dedicated	256	Virtex-6	6600	–	–	93	2130	0.47
[14]	Edward	Double and Add	Dedicated	256	Virtex-7	6500	–	–	104	1900	0.53
[12]	Edward	Montgomery	Dedicated	256	Virtex-6	9246	33,238	262,650	161	1630	0.61
[12]	Edward	Montgomery	Dedicated	256	Virtex-7	8873	32,781	262,650	178	1480	0.67
[15]	Edward	Montgomery	Dedicated	233	Virtex-6	1245	3878	718,805	107	6720	0.15
[16]	Edward	Montgomery	Dedicated	233	Virtex-7	2662	24,727	3244	179	18.10	55.24
[17]	Huff	Double and Add	Dedicated	233	Virtex-7	7017	–	13,057	434	30.08	33.24
[13]	Huff	Double and Add	Dedicated	233	Virtex-7	7123	–	15,495	188	82.4	12.13
[18]	Weierstrass	Montgomery	Unified	233	Virtex-7	8866	23,017	5635	271	20.78	48.12
	Huff	Double and Add						12,554		46.32	21.58
TW	Weierstrass	Montgomery	Unified	233	Virtex-6	6109	19,869	3607	247	14.60	68.49
	Edward	Montgomery						3472		14.05	71.17
	Huff	Double and Add						8913		36.08	27.71
	Weierstrass	Montgomery	Unified	233	Virtex-7	7423	21,917	3607	294	12.26	81.56
	Edward	Montgomery						3472		11.80	84.74
	Huff	Double and Add						8913		30.31	32.99

The design of [19] supports different binary fields (233, 283, 409 and 571). Cycles and latency values are for a 233-bit size. The design of [15] uses $d_1 = d_2$ and operates PM in $\omega - coordinates$ (like our provided Algorithm 2). The results for the design of [16] are reported for $d_1 = d_2 = 26$. This design also computes PM in $\omega - coordinates$. The design of [18] implements Montgomery and Double and ADD PM algorithms for Weierstrass and Huff curves. TW represents the implementation of our unified accelerator architecture.

Comparison with dedicated designs of Weierstrass curve [9,10]. The comparison to dedicated designs of [9,10] shows that our unified accelerator utilizes more FPGA resources in slices and LUTs. The reason is the flexibility in our design to support three different ECC curves, i.e., Weierstrass, Edward, and Huff. In contrast, the dedicated designs of [9,10] are specific to Weierstrass curve implementation on Virtex-7 FPGA. As shown in column eight of Table 3, comparing clock cycle utilization is impossible as the related data are unavailable in the reference designs. On the same Virtex-7 FPGA device, our unified accelerator can

operate at a maximum of 294 MHz, while the designs of [9,10] can operate at maximum frequencies of 370 and 379 MHz, respectively. This is due to the flexibility considered in our accelerator. Apart from the area and frequency comparison, our unified accelerator requires 1.30 (ratio of 16.01 with 12.26) and 1.16 (ratio of 14.25 with 12.26) times lower computation time as compared to the dedicated architectures of [9,10], respectively. The reported latency is the ratio of clock cycle counts over circuit frequency in Equation (5); thus, we believe the designs of [9,10] take higher cycle counts compared to our unified design, which is why their architectures require more computational time than the unified design of this work.

Comparison with dedicated Edward curve designs of [12,14–16]. The Edward designs over the prime field with 256-bit length are described in [12,14]. On Virtex-6 and Virtex-7 FPGA devices, our unified accelerator utilizes lower slices and LUTs, as shown in columns seven and eight of Table 3. This is due to a lower key length of 233 bits implemented in our design, while a 256-bit key length is supported in the reference designs of [12,14]. The last two columns of Table 3 (columns ten and eleven) reveal that the proposed unified accelerator can operate on a higher circuit frequency and takes a lower computation time than the Edward curve accelerators of [12,14]. As we stated earlier, our accelerator supports a shorter 233-bit key length than [12,14], where a 256-bit supported accelerator is implemented—this is the reason for the lower computation time in our accelerator design.

