Very Fast Elliptic Curve Cryptography Public-key Generator on Koblitz Curves

Gebre Tsegay
Dept. of Computer Engineering
Defense Institute of Advanced Technology (DIAT)
Pune, India
gtswefri@yahoo.com

MM. Kuber
Dept. of Computer Engineering
Defense Institute of Advanced Technology (DIAT)
Pune, India
mmkuber@gmail.com,mmkuber@yahoo.com

***Abstract*—This paper presents Xilinx vertex-5 FPGA-based very fast elliptic curve cryptography public key generator on Koblitz curves targeting for applications requiring high speed. The generator supports both fast computation of point multiplication using window method and multiple point multiplications with joint sparse form representations. In order to achieve better performance, optimized operation-specific processing units are utilized. Throughput is enhanced using pipelining techniques. The generator computes point multiplication on average in 16.95 µs and achieves a maximum of 160,920 operations per second. A 2-term multiple point multiplication requires 35.12 µs with a maximum of 61,719 operations per second.**

Keywords- Elliptic curve cryptography, Field programmable gate array, Elliptic curve public key generatorr, very fast elliptic curve cryptography public key generatory.

I. INTRODUCTION

Elliptic curve cryptography offers the same level of security as traditional public key cryptography with considerably shorter keys [2]. So, it has replaced traditional public-key cryptosystems in environments where short keys are important. Public-key cryptosystems require considerable computation time, hence, the fact that elliptic curve cryptography has been shown to be faster than traditional public-key cryptosystems [2] is of great importance. Speed can be further increased by using a special class of elliptic curves referred to as Koblitz curves [3].

This paper discusses FPGA-based implementation of elliptic curve cryptography which has attained considerable interest in both cryptography and FPGA communities. The interest comes from the fact that public-key cryptosystems require a lot of computation which often yield a need for hardware accelerator. FPGAs are feasible alternatives for cryptographic implementations because of the combination of fast performance and flexibility.

This paper is organized as follows: In section II preliminaries are presented with sub-sections A,B,C and D. Next the main algorithms of this paper window method and multiple point multiplication are discussed. Finally, the implementation followed by analysis, results and conclusion will be discussed.

II. PRELIMINARIES

Elliptic curve defined over finite binary fields, denoted by F_2^m, are commonly used in practical cryptosystems. These curves are called binary curves. In this paper, binary curves of the following form are considered:

$$E: y^2 + xy = x^3 + ax^2 + b, \quad (1)$$

where a,b€F_2^m with $b \neq 0$. The additive operation of the group is referred to as point addition and it is defined as $P_3 = P_1 + P_2$ where $_{Pi}$ € F_2^m. Point multiplication, which is a basic component of every elliptic curve cryptosystem, is defined by using point additions as follows:

$$Q = kP = P + P + P \ldots P, \quad (2)$$

where Q,P€E(F_2^m) and k is an integer. P is called the base point and Q is the result point. Point multiplication decomposes into three levels of hierarchy from top to bottom as follows: point multiplication, point operation and finite field arithmetic. These hierarchy levels are discussed in the following sections. Each of the levels of the hierarchy is considered in sections A,B and C. Finally, Koblitz curves are discussed in section D.

A. Point Multiplication

Point multiplication is calculated using point doubling and point addition. Point doubling is a special case of point addition in which $P_3 = P_1 + P_2$, where $P_1 = P_2$ and point addition is calculated as $P_3 = P_1 + P_2$ where $P_1 \neq P_2$. Both point additions and point doublings are used in computing Eq. (2) when the integer k is represented with binary expansion as

$$K = \sum_{i=0}^{\ell-1} k_i \, 2^i \quad (3)$$

where k_i €{0,1}. Point doubling is done for all values of k_i whereas, point addition is not needed if $k_i = 0$. So, it is crucial to reduce the number of nonzeros. A simple option is to use a signed-bit representation, i.e., k_i€{0, ±1}, called non-adjacent form (NAF). NAF has the property that adjacent bits are never both nonzero. Every k has a unique

NAF and it has the minimum number of nonzeros among all signed-bit representations [6].

