IET Computers & Digital Techniques

*Research Article*

IET Journals
The Institution of Engineering and Technology

# High-performance elliptic curve cryptography processor over NIST prime fields

ISSN 1751-8601
Received on 26th March 2016
Revised 15th July 2016
Accepted on 20th August 2016
E-First on 30th November 2016
doi: 10.1049/iet-cdt.2016.0033
www.ietdl.org

*Md Selim Hossain*[1] ✉, *Yinan Kong*[1], *Ehsan Saeedi*[1], *Niras C. Vayalil*[1]

[1]*Department of Engineering, Macquarie University, Sydney, NSW 2109, Australia*

✉ *E-mail: md.selim.hossain@mq.edu.au*

**Abstract:** This study presents a description of an efficient hardware implementation of an elliptic curve cryptography processor (ECP) for modern security applications. A high-performance elliptic curve scalar multiplication (ECSM), which is the key operation of an ECP, is developed both in affine and Jacobian coordinates over a prime field of size $p$ using the National Institute of Standards and Technology standard. A novel combined point doubling and point addition architecture is proposed using efficient modular arithmetic to achieve high speed and low hardware utilisation of the ECP in Jacobian coordinates. This new architecture has been synthesised both in application-specific integrated circuit (ASIC) and field-programmable gate array (FPGA). A 65 nm CMOS ASIC implementation of the proposed ECP in Jacobian coordinates takes between 0.56 and 0.73 ms for 224-bit and 256-bit elliptic curve cryptography, respectively. The ECSM is also implemented in an FPGA and provides a better delay performance than previous designs. The implemented design is area-efficient and this means that it requires not many resources, without any digital signal processing (DSP) slices, on an FPGA. Moreover, the area–delay product of this design is very low compared with similar designs. To the best of the authors' knowledge, the ECP proposed in this study over $\mathbb{F}_p$ performs better than available hardware in terms of area and timing.

## 1 Introduction

The demand for secure transactions over the network and associated appliances has increased rapidly in recent times. Advanced communication systems require secure information transmission in different areas such as health care, confidential systems, storage, and financial services. For these applications, asymmetric cryptography, or public-key cryptography (PKC), plays a vital role in passing secured information among different devices. PKC offers an important type of technology for key agreement, encryption/decryption, and digital signatures. The algorithm associated with PKC [e.g. elliptic curve cryptography (ECC)] should be designed so that it requires minimal resources with the assurance of high security and throughput. A high-performance hardware implementation is vital for ECC, especially to speed up the calculations in an elliptic curve cryptography processor (ECP) [1, 2].

### *1.1 Related work*

ECC was first proposed in the mid-1980s by Koblitz [3] and Miller [4]. The ECC and Rivest–Shamir–Adleman (RSA) [5] cryptosystems are the two most popular and powerful public-key encryption methods for cryptographic applications. However, ECC can provide the same level of security as the traditional RSA cryptosystem with a significantly shorter key. For example, a 256-bit ECC over a prime field provides the same level of security as a 3072-bit RSA. In addition, less memory and hardware resources are required to implement elliptic curve cryptosystems [6–9]. This smart and attractive feature makes ECC very popular for resource-constrained devices such as smart cards, credit cards, pagers, personal digital assistants, and cellular phones. The IEEE [10] and National Institute of Standards and Technology (NIST) [8] have standardised elliptic curve (EC) parameters over GF($p$) and GF($2^m$) for PKC. Moreover, Certicom has provided NIST-recommended EC domain parameters, which are standard for efficient cryptography in SEC2 (Standards for Efficient Cryptography) [9]. Field-programmable gate array (FPGA) technology has been used for hardware implementation of an ECP; it assures low cost, better performance, shorter design time, and greater system flexibility, for instance updating algorithms.

To date, several FPGA-based hardware implementations of ECPs over a prime field have been proposed [1, 2, 6, 7, 11–19]. The core operation, of elliptic curve scalar/point multiplication (ECSM/ECPM), is defined as $R = k \cdot P$, where the multiplication of an EC point $P$ by a scalar $k$ provides the resultant point $R$ [20]. Scalable/flexible FPGA-based ECPs were proposed by Loi and Ko [1] and Ananyi *et al.* [13], respectively. Both these ECPs support all five prime-field elliptic curves recommended by NIST. ECPs over GF($p$) on an FPGA were proposed in [2, 14], and a parallel architecture unit is used for ECC. In [15], they also synthesised their ECSM architecture on a 130 nm CMOS application-specific integrated circuit (ASIC). They used the double-and-add always algorithm for implementing ECSM. In [15], Lai and Huang proposed a dual-field ECP, implemented on a TSMC 130 nm CMOS ASIC. Marzouqi *et al.* [6] and Vliegen *et al.* [12] proposed an FPGA-based ECP over an NIST prime field $\mathbb{F}_{256}$ on a Xilinx Virtex-5 and Virtex-II FPGA, respectively. A programmable PKC coprocessor was proposed by Mentens *et al.* [18] and a reconfigurable modular arithmetic logic unit for PKC was developed by Sakiyama *et al.* [19]. These ECPs were implemented over prime field $\mathbb{F}_{256}$ on a Xilinx Spartan-3 FPGA. Ahmadi and Afzali-Kusha [16] and Fan *et al.* [17] proposed ECPs over a prime field $\mathbb{F}_{192}$ on 0.13 µm CMOS and Xilinx Virtex-II FPGA, respectively. Although a few high-speed ECPs have been reported over $\mathbb{F}_p$, most are only area-efficient or are superior only in terms of speed. To have a trade-off between speed and area complexities, a well-designed and efficient ECP is a better option for modern cryptographic applications.

### *1.2 Our contribution*

In this paper, we present a new ECSM with a focus on a system-level description of an efficient FFMA for implementing area-efficient and faster ECP hardware over prime field $\mathbb{F}_p$. The major contributions of this paper are as follows:

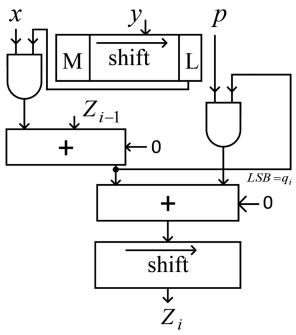


**Fig. 1** *Hardware architecture for a modular multiplier/squarer using Montgomery method [23, 24]*

i. We have proposed a novel elliptic curve scalar multiplication (ECSM) architecture using point doubling and point addition (PDPA) hardware in Jacobian coordinates; it is the fastest hardware implementation both in ASIC and FPGA.
ii. We have developed an architecture for Jacobian-to-affine coordinate conversion, serial-in parallel-out (SIPO), and parallel-in serial-out (PISO) at the top level in order to interface the I/O ports of the ECC processor (ECP) due to the limited number of pins on the FPGA.
iii. We have also proposed an efficient ECP in affine coordinates in which ECSM operations are achieved in a very low area (around 9 K slices without using any DSP slices) and latency (20% less than recent implementations).
iv. We have proposed a new hardware for a combined EC group operation named point doubling and point addition (PDPA) to develop a high-performance ECP in Jacobian coordinates.
v. We have designed an optimised data-flow architecture for EC group operations such as point doubling (PDBL) and point addition (PADD) for the ECP in affine coordinates.
vi. We have proposed and developed high-performance finite-field modular arithmetic (FFMA), for example modular addition, subtraction, multiplication, and inversion algorithms, with hardware architectures over GF($p$) ($\mathbb{F}_p$). To improve these FFMAs and hence the EC group operations, efficient algorithmic reformulations underlying the NIST prime field and architectural optimisation schemes are proposed.

## 2 Mathematical background

In this section, a brief introduction to abstract algebra, field and group theories relevant to the ECP used in this hardware implementation are presented.

### *2.1 Elliptic curve cryptography*

ECC can be implemented in either prime fields ($\mathbb{F}_p$) or binary fields (GF($2^m$)); both provide almost the same level of security. To design an efficient FFMA, use of an EC over a prime field has been intensively investigated. An EC defined over $\mathbb{F}_p$ provides a group structure that is used to implement cryptographic systems. The group operations are PDBL and PADD. We have implemented all EC operations in both affine and Jacobian coordinates. An elliptic curve E over GF($p$) in affine coordinates is the set of solutions for an equation such as

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

where $x, y, a, b \in$ GF($p$) with

$$4a^3 + 27b^2 \neq 0.$$

The coefficients $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ are defined by (1). The number of points on elliptic curve E is represented by $\#E(\mathbb{F}_p)$. It is defined over $\mathbb{F}_p$ as $nh$, where $n$ is the prime order of the curve and the integer $h$ is a co-factor such as $h = \#E(\mathbb{F}_p)/n$. A detailed elliptic curve group operation in affine coordinates is found in [3, 4, 20], and described in Section 3.3.

