Low Latency Scheduling of Point Multiplication Featuring High Speed GF (2^m) Multiplier Suitable for FPGA Implementation

I.Blessing Meshach Dason
Department of ECE
Jayaraj Annapackiam CSI College of Engineering
Thoothukudi, India
blessingmeshach@gmail.com

N. Kasthuri
Department of ECE
Kongu Engineering College
Erode, India
kasthuri@kongu.ac.in

***Abstract*—Point Multiplication (PM) is the most tedious and timeconsuming operation in Elliptic Curve Cryptography (ECC). Performing PM using a standalone hardware platform – the Field Programmable Gate Array (FPGA) is much necessary for high speed, secured communication. In this paper, we employ Lopez-Dahab (LD) Montgomery PM algorithm to perform PM faster, resisting the side channel attacks. We propose a careful scheduling of LD algorithm to reduce the number of clock cycles; thus reducing latency. Speed of PM fully depends on the speed of finite field multiplier. We implemented the finite field Montgomery modular multiplier in several FPGAs and found that Virtex6 FPGA achieves the highest speed (14.81 ns).**

Keywords—Point Multiplication (PM), Elliptic Curve Cryptography (ECC), Field Programmable Gate Array (FPGA),Lopez-Dahab (LD) Montgomery PM Algorithm

I. INTRODUCTION

The need for sensitive data communications such as bank transactions is on very high rise. The process of carrying out such a secured communication is termed as Cryptography. Cryptography is categorized into two types: Symmetric key cryptography (Private key cryptography) and Asymmetric key cryptography (Public key cryptography). Distribution of private key before establishing a communication is a difficult process over internet communication. Therefore, asymmetric key cryptography which shares public key over the internet is of great interest for establishing a secured communication. Security of Public key cryptography depends on the difficulty to solve a mathematical hard problem. Very common algorithms of this type are: RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Key Cryptography). ECC is increasingly popular nowadays because it uses smaller key size which suits high-speed environments [1]. The security of ECC depends on the mathematical hardness to solve a Discrete Logarithmic Problem in which given the public key, the private key can only be calculated in Exponential time. To make it more superior over RSA, ECC also supports faster computation.

The most time consuming operation of ECC is the Scalar Point Multiplication (PM) which computes the public key Q by multiplying a base point of the elliptic curve P and the private key (Integer) K. Latency is very high for software implementation of PM. Therefore, developing a standalone hardware platform with low latency and increased performance is the need of the hour [2]. For hardware implementation of ECC, Elliptic curves based on binary extension field are more suitable than the prime field counterpart [1]. Two common bases for representing finite field elements are: polynomial bases and normal bases. Polynomial arithmetic is regular and therefore they are much suitable for scheduling of operations. This paper discusses arithmetic over binary extension field GF (2m) in polynomial bases.

The aim of this paper is to reduce the number of clock cycles (latency) required to perform point multiplication. The implementations in [1], [2], [4], [5], [7], [8], [10] achieves this task by means of pipelining the operations and by parallel processing. This paper selects an algorithm in view of this task and proposes a careful scheduling. A high speed field multiplier implementation is studied in several FPGA devices selecting an optimum one.

The rest of the paper is organized as below. Section II gives the mathematical background of ECC and discusses the selection of appropriate coordinates to perform scalar multiplication with reduced cost. Section III selects a suitable multiplication algorithm from the literature for parallelizing the operations of ECC. Section IV gives the proposed scheduling of ECC operations facilitating high speed ECC design. A finite field multiplier is also implemented and results are presented. Finally, Section V concludes the paper.

II. MATHEMATICAL BACKGROUND

A. ECC Hierarchy

The most fundamental and time consuming operation in ECC is the Point Multiplication (PM) operation, Q=KP. Here, the public key is computed by adding a point P itself 'K' number of times.

$$Q = KP = P + ...P + P \tag{1}$$

On the other hand, calculating the private key from the public key is very difficult. This mathematical hard problem is called Elliptic Curve Discrete Logarithmic Problem (ECDLP).

