FPGA Implementation of a Fast Scalar Point Multiplier for an Elliptic Curve Crypto-Processor

Satvik Maurya and Vaishali Ingale

Abstract This paper presents a fast scalar point multiplier for an elliptic curve crypto-processor in the field GF (2^{163}). Elliptic curve-based cryptographic algorithms have been in wide use since the early 2000s after being introduced in 1986. With the ever-increasing need for information security, it is essential for systems to perform the required operations in a fast and efficient manner. In this work, a hybrid type Karatsuba multiplier has been used for fast field multiplications and a dedicated inverter module based on the extended Euclidean algorithm is used for fast field inversions. The point multiplication is performed using the standard double-and-add algorithm for which the point doubling and point addition are done using standard projective coordinates. The use of the fast multipliers and field inverters makes the implementation a fast one as compared to other high-performance implementations that have been reported over the years, at the cost of increased resource usage. The results obtained, however, justify this increased resource usage as the point multiplication is the most time-intensive operation in the encryption and decryption process of elliptic curve cryptography.

Keywords FPGA · Karatsuba multiplier · Extended Euclidean algorithm · Elliptic curve cryptography (ECC) · Scalar point multiplication

1 Introduction

Cryptography, in its most classical form, has been around for thousands of years. From Julius Caesar, who used a simple substitution cipher to relay messages to his generals, to the Second World War, in which the breaking of the German cipher

S. Maurya (✉)
Delhi Technological University, Delhi, India
e-mail: satvik_bt2k14@dtu.ac.in

V. Ingale
College of Engineering, Pune, India
e-mail: vvi.extc@coep.ac.in

M. L. Kolhe et al. (eds.), *Advances in Data and Information Sciences*, Lecture Notes in Networks and Systems 38, https://doi.org/10.1007/978-981-10-8360-0_14

hastened the end of the war, cryptography and subsequently cryptanalysis have been a part of society for a very long time. Till the 1970s, cryptography was an obscure field, used primarily by the military and spy agencies. With the advent of modern computers, and along with it the invention of Data Encryption Standard (DES) and most importantly the RSA public-key cryptography algorithm, the field of cryptography was brought to the public domain. With the increasing need to protect data from unauthorized usage along with the ever-increasing computational power available, new algorithms are constantly being developed. Elliptical curve cryptography is an algorithm which has gained popularity in recent times, even though it has been around since the 1980s.

ECC is a public-key cryptographic algorithm first developed in 1985 by Neal Koblitz and Victor S. Miller, and it has been standardized by [1–3]. Like the RSA, ECC functions in a finite field, which can be a prime field represented by GF(p) or a binary field GF(2^m), where p and m are prime numbers. The chief reason for the popularity of ECC is the smaller key size required to encrypt data as compared to the RSA algorithm. For example, the security offered by a 1024-bit key in RSA can be offered by a 160-bit key in ECC. Both RSA and ECC are used in applications where a secure communication is to take place between two entities, and thus, for applications such as IoT, Near-Field Communication (NFC), Bitcoin, their use is extensive. The smaller key requirement of ECC thus enables it to have lesser hardware, making it suitable for mobile applications.

The functionality of ECC is based on the manipulation of points present on the elliptic curve, the most important being the scalar point multiplication $Q = kP$ which is the addition of P to itself $k - 1$ times. The security of ECC is thus based on this manipulation, that given two points P and Q on the elliptic curve; it is difficult to calculate the value of k. This problem is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP).