B. Point Operations

Point doubling, point addition, and point subtraction all require an inversion in F_2^m. Inversions are very expensive and it can be avoided by using projective coordinates where a point is represented with three coordinates as (X, Y, Z). We consider López-Dahab coordinates (LD) [12], where a point (X, Y, Z) represents the point $(X/Z, Y/Z^2)$ in affine (A). The LD coordinates allow an efficient mixed coordinate point - subtraction. If $P_1 = (X_1, Y_1, Z_1)$ is in LD and $P_2 = (x_2, y_2)$ is in A, point addition $P_3 = (X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$ is given as follows [9].

$$\begin{aligned} A &= Y_1 + y_2 Z_1^2;\ B = X_1 + x_2 Z_1; \\ C &= B Z_1;\ Z_3 = C^2; \\ D &= x_2 Z_3;\ X_3 = A^2 + C(A + B^2 + aC); \\ Y_3 &= (D + X_3)(AC + Z_3) + (y_2 + x_2) Z_3^2 \end{aligned} \quad (4)$$

C. Finite Field Arithmetic

We chose polynomial basis as polynomial multiplication is fast. Adding and squaring require only one clock cycle. For computing inversions, we used Fermat's Little Theorem as proposed by Itoh and Tsujii in [7].

D. Koblitz Curves

In 1985, Koblitz [1] and Miller [10] independently proposed the use of the additive finite abelian group of points on elliptic curves defined over a finite field for cryptographic applications. The Koblitz curves [1], or anomalous binary curves, are

$$E_k: y^2 + xy = x^3 + ax^2 + 1, \quad (5)$$

where a€{0,1}. The main advantage of Koblitz curves is that the Frobenius endomorphism of F_2^m acts on points via $\tau(x, y) = (x^2, y^2)$ and is essentially free to compute. Because τ satisfies $(\tau^2 + 2)P = \mu\tau(P)$ for all points P € $E(F_2^m)$ where $\mu = (-1)^{1-a}$, we can consider τ as a complex number satisfying $\tau^2 - \mu\tau + 2 = 0$, i.e., $\tau = (\mu + \sqrt{-7})/2$. Thus, computing kP, where k€Z and P€E_k (F_2^m), can be done using a representation of k involving powers of τ instead of the usual binary representation using powers of 2, yielding a point multiplication algorithm similar to the binary "double-and-add" method in which the point doublings are replaced by applications of the Frobenius [12]. Solinas [12] shows how the non-adjacent form (NAF) and window-NAF methods can be extended to τ-adic expansions.

III. WINDOW METHODS

If enough storage space is available, point multiplication can be sped up with window methods which involve pre-computations with P and process w bits of k at a time. The easiest way to width-w τNAF is by replacing certain strings of 0, 1, and -1 with window values. The resulting representation has an average weight of H(k) = l/(w + 1). The pre-computed points, P3, P5, P7 and P9 are given in Table 1.

An algorithm for width-w window point multiplication in Koblitz curves is given in Alg. 1.

Algorithm 1. Window algorithm
Input: Integer k, point P
Output: Result point Q = kP
$k_{l-1}, k_{l-2} \ldots k_1 k_0 \leftarrow$ w-τNAF(k)
P1, P3, . . . , $P2^{(w-1)} - 1 \leftarrow$ Pre-compute(P)
Q ← O
for i = l − 1 down to 0 do
Q ← Φ(Q)
  if $k_i \neq 0$ then
    Q ← Q + sign(k_i)P|k_i|
  end if
end for
Q ← xy(Q)

IV. MULTIPLE POINT MULTIPLICATION

Sum of n elliptic curve point multiplications is called multiple point multiplication and it is defined by

$$Q = \sum_{i=0}^{n} K^i P^i \quad (6)$$

where k_iP_i are point multiplications as defined by (2). All n point multiplications can be computed simultaneously with the so-called Shamir's trick [6]. Consider the case n = 2. The integers are represented as a table with k(1) and k(2) as rows. First, P1 + P2 is pre-computed. Analogously with the double-and-add algorithm, point multiplication proceeds column by column so that P1 is added if the column is 10, the point P2 if 01, and the pre-computed point if 11.

