Journal of Circuits, Systems, and Computers
Vol. 19, No. 2 (2010) 425–433

DOI: 10.1142/S0218126610006153

World Scientific
www.worldscientific.com

A SINGLE FORMULA AND ITS IMPLEMENTATION IN FPGA FOR ELLIPTIC CURVE POINT ADDITION USING AFFINE REPRESENTATION*

M. MORALES-SANDOVAL

Polytechnic University of Victoria, Information Technology Department,
C. Luis Caballero 1200, Ciudad Victoria, Tamaulipas, 87070, Mexico
mmoraless@upv.edu.mx

C. FEREGRINO-URIBE

Computer Science Department, National Institute for Astrophysics,
Optics and Electronics, L. Enrique Erro 1,
Tonantzintla, Puebla, 72840, Mexico
cferegrino@inaoep.mx

R. CUMPLIDO

Computer Science Department, National Institute for Astrophysics,
Optics and Electronics, L. Enrique Erro 1,
Tonantzintla, Puebla, 72840, Mexico
rcumplido@inaoep.mx

I. ALGREDO-BADILLO

University of Istmo, Ciudad Universitaria S/N,
Sto. Domingo Tehuantepec, Oaxaca, 70760, Mexico
algredabadillo@Sandunga.unistmo.edu

Received 26 June 2008
Accepted 24 September 2009

A formula for point addition in elliptic curves using affine representation and its implementation in FPGA is presented. The use of this new formula in hardware implementations of scalar multiplications for elliptic curve cryptography has the main advantages of: (i) reducing area for the implementations of elliptic curve point addition, and (ii) increasing the resistance to side channel attacks of the hardware implementation itself. Hardware implementation of scalar multiplication for elliptic curve cryptography using this new formulation requires low area resources while keeping high performance compared to implementations using projective coordinates, which are usually considered faster than the affine coordinates.

Keywords: Elliptic curve cryptography; point addition; hardware architecture; affine coordinates.

*This paper was recommended by Regional Editor Majid Ahmadi

1. Introduction

Elliptic curve cryptography (ECC) is a kind of public key cryptography founded on the mathematical properties of elliptic curves.[1,2] An elliptic curve over a field K is the set of points $P = (x, y) \in K \times K$ satisfying a non-singular Weierstrass equation Eq. (1):

$$E(K) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\,. \tag{1}$$

The set $E(K)$ together with the point O forms an additive abelian group $S = (E(K) \cup O, +)$. The security of elliptic curve cryptography is based on the difficulty to solve the discrete logarithm problem defined on S.

The "+" operation in the group S for elliptic curve point addition is defined for two different operations: addition *ECC-Add* to sum two distinct points $P, Q \in E(K)$ and doubling *ECC-Dbl* to sum a point $P \in E(K)$ to itself. Each of these operations is defined in terms of field operations in K such as inversions, multiplications, squarings and additions. The definition of each operation, ECC-Add and ECC-Dbl, varies accordingly to the coordinate system used to represent the points of the elliptic curve $E(K)$. ECC-Add and ECC-Dbl operations obey to geometrical interpretations to ensure the closure property of S.

The scalar multiplication dP is the result of adding the point $P \in E(K)$ to itself $d-1$ times, that is, $dP = \underbrace{P + P + P + \cdots + P}_{d-1 \text{ sums}}$.

The scalar d is in the range $[1, n-1]$, where n is the order of P, that is the smallest n such that $nP = O$. The scalar multiplication is the most time consuming operation in cryptographic schemes based on elliptic curves such as digital signatures and bulk encryption. In these schemes, a scalar d is the private key while the public key is the elliptic curve point dP, for a known point P. The main objective for breaking the system is to find the scalar d given the points dP and P, that is, the main objective is to solve the elliptic curve discrete logarithm problem.

Being dP the most time consuming operation in ECC, most of the related work on elliptic curve cryptography is proposed for efficient implementations of this operation in hardware.[3–10] However, the hardware implementation of dP should be not only efficient but resistant to side channel attacks.[11] In these attacks extra source information such as timing, power consumption, electromagnetic leaks or even sound can be exploited to break the system.

The traditional method for computing dP is the binary method. It parses every bit value of scalar d and executes at each iteration one ECC-Dbl operation followed by one ECC-Add only if the current bit value of d is "1". The direct hardware implementation of this dP method is vulnerable to side channel attacks, such as the SPA (Simple Power Analysis). In SPA, the attacker measures the power produced by the hardware executing the operation dP and tries to reveal the private key from those traces. An SPA attack for the hardware implementation of the binary method for dP is possible because ECC-Add and ECC-Dbl are different and they will produce

different power traces. Due the operations ECC-Add and ECC-Dbl are strongly related to the d's bits, the security of the system could be compromised.