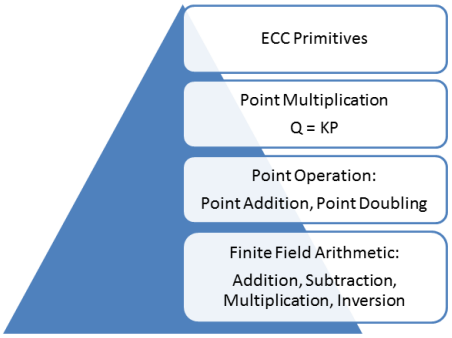

Fig. 1. ECC Hierarchy Diagram

Point Multiplication is composed of point addition and point doubling down the hierarchy as shown in Fig. 1. The underlying mathematics of the point operation is the finite field arithmetic.

B. Finite Field Arithmetic Over GF (2m)

Finite Field Arithmetic is based on Galosis Field (GF) theory, named after a French Mathematician Evariste Galosis. The set of elements of a finite field GF satisfy the properties of an Abelian group. GF is widely classified into: the prime field GF (p) and binary extension field GF (2m) [6]. Binary extension field GF (2m) is much suitable for hardware implementation because of its carry free operations. As a result, the critical path delay is smaller and operation is faster.

For GF (2m) operation, arithmetic on non supersingular elliptic curves is concentrated to prevent possible breaking algorithms [3]. The non supersingular curve is expressed by the following equation:

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

Where a,b Ɛ GF (2^m), b ≠ 0 and Ơ be the point at infinity.

Let P = (x1, y1) Ɛ E and Q = (x2, y2) Ɛ E, then P+Q = (x3, y3) also Ɛ E. When P ≠ Q point addition is performed and when P = Q the operation is point doubling. Formulas to perform these operations are:

$$x_3 = \begin{cases} \lambda^2 + \lambda + x_1 + x_2 + a, & when\ P \neq Q \\ x_1^2 + \dfrac{b}{x_1^2}, & when\ P = Q \end{cases} \tag{3}$$

$$y_3 = \begin{cases} \lambda(x_1 + x_3) + x_3 + y_1, & when\ P \neq Q \\ x_1^2 + \left(x_1 + \dfrac{y_1}{x_1}\right)x_3 + x_3, & when\ P = Q \end{cases} \tag{4}$$

Where $\lambda = \dfrac{y_1 + y_2}{x_1 + x_2}$

Equations (3) and (4) perform point addition and point doubling operations in affine coordinate system which involve costly finite field inversions. Affine coordinate system uses only two coordinates x, y with x, y Ɛ GF (2m). There is an alternative coordinate system called projective coordinate system that involves a redundant coordinate z which can be derived from the other two coordinates. However, projective coordinate system avoids finite field inversions while performing point addition and point doubling [4].

C. Lopez-Dahab Projective Coordinate system

Point multiplication Q=KP cannot be computed simply by means of a straight forward multiplication approach. It is actually performed by repetitive addition of the point P to itself by K number of times [1]. Many point multiplication algorithms are suggested in the literature such as Binary point, Non-Adjacent Form (NAF), Windowing methods, Montgomery method. Lopez and Dahab suggested a modified form of Montgomery point multiplication based on projective coordinate system [5].

The curve equation of Lopez-Dahab projective coordinate system is:

$$E: y^2 + xyz = x^3 z + ax^2 z^2 + bz^4 \tag{5}$$

Where a,b Ɛ GF (2m), b ≠ 0 and Ơ be the point at infinity.

Equation (5) is related to affine coordinate system as: (x: y: z) → (x/z, y/z2). Using projective coordinate system in point addition and point doubling avoids costly finite field inversions.

Lopez-Dahab's modified version involves the functions Madd, Mdouble to perform addition, doubling operations inside its main loop. These operations are performed in the projective coordinates [5]. Addition function Madd computes (Xadd, Zadd) using the formulas specified in (6). The input arguments of Madd are X1, Z1, X2, Z2.

$$\begin{aligned} Z_{add} &= (X_1.Z_2 + X_2.Z_1)^2 \\ X_{add} &= x.Z_{add} + (X_1.Z_2).(X_2.Z_1) \end{aligned} \tag{6}$$

Equation (6) shows Madd utilizes the following number of finite field arithmetic elements: 2 Adders, 6 Multipliers and 1 Squarer.

Function Mdouble computes (Xdouble, Zdouble) given the input arguments X2, Z2 as defined in (7).

$$\begin{aligned} Z_{double} &= (X_2.Z_2)^2 \\ X_{double} &= X_2^4 + b.Z_2^4 \end{aligned} \tag{7}$$

Equation (7) shows Mdouble utilizes the finite field elements: 1 Adder, 2 Multipliers and 5 Squarers.