The relative ease with which software applications can be compromised has resulted in the general popularity of hardware approaches towards cryptography. Dedicated crypto-processors not only perform the encryption faster, they also provide security at a physical level from potential attackers as they cannot be compromised without direct physical access to them. Hardware approaches towards ECC can be broadly classified into two main categories based on the finite field used. Approaches which use the prime field GF(p) are few as compared to the approaches which use binary fields GF(2^m) [4]. This skew is due to the difficulties involved with large prime numbers on hardware as well as the ease with which binary fields can be represented on hardware. As the scalar point multiplication is the most hardware and time-intensive operation involved in any elliptic curve application such as the elliptic curve Diffie–Hellman (ECDH) key protocol or the Elliptic Curve Digital Signature Algorithm (ECDSA), it is this operation that has been the prime area of focus for crypto-processor systems. A high-speed crypto-processor implementation using the Montgomery Point Multiplication Algorithm was presented in [5, 6] which used pipelining as well as multiple multipliers and squarers for reducing the latency of operations. Dual field approaches which implement both prime and binary fields are also becoming popular, such as the architecture proposed in [7]. In general,

most papers on the hardware implementation of ECC alternate between the various point multiplication algorithms [8–11] and the inversion algorithm used, with the Itoh-Tsujii being the most efficient algorithm in this case [4].

This paper presents an implementation of an elliptic curve crypto-processor in the finite field of GF (2^{163}). The point multiplication has been accomplished using standard projective coordinates, and the extended Euclidean algorithm has been used for the inversion. To perform quick field multiplications, a hybrid Karatsuba multiplier has been used. The structure of the paper is as follows: Sect. 2 presents the basics of ECC as well as the architecture of the modules used in this work, Sect. 3 presents the implementation and results, and conclusions are put forth in Sect. 4.

2 Methodology

2.1 Elliptic Curve Cryptography Preliminaries

The elliptic curve E for a binary field GF(2^m) is given below:

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

where $a \in [0, 1]$ and $b \neq 0$. Any operation on a point that satisfies the curve E follows a specific set of group laws which govern how the operation can be performed. These group laws differ according to the field in which the elliptic curve is defined.

Elliptic curves have a wide range of modular arithmetic operations associated with them. These include the field multiplication, field squaring, field reduction, field addition and the field inversion operations. As mentioned earlier, the scalar point multiplication is the most important operation in ECC. The scalar point multiplication involves repeated point doublings and point additions which in turn require the field arithmetic operations mentioned above. Depending on the algorithm used for the point multiplication, field inversion can be required for every point doubling and addition or just once for every point multiplication. This variability comes from the use of coordinate systems such as standard projective coordinates and the Lopez-Dahab projective coordinates that differ from the one in which the curve E is defined in. The field arithmetic as well as the group laws governing elliptic curves are given in detail in [12].

2.2 Proposed Design

The architecture for the proposed arithmetic unit for implementing a point multiplication for ECC in GF (2^m) is shown in Fig. 1. A hybrid Karatsuba multiplier has been used for the field multiplication along with a field inverter based on the extended

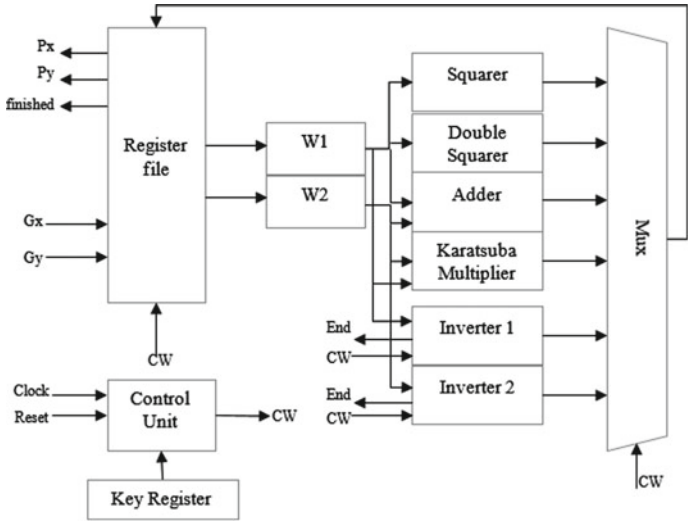

Fig. 1 Architecture for the arithmetic unit

Euclidean algorithm. The dedicated hardware made for field inversions guarantee a fast inversion process at the cost of increased area usage.