One approach for preventing SPA attacks is to rewrite the addition formulas ECC-Add and ECC-Dbl so that a single formula can be used for both kinds of point sums, indifferently.[12] This approach has been considered in the literature for Weierstrass curves using affine,[13,14] projective coordinates,[3] and for special forms of the elliptic curve.[15]

This work presents a single formula for point addition in Weierstrass elliptic curves using affine coordinates and its hardware implementation in an FPGA, well suited for hardware implementations of scalar multiplication dP with resistance to side channel attacks.

The new formulation is derived from an analysis when both ECC-Add and ECC-Dbl are implemented in hardware, different to the unified formula proposed by Brier *et al.*,[13,14] where the formulation is derived from a mathematical approach using the geometrical interpretation of operations ECC-Add and ECC-Dbl. The new formulation presented in this work has the property of being a single formula with fixed and well defined operations for performing both ECC-Add and ECC-Dbl operation. This regularity in the new formulation matches very well with the assumptions previously mentioned of having an indistinguishable formulation for point addition in order to prevent SPA attacks in hardware implementations of scalar multiplication.

The next section describes the new formulations for point addition and the advantages of using affine instead of projective representation.

2. A New Single Formula for Point Addition

A software or hardware implementation of the scalar multiplication implies choosing the algorithms to perform finite field arithmetic, selecting the coordinate system to represent the elliptic curve points and selecting the algorithm to compute dP. Most of the works reported in the literature argue that López-Dahab coordinates, a kind of projective coordinates, are the best way to represent the elliptic curve points.[6,8,10] This argument is based on the fact that field inversion is a very time consuming operation, requiring for its computation the same time required to compute six or more field multiplications. However, for small area implementations, affine coordinates are better preferred because they require less field operations and also less intermediate registers during the computations, which could result in higher performance and lower hardware requirements.

Addition and doubling operations are very similar in affine representation for elliptic curves defined on binary fields GF(2^m). An elliptic curve defined on GF(2^m) is the set of points satisfying the equation Eq. (2):

$$E(GF(2^m)) : y^2 + xy = x^3 + ax^2 + b\,. \tag{2}$$

Given the points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$, the operations ECC-Add $(P, Q) = (x_{ADD}, y_{ADD})$ and ECC-Dbl$(P) = (x_{DBL}, y_{DBL})$ are shown from Eqs. (3) to (8):[16]

$$\lambda_1 = \frac{y_Q + y_P}{x_Q + x_P}\,, \tag{3}$$

$$x_{ADD} = \lambda_1^2 + \lambda_1 + x_Q + x_P + a\,, \tag{4}$$

$$y_{ADD} = \lambda_1(x_P + x_{ADD}) + x_{ADD} + y_P\,, \tag{5}$$

$$\lambda_2 = x_P + \frac{y_P}{x_P}\,, \tag{6}$$

$$x_{DBL} = \lambda_2^2 + \lambda_2 + a\,, \tag{7}$$

$$y_{DBL} = {x_P}^2 + \lambda_2 x_{DBL} + x_{DBL}\,. \tag{8}$$

The Eq. (8) can be rewritten using Eq. (6). So, Eq. (8) becomes Eq. (9):

$$y_{DBL} = \lambda_2(x_P + x_{DBL}) + x_{DBL} + y_P\,. \tag{9}$$

Both ECC-Add and ECC-Dbl operations require to perform one division, one multiplication and one squaring. The ECC-Add operation requires to perform nine additions and the ECC-Dbl requires six. Although both kinds of elliptic curve point addition use almost the same number of operations, the way in which each one is defined is different. This implies a dedicated module when scalar multiplication is implemented in hardware. These different modules have different power traces that could be used in side channel attacks.

The single formula for operations ECC-Add and ECC-Dbl in affine coordinates aims: (i) to reduce hardware resources for implementing the addition operation in elliptic curves, used for performing scalar multiplications, and (ii) to increase the resistance of the dP hardware implementation to side channel attacks. The main idea behind the proposed formula is to unify the ECC-Add and the ECC-Dbl operations by multiplexing data according to the operation being performed. Such multiplexing is implemented by introducing the operation $s_0 \cdot x$, which is the bitwise AND operation of bit s_0 with each bit-value of x.