It is interesting to notice that (6) and (7) do not involve Y coordinate as well as costly inversions. Finally, at the end of the Lopez-Dahab algorithm the conversion function Mxy converts the projective coordinates back to affine coordinates. Function Mxy computes the x and y coordinates of public key Q. Conversion formula is as specified in (8). The input arguments of Mxy are X1, Z1, X2, Z2.

$$\begin{aligned} x_3 &= X_1/Z_1 \\ y_3 &= \begin{cases} ((x + X_1)/Z_1)[(X_1 + x.Z_1)(X_2 + x.Z_2) \\ + (x^2 + y)(Z_1.Z_2)](x.Z_1.Z_2)^{-1} + y \end{cases} \end{aligned} \tag{8}$$

III. POINT MULTIPLICATION ALGORITHMS

A. Basic Algorithm

Point multiplication plays a crucial role in the design of ECC. The inputs to a point multiplier are: an integer K and a base point P. Several point multiplication algorithms exist in the literature [6]. The basic approach is given in Algorithm 1.

For each bit of binary K, point doubling is performed and if $K_i = 1$, point addition is also performed after point doubling operation.

Input: Point $P = (x, y) \in GF(2^m)$ and
Integer $K = K_{l-1},.. K_1K_0$
Output: $Q(x_3, y_3) = K. P$
Initialize: $Q := O$ which is the point at infinity
for i in (l-1) to 0 do
  $Q := 2Q;$
  if $K_i = 1$ *then*
    $Q := Q+P;$
  end if
end loop
Return $Q(x_3, y_3)$

Fig. 2. Algorithm 1: Binary Point Multiplication

B. Montgomery Point Multiplication Algorithm

Peter L. Montgomery proposed an alternate approach in which both point addition and point doubling are executed for every bit of K. The advantage of using Montgomery (Algorithm 2) is that since same operation is performed in the main loop iterations and the difference between P_2 and P_1 is invariant, the algorithm is highly resistive to side channel power attacks [7].

Input: Point $P = (x, y) \in GF(2^m)$ and
Integer $K = K_{l-1},.. K_1K_0$
Output: $Q(x_3, y_3) = K. P$
Initialize: $P_1 := P; P_2 := 2P;$
for i in (l-2) to 0 do
  if $K_i = 1$ *then*
    $P_1 := P_1+P_2; P_2 := 2P_2;$
  else
    $P_2 := P_2+P_1; P_1 := 2P_1;$
  end if
end loop
Return $Q = P_1(x_3, y_3)$

Fig. 3. Algorithm 2: Montgomery Point Multiplication [5]

Montgomery method reduces the number of registers needed for field arithmetic operation, yet it does not have speed advantage since the basic operations involve costly inversions inside the main loop.

C. Lopez Dahab Montgomery Point Multiplication Algorithm

Lopez and Dahab proposed a modified version of the Montgomery algorithm in which the point addition and point doubling operations can be executed in parallel. In the projective version of the algorithm, the usage of Y-coordinates and point inversions are avoided in loop operation. However, there are inversions when projective coordinates are converted back to affine coordinates. Algorithm 3 presents Lopez Dahab (LD) Projective coordinate version of Montgomery point multiplication.

Algorithm3 also supports parallel execution of operations. Hence LD Montgomery Point Multiplier is more suitable for high speed ECC Processor implementation.

Input: Point $P = (x, y) \in GF(2^m)$ and
Integer $K = K_{l-1},.. K_1K_0$
Output: $Q(x_3, y_3) = K. P$
Initialize: $X_1 := x; Z_1 := 1; X_2 := x^4+b; Z_2 := x^2;$
for i in (l-2) to 0 do
  if $K_i = 1$ *then*
    $Madd\ (X_1, Z_1, X_2, Z_2);\ Mdouble\ (X_2,Z_2);$
  else
    $Madd\ (X_2, Z_2, X_1, Z_1);\ Mdouble\ (X_1,Z_1);$
  end if
end loop
Return $(Q(x_3, y_3) = Mxy(X_1, Z_1, X_2, Z_2))$

Fig. 4 Algorithm 3: Montgomery LD Projective Point Multiplication [5]

IV. HIGH SPEED ECC DESIGN AND IMPLEMENTATION

A. Lopez Dahab's Point Operations Approach

Point operations of Lopez Dahab algorithm functions Madd and Mdouble are as specified in Table I. Function Madd involves 2 A, 4 M, 1 S operations and Mdouble involves 1 A, 2 M, 4 S operations. Here A, M and S represent the costs of finite field addition, multiplication and squaring elements respectively. The number of registers that are involved in Table 1 is six.