The register file contains the registers used for storing the values of the intermediate as well as final coordinates of the point multiplication process. The registers W1 and W2 are the two working registers that are loaded with the values that need to be operated on. The working registers form the input to the arithmetic blocks, namely the squarer, double squarer, adder, multiplier and field inverters. Working register W1 is used to provide the singular inputs to the adder, squarer, double squarer and Inverter1 as well as the dual inputs to the multiplier along with working register W2. W2 also provides the input to the second inverter module, Inverter2. It should be noted that in a field of GF(2^m), all registers and wires are m-bits wide.

The algorithm that has been used for the scalar point multiplication is called the left-to-right double-and-add algorithm. Starting from the MSB of the key, every bit is checked whether it is a 0 or a 1. Depending on the value of the current bit, a point doubling and a point addition are performed on the points. The algorithm is given in Fig. 2.

Thus, if the present bit of the key is 1, a point addition is performed after the point doubling. The total length of the key in this case will decide the worst-case execution time of the entire point multiplication.

The point addition and doubling are part of the group law which governs the elliptic curve being used. When standard affine coordinates (X, Y) are being used, the point doubling and point additions require a field inversion every time they are executed. Because the field inversion is the most time-intensive operation of the point multiplication process, a better alternative is to use a different coordinate system which requires a field inversion only once during the entire point

Left-to-right double and add algorithm for scalar Point Multiplication

Input: Generator point $G \epsilon E(GF(2^m))$, key $k = (k_{l-1}, \ldots \ldots k_0)$.

Output: Product $P = kG, P \epsilon E(GF(2^m))$

1. $P \leftarrow \infty$
2. *For* i *from* $l-1$ *to* 0
   2.1. $P \leftarrow 2P$
   2.2. *If* $k_i = 1, P \leftarrow P + G$
3. *Return*(P)

Fig. 2 Algorithm for scalar point multiplication

Fig. 3 Point arithmetic using standard projective coordinates

Point Doubling and Point addition using Standard Projective Coordinates

$E: y^2 + xy = x^3 + ax + b, a = 1, b = 1$

Point Doubling: $\mathbf{2}(X_1, Y_1, Z_1) = (X_3, Y_3, Z_3)$

1. $Z_3 = X_1^2 . Z_1^2$
2. $X_3 = X_1^4 + b . Z_1^4$
3. $Y_3 = b . Z_1^4 . Z_3 + X_3 (a . Z_3 + Y_1^2 + b . Z_1^4$

Point Addition: $(X_1, Y_1, Z_1) + (G_x, G_y, \mathbf{1}) = (X_3, Y_3, Z_3)$

1. $A = Y_2 . Z_1^2 + Y_1$
2. $B = X_2 . Z_1 + X_1$
3. $C = Z_1 . B$
4. $D = B^2 . (C + a . Z_1^2)$
5. $Z_3 = C^2$
6. $E = A . C$
7. $X_3 = A^2 + D + E$
8. $F = X_3 + X_2 Z_3$
9. $G = X_3 + Y_2 Z_3$
10. $Y_3 = E . F + Z_3 . G$

multiplication process. One such system is the standard projective coordinate system which introduces an additional coordinate Z which reduces the point doubling and addition operations to field multiplications, squaring and addition. The projective coordinates can be converted back into affine coordinates by using a suitable transformation. A method of point doubling and point addition using standard projective coordinates is given in [13]. During point addition, the point P is added to the starting point G. Thus, the generator points (G_x, G_y) in affine coordinates can be converted into the projective coordinate $(G_x, G_y, 1)$. This can be verified using the projective to affine conversion $(X, Y, Z) \rightarrow (X/Z, Y/Z^2)$. Thus, the final coordinates in P can be converted into affine coordinates to obtain the points (P_x, P_y). The algorithm for point doubling and point addition is given in Fig. 3.