By introducing the $s_0 \cdot x$ operation in the original formulas for point addition and applying boolean reductions, the new formulas to perform an ECC-Add operation if $s_0 =$ "1" or an ECC-Dbl operation if $s_0 =$ "0", is the operation $(X, Y) =$ POINT_ADDITION(P, Q, s_0), where X and Y are defined as in Eqs. (11) and (12):

$$\lambda = \frac{s_0 \cdot y_Q + y_P}{s_0 \cdot x_Q + x_P}\,, \tag{10}$$

$$X = (\lambda + s_0 \cdot x_P)^2 + \lambda + s_0 \cdot x_Q + x_P + a\,, \tag{11}$$

$$Y = (\lambda + s_0 \cdot x_P)(x_P + X) + X + y_P\,. \tag{12}$$

Table 1. Complexity of proposed single formulation against related work.

Field operation	POINT_ADDITION	Unified formula in Refs. 13 and 14
Division	1	1
Multiplication	1	4
Squaring	1	3
Additions	10	13

The new formulation POINT_ADDITION for both ECC-Add and ECC-Dbl requires the following field operations: ten additions, one division, one multiplication and one squaring. That is, the new formula requires one more addition in the case of the ECC-Add operation and four additions in the case of the ECC-Dbl. Field additions in GF(2^m) are trivial operations implemented as XOR operations so this difference has not a serious impact in the timing to compute any of the two elliptic curve point additions. Instead of having two distinct hardware modules for each ECC elliptic curve point addition operation, a single hardware module is provided thus resulting in smaller area requirements.

Table 1 compares the complexity in terms of field operations of the proposed formulation and the formulation previously studied by Brier *et al.*[13,14]

The new formulation POINT_ADDITION uses less field operations, which results in faster execution time. Division and multiplication are the critical field operations in finite fields. In GF(2^m), typical latencies for these operations are $2m-1$ and m, respectively. This means that the proposed formulation in this work is about 2.6 times faster that the formulation reported in the literature.[13,14]

2.1. *Implementation of POINT_ADDITION formulation*

The GF(2^m) field operations used in elliptic curve point addition are well suited to be implemented in hardware using polynomial basis. Let $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ (where $f_i \in \{0,1\}$) be an irreducible polynomial of degree m over GF(2). The polynomial $f(x)$ is called the reduction polynomial. For each reduction polynomial there exists a polynomial basis representation. In such a representation, each element of GF(2^m) corresponds to a binary polynomial of degree less than m. That is, for each $e \in$ GF(2^m) there exist m numbers $e_i \in \{0,1\}$ such that

$$e = e_{m-1}x^{m-1} + \cdots + e_1 x + e_0\,.$$

The element $e \in$ GF(2^m) is usually denoted by the bit string $(e_0, e_1, \ldots, e_{m-1})$ of length m. Arithmetic in GF(2^m) using polynomial basis is arithmetic of polynomials modulo $F(x)$.

Figure 1 shows the data flow for the point addition module. Since field addition is an XOR operation and squaring can be implemented using combinatorial logic, the whole latency for point addition is the latency of a field division plus the one of a field multiplication. In Fig. 1, the combinatorial operations like AND and XOR are

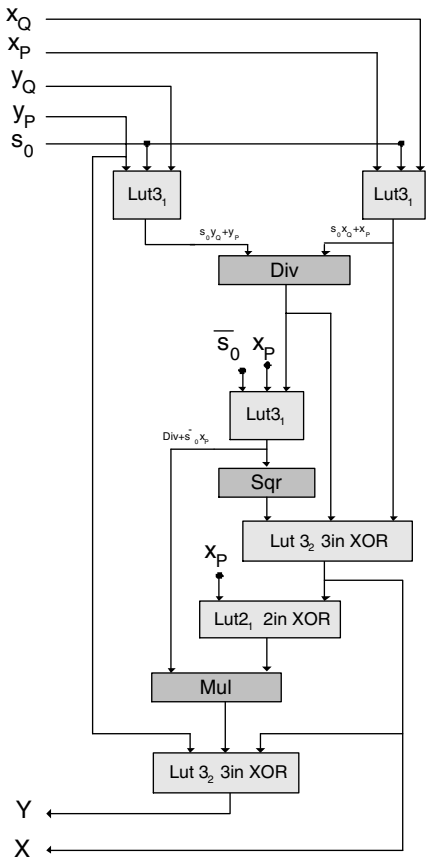

Fig. 1. Point addition diagram block.

represented as black boxes of two or three m-bit inputs. The black boxes are well mapped to LUTs (Look Up Table), which are elements in FPGAs that implement any Boolean function of up to 4-inputs.