Generalization of Shamir's trick for n point multiplications is straightforward but requires more pre-computations. Because zero columns do not require point additions, it is possible to reduce computational cost by representing k(i) with signed-binary representations and choosing the representations which maximize the number of zero columns. A representation maximizing the number of zero columns is called τ -adic joint sparse form (τJSF). An algorithm for computing τJSF for n = 2 was presented in [10].

An algorithm for n-term multiple point multiplication on Koblitz curves is given in Alg. 2.

Algorithm 2. Multiple point multiplication
Input: n integers k(1), . . . , k(n), n points P(1), . . . , P (n)
Output: Result point $Q = \sum_{i=0}^{n} K^i P^i$
$k_{l-1}k_{l-2} \ldots k_1 k_0 \leftarrow \tau$JSF(k(1), . . . , k(n))
P1, P2, . . . ,P(3^n-1)/2 ← Pre-compute(P(1), . . . , P (n))
Q ← O
for i = l − 1 down to 0 do
  Q ← Φ(Q)

```
        if k_i ≠ 0 then
            Q ← Q + sign(k_i)P|k_i|
        end if
    end for
Q ← xy(Q)
```

V. IMPLEMEMNTATION

The aim of the fast generator is to provide high throughput. The motivation for designing the generator is to use it for PLA. So, it must support both 1-term and 2-term (multiple) point multiplications which are used in signing and verifying packets.

Algs. 1 and 2 share a common structure, i.e. both require conversion for integer(s), pre-computation, have the same for-loop, and convert the point Q back to A in the end. This paper utilizes this common structure in implementing the algorithms and use operation-specific processing units in order to increase efficiency.

The accelerator operates as follows. First, integer(s) k(i) and point(s) P(i) are sent to the τNAF/JSF converter and pre-processor, respectively. The converter converts integer(s) to either width-4 τNAF or τJSF and saves the result into a buffer. Simultaneously, the pre-processor performs pre-computations and stores points into the registers. When both converter and pre-processor are ready, the main processor executes the for-loop of Algs. 1 and 2. The control logic selects one of the pre-computed points according to the current k_i and the main processor adds or subtracts it to or from a temporary value Q = (X, Y, Z) and performs the following Frobenius endomorphisms. When the for-loop has been executed, the control logic enables the postprocessor which computes back the affine representation of the point Q.

A. *τNAF/JSF Converter*

The τNAF/JSF converter supports computation of two representations: width-4 τNAF and 2-term τJSF. The converter first converts all integers into τNAF simultaneously with two τNAF converters which are implemented as presented in [7]. Parallel processing was used in order to minimize total computation time.

A. *Preprocessor*

The pre-processor is based on the architecture presented in [5], but it uses polynomial basis instead of normal basis. Fig.1 depicts the architecture of the pre-processor. The storage RAM is used for storing temporary variables during pre-computations and it is implemented with embedded memory. The pre-processor is controlled by a finite state machine.

Points that are pre-computed in the pre-processor are listed in Tables 1 and 2. They are represented in affine coordinate so that (4) can be used in the for-loop. Pre-computations utilize unified point addition and subtraction formulae which compute both $P_1 + P_2$ and $P_1 - P_2$ with only

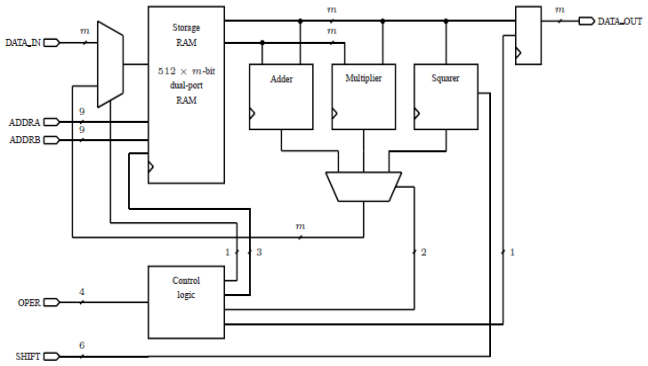


Figure 1. Pre-processor

TABLE 1. Pre-computed points for single point multiplication.