Over $GF(2^{233})$, a dedicated Montgomery-based PM accelerator architecture of the Edward curve is presented in [15]. On Virtex-6 FPGA, their architecture utilizes 4.90 (ratio of 6109 with 1245) times lower slices than our unified accelerator. On the other hand, our accelerator utilizes 207.02 (ratio of 718,805 with 3472) times lower clock cycles. In our design, we use a bit-parallel Karatsuba multiplier for polynomial multiplications, resulting in one multiplication in one clock cycle with an area overhead. On the other hand, a serial multiplication architecture is incorporated in [15] to optimize the hardware resources with clock cycles overhead. There is always a tradeoff between area and performance (clock cycles). The use of serial and parallel multiplication approaches affects the critical path of the PM circuits. More precisely, our accelerator can operate on a maximum of 247 MHz, while the design of [15] can operate on 107 MHz. As shown in the last column of Table 3, the larger clock cycle utilization and lower circuit frequency in [15] result in a longer computation time (latency) compared with our unified accelerator.

Another Montgomery-based dedicated PM accelerator of the Edward curve over $GF(2^{233})$ is described in [16]. If we compare our implementation results on Virtex-7 FPGA, the design of [16] is more area-efficient (in slices), as their design comprises a 32-bit digit parallel multiplier—we employed a bit-parallel Karatsuba multiplier. Moreover, their design takes a lower number of clock cycles of 3244 for one PM computation, while we utilized 3472 cycles. However, our accelerator architecture is 1.64 (ratio of 294 with 179) times faster in circuit frequency. This is due to a shorter critical path delay in our PM architecture, accomplished by an efficient Karatsuba implementation. The higher circuit frequency in our accelerator results in lower computation time (latency), as shown in the last column of Table 3.

Comparison with dedicated Huff curve designs of [13,17]. On a similar Virtex-7 FPGA device, our unified accelerator takes a higher number of bit slices (7423) as compared to the dedicated Huff designs of [17] (7017) and [13] (7123). The reason is our accelerator's support for three different curves (Weierstrass, Edward, and Huff). Our efficient memory replacements of storage elements and the implementation of the quad block variant of an Itoh–Tsujii inversion algorithm result in lower clock cycles. As you can see in column nine of Table 3, the dedicated designs of [13,17] take 13,057 and 15,495 cycles, while our accelerator to implement the Huff curve takes only 8913 cycles. Due to four-stage pipelining, the design of [17] has an efficient in-circuit frequency, but for the computation time, our accelerator computes one PM in almost the equivalent time (30.31 μs) consumed in [17] (30.08 μs). The last two columns in Table 3 show that the proposed accelerator is more efficient in clock frequency and computation time (latency) compared with a dedicated Huff curve architecture of [13].

Comparison with flexible/unified architectures of [18,19]. If we compare our unified accelerator twith a flexible implementation of [19] on Virtex-6 FPGA, our unified implementation is 5.85 (ratio of 116,241 with 19,869) times more area-efficient (in terms of LUTs). We cannot compare the slices as the reference design does not give this information. It is essential to note that the architecture of [19] is flexible as it implements four different NIST-specified binary field key lengths of 233, 283, 409, and 571. In contrast, our design is flexible as we implement three different curves (Weierstrass, Edward, and Huff) using a single key length of 233 bits. The clock cycle comparison shows that the design of [19] requires 2609 cycles for 233-bit key-length while for the identical key-length, our unified accelerator takes 3607 cycles—that is, 1.38 times higher than [19]. Instead of the area and cycle requirements, the last two columns of Table 3 demonstrate that our accelerator can operate on the maximum circuit frequency and takes a lower computation time than [19].

For the identical $GF(2^{233})$ field, a unified accelerator over Weierstrass and Huff curves is described in [18]. Similar to our unified accelerator, Montgomery and Double and ADD PM algorithms are implemented for the Weierstrass and Huff curves, as shown in columns two and three of Table 3. On the exact Virtex-7 FPGA device, our design is 1.19 (ratio of 8866 with 7423) and 1.05 (ratio of 23,017 with 21,917) times more efficient in slices and LUTs. Furthermore, columns nine to eleven of Table 3 show that we utilize lower clock cycles, achieve higher circuit frequency and take less computation time (latency) for implementing the Weierstrass and Huff curves. Our efficient replacement of storage elements (described in Section 3) for executing PA and PD instructions results in lower clock cycle counts. The efficient implementation of our Karatsuba multiplication architecture allows us to minimize hardware resources. In contrast, a digit-parallel multiplication design with a digit size of 32 bits is described in [18]. The efficient implementation of an FSM-based dedicated controller has the benefit of minimizing the critical path delay, resulting in higher circuit frequency. In addition to other design parameters (area, clock cycles, operating frequency, and latency), the last column of Table 3 shows that the proposed unified design outperforms the most recent state-of-the-art designs of Weierstrass, Edward, and Huff curves in terms of throughput.