Let $P = (x, y)$ be a point in an affine coordinate system; the projective coordinates $P = (X, Y, Z)$ are given by the following:

$$X = x; \quad Y = y; \quad Z = 1. \tag{2}$$

The projective point $P = (X, Y, Z)$, $Z \neq 0$ corresponding to the affine point ($P = (x, y)$) is given by

$$x = X/Z^2; \quad y = Y/Z^3. \tag{3}$$

Using (1)–(3), the projective form of the Weierstrass equation of the elliptic curve becomes

$$Y^2 = X^3 + aXZ^4 + bZ^6. \tag{4}$$

EC group operations formulae in Jacobian coordinates are given in [20, 21]; the ECSM $R = kP$, which is the most important operation in ECC has been implemented in Jacobian coordinates.

## 3 Hardware architecture over GF($p$) for ECC

This section presents all algorithms and hardware architectures related to FFMA and ECC which are important for the ECP. All parameters and standards for NIST elliptic curves over $\mathbb{F}_{224}$ and $\mathbb{F}_{256}$ are listed in [20].

### *3.1 Modular multiplier/squarer over $\mathbb{F}_p$*

In order to implement the ECSM, modular multiplication is mandatory because this is the most crucial operation for the ECP over the prime field, presented as

$$Z = (x \times y)(\bmod\ p). \tag{5}$$

The Montgomery modular multiplication (MMM) method has been used to implement the ECP in affine coordinates also, because modular multiplication can be performed very efficiently. In 1985 the well-known Montgomery multiplication algorithm [22] was shown to be an efficient method for performing modular multiplication. This method calculates the Montgomery product using a series of simple additions and right shifts. A hardware architecture of the MMM/squarer is presented in Fig. 1. This method avoids the need for costly trial division by modulus $p$, and keeps the intermediate result bounded to $(m+2)$ bits throughout the calculation. The Montgomery modular product is given by

$$\begin{aligned} R &= \text{Montgpro}(x, y, p) \\ &= x \times y \times 2^{-(m+2)}(\bmod\ p). \end{aligned} \tag{6}$$

The output of a MMM is a factor $2^{-(m+2)}$ times the expected result, where $m$ is the field size in bits. In order to obtain the exact result, the output must be multiplied by $2^{(m+2)}$ to remove the $2^{-(m+2)}$, which is the extra factor of the output. The size of the adders used for this must be $(m+2)$ bits to handle the intermediate result at each iteration; $(m+2)$ iterations are required to obtain an output in the range between 0 and $2m-1$ for multiplicands up to twice $m$. One modular correction is then required to ensure that the output is in the range between 0 and $m-1$ inclusive, taking only one extra clock cycle (CC) [23–25].

An efficient algorithm also proposed for modular multiplication is shown in Algorithm 1 (see Fig. 2), and based on interleaved modular multiplication [26]. Fig. 3 shows the proposed architecture for modular multiplication over prime field $\mathbb{F}_p$. In this method, the multiply-by-two operation is performed by a simple left-shift operation. The and-gate operation is: one bit $A[i]$ of the first operand $A$ is multiplied by the whole second operand $B$ bitwise, and then added to the intermediate result. The intermediate result $C3$ is then reduced with respect to the modulus $p$ by two subtractions operating in parallel until the values $C4$ and $C5$ are

**Algorithm 1**

**Input:** Prime $p$ and $A$, $B \in [1, p - 1]$
**Output:** $C = (A * B) \bmod p$
1: $C = 0$; $p2 = 2 * p$ (pre-computed) ;
2: **for** $i = m - 1$ downto 0 **do**
3: $C1 = C$; $C2 = 2 * C1$ (left-shift operation);
4: $I1 = A[i] * B$ (and-gate operation);
5: $C3 = C2 + I1s$; $C4 = C3 - p$; $C5 = C3 - p2$ ($p2 = 2p$);
6: **if** $C3 \geq p$ **then** $C6 = C4$;
7: **elsif** $C3 \geq p2$ **then** $C6 = C5$; **else** $C6 = C3$; **end if** $C = C6$;
8: **end for**
9: Return $C$

**Fig. 2** ***Algorithm 1:*** *Proposed algorithm for modular multiplication in GF(p)*

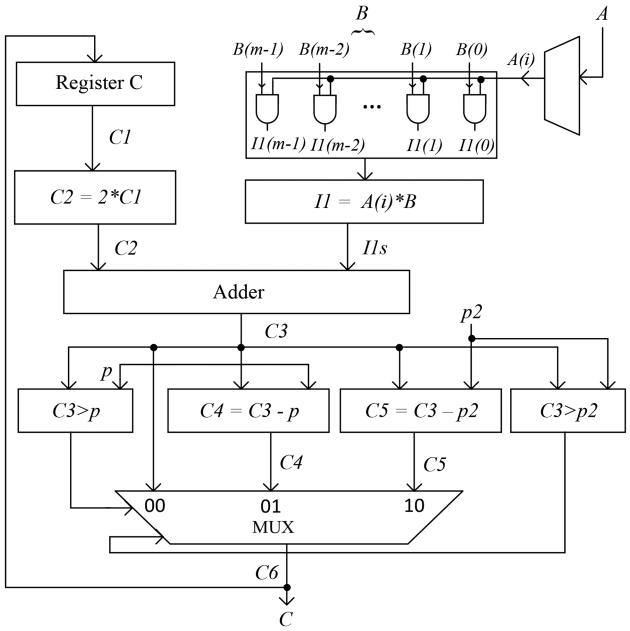


**Fig. 3** *Proposed hardware architecture for a modular multiplier/squarer*

**Algorithm 2**

**Input:** Prime $p$ and $a \in [1, p - 1]$
**Output:** $R = a^{-1} \bmod p$
1: $u = a$; $v = p$; $x = 1$; $y = 0$ ;
2: **while** $u \neq 1$ and $v \neq 1$ **do**
3: **while** $u$ is even **do** $u = u/2$;
4: **if** $x$ is even **then** $x = x/2$; **else** $x = (x + p)/2$; **end**
5: **end**
6: **while** $v$ is even **do** $v = v/2$;
7: **if** $y$ is even **then** $y = y/2$; **else** $y = (y + p)/2$; **end**
8: **end**
9: **if** $u \geq v$ **then**
10: $u = u - v$ ; **if** $x > y$ **then** $x = x - y$;
11: **else** $x = (x + p - y)$;
12: **else**
13: $v = v - u$ ; **if** $y > x$ **then** $y = y - x$;
14: **else** $y = (y + p - x)$; **end**
15: **end**
16: **end**
17: **if** $u = 1$ **then** $R = x \bmod p$; **elseif** $v = 1$ **then** $R = y \bmod p$;
18: **end**
19: Return $R$ (At this instant, $R = a^{-1} \bmod p$)

**Fig. 4** ***Algorithm 2:*** *Binary algorithm for inversion in GF(p) [20]*

smaller than the modulus. For doing this, two subtractions and two comparisons are required per iteration. These operations (for $C4$ and $C5$) are running in parallel where $p2$ is pre-computed. Then we need a two-bit multiplexer to select which result is correct. In this architecture, parallelisation in operations and pre-computations (to get $p2$) are used to calculate all intermediate results. The latency of this method is mostly and-gate, addition, and subtraction. This method requires $m + 1$ cycles to compute the final result of modular multiplication, where $m$ is the maximum bit length of the operands $A$, $B$ or $p$ and $m \simeq \lceil \log_2 p \rceil$. Therefore, we have designed a high-performance modular multiplier for an ECP in Jacobian coordinates. A modular squarer is similar to a modular multiplier except that only one input is required for a modular squarer rather than the two inputs for a modular multiplier; otherwise all other operations are the same as for modular multiplication.

### 3.2 Modular inversion over $\mathbb{F}_p$

Modular inversion over a prime field is the most expensive operation in ECC hardware, and it is mandatory for ECSM in affine coordinates. An efficient algorithm has been used for inversion based on the binary method, which is well known as the binary inversion algorithm shown in Algorithm 2 (see Fig. 4) [20].