The field squaring, double squaring, multiplication and addition are performed using entirely combinational circuits which thus enable these operations to be performed in a single clock cycle. The addition, squaring and double squaring can be done using simple XOR operations. The various modules including the control logic are explained below:

I. *The Karatsuba Multiplier*: The Karatsuba multiplier is one of the many alternatives available for designing a finite field multiplier. The efficiency of the Karatsuba multiplier, especially for large number multiplication, has made it one of the most popular choices for the finite field multiplier module in ECC cryptosystems. The Karatsuba multiplier used in this paper differs from the traditional recursive Karatsuba multiplier that requires operands with a bit length of an integral power of 2. To accommodate the fields and word lengths required in ECC, a hybrid Karatsuba multiplier is used as presented in [14]. The 163-bit multiplication is split into two multiplications with lengths 81 and 82 bits, respectively. These are further split into 40- and 41-bit multiplications, and the remainder of the tree can be grown likewise. The last layer of multipliers, namely the 20-bit and 21-bit multipliers, can be implemented using simple AND and XOR logic. The properties of polynomial arithmetic in GF(2^m) do not allow the generation of any carries during any arithmetic operation, and thus, combining the multipliers together to obtain the final product requires simple XOR operations. It should be noted that a field multiplication of $2m$-bit numbers will result in a product with $(2m - 1)$ bits. The modular reduction of this product according to the irreducible polynomial of that field is done using a reduction module that is present with the multiplier. This reduction unit has not been shown in the diagram, but it converts the $(2m - 1)$ bit product to a m-bit product. The multiplier tree for the field GF (2^{163}) is shown in Fig. 5.

II. *Finite Field Inversion*: The finite field inversion has been performed using the extended Euclidean algorithm for binary fields, as shown in Fig. 4. The extended Euclidean algorithm works on the principle of continuously dividing the polynomial until the remainder is 1. While the area requirement of such an inversion process is larger than other approaches such as the Itoh-Tsujii [15], it has the advantage of being faster. The inverter block shown in Fig. 1 also has the control word input from the control unit and an "End" output which signals the end of the inversion process. The control unit reads this output.

III. *Control Unit*: The control unit controls which register values are transferred to the working registers as well as the selection of outputs of the various arithmetic modules. The state diagram for the control unit is shown in Fig. 6. The control unit issues a control word (CW) which is specific to the present state of the control unit. The states are broadly classified in four different types: the initial state, which initializes all registers and is active until the active low asynchronous reset is enabled; the point doubling and point addition states which are switched according to the present bit of the input key; the projective to affine conversion state, which is entered when the point multiplication is complete. The projective to affine state issues the control word for starting the inversion process in the inverters and reads the "End" output of both inverters. The inverter outputs are transferred to the register file only after both inverters have completed their operations. Once the point multiplication has been completed, the control unit asserts the "finished" signal which indicates the end of the point multiplication process. The final outputs are available from the register file from the P_x, P_y registers.

Finite field inversion using the extended Euclidean algorithm

Input: A polynomial a, irreducible polynomial p of the field $GF(2^m)$

Output: $b = a^{-1} mod(p)$

1. $u \leftarrow a, v \leftarrow p$
2. $g_1 \leftarrow 1, g_2 \leftarrow 0$
3. While $(u \neq 1 \; and \; v \neq 1)$ do
   - 3.1. if $(u[0] = 0)$ then,
     - 3.1.1.1. $u \leftarrow (u \gg 1)$
     - 3.1.1.2. If $(g_1[0] = 0)$, then $g_1 \leftarrow (g_1 \gg 1)$; else $g_1 \leftarrow (g_1 + p) \gg 1$
   - 3.2. If $(v[0] = 0)$ then,
     - 3.2.1.1. $v \leftarrow (v \gg 1)$
     - 3.2.1.2. If $(g_2[0] = 0)$, then $g_2 \leftarrow (g_2 \gg 1)$; else $g_2 \leftarrow (g_2 + p) \gg 1$
   - 3.3. If $u > v$, then $u \leftarrow u + v, g_1 \leftarrow g_1 + g_2$; else $v \leftarrow v + u, g_2 \leftarrow g_2 + g_1$
4. If $u = 1$, $return(g_1)$; else $return(g_2)$