6. Conclusions

This paper presents an area-aware unified hardware accelerator of Weierstrass, Edward, and Huff curves over $GF(2^{233})$ for PM computation on FPGA. It has been demonstrated that replacing storage elements, as well as utilizing the modular square block after the multiplier unit to operate the quad-block variant of a modular inversion, helps reduce the hardware resources used in ECC designs. In addition, using a common adder, multiplier, and square unit among different Weierstrass, Edward, and Huff curves is also beneficial to reduce hardware resources. This type of strategy is not only applicable to ECC but can be utilized in any digital system design. On the Virtex-7 FPGA device, our unified accelerator achieves a maximum clock frequency of 294 MHz and requires 7423 slices. The comparison with state-of-the-art solutions shows that the proposed unified architecture provides a large design space between the processing time and various protection levels. Hence, it benefits cryptographic applications that need different security levels simultaneously.

Author Contributions: Conceptualization, M.A. and M.R.; methodology, M.M.; software, O.S.S. and M.H.S.; validation, M.R.; formal analysis, M.A.; investigation, O.S.S. and M.M.; resources, M.R. and M.H.S.; data curation, O.S.S. and M.M.; writing—original draft preparation, M.M.; writing—review and editing, M.A. and M.R.; visualization, M.R.; supervision, M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Deanship of Scientific Research at Umm Al-Qura University under grant number 22UQU4320020DSR01.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the Deanship of Scientific Research at Umm Al-Qura University for supporting this work.

Conflicts of Interest: The authors declare that they have no conflict of interest.