Pre-computed points for ω = 4 bits			
ω	**Point**	**Operation**	**Arithmetic**
0001	**P1**	P	**(x, y)**
010 î	**P3**	$\Phi^2(P) - (P)$	**(x^4,y^4) + (x, y + x)**
0 101	**P5**	$\Phi^2(P) + (P)$	**(x^4,y^4) + (x, y)**
1 00 î	**P7**	$\Phi^3(P) - (P)$	**(x^8,y^8) + (x, y + x)**
1001	**P9**	$\Phi^3(P) + (P)$	**(x^8,y^8) + (x, y)**

TABLE 2. Pre-computed points for multiple point multiplication.

K_2k_1	point	K_2k_1	point
00	R0 = 0		
01	R1 = P1	0î	-R1
10	R2 = P2	î0	-R2
1î	R3 = R2 + R1	î1	-R3
11	R4 = R2 - R1	îî	-R4

one inversion [6]. The number of inversions is further reduced using the so-called Montgomery's trick which trades inversions to multiplications [6]. If n = 2, only two points need to be pre-computed.

B. *Main Processor*

The main processor implements the for-loop of Algs. 1 and 2. This for-loop dominates the computational requirements and, hence, its efficient computation is necessary. The sequential nature of the loop forbids computing point operations in parallel. Parallelism can be used in Frobenius endomorphisms (squaring for all coordinates computed in parallel) and point additions. However, data dependencies usually prevent efficient use of parallelism in point additions and cause poor latency-area products. Details of the method are available in [8]. The method computes consecutive point additions and Frobenius endomorphisms efficiently with parallel field multipliers by interleaving successive operations. The key observation is that the computation of Z3 in (4) does not require Y1. Hence, the Z coordinate of the next point addition can be computed simultaneously with the Y coordinate of the previous point addition. The method redefines (4) so that the computation is performed with eight sub-computations each including one multiplication. We use four multipliers so that one of them is

devoted for the Z coordinate computations, (7)–(8), one for the X coordinate computations, (9)–(10), and two for the Y coordinate computations, (11)–(14).

$$z_0 : E = x_2 Z_1 \quad (7)$$

$$z_1 : \begin{cases} C = Z1(E + X_1) \\ F = aC + (E + X_1)^2 \\ Z_3 = C^2 \end{cases} \quad (8)$$

$$x_0 : G = y_2 Z_1^2 \quad (9)$$

$$x_1 : X3 = C + (F + G + Y_1) + (G + Y1)^2 \quad (10)$$

$$y_0 : H = C(G + Y_1) + Z_3 \quad (11)$$

$$y_1 : D = x_2 Z_3 \quad (12)$$

$$y_2 : J = Z_3^2 (x_2 + y_2) \quad (13)$$

$$y_3 : Y_3 = H(D + X_3) + J \quad (14)$$

The computation schedule, given in Fig. 2, clearly shows that the effective critical path is 2 multiplications per point addition. Because each multiplier is used for only two sub-computations, specialized processing units optimized for these sub-computations were designed. The processing units consist of a multiplier and several adders and squarers as shown in Fig. 3. The processing units compute the sub-computations so that their latency is the latency of multiplication plus one clock cycle.

When a result coordinate is ready (after z_1, x_1, and $y_{2/3}$), a squarer is used for computing Frobenius endomorphisms for that coordinate. Each Frobenius endomorphism requires one clock cycle. This architecture is presented in more detail in [8].