Fig. 4 Extended Euclidean algorithm for field inversion

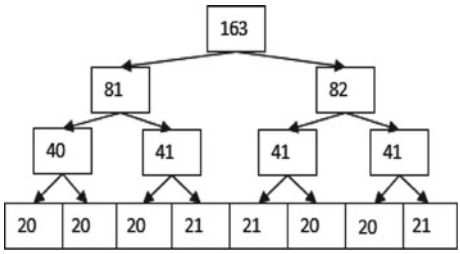

Fig. 5 Tree for 163-bit Karatsuba multiplier

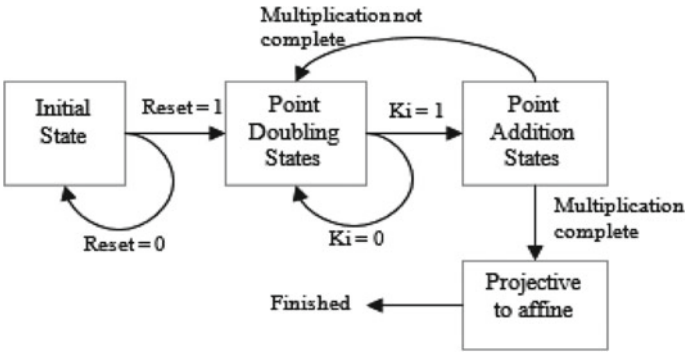

Fig. 6 FSM for PM control logic

From the point doubling and point addition algorithms, these operations will require a fixed number of states and hence a fixed number of clock cycles for execution.

Another aspect of this architecture is that the key register is placed separately from the register file. This physical separation of the key wherein it cannot be accessed by any unauthorized means introduces the hardware-level protection for the key. This physical boundary between the key and any malicious entity is one of the most important advantages offered by hardware-based implementations over conventional software implementations of cryptographic algorithms.

3 Discussions

The proposed design was implemented using the binary field GF(2^{163}) with the standard NIST recommended irreducible polynomial $p = x^{163} + x^7 + x^6 + x^3 + 1$. The curve equation was taken to be the NIST recommended binary curve E: $y^2+xy = x^3 + x + 1$, i.e. $a = b = 1$. The proposed architecture makes use of a 163-bit hybrid Karatsuba multiplier for field multiplications and dedicated inverter modules for field inversions. While this will increase the speed of execution, the associated trade-off in the area usage can be seen directly. The key register was 32-bit long. The comparisons of the proposed architecture with other published papers using the same FPGA and binary field GF(2^{163}) are given in Table 1. The number of clock cycles and the time are the values obtained for one point multiplication.

From the table, it can be easily seen that area-efficient implementations are much slower than the proposed design. However, the proposed design suffers from a very low maximum clock frequency which makes the time required for one point multiplication comparable to the other implementations. But implementations with an area usage which is comparable to the proposed design too require more time for the execution of one point multiplication.

The area of the proposed design is large due to the use of the dedicated inverter modules and the Karatsuba multiplier. The resource usage of these modules is given in Table 2.

The area usage indicates that using a more area-efficient multiplier can reduce the overall area requirement, at the cost of increased latency. The use of an inversion algorithm such as the Itoh-Tsujii algorithm can further reduce area usage as this

Table 1 Comparison of proposed design with other published works

Ref.	Slices	LUTs	Clock cycles	F_{max} (MHz)	Time (μs)	Field size	FPGA
[16]	16,209	26,364	3,010	154	19.55	163	Virtex-4
[17]	12,834	22,815	3,372	196	17.2	163	Virtex-4
[5]	3,536	6,672	4,168	290	14.39	163	Virtex-4
[18]	–	27,889	2,128	133	16	163	Virtex-4
[8]	4,080	7,719	4,050	197	20.56	163	Virtex-4
Ours	14,255	27,111	918	71	12.8	163	Virtex-4

Table 2 Resource usage of various modules in the proposed design

Module	Slices	LUTs	FPGA
Hybrid 163-bit Karatsuba multiplier	6,304	1,0982	Virtex-4
Field inverter	1,365	2,581	Virtex-4
Double squarer	181	315	Virtex-4
Squarer	95	165	Virtex-4
Adder	94	163	Virtex-4

algorithm uses the existing squaring and addition modules. The other field arithmetic modules, namely the adder, squarer and double squarer, have a negligible resource usage as compared to the multiplier and inverter.