TABLE I. LD ADDITION AND DOUBLING OPERATIONS

Function Madd $(X_{add}, Z_{add}) \leftarrow$ ***Madd*** (X_1, Z_1, X_2, Z_2)	**Function Mdouble** $(X_{double}, Z_{double}) \leftarrow$ ***Mdouble*** (X, Z)
1. $Z_1 \leftarrow Z_1 . X_2$	1. $Z \leftarrow Z^2$
2. $X_1 \leftarrow Z_2 . X_1$	2. $T \leftarrow Z^2$
3. $T \leftarrow Z_1 + X_1$	3. $T \leftarrow b . T$
4. $X_1 \leftarrow X_1 . Z_1$	4. $X \leftarrow X^2$
5. $Z_1 \leftarrow T^2$	5. $Z \leftarrow X . Z$
6. $T \leftarrow x . Z_1$	6. $X \leftarrow X^2$
7. $X_1 \leftarrow X_1 + T$	7. $X \leftarrow X + T$
8. Return (X_{add}, Z_{add})	8. Return (X_{double}, Z_{double})

LD Algorithm is very much suitable for high speed design since it supports parallelizing the operations. Parallel operations should be carefully scheduled to prevent any idle clock cycle due to data dependency between the operations while pipelining. Hence careful scheduling is very much important for high speed design.

B. Proposed Scheduling of Operations

High speed design utilizes more than one multiplier for point operations [8]. Design utilizing two multipliers would be optimum in terms of area-time metric. The dataflow graph of proposed two multiplier design is shown in Fig. 5. Latency for point operations in Fig. 5 is 3M clock cycles, where M is the latency of a finite field multiplier.

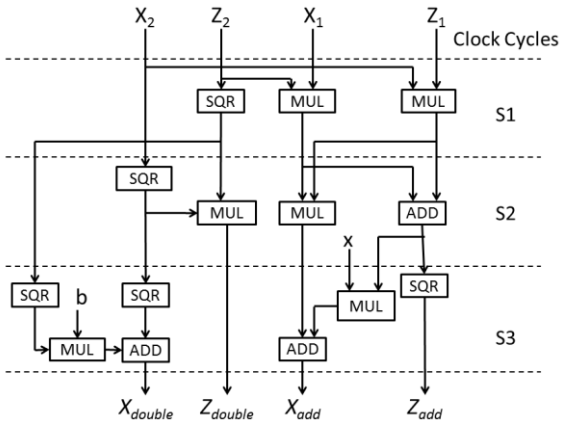

Fig. 5. Proposed data dependency graph for Madd and Mdouble utilizing two multipliers in parallel

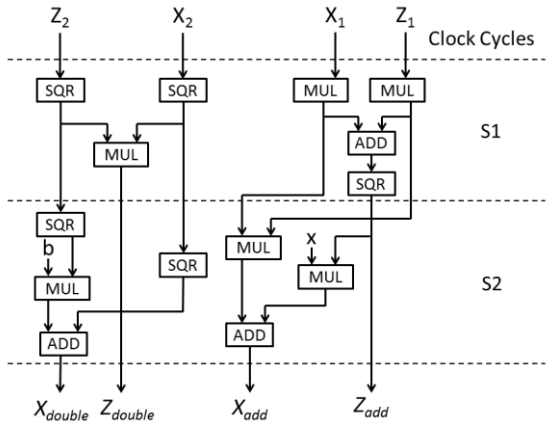

Fig. 6. Proposed data dependency graph for Madd and Mdouble utilizing three multipliers in parallel

High speed design can be achieved by operating three finite field multipliers in a parallel fashion as proposed in Fig. 6. Since only X and Z coordinates are only involved in LD Montgomery operations, the data dependency is much reduced and it is possible to complete an iteration of point arithmetic in just two clock cycles. Latency of Fig. 6 is just 2M, M being the latency of a finite field multiplier.