The modular inversion over the prime field is accomplished using a series of additions, subtractions, and shift operations [27]. This algorithm works iteratively, and at every step either $u$ or $v$ decreases by at least one in bit length. The result of modular inversion is achieved after $2m$ iterations, where $m$ is the maximum bit length of $p$ and $a$. The inverse of an integer $a$ modulo $p$ is defined as an integer $R$ such that $a \cdot R \equiv 1 (\bmod\ p)$. This classical definition of the modular inversion operation can be presented as

$$R = a^{-1} (\bmod\ p) \tag{7}$$

where $a$ is an integer. From (7), the inverse of $a$ exists if and only if $a$ is relatively prime with $p$. Therefore, the greatest common divisor (GCD) of $a$ and $p$ must be 1, or $\gcd(a, p) = 1$. From Algorithm 2 (Fig. 4), four registers – $u$, $v$, $x$, and $y$ – are essential to implement a hardware architecture for inversion over a prime field. The calculation of divisions such as $u/2$, $v/2$, $x/2$, and $y/2$ depends upon parity and magnitude comparisons of the $m$-bit registers named $u$, $v$, $x$, and $y$. Two multiplexers are used to select $u$ or $v$ and several multiplexers are used to select $x$ or $y$ as appropriate. The least significant bit determines (1 indicates odd, 0 indicates even) the parity of any number. However, exact comparisons can be attained only through full $m$-bit subtraction, and this contributes a major delay before decisions regarding the next calculation can be made. We have used $m$-bit carry-propagation adders to execute the additions or subtractions. The implemented designs compute all possible values like $x$, $x/2$, $(x + p)/2$, $(x - y)/2$, $(x + p - y)/2$ or $y$, $y/2$, $(y + p)/2$, $(y - x)/2$, $(y + p - x)/2$ simultaneously to save time, and multiplexers are used for selecting the new values of $x$ and $y$. Using this algorithm, one modular inversion over prime field $\mathbb{F}_p$ takes an average $1.33m$ CCs for an $m$-bit modular inversion.

### 3.3 Proposed EC group operations

The EC group operations (PDBL and PADD) are the building blocks of FFMA. There are different techniques for a data-flow architecture such as balancing the architecture, which minimises the power consumption and reduces the longest path for better performance. Parallelisation in operations and pre-computations can be used for further improvement, and we have used all these techniques to increase the throughput rate of EC group operations for a high-performance ECP. As one can see in level 2 in Fig. 5*a*, one inversion, one squaring, and one addition module are operating in parallel. The $2P$ ($2Px$, $2Py$) value of PDBL is pre-computed, and used for the PADD module in Fig. 5*b* if $x_1 = y_1$ and $x_2 = y_2$. Using this parallel operation and the pre-computation technique, the data path is reduced for the PDBL and PADD operations. In Figs. 5*a* and *b*, the latencies of PDBL and PADD are $5m + 12$ and $5m + 10$, respectively, in affine coordinates. Hence, we have designed a high-performance ECSM in affine coordinates for an FPGA. In addition, we propose a novel PDPA technique for computing EC group operations together, as shown in Fig. 5*c*. As can be seen from Figs. 5*a* and *b*, 11 steps are needed to perform the PDBL operation and 9 steps are needed to perform the PADD operation whereas a combined PDPA module needs only 12 steps/levels. Parallelisation in operations is used to increase the latency and throughput of EC group operations. As we can see in level 1 in

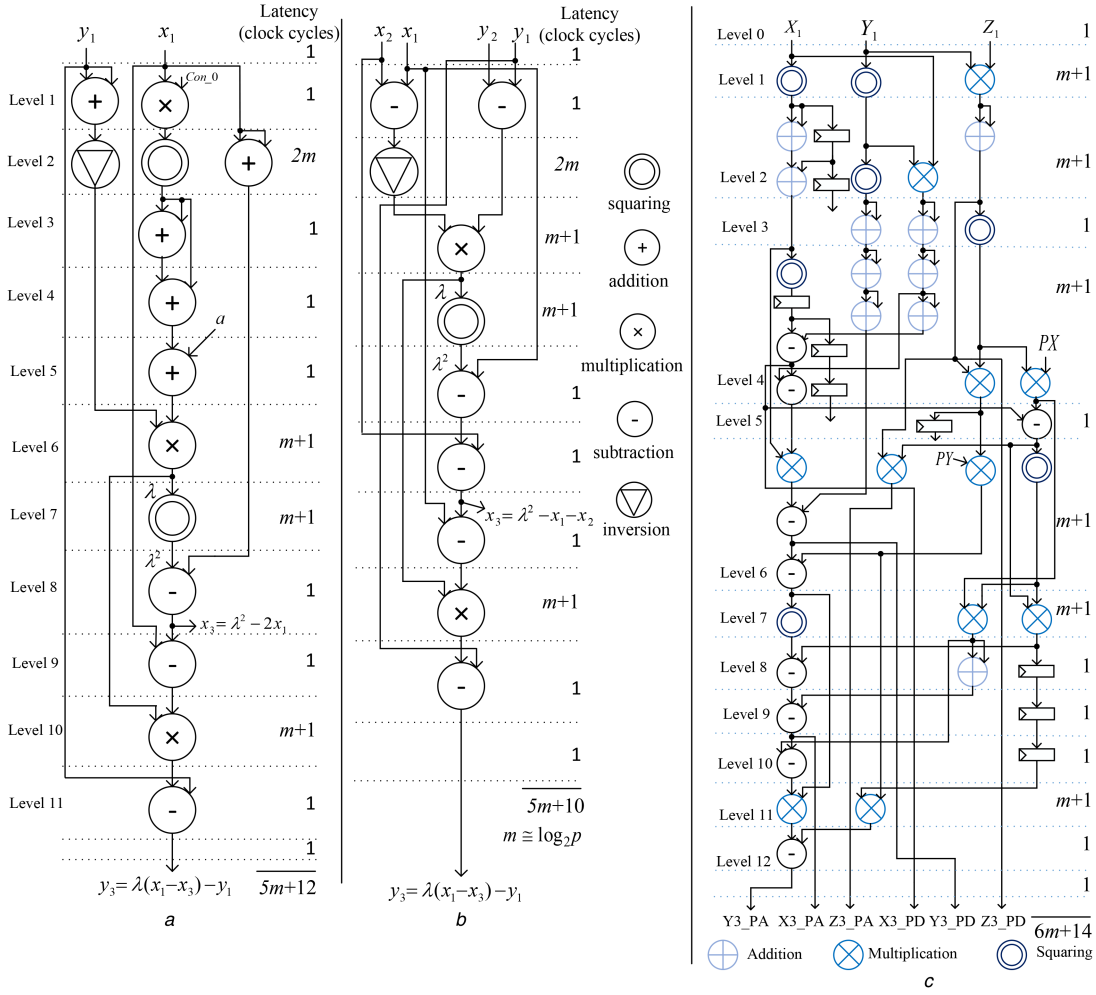


**Fig. 5** *Proposed hardware architecture for EC*
***(a)*** PDBL, ***(b)*** PADD, ***(c)*** PDPA

Fig. 5*c*, two squarings and one multiplication module are operating in parallel. Similarly in level 6, three multiplications and one squaring module are running in parallel. Therefore, the number of levels in the data path is reduced, and the overall latency of the PDPA module is reduced to $6m + 14$ in Jacobian coordinates. Using this efficient PDPA hardware, we have designed a high-performance ECP in Jacobian coordinates. Figs. 5*a–c* depict the proposed architecture of the PDBL, PADD, and PDPA operations, respectively. The costs of PDBL, PADD and PDPA over $\mathbb{F}_p$ are 3 MUL + 2 SQ + 1 INV + 5 ADD + 3 SUB, 2 MUL + 1 SQ + 1 INV + 6 SUB, and 11 MUL + 7 SQ + 10 ADD + 9 SUB, respectively, where MUL, SQ, INV, ADD, and SUB are the costs of modular multiplication, squaring, inversion, addition, and subtraction, respectively.