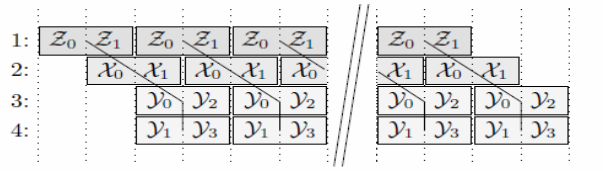
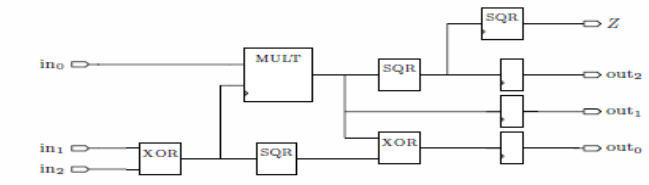
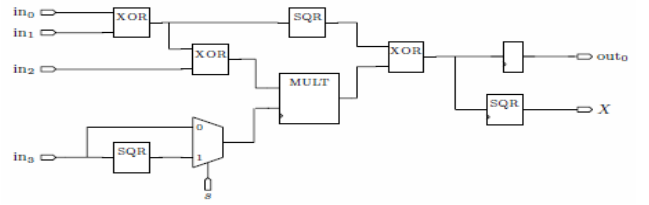


Figure 2. Computation schedule of the main processor. Operations connected with a line belong to the same point addition.

(a)



(b)



(c)

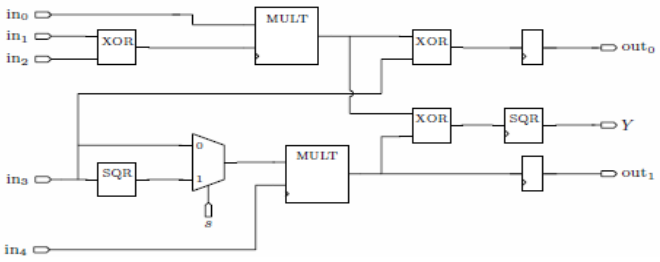


Figure. 3. Processing units for (a) z_0 and z_1, (b) x_0 and x_1, and (c) y_0, y_1, y_2, and y_3.

B. *Postprocessor*

The postprocessor maps the output of the main processor from LD to A, i.e. it computes the last line of Algs. 1 and 2. The computation requires one inversion, one squaring, and two multiplications, of which the inversion is the most complex operation by far. Inversions can be computed with the Fermat's Little Theorem as suggested by Itoh and Tsujii [7] because it uses successive squaring and multiplications which allows reusing the same hardware for inversion and other operations required. The architecture of the postprocessor was presented in [8] and it is depicted in Fig. 4.

VI. Analysis and Optimizations

The latencies of operations needed in Algs. 1 and 2 are listed in Table 3. The latencies of pre-processor, main-processor, and postprocessor depend on the latency of multiplication in F_2^m denoted by M.

Digit-serial multipliers are used and their latencies are given by

$$M = \lceil m/D \rceil + 1 \quad (15)$$

where m is the field size (m = 163) and D is the digit size. The digit size, D, defines both the latency and the size of a multiplier. Different D can be used for multipliers in different parts of the generator with the exception that all multipliers of the main processor must have the same D.

If fast multipliers (large D) are used, the constant latency of the τNAF/JSF converter becomes a bottleneck. In order to avoid this, we optimized the generator for 1-term point multiplication by choosing D so that τNAF/JSF conversions are computed slightly faster than the other operations of window point multiplications. Hence, we selected D = 4 for

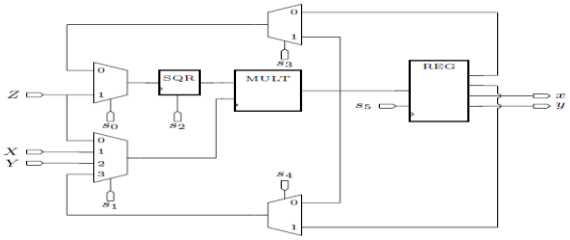


Figure 4. Postprocessor

the pre-processor, D = 13 for the main processor, and D = 3 for the postprocessor. The postprocessor was selected to be faster than the τNAF/JSF converter in order to ensure that inversions do not become the bottleneck.