C. FPGA Implementations and Comparisions

Hardware implementation of the proposed design involves finite field addition, multiplication and squaring of operations. Addition and squaring circuits are simple combinational circuits. Performance of the field multiplier circuit determines the overall performance of ECC scalar multiplication. GF (2m) field multiplier has two parts in it: GF (2) multiplication followed by reduction. We have selected Montgomery modular multiplier since reduction part is easy by just ignoring the higher power terms. Over GF (2163), a NIST recommended curve [9], the architecture of modular multiplier [10] was designed using VHDL – the hardware description language. Using Xilinx ISE 14.5 tool, the design was implemented on Virtex4 (XC4VFX12), Virtex5 (XC5VLX20T), Virtex6 (XC6VCX75T), Spartan3E (XC3S100E), Spartan6 (XC6SLX4). Table II provides the comparison of different implementations.

TABLE II. VARIOUS FPGA IMPLEMENTATIONS OF FINITE FIELD MULTIPLIER – COMPARISON

FPGA	# of 4 input LUTs occupied	# of IO Buffers utilized	Delay in ns
Virtex4	56	24	24.041
Virtex5	149	24	23.313
Virtex6	149	24	14.81
Spartan6	162	24	24.363
Spartan3E	168	24	49.759

Our implementations show that Virtex6 FPGA achieves the highest speed taking only 14.81 ns involving 149 LUTs. Over GF (2^{163}), the total number of Clock cycles involved in point addition and doubling operations is 2*162=324 CCs for a high speed design.

V. CONCLUSION

This paper presented an approach to perform the most tedious operation of ECC, the scalar PM with reduced latency. LD projective coordinate system was selected aiming to avoid costly field inversions. Montgomery LD algorithm which supports parallelizing the finite field operations was carefully scheduled to reduce latency. Utilizing three multipliers in parallel, the proposed scheduling was able to complete the main loop of LD algorithm in two clock cycles. Now, the latency wholly depends on the architecture of GF multiplier. Implementing the fast modular multiplier in several FPGAs we conclude that Virtex6 FPGA is the best candidate for high speed ECC scalar PM. Over GF (2163), Virtex6 FPGA takes only 14.81 ns involving 149 LUTs. The number of Clock cycles in the main loop of LD algorithm is 324 CCs for the proposed high speed design.

REFERENCES

[1] Zia-Uddin-Ahamed Khan and M. Benaissa, "Throughput/Area Efficient ECC Processor using Montgomery Point Multiplication on FPGA," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 62, no. 11, pp. 1078-1082, Nov. 2015.

[2] F. Rodriguez-Henriquez, N. Saqib, and A. Díaz-Pérez, "A fast parallel implementation of elliptic curve point multiplication over GF(2m)," Microprocessors Microsyst., vol. 28, no. 5-6, pp. 329-339, 2004.

[3] Beth. T and Schaefer. F, "Non Supersingular Elliptic Curves for Public Key Cryptosystems" Proceedings of the 10th annual

international conference on Theory and application of cryptographic techniques, pp. 316-327, Jan. 1995.

[4] Orlando, Gerardo and Christof Paar, "A High Performance Reconfigurable Elliptic Curve Processor for GF(2m)." in Lecture Notes in Computer Science: CHES 2000 Proceedings, pp. 41-56, 2000.

[5] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation," in Lecture Notes in Computer Science: CRYPTO 1999 Proceedings, pp. 316–327, 1999.

[6] D. Hankerson, A. J. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography. New York, NY, USA: Springer-Verlag, 2004.

[7] C. K. Koc and T. Acar, "Montgomery multiplication in GF(2k)", Designs, Codes and Cryptography, 14(1), pp. 57-68, April 1998.

[8] Reza Azarderakhsh and Arash Reyhani-Masoleh, "High-Performance Implementation of Point Multiplication on Koblitz Curves," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 60 (1). pp. 41-45, Jan. 2013.

[9] National Institute of Standards and Technology (NIST), "Recommended elliptic curves for federal government use," Jul. 1999. [online]. Available: http://csrc.nist.gov/encryption

[10] G. D. Sutter, J.-P. Deschamps, and J. L. Imana, "Efficient elliptic curve point multiplication using digit-serial binary field operations," IEEE Trans. Ind. Electron., vol. 60, no. 1, pp. 217–225, Jan. 2013.