### *3.4 Proposed ECSM*

ECSM over $\mathbb{F}_p$ is the key operation of an ECP; it is computationally the most expensive. However, we have designed a high-performance ECSM using efficient group operations and FFMA units. The basic operation of ECSM is defined as $kP$, where $k$ is a positive integer and $P$ is a point on the elliptic curve E defined over a prime field $\mathbb{F}_p$. Various methods exist for implementing ECSM: the binary method, the non-adjacent form (NAF) method, and the Montgomery method. The easiest way to implement ECC is the binary method (left to right) [20], shown in Algorithm 3 (see Fig. 6). Using this algorithm, on average $m$ PDBL and $m/2$ PADD operations are required for an ECSM in affine coordinates. An ECSM architecture in affine coordinates over $\mathbb{F}_p$ using separate PDBL and PADD is presented in Fig. 7. In this ECSM architecture, the PDBL module computes $2Q$ $(Q2x, Q2y)$ and the PADD module computes $P + 2Q$ $(Q2px, Q2py)$. The comparator module is used for comparison between the output of the PDBL module and the input of the PADD module. When the two inputs of PADD are equal (e.g. $Px = Q2x$ and $Py = Q2y$) then the output of MUX2 is the same as $2P(2Px, 2Py)$ if key = 1. This result occurs when key $= n + 1$; in this case if the input of the PDBL module is $(n + 1)/2$ then the output of this module is $(n + 1)P = 1P(Px, Py)$ which is the same as the other input of the PADD module. The pre-computed value of $2P$ ($2Px$, $2Py$) from the PDBL module is used in MUX1 for this comparison. The output of MUX2 always depends upon the bit pattern of the input key. For a bit pattern of key = 0, the output of PDBL goes directly to the input of MUX2, this appears at the output of MUX2. When key =1 the output of PDBL goes to the input of PADD and the output of MUX2 is the same as the output of PADD. In this architecture, $5m + 12$ and $5m + 10$ CCs are required for computing PDBL and PADD, respectively, in affine coordinates. The total number of CCs for computing ECSM in affine coordinates is defined by (8). Total average CCs for ECP in affine coordinates

$$\begin{aligned} &= m \times (\text{PDBL CC}) + (m/2) \times (\text{PADD CC}) \\ &= (m(5m + 12) + (m/2)(5m + 10)) \\ &= (7.5m^2 + 17m) \end{aligned} \tag{8}$$

To have a high-performance ECP in Jacobian coordinates, we have proposed a novel ECSM architecture using our proposed PDPA module. Most ECC hardware implementations in the literature have used separate PDBL and PADD modules, and require more CCs than our design. Also, most of the ECC hardware has been implemented in FPGA form. We have found a few hardware implementations targeting an ASIC. The proposed hardware is implemented in both FPGA and ASIC. Fig. 8 shows our proposed hardware architecture of ECSM in Jacobian

**Algorithm 3**

**Input:** $k = (k_{m-1,\ldots,k_1,k_0})_2$, $P(x, y) \in E(\mathbb{F}_p)$
**Output:** $Q(x, y) = k.P(x, y)$, where $Q(x, y), P(x, y) \in E(\mathbb{F}_p)$
1: $Q = 0$ ;
2: **for** i = $m$ - 1 to 0 **do**
3: $Q = 2Q$;
4: **if** $k(i)$ = '1' **then**
5: $Q = P + Q$ ;
6: **end**
7: **end for**
8: Return $(Q(x, y))$

**Fig. 6** ***Algorithm 3:*** *Binary method (left to right) for point multiplication*

ECSM, Jacobian to affine, and PISO. The upper module is SIPO which is required to send data serially and receive data in parallel. The second module contains ECSM, which is the main operation of the ECP. The Jacobian to affine conversion module is the third level, and is needed to get the result in affine coordinates. The bottom level is PISO, which sends the final results serially. The top module of ECP needs only five pins, two for input and three for output. Total CCs for ECP in Jacobian coordinates

$$\begin{aligned} &= (m-1) \times \text{PDPA} + \text{Jacobian to affine} + \text{SIPO} + \text{PISO} \\ &= (m-1) \times (6m+14) + (6m+6) + (m+2) + (m+1) \\ &= (6m^2 + 16m - 5) \end{aligned} \quad (9)$$

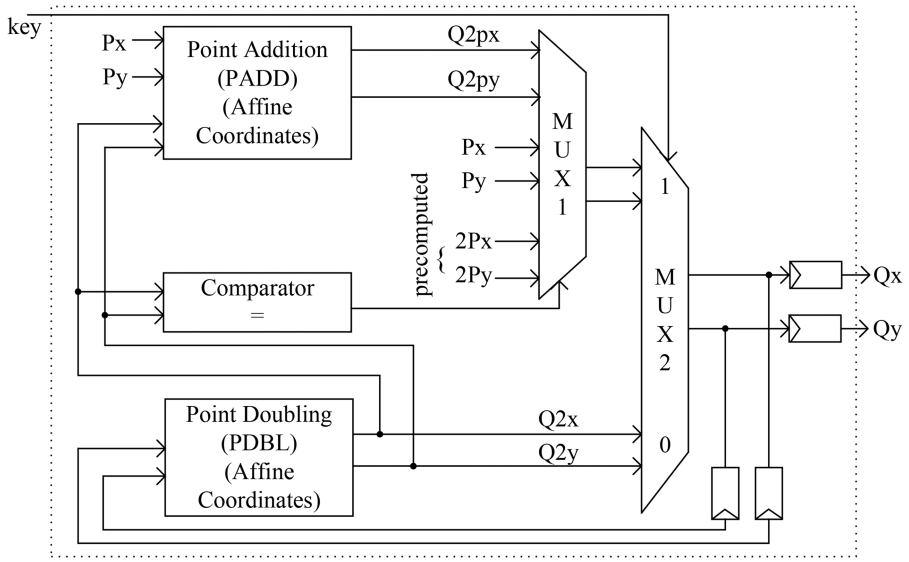


**Fig. 7** *Overall hardware architecture of proposed ECSM in affine coordinates for prime field*

coordinates over the prime field $\mathbb{F}_p$. In this architecture, $6m + 14$ CCs are needed for the PDPA module including two cycles are needed for the two registers to store all intermediate results. Equation (9) represents the total number of CCs for ECP in Jacobian coordinates. As we can see in Fig. 8, the PDPA module performs the EC group operations (PDBL and PADD) together, and these results are stored in the register module Register_PDPA. The select logic module generates a 'sel2s' signal for the MUX_1_NEW module. In the MUX_1_NEW module of Fig. 8, when 'sel2s = 00' then PADD results from the Register_PDPA module go to the output, which is the same as the input of the MUX_2_NEW module. Similarly, when 'sel2s = 01' then 1*P* (*Px*, *Py*, *Pz*) results, and when 'sel2s = 10' then 2*P* (2*Px*, 2*Py*, 2*Pz*) results, which are pre-computed, go to the input of MUX_2_NEW. Hence, the inputs of the MUX_2_NEW module are the PDBL and PADD results and the output of this module is either PDBL or PADD, depending upon the bit pattern of input key. When the bit pattern of key is high, then the output is the PADD result otherwise the PDBL result, which is stored in register Register_Count_PDPA. The counter module of this architecture acts as a control unit, and decides when the results of this register will be passed to the next input of the PDPA module. There $m - 1$ cycles are required to compute the final result of this ECSM module, where each cycle needs $6m + 14$ CCs (CCs for PDPA). The final results of this ECSM module are in Jacobian coordinates, and need conversion to verify the result in affine coordinates. Fig. 9*a* shows the hardware architecture for conversion from Jacobian to affine coordinates. As can be seen from this figure, $6m + 6$ cycles are required to get the result in affine coordinates. The complexity of this module is 4MUL + 1SQ + 1INV, where INV is the cost of modular inversion which is the most expensive operation in the prime field. However, to get the result in affine coordinates, only one modular inversion is needed. We have also designed serial-in parallel-out (SIPO) and parallel-in serial-out (PISO) modules to reduce the pin numbers of the top module. The top module of the final ECP is shown in Fig. 9*b* and are the building blocks of four modules, namely SIPO,

## 4 Implementation results and performance analysis

This section presents the implementation results for our proposed design. The design has been extensively simulated using both ModelSim PE and ISim, and synthesised using Xilinx ISE 14.7 synthesis technologies with an optimised goal of 'Speed'. All simulation results are verified using high-level Maple software. The proposed ECP hardware in Jacobian coordinates is also synthesised using Synopsys Design Compiler with United Microelectronics (UMC) standard logic-cell library (65 nm, 1.2 V, 25°C) for normal case analysis. In our experiment, the Synopsys Design Compiler gives better results using our new proposed hardware. We have compared the overall performance of the ECSM with those in the available literature. The detailed results are given in Table 5.

Modular addition and subtraction are implemented very efficiently; both operations take only one CC. The computation time of modular addition and subtraction is only 1.13 ns in Kintex-7 (XC7K325T-2FFG900) FPGA and 3.51 ns in Virtex-5 (XC5VLX330-2FF1760) FPGA for a prime field of either $\mathbb{F}_{224}$ or $\mathbb{F}_{256}$. Both designs were also synthesised using 65 nm CMOS technology with a target clock of 1 ns. Thus, the area of modular addition, subtraction, and combined addition and subtraction consumes only 7.96 *K*gates (0.01655 mm$^2$), 16.17 *K*gates (0.0336 mm$^2$), and 18.38 *K*gates (0.038 mm$^2$), respectively.