4 Conclusions

This paper presents a fast implementation of the scalar point multiplier required for elliptic curve cryptography. The area usage, while large as compared to other similar implementations, is a justified trade-off for the speed the design possesses. The number of clock cycles required for the completion of one point multiplication is the lowest and thus shows the high-speed nature of the design, which can be used for many high-performance applications which do not have tight area constraints. The resource usage can be lowered by using other inversion algorithms and multipliers, but these optimizations will cause an increase in latency. The architecture for an entire elliptic curve crypto-processor using the elliptic curve Diffie–Hellman protocol can be constructed around this scalar point multiplier. A viable implementation of such a processor on an FPGA can use the embedded MPU found on most modern FPGAs such as the Artix-7.

References

1. Institute of Electrical and Electronics Engineers (2000) P1363 standard specifications for public key cryptography, NY
2. American National Standards Institute (1999) X 9.62 public key cryptography for the financial services industry: elliptic curve digital signature algorithm (ECDSA), Washington
3. National Institute of Standards and Technology (1994) FIPS 186-digital signature standard, Gaithersburg
4. Dormale GM, Quisquater J-J (2007) High speed hardware implementations of elliptic curve cryptography, a survey. Elsevier J Syst Archit 53:72–84
5. Khan ZUA, Benaissa M (2015) Throughput/area efficient ECC processor using montgomery point multiplication on FPGA. IEEE Trans Circuits Syst II Express Briefs 62(11):1078–1082

6. Khan ZUA, Benaissa M (2016) High-speed and low-latency ECC processor implementation over GF(2m) on FPGA. IEEE Trans Very Large Scale Integr (VLSI) Syst 25(1):165–176
7. Liu Z, Liu D (2016) An efficient and flexible hardware implementation of the dual-field elliptic curve cryptographic processor. IEEE Trans Ind Electron 64(3):2353–2362
8. Ansari B, Hasan MA (2008) High-performance architecture of elliptic curve scalar multiplication. IEEE Trans Comput 57(11):1443–1453
9. Roy S, Rebeiro C, Mukhopadhyay D (2013) Theoretical modeling of elliptic curve scalar multiplier on LUT-based FPGAs for area and speed. IEEE Trans VLSI Syst 21(5):901–909
10. Choi HM, Hong CP, Kim CH (2008) High performance elliptic curve cryptographic processor over GF(2163). In: 4th IEEE international symposium on electronic design, test and applications
11. Mahdizadeh H, Masoumi M (2013) Novel architecture for efficient FPGA implementation of elliptic curve cryptographic processor over GF(2163). IEEE Trans VLSI Syst 21(12):2330–2333
12. Hankerson D, Menezes A, Vanstone S (2004) Guide to elliptic curve cryptography. Springer
13. Anoop MS (2007) Elliptic curve cryptography: an implementation guide
14. Rebeiro C, Mukhopadhyay D (2007) Hybrid masked Karatsuba multiplier for 233(2) GF. In: 11th IEEE VLSI design and test symposium, Kolkata
15. Kodali RK, Amanchi CN, Kumar S, Boppana L (2014) FPGA implementation of Itoh-Tsujii inversion algorithm. In: IEEE international conference on recent advances and innovations in engineering, Jaipur
16. Chelton WN, Benaissa M (2008) Fast elliptic curve cryptography on FPGA. IEEE Trans VLSI Syst 16(2):198–205
17. Azarderakhsh R, Reyhani-Masoleh A (2012) Efficient FPGA implementations of point multiplication on binary edwards and generalized Hessian curves using Gaussian normal basis. IEEE Trans VLSI Syst 20(8):1453–1467
18. Fournaris AP, Zafeirakis J, Koufopavlou O (2014) Designing and evaluating high speed elliptic curve point multipliers. In: Euromicro conference on digital systems design