A dP co-processor was implemented for evaluating the POINT_ADDITION operation. It implements the add and double method resistant to SPA attacks proposed by Coron.[17] This method parses each bit of the scalar $d = (1, s_{k-2}, \ldots, s_0)_2$ and performs an ECC-Dbl operation followed by an ECC-Add operation. The co-processor computes dP after $(k \cdot \text{ECC\_Add} + (k-1) \cdot \text{ECC\_Dbl})$ operations. Being L the latency in clock cycles of the POINT_ADDITION module, the co-processor delivers the final value dP in $L(2k-1)$ clock cycles.

3. Results

The dP co-processor using the POINT_ADDITION module was implemented in a V4 Xilinx's FPGA device for validation and performance analysis. The GF(2^m)

arithmetic modules direct division, serial multiplication, and combinatorial squarer used in the POINT ADDITION module were previously reported by Morales-Sandoval *et al.*:[18] direct division, serial multiplication, and combinatorial squarer. The latency of the POINT_ADDITION module is mainly determined by the latency of the divider ($2m - 1$ clock cycles) and the latency of the multiplier (m clock cycles), resulting in $L = 3m - 1$. So, the whole latency of the co-processor for computing dP using binary fields, affine representation and the Coron's binary method is $(3m - 1)(2k - 1)$.

Area and performance results for all the modules of the hardware dP co-processor are shown in Tables 2 and 3. Table 4 shows the time to compute dP using the new single formula and compares those results against related work.

Table 2. Synthesis results for the GF(2^m) arithmetic units optimized by speed and area.

Security level		113		131		163	
Optimization		Speed	Area	Speed	Area	Speed	Area
Divider[a]	Slices	730	459	848	529	1044	654
	Freq. (MHz)	156	96	152	92	151	90
Multiplier[b]	Slices	193	129	231	149	286	183
	Freq. (MHz)	298	276	291	270	283	260
Squarer[c]	Slices	32	32	75	75	95	95

[a]Direct division, latency of $2m - 1$ clock cycles.
[b]Serial multiplication, latency of m clock cycles.
[c]Combinatorial squaring, latency of one clock cycle.

Table 3. Synthesis results for the Point addition module optimized by speed and area.

Security level	113		131		163	
Optimization	Speed	Area	Speed	Area	Speed	Area
Slices	1462	1174	1948	1590	2418	1966
Freq. (MHz)	165.20	124.22	152.06	101.94	151.70	96.47

Table 4. Devices used, area consumption and execution time of dP implementations in GF(2^m) (synthesis optimized for speed).

Ref.	m	Device	Area	Freq.	Time (ms)
3	179	XCV800	10,626 slices	52 MHz	2.47
4	113	AT94K40 Amtel	38.4 Kgates	12 MHz	10.9
5	163	XCV2000E	19,000 slices	66.4 MHz	0.14
7	160	XCV800	150 Kgates	47 MHz	3.81
9	163	V2Pro	4,749 slices	—	0.49
This work	113	XC4VFX12	2,405 slices	100 MHz	0.52
This work	131	XC4VFX12	2,871 slices	100 MHz	0.69
This work	163	XC4VFX12	3,528 slices	100 MHz	1.07

The timing to compute dP in this work using a single hardware module for point addition in affine coordinates is better than other works that have used projective coordinates, like in Ernst *et al.* (10.9 ms for $m = 113$),[4] Mentens *et al.* (3.8 ms for $m = 160$),[7] or Batina *et al.* (2.47 ms for $m = 179$).[3] The use of projective coordinates supposes a better performance because inversions are avoided in each point addition operation at the cost of more multiplications. Other works using projective coordinates perform dP faster than the implementation presented in this article but they use higher area resources. For example, Sakiyama *et al.*[9] uses 4,749 slices from a Virtex2 Pro FPGA and performs dP in the field $GF(2^{163})$ in 0.49 ms. In the work of Gura *et al.*,[5] the area required is 19,000 slices from a Virtex2 FPGA while the dP operation in the field $GF(2^{163})$ is computed in 0.14 ms. The area used in Ref. 5 is six times bigger that the area used by the co-processor proposed, and the one used in Ref. 3 is three times bigger. In addition, the hardware implementation of the binary method for computing scalar multiplications dP using the single formula for point addition will be more resistant to side channel attacks.