Table 1 presents modular multiplication and inversion results and performance over the prime field $\mathbb{F}_p$. A Kintex-7 FPGA has been used as the hardware platform for the ECP in affine coordinates because the Virtex-II FPGA is now obsolete. A new modular multiplication algorithm and architecture has been designed, and implemented in both Kintex-7 and Virtex-5 FPGA, and the results are shown in Table 2. We have implemented this new architecture on a Virtex-5 FPGA also, and its performance is slightly worse than with a Kintex-7 FPGA. As can be seen from

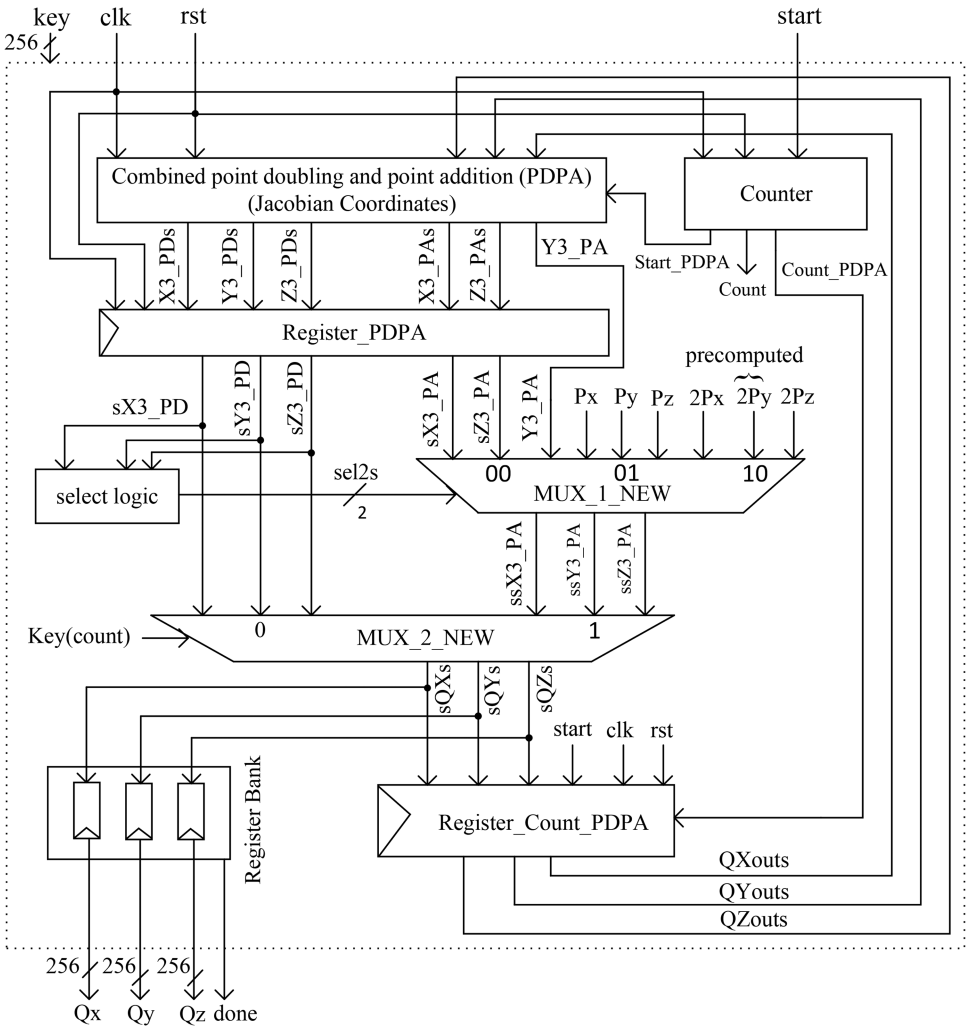


**Fig. 8** *Overall hardware architecture of proposed ECSM in Jacobian coordinates*

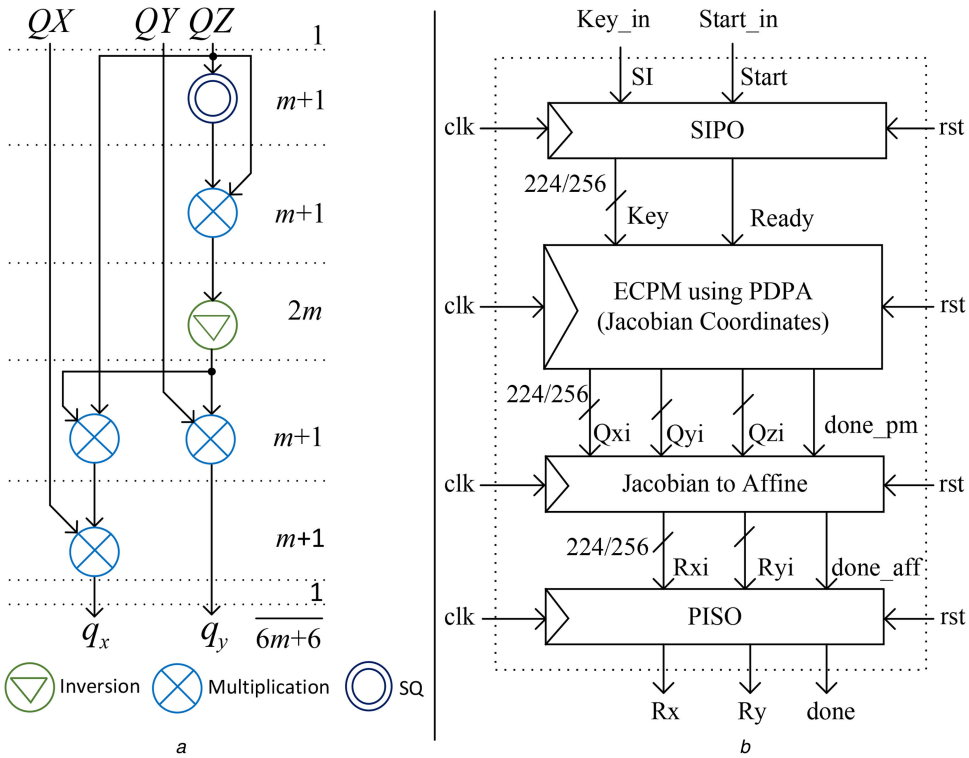


**Fig. 9** *Proposed hardware architecture of*
***(a)*** Jacobian to affine conversion and ***(b)*** Top module of ECP

Table 2, this new design on a Kintex-7 FPGA shows better performance than our Montgomery modular multiplication method. Using this new modular multiplication, we have designed a high-performance ECP in Jacobian coordinates. A modular multiplication over the prime field for an ECSM is implemented by Ghosh *et al.* [2]. They have also used the interleaved modular multiplication method. The hardware resources of their design are 4657 and 5379 slices, and the computation times are 6.00 and 7.30 µs, respectively, to achieve 224-bit and 256-bit modular multiplications. Their design takes $k$ CCs for $k$-bit modular multiplication, which is almost the same as our design, but we have achieved a more area-efficient design. A 256-bit modular multiplication using the Montgomery method on a Virtex-II FPGA over the field $\mathbb{F}_{256}$ is presented in [24, 28], and their implemented modular multiplications require 18.28 and 4.06 µs, respectively. However, our designed modular multiplier using the Montgomery method over $\mathbb{F}_{256}$ on a Xilinx Kintex-7 FPGA takes only 605 slices and 1.68 µs. The throughput rate of our design is also higher than the other available designs.

We have also designed a modular inversion for ECP in both affine and Jacobian coordinates. In [2], the authors propose a modular inversion for ECC over the NIST prime fields $\mathbb{F}_{192}$, $\mathbb{F}_{224}$, and $\mathbb{F}_{256}$. Their design takes $2m$ CCs for an $m$-bit modular inversion. A 256-bit modular inversion on a Virtex-II FPGA is proposed in [7, 12, 24], and their implemented modular inversion requires 15.22, 1160, and 6.4 µs, respectively. In [7, 12, 24], their designs consume14,844, 2085, and 5477 slices, respectively. Of them, the McIvor design consumes more area than all other available designs and Vliegen's design takes more time than all other designs. We have a trade-off between area and speed, and take only 1480 slices and 2.33 µs for a 256-bit modular inversion. Our design takes an average $1.33m$ CCs for an $m$-bit modular inversion. We have also achieved a higher clock frequency and less delay than all the other designs.

Most of the modular multiplication architecture was implemented on an FPGA. We have also synthesised our new modular multiplication [Algorithm 1 (Fig. 2) and Fig. 3] using UMC 65 nm CMOS technology; results are shown in Table 3. This indicates that the proposed design is very fast as well as area-efficient. It takes only 468 ns with an area of 0.0275 mm$^2$ (13.26 *K*gates) for 256-bit modular multiplication. The throughput rate of our design is 547 Mbps, which is very high for a prime field of either $\mathbb{F}_{224}$ or $\mathbb{F}_{256}$.