VII. Results and Comparisons

The architecture was described in VHDL and synthesized for Xilinx vertex-5 FPG with ISE 9.1 design software. Table 3 presents the area consumption of the generator and its components as given by the design software. The main processor expectedly dominates in the area consumption.

Computation times are presented in Table 4. The computation time is the time in which the generaor computes a single operation with an average H(k) when the pipeline is empty, i.e. when no wait delays occur. In all cases throughput is bounded by the main processor.

Our generator is clearly faster than most published implementations, although comparisons are difficult between different FPGAs.

The superiority of Koblitz curves over general curves is evident. The fastest general curve implementation using the same field size (m = 163) was recently presented in [6] and it achieves computation time 19.55μs and throughput 51,120 ops in a Virtex-4 FPGA. These values are inferior to our accelerator.

TABLE 3. Area Consumption

Component	***LUTs***	***FFs***	***RAMs***
Converter	5,238	3,543	7
Pre-processor	2,934	1,543	14
Main processor	15,073	10,978	0
Postprocessor	2,857	2,984	0

TABLE 4. Latencies

Operations	***Latency(clock cycles)***
Conversion, w-Tnaf	***500***
Conversion, tJSF	***500***
Pre-computation, w=4	***18M + 300***
Pre-computation, n=2	***13M + 414***
Pre-processor	***16M + 656***
Main processor	***2H(k)(M + 1) + l + 6***
Postprocessor	***11M + 175***

VIII. Conclusions

We described an FPGA-based very fast elliptic curve cryptography public key generator on NIST K-163 Koblitz curve. The generator utilizes window methods and multiple point multiplications. It uses dedicated processing units for different parts of the algorithms and supports pipelined computation.

References

[1] N. Koblitz, Elliptic curve cryptosystems, Math. Comput 48 (1987) 203–209.

[2] J. Goodman and A. Chandrakasan. An energy-efficient reconfigurable public-key cryptography processor. IEEE J. Solid-State Circuits, 36(11):1808–1820, Nov. 2001.

[3] N.Koblitz. CM-curves with good cryptographic properties. In Advances in Cryptology, CRYPTO '91, volume 576 of lecture notes in Computer Sci., pages 279-287. Springer 1991.

[4] C. Candolin, J. Lundberg, and H. Kari. Packet level Authentication in military networks. In Proc. 6th Australian Information Warfare & IT Security Conf., Geelong, Australia, Nov. 2005.

[5] K. J¨arvinen and J. Skytt¨a. On parallelization of high-speed processors for elliptic curve cryptography. IEEE Trans. VLSI. pp. 109 -118. Springer 2008.

[6] K. J¨arvinen, J. Forsten, and J. Skytt¨a. FPGA design of selfcertified signature verification on Koblitz curves. In Cryptographic Hardware and Embedded Systems CHES 2007, volum4727 of Lecture Notes in Computer Sci. pages 256-271. Springer 2007.

[7] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. Inform.Comput.,78(3):171–177,Sept.1988.

[8] K. J¨arvinen and J. Skytt¨a. Fast point multiplication on Koblitz curves: Parallelization method and implementations. Microproc. Microsyst, Vol. 33. pp. 106-116. Springer 2009.

[9] E. Al-Daoud, R. Mahmod, M. Rushdan, A. Kilicman, A new addition formula for elliptic curves over GF(2n), IEEE Trans. Comput. 51 (8) (2002) 972–975.

[10] V. Miller, Use of elliptic curves in cryptography, in: Advances in Cryptology, CRYPTO'85, Lecture Notes in Computer Sci., pp. 417-426. Springer1986. .

[11] J. A. Solinas. Efficient arithmetic on Koblitz curves. Des. Codes Cryptography, 19(2–3):195–249, 2000.

[12] J. López, R. Dahab, Improved algorithms for elliptic curve arithmetic in GF(2n), in: Selected Areas in Cryptography, SAC'98, Lecture Notes in Computer Science, vol. 1556, Springer, 1998, pp. 201-227.