References

1. Miller, V.S. Use of Elliptic Curves in Cryptography. In Proceedings of the Advances in Cryptology—CRYPTO '85 Proceedings; Williams, H.C., Ed.; Springer: Berlin/Heidelberg, Germany, 1986; pp. 417–426.
2. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
3. Noori, D.; Shakeri, H.; Niazi, T.M. Scalable, efficient, and secure RFID with elliptic curve cryptosystem for Internet of Things in healthcare environment. *Eurasip J. Inf. Secur.* **2020**, *13*, 1–11.
4. Calderoni, L.; Maio, D. Lightweight Security Settings in RFID Technology for Smart Agri-Food Certification. In Proceedings of the 2020 IEEE International Conference on Smart Computing (SMARTCOMP), Bologna, Italy, 14–17 September 2020; pp. 226–231. [CrossRef]
5. Kumar, K.A.; Krishna, A.V.N.; Chatrapati, K.S. New secure routing protocol with elliptic curve cryptography for military heterogeneous wireless sensor networks. *J. Inf. Optim. Sci.* **2017**, *38*, 341–365. [CrossRef]
6. Gulen, U.; Baktir, S. Elliptic Curve Cryptography for Wireless Sensor Networks Using the Number Theoretic Transform. *Sensors* **2020**, *20*, 1507. [CrossRef] [PubMed]
7. NIST. Recommended Elliptic Curves for Federal Government Use. 1999. Available online: https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf (accessed on 17 December 2022).
8. Hankerson, D.; Menezes, A.J.; Vanstone, S. Guide to Elliptic Curve Cryptography 2004. pp. 1–311. Available online: https://link.springer.com/book/10.1007/b97644 (accessed on 28 December 2022).
9. Khan, Z.U.A.; Benaissa, M. Throughput/Area-efficient ECC Processor Using Montgomery Point Multiplication on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2015**, *62*, 1078–1082. [CrossRef]
10. Imran, M.; Pagliarini, S.; Rashid, M. An Area Aware Accelerator for Elliptic Curve Point Multiplication. In Proceedings of the 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, Scotland, 23–25 November 2020; pp. 1–4. [CrossRef]
11. Imran, M.; Rashid, M.; Jafri, A.R.; Kashif, M. Throughput/area optimised pipelined architecture for elliptic curve crypto processor. *IET Comput. Digit. Tech.* **2019**, *13*, 361–368. [CrossRef]
12. Islam, M.M.; Hossain, M.S.; Hasan, M.K.; Shahjalal, M.; Jang, Y. FPGA Implementation of High-Speed Area-Efficient Processor for Elliptic Curve Point Multiplication Over Prime Field. *IEEE Access* **2019**, *7*, 178811–178826. [CrossRef]
13. Rashid, M.; Imran, M.; Kashif, M.; Sajid, A. An Optimized Architecture for Binary Huff Curves with Improved Security. *IEEE Access* **2021**, *9*, 88498–88511. [CrossRef]
14. Islam, M.M.; Hossain, M.S.; Hasan, M.K.; Shahjalal, M.; Jang, Y.M. Design and Implementation of High-Performance ECC Processor with Unified Point Addition on Twisted Edwards Curve. *Sensors* **2020**, *20*, 5148. [CrossRef] [PubMed]
15. Lara-Nino, C.A.; Diaz-Perez, A.; Morales-Sandoval, M. Lightweight elliptic curve cryptography accelerator for internet of things applications. *Ad. Hoc. Netw.* **2020**, *103*, 102159. [CrossRef]
16. Sajid, A.; Rashid, M.; Imran, M.; Jafri, A.R. A Low-Complexity Edward-Curve Point Multiplication Architecture. *Electronics* **2021**, *10*, 1080. [CrossRef]
17. Rashid, M.; Imran, M.; Jafri, A.R.; Mehmood, Z. A 4-Stage Pipelined Architecture for Point Multiplication of Binary Huff Curves. *J. Circuits Syst. Comput.* **2020**, *29*, 2050179. [CrossRef]
18. Imran, M.; Rashid, M.; Raza Jafri, A.; Najam-ul Islam, M. ACryp-Proc: Flexible Asymmetric Crypto Processor for Point Multiplication. *IEEE Access* **2018**, *6*, 22778–22793. [CrossRef]
19. Zhao, X.; Li, B.; Zhang, L.; Wang, Y.; Zhang, Y.; Chen, R. FPGA Implementation of High-Efficiency ECC Point Multiplication Circuit. *Electronics* **2021**, *10*, 1252. [CrossRef]
20. Bernstein, D.J.; Lange, T.; Rezaeian Farashahi, R. Binary Edwards Curves. In *Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2008*; Oswald, E., Rohatgi, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 244–265.
21. Joye, M.; Tibouchi, M.; Vergnaud, D. Huff's model for elliptic curves. In *Proceedings of the International Algorithmic Number Theory Symposium*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 234–250.
22. Devigne, J.; Joye, M. Binary Huff Curves. In *Proceedings of the Topics in Cryptology—CT-RSA 2011*; Kiayias, A., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 340–355.
23. Ghosh, S.; Kumar, A.; Das, A.; Verbauwhede, I. On the implementation of unified arithmetic on binary huff curves. In *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 349–364.
24. Cho, S.M.; Jin, S.; Kim, H. Side-channel vulnerabilities of unified point addition on binary huff curve and its Countermeasure. *Appl. Sci.* **2018**, *8*, 2002. [CrossRef]

25. Rashid, M.; Imran, M.; Jafri, A.R.; Al-Somani, T.F. Flexible Architectures for Cryptographic Algorithms—A Systematic Literature Review. *J. Circuits Syst. Comput.* **2019**, *28*, 1930003. [CrossRef]
26. Imran, M.; Abideen, Z.U.; Pagliarini, S. An Open-source Library of Large Integer Polynomial Multipliers. In Proceedings of the 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Vienna, Austria, 7–9 April 2021; pp. 145–150. [CrossRef]
27. Itoh, T.; Tsujii, S. A fast algorithm for computing multiplicative inverses in GF (2m) using normal bases. *Inf. Comput.* **1988**, *78*, 171–177. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.