Hardware implementation results for PDBL and PADD are presented in Table 4. PBDL over the prime fields $\mathbb{F}_{224}$ and $\mathbb{F}_{256}$ is performed in Xilinx Kintex-7 in 6.20 and 7.65 µs, respectively. Similarly, PADD needs computation times of 6.09 and 7.57 µs with lower hardware resources. Based on our implementation results, we note that the computation times for both operations are much less due to the efficient implementation of FFMA. We have achieved a high throughput rate of almost 34 Mbps for both operations. In addition, a novel PDPA architecture has been designed (shown in Fig. 5*c*) with the latency of $6m + 14$, which is very small compared with other available designs in the literature.

Table 5 represents the ECSM results and performance comparisons with similar designs over the NIST prime field $\mathbb{F}_p$. Xilinx Kintex-7 and Virtex-5 FPGAs have been used for this work as the hardware implementation platforms because the Virtex-II

**Table 1** Performance analysis of FFMA for ECP in affine coordinates over $\mathbb{F}_p$ on FPGA

| FFMA operation | Platform (technology) | Field, bit length | Area (slices) | Frequency, MHz | Time, µs | AT | Throughput rate, Mbps |
|---|---|---|---|---|---|---|---|
| modular multiplication | Kintex-7 | $\mathbb{F}_p$ 224 | 506 | 163.76 | 1.37 | 0.68 | 163.50 |
| | | $\mathbb{F}_p$ 256 | 605 | 152.71 | 1.68 | 1[a] | 152.38 |
| modular inversion | Kintex-7 | $\mathbb{F}_p$ 224 | 1263 | 156.27 | 1.90 | 0.70 | 117.90 |
| | | $\mathbb{F}_p$ 256 | 1480 | 146.38 | 2.33 | 1[b] | 109.87 |

[a]The normalisation factor of $A \times T$ for modular multiplication is 1/0.001018 = 982.3183, where $A$ is area (slices) and $T$ is time (s).

[b]The normalisation factor of $A \times T$ for modular inversion is 1/0.00344692 = 290.114, where $A$ is area (slices) and $T$ is time (s).

**Table 2** FPGA implementation of modular multiplication for ECP in Jacobian coordinates

| Circuit | Platform (technology) | Field, bit length | Area (slices) | Frequency, MHz | Time, µs | AT[a] | Throughput rate, Mbps |
|---|---|---|---|---|---|---|---|
| Modular multiplication | Kintex-7 | $\mathbb{F}_p$ 224 | 365 | 130.49 | 1.71 | 0.63 | 130.99 |
| | | $\mathbb{F}_p$ 256 | 397 | 135.89 | 1.88 | 0.76 | 136.17 |
| | Virtex-5 | $\mathbb{F}_p$ 224 | 362 | 82.39 | 2.72 | 1 | 82.39 |
| | | $\mathbb{F}_p$ 256 | 420 | 78.22 | 3.27 | 1.40 | 78.29 |

[a]The normalisation factor for $A \times T$ is 1/0.001018 = 982.3183, where $A$ is area (slices) and $T$ is time (s).

**Table 3** ASIC implementation of modular multiplication for ECP in Jacobian coordinates over $\mathbb{F}_p$

| Circuit | Platform (technology) | Field, bit length | Area | Cycles | Frequency, MHz | Time, µs | Throughput rate, Mbps |
|---|---|---|---|---|---|---|---|
| modular multiplication | 65 nm CMOS[a] | $\mathbb{F}_p$ 224 | 0.0266 mm$^2$/12.79 *K*gates | 225 | 549.45 | 0.409 | 547.00 |
| | | $\mathbb{F}_p$ 256 | 0.0275 mm$^2$/13.26 *K*gates | 257 | 549.45 | 0.468 | 547.00 |

[a]Used UMC standard logic cell library (65 nm, 1.2 V, 25C) for normal case analysis.

**Table 4** EC group operation (PDBL and PADD) results in affine coordinates for GF($p$) on Kintex-7

| Group Opn | $\|p\|$, bits | Area (slices) | Latency (CCs) | Avg time, µs at $f$, MHz | Throughput rate, Mbps |
|---|---|---|---|---|---|
| PDBL | 224 | 4130 | $5m + 12$ | 6.20 at 156.56 | 36.13 |
| | 256 | 4597 | $5m + 12$ | 7.65 at 146.40 | 33.46 |
| PADD | 224 | 3459 | $5m + 10$ | 6.09 at 156.56 | 36.78 |
| | 256 | 3861 | $5m + 10$ | 7.57 at 146.40 | 33.82 |

$m \simeq \lceil \log_2 p \rceil$ and CC = number of clock cycles.

FPGA is now obsolete. In affine coordinates, we achieve a scalar multiplication in 3.05 ms at the frequency of 140.7 MHz and 4.70 ms at the frequency of 119.2 MHz in a Xilinx Kintex-7 FPGA over the prime fields $\mathbb{F}_{224}$ and $\mathbb{F}_{256}$, respectively. Our proposed ECSM provides around 20% better delay performance than previous designs, and is also area-efficient: it takes only 8.4*K* slices and 9.3*K* slices, respectively, without using any DSP slices. Most ECC architectures in the available literature have used separate PBDL and PADD modules. We have proposed a new ECSM hardware in Jacobian coordinates using PDPA (combined PDBL and PADD modules). There are few hardware implementations on ASIC, most being implemented in FPGA. Our proposed ECP was also synthesised using UMC 65 nm CMOS technology. To make a fair comparison with other ASIC implementations, we used post-synthesis results of our new design. Our designed ECP in Jacobian coordinates takes 0.56 and 0.73 ms over the prime fields $\mathbb{F}_{224}$ and $\mathbb{F}_{256}$, respectively, with a clock frequency of 546.5 MHz. We have also achieved an area-efficient design which requires less than 1 $mm^2$ area on 65 nm CMOS ASIC. The proposed new ECP in Jacobian coordinates takes less time than previous designs on FPGA. It takes 2.36 ms for the prime field $\mathbb{F}_{224}$ and 3.27 ms for the prime field $\mathbb{F}_{256}$ in a Kintex-7 FPGA. This new ECP architecture was also implemented in a Virtex-5 FPGA. All the available results for our new designs are presented in the first, second, and fourth rows of Table 5.

The ECP proposed by Loi and Ko [1] provides the fastest ECPs, highest frequency, and a better area and time (AT) product. Their design requires fewer slices than our design, however they need