4. Concluding Remarks

A new formula for ECC-Add and ECC-Dbl operation in elliptic curve cryptography using affine representation and its hardware implementation was presented. This new formulation performs as well as those using projective representation. The proposed unified formula reduces hardware while keeping the complexity of operations ECC-Add and ECC-Dbl as in its original form, which is mainly determined by the computational cost of one field division and one field multiplication. The whole latency of the point addition operation could be reduced by using better performing GF(2^m) arithmetic modules. This work provided an single formula for ECC point addition that makes the ECC-Add and ECC-Dbl operations indistinguishable, which could increase the security of the hardware implementation of dP against side channel attacks.

References

1. N. Koblitz, S. Vastone and A. Menezes, The state of elliptic curve cryptography, *Designs, Codes and Cryptography* **19** (2000) 173−193.
2. V. Miller, Use of elliptic curves in cryptography, *Proc. Advances in Cryptology, CRYPTO'85*, Santa Barbara, CA (1985), pp. 417−426.
3. L. Batina, N. Mentens, B. Preneel and I. Verbauwhede, Balanced point operations for side-channel protection of elliptic curve cryptography, *IEE Proc. Information Security*, October 2005, pp. 57−65.
4. M. Ernst, M. Jung, F. Madlener, S. Huss and R. Blumel, A reconfigurable system on chip implementation for elliptic curve cryptography over GF(2^n), *Proc. CHES'2002*, Vol. 2523 of LNCS, Redwood Shores, CA (2002), pp. 381−399.
5. N. Gura, S. C. Shantz, H. Eberle, S. Gupta and V. Gupta, An end to end systems approach to elliptic curve cryptography, *Proc. CHES 2002*, Vol. 2523 of LNCS, Redwood Shores, CA (2002), pp. 349−365.

6. J. Lutz and A. Hasan, High performance FPGA based elliptic curve cryptographic co-processor, *ITCC'04*: *Int. Conf. Information Technology*: *Coding and Computing* (2004), pp. 486−492.
7. N. Mentens, S. Berna and B. Preneel, An FPGA implementation of an elliptic curve processor GF(2^m), *Proc. 14th ACM Great Lakes Symp. VLSI*, Boston, MA (2004), pp. 454−457.
8. G. Orlando and C. Paar, A high-performance reconfigurable elliptic curve processor for GF(2^m), *Proc. 2nd Int. Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, Vol. 1965 of Lecture Notes in Computer Science, Worcester, MA (2000), pp. 41−56.
9. K. Sakiyama, L. Batina, B. Preneel and I. Verbauwhede, Superscalar coprocessor for high-speed curve-based cryptography, *Proc. CHES 2006*, Vol. 4249 (2006), pp. 415−429.
10. N. Saquib, F. Rodriguez and A. Diaz, A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2^n) *Proc. 11th Reconfigurable Architectures Workshop, RAW'04*, Sta. Fe, USA (2004), pp. 26−27.
11. M. Joye, Elliptic curves and side-channel analysis, *ST J. Syst. Res.* **4** (2003) 17−21.
12. B. Chevallier-Mames, M. Ciet and M. Joye, Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity, *IEEE Trans. Comput.* **53** (2004) 760−768.
13. E. Brier, I. Dechene and M. Joye, Unified point addition formulae for elliptic curve cryptosystems, *Embedded Cryptographic Hardware*: *Methodologies and Architectures* (Nova Science Publishers, New York, 2004).
14. E. Brier and M. Joye, Weierstrass elliptic curves and side-channel attacks, *Public Key Cryptography — PKC 2002*, LNCS, Vol. 2274 (Springer-Verlag, 2002), pp. 335−345.
15. T. Izu and T. Takagi, A fast parallel elliptic curve multiplication resistant against side channel attacks, *PKC'02*: *Proc. 5th Int. Workshop on Practice and Theory in Public Key Cryptosystems*, London, UK (2002), pp. 280−296.
16. IEEE P1363 Committee, Standards specification for public key cryptography (1998), http://grouper.ieee.org/groups/1363/.
17. J. S. Coron, Resistance against differential power analysis for elliptic curve cryptosystems, *Proc. Cryptographic Hardware and Embedded Systems* (1999), pp. 292−302.
18. M. Morales-Sandoval and C. Feregrino-Uribe, GF(2^m) arithmetic modules for elliptic curve cryptography, *Proc. 3rd Int. Conf. ReConFigurable Computing and FPGAs* (*ReConFig06*), Colima, Mexico, September 2006, pp. 176−183.