**Table 5** Comparison between our ECC design and similar work over GF(*p*)

| Circuit | Platform | Field, bit length | Reported area (slices) | *K*cycles | Time, ms/ ECSM at *f*, MHz | Area×time (*AT*) | Relative *AT* | Throughput rate, kbps |
|---|---|---|---|---|---|---|---|---|
| this work [a] | 65 nm CMOS | $\mathbb{F}_p$ 224 | 0.81 $mm^2$/393*K*gates | 304.6 | 0.56 at 546.5 | — | — | 400 |
| (Jacobian) | | $\mathbb{F}_p$ 256 | 0.93 $mm^2$/447*K*gates | 397.3 | 0.73 at 546.5 | — | — | 350 |
| this work [b] | Kintex-7 | $\mathbb{F}_p$ 224 | 9.7*K* | 304.6 | 2.36 at 128.9 | 22.89 | 0.52 | 94.91 |
| (Jacobian) | | $\mathbb{F}_p$256 | 11.3*K* | 397.3 | 3.27 at 121.5 | 36.95 | 0.84 | 78.28 |
| this work [c] | Kintex-7 | $\mathbb{F}_p$ 224 | 8.4*K* | 429.7 | 3.05 at 140.7 | 25.62 | 0.59 | 73.44 |
| (Affine) | | $\mathbb{F}_p$ 256 | 9.3*K* | 560.7 | 4.70 at 119.2 | 43.71 | 1 | 54.41 |
| this work [d] | Virtex-5 | $\mathbb{F}_p$ 224 | 10.7*K* | 304.6 | 3.64 at 83.65 | 38.95 | 0.89 | 61.54 |
| (Jacobian) | | $\mathbb{F}_p$ 256 | 12.3*K* | 397.3 | 5.26 at 75.43 | 64.70 | 1.48 | 48.67 |
| Loi and Ko [1] | Virtex-4 | $\mathbb{F}_p$ 192 | 7.02*K* + 8 DSPs | 429.7 | 2.36 at 182.0 | 16.57 | 0.38 | 81.32 |
| | | $\mathbb{F}_p$ 224 | | 666.7 | 3.66 at 182.0 | 25.70 | 0.59 | 61.2 |
| | | $\mathbb{F}_p$ 256 | | 993.2 | 5.46 at 182.0 | 38.30 | 0.88 | 46.91 |
| | | $\mathbb{F}_p$ 384 | | 2968.4 | 16.31 at 182.0 | 114.50 | 2.62 | 23.54 |
| | | $\mathbb{F}_p$ 521 | | 7048.9 | 38.73 at 182.0 | 271.88 | 6.22 | 13.45 |
| Lee *et al.* [11] | 90 nm CMOS | $\mathbb{F}_p$ 224 | 1.12 $mm^2$/313*K*gates | 127.2 | 0.59 at 217.0 | — | — | 379.66 |
| | | $\mathbb{F}_p$ 256 | 1.12 $mm^2$/313*K*gates | 165.1 | 0.76 at 217.0 | — | — | 336.84 |
| Marzouqi *et al.* [6] | Virtex-5 | $\mathbb{F}_p$ 256 | 10.2*K* | 442.2 | 6.63 at 66.7 | 67.63 | 1.55 | 38.61 |
| Ghosh *et al.* [2] | Virtex-II Pro | $\mathbb{F}_p$ 192 | 9.0*K* | 192.2 | 4.47 at 43.0 | 40.11 | 0.92 | 42.95 |
| | | $\mathbb{F}_p$ 224 | 10.4*K* | 260.0 | 6.50 at 40.0 | 67.51 | 1.54 | 34.46 |
| | | $\mathbb{F}_p$ 256 | 12.0*K* | 338.0 | 9.38 at 36.0 | 112.12 | 2.57 | 27.29 |
| Vliegen *et al.* [12] | Virtex-II | $\mathbb{F}_p$ 256 | 2.1*K* + 7 DSPs | 1074.83 | 15.76 at 68.2 | 32.86 | 0.75 | 16.24 |
| Ananyi *et al.* [13] | Virtex-II Pro | $\mathbb{F}_p$ 192 | 20.8 *K* + 32 DSPs | 288.0 | 4.80 at 60.0 | 99.84 | 2.28 | 40.00 |
| | | $\mathbb{F}_p$ 224 | 20.8 *K* + 32 DSPs | 348.0 | 5.80 at 60.0 | 120.64 | 2.76 | 38.62 |
| | | $\mathbb{F}_p$ 256 | 20.8 *K* + 32 DSPs | 414.0 | 6.90 at 60.0 | 143.52 | 3.28 | 37.10 |
| | | $\mathbb{F}_p$ 384 | 20.8*K* + 32 DSPs | 1194.0 | 19.90 at 60.0 | 413.92 | 9.47 | 19.30 |
| | | $\mathbb{F}_p$ 521 | 20.8*K* + 32 DSPs | 2736.0 | 45.60 at 60.0 | 948.48 | 21.70 | 11.43 |
| Ghosh *et al.* [14] | Virtex-4 | $\mathbb{F}_p$ 192 | 14.9*K* | 227.9 | 4.30 at 53.0 | 63.89 | 1.46 | 44.65 |
| | | $\mathbb{F}_p$ 224 | 17.3*K* | 305.5 | 6.50 at 47.0 | 112.65 | 2.58 | 34.46 |
| | | $\mathbb{F}_p$ 256 | 20.1*K* | 395.6 | 9.20 at 43.0 | 185.13 | 4.24 | 27.83 |
| Ghosh *et al.* [14] | 130 nm CMOS | $\mathbb{F}_p$ 192 | —/123.1*K*gates | 187.68 | 1.36 at 138 | — | — | 141.17 |
| | | $\mathbb{F}_p$ 224 | —/143.9*K*gates | 253.5 | 1.95 at 130 | — | — | 114.87 |
| | | $\mathbb{F}_p$ 256 | —/167.5*K*gates | 331.1 | 3.01 at 110.0 | — | — | 85.05 |
| Lai and Huang [15] | 130 nm CMOS | $\mathbb{F}_p$ 160 | —/169.4*K*gates | 74.02 | 0.36 at 208 | — | — | 444.44 |
| | | $\mathbb{F}_p$ 256 | —/197.0*K*gates | 252.1 | 1.21 at 208.0 | — | — | 211.57 |
| Ahmadi and Afzali-Kusha [16] | 130 nm CMOS | $\mathbb{F}_p$ 192 | — | 9712.5 | 525 at 18.5 | — | — | 0.37 |
| Fan *et al.* [17] | Virtex-II Pro | $\mathbb{F}_p$ 192 | 3.2 *K* + 16 DSPs | 920.7 | 9.90 at 93.0 | 31.41 | 0.72 | 19.39 |
| Mentens *et al.* [18] | Spartan-3 | $\mathbb{F}_p$ 256 | 4.8 *K* + 66 DSPs | 1768.8 | 26.80 at 66.0 | 129.33 | 2.96 | 9.55 |
| McIvor *et al.* [7] | Virtex-II Pro | $\mathbb{F}_p$ 256 | 15.8 *K* + 256 DSPs | 152.3 | 3.86 at 39.5 | 60.81 | 1.39 | 66.32 |
| Sakiyama *et al.* [19] | Spartan-3 | $\mathbb{F}_p$ 256 | 27.6 *K* | 708.0 | 17.7 at 40.0 | 488.47 | 11.18 | 14.46 |

The normalisation factor for relative $A \times T$ is 43.71, where $A$ is area (slices) and $T$ is time (s).

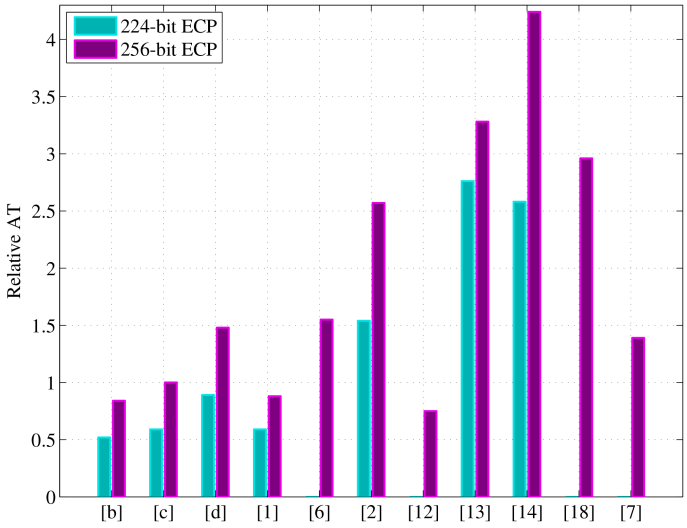


**Fig. 10** *Comparison of relative AT values between our ECC design ([b]–[d]) and similar work*

eight DSP slices and more CCs for implementation, whereas we have no need for any DSP slices for our proposed design. A new architecture of ECSM using PDBL and PADD modules was implemented in 90 nm CMOS technology by Lee *et al.* [11]. Their proposed design requires more than 1 $mm^2$ of area, whereas our proposed design requires less than 1 $mm^2$ of area. Besides, our design is faster and has a higher throughput rate than their proposed ECSM. A 256-bit ECP presented in [6] takes a similar number of CCs, but needs more area and time to compute a scalar multiplication than our design, and the AT value of their design is almost twice our AT. An ECSM is presented by Ghosh *et al.* [2], and their ECP requires 6.50 and 9.38 ms over the fields $\mathbb{F}_{224}$ and $\mathbb{F}_{256}$, respectively, to achieve a scalar multiplication. However, our ECP requires only one-third the computation time of their design. Our two implementations show a similar performance to their design in terms of area and CCs. ECPs over the field $\mathbb{F}_{256}$ are presented by Vliegen *et al.* [12], Mentens *et al.* [18], and Sakiyama *et al.* [19], and their proposed crypto-processors require 15.76, 26.80, 3.86, and 17.70 ms, respectively, for a typical scalar/point multiplication. We can see in Table 5 that their design has more delay than our proposed design. Besides, the throughput rate of our design is better than the others. The ECP proposed by Ananyi *et al.* [13] provides results for all five NIST-recommended prime curves, and their design takes between 4.80 ms (192-bit ECC) and 45.60 ms (521-bit ECC) to compute a scalar multiplication. Their proposed ECP requires 20.8*K* slices and 32 DSP slices on a Virtex-II Pro FPGA, which is more than our designs. A parallel ECP for $\mathbb{F}_p$ is presented by Ghosh *et al.* [14]; their design takes almost double the area and time of our design. In [14, 15], they implemented their design on TSMC 130 nm CMOS technology using separate PDBL and PADD modules, however we have synthesised our new ECP using a PDPA module on 65 nm CMOS. Their design takes more computation time than ours. Ahmadi and Afzali-Kusha [16] and Fan *et al.* [17] also implemented ECPs over the prime field $\mathbb{F}_{192}$ on different platforms, and their cryptographic processors are slower than our processor. Although the McIvor *et al.* [7] design provides a better result in terms of time, area–time (AT) product, and throughput rate, it requires an additional 256 DSP slices for hardware implementation. On the other hand, we have implemented the ECP with almost the same AT product without any DSP slices on the FPGA.

It can be noted that the ECPs are implemented on different platforms and employ different hardware resources, so it is difficult to state which design is the best. However, the best indicator for efficient design is the product of area and time (AT), and throughput rate. For this reason, we have calculated the AT and throughput rate for all the available designs, and make a comparison of these AT and throughput rates with our design. All these calculated AT values, relative AT values, and throughput rates have been presented in columns seven to nine of Table 5. The relative AT product comparison of similar work is shown in Fig. 10. In Fig. 10, [b]–[d] represent our implementation results (both 224-bit and 256-bit ECCs) and [6, 7, 12, 18] illustrate reference implementations for 256-bit ECC only. This figure also demonstrates that our proposed designs require lower AT than most of the similar designs in the available literature. Note that, of all the available designs, in terms of AT value the Loi and Ko [1] and Vliegen *et al.* [12] designs perform the best. However, we have achieved a high throughput rate compared with the designs in [1, 12]. Besides, their designs require additional DSP slices for hardware implementation. The ECP designs in [2, 7, 13, 14] need fewer CCs for a scalar multiplication, but would require more slices than our design. We require less hardware resources for our design compared with other previous designs. Beside, our new ECP in Jacobian coordinates is not only area-efficient, but also faster than all other designs, even better than our previous design. From the performance analysis and comparison of different ECPs over the prime field in Table 5, some are only area-efficient or some are better in terms of only speed; it can be concluded that our ECP performs better than other comparable designs.

## 5 Conclusion

A fast, high-performance ECP over prime field GF($p$) is developed using efficient FFMA, EC group operations, and ECSM. Our design supports two NIST prime fields of the five NIST-recommended primes $p$, with sizes 224 and 256 bits. A novel PDPA technique is proposed to perform EC group operations in parallel, aimed at reducing the number of steps in the PDBL and PADD operations and decreasing the overall latency of the ECP. Thus, we have designed a faster ECP in Jacobian coordinates which takes 0.56 ms for $\mathbb{F}_{224}$ and 0.73 ms for $\mathbb{F}_{256}$ in ASIC 65 nm CMOS. We also present the ECSM results of our new proposed design in Xilinx Kintex-7 and Virtex-5 FPGAs. The proposed ECP in Jacobian coordinates takes between 2.36 and 3.27 ms on a Xilinx Kintex-7 FPGA and between 3.64 and 5.26 ms on a Xilinx Virtex-5 FPGA. In addition, our proposed ECP in affine coordinates on a Xilinx Kintex-7 FPGA takes between 3.05 and 4.70 ms to execute a typical scalar/point multiplication, which represents the fastest hardware implementation result in an affine coordinate system. The hardware architecture delivers a high-performance operation with fewer hardware resources. The implemented design is optimised by using different techniques such as balancing the PDBL and PADD architectures, parallelisation in operations, and pre-computations, for obtaining higher performance. Based on the overall performance analysis and comparisons of different ECPs over the prime field $\mathbb{F}_p$, it can be concluded that this design provides better performance than others in terms of the area, delay, AT, and throughput rate.

## 6 References


[1] Loi, K.C.C., Ko, S.B.: 'Scalable elliptic curve cryptosystem FPGA processor for NIST prime curves', *IEEE Trans. VLSI Syst.*, 2015, **23**, (11), pp. 2753–2756

[2] Ghosh, S., Mukhopadhyay, D., Roychowdhury, D.: 'Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable GF(p) arithmetic unit', *IEEE Trans. Circuits Syst. I*, 2011, **58**, (8), pp. 1798–1812

[3] Koblitz, N.: 'Elliptic curve cryptosystems', *Math. Comput.*, 1987, **48**, pp. 203–209

[4] Miller, V.S.: 'Use of elliptic curves in cryptography'. Proc. CRYPTO 1985, 1986, pp. 417–426

[5] Rivest, R.L., Shamir, A., Adleman, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, 1978, **21**, (2), pp. 120–126

[6] Marzouqi, H., Al-Qutayri, M., Salah, K.: 'An FPGA implementation of NIST 256 prime field ECC processor'. Proc. IEEE ICECS, December 2013, pp. 493–496

[7] McIvor, C., McLoone, M., McCanny, J.: 'Hardware elliptic curve cryptographic processor over GF(p)', *IEEE Trans. Circuits Syst. I*, 2006, **53**, (9), pp. 1946–1957

[8] '*National Institute of Standards and Technology, Digital Signature Standard, FIPS Publication 186-2*' (NIST, Gaithersburg, MD, USA, 2000)

[9] 'SEC 2: Recommended elliptic curve domain parameters, standards for efficient cryptography, Certicom Research', 2000

[10] 'IEEE standard specifications for public-key cryptography', IEEE Std 1363-2000, August 2000, pp. 1–228

[11] Lee, J.-W., Chung, S.-C., Chang, H.-C*., et al.*: 'Efficient power-analysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-processing-element architecture', *IEEE Trans. VLSI Syst.*, 2014, **22**, (1), pp. 49–61

[12] Vliegen, J., Mentens, N., Genoe, J*., et al.*: 'A compact FPGA-based architecture for elliptic curve cryptography over prime fields'. Proc. IEEE Int. Conf. ASAP, July 2010, pp. 313–316

[13] Ananyi, K., Alrimeih, H., Rakhmatov, D.: 'Flexible hardware processor for elliptic curve cryptography over NIST prime fields', *IEEE Trans. VLSI Syst.*, 2009, **17**, (8), pp. 1099–1112

[14] Ghosh, S., Alam, M., Chowdhury, D.R., *et al.*: 'Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks', *Comput. Electr. Eng.*, 2009, **35**, (2), pp. 329–338

[15] Lai, J.Y., Huang, C.T.: 'A highly efficient cipher processor for dual-field elliptic curve cryptography', *IEEE Trans. Circuits Syst. II*, 2009, **56**, (5), pp. 394–398

[16] Ahmadi, H., Afzali-Kusha, A.: 'Low-power low-energy prime-field ECC processor based on montgomery modular inverse algorithm'. Proc. Euromicro Conf. DSD, August 2009, pp. 817–822

[17] Fan, J., Sakiyama, K., Verbauwhede, I.: 'Elliptic curve cryptography on embedded multicore systems', *Des. Autom. Embed. Syst.*, 2008, **12**, (3), pp. 231–242

[18] Mentens, N., Sakiyama, K., Batina, L*., et al.*: 'A side-channel attack resistant programmable PKC coprocessor for embedded applications'. Proc. Int. Conf. SAMOS, July 2007, pp. 194–200

[19] Sakiyama, K., Mentens, N., Batina, L*., et al.*: '*Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems*' (Springer Berlin Heidelberg, 2006), pp. 347–357

[20] Hankerson, D., Menezes, A.J., Vanstone, S.: '*Guide to elliptic curve cryptography*' (Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003)

[21] Longa, P., Miri, A.: 'Fast and flexible elliptic curve point arithmetic over prime fields', *IEEE Trans. Comput.*, 2008, **57**, (3), pp. 289–302

[22] Montgomery, P.L.: 'Modular multiplication without trial division', *Math. Comput.*, 1985, **44**, (170), pp. 519–521

[23] Byrne, A., Meloni, N., Crowe, F*., et al.*: 'SPA resistant elliptic curve cryptosystem using addition chains'. Proc. Int. Conf. ITNG, April 2007, pp. 995–1000

[24] Daly, A., Marnane, W., Kerins, T*., et al.*: 'An FPGA implementation of a GF(p) ALU for encryption processors', *Microproces. Microsyst.*, 2004, **28**, (56), pp. 253–260, Special Issue on FPGAs: Applications and Designs

[25] Hossain, M.S., Kong, Y.: 'FPGA-based efficient modular multiplication for elliptic curve cryptography'. Proc. ITNAC, November 2015, pp. 191–195

[26] Bunimov, V., Schimmler, M.: 'Area and time efficient modular multiplication of large integers'. Proc. IEEE Int. Conf. ASAP, June 2003, pp. 400–409

[27] Hossain, M.S., Kong, Y.: 'High-performance FPGA implementation of modular inversion over $F_{256}$ for elliptic curve cryptography'. Proc. IEEE Int. Conf. DSDIS, December 2015, pp. 169–174

[28] Cheng-hua, D., Yi, L., Yong-tao, C.: 'A 3-stage pipelined large integer modular arithmetic unit for ECC'. Proc. Int. Symp. IEEC, May 2009, pp. 519–523