Low Power Elliptic Curve Cryptography

Maurice Keller[⋆] and William Marnane

Dept. of Electrical and Electronic Engineering
University College Cork, Cork City, Ireland
{mauricek,liam}@rennes.ucc.ie

Abstract. The designer of an elliptic curve processor is faced with many design choices that include the algorithm and coordinate system to be used. The power consumption of elliptic curve processors is becoming increasingly important as such processors find new uses in power constrained environments. This paper studies the effect that algorithm and coordinate choices have on the power consumption and energy per point multiplication of an FPGA based, reconfigurable elliptic curve processor.

1 Introduction

Elliptic curve cryptography has become a popular choice for implementing public key cryptosystems due to its ability to offer a similar level of security to traditional systems, such as RSA, but with smaller key lengths [1]. The smaller key length results in smaller memory and bandwidth requirements for elliptic curve cryptosystems. This makes them suitable for use in resource constrained environments which require security protocols, such as wireless sensor networks.

Elliptic curve point scalar multiplication is the computation of $Q = [k]P$, where Q and P are points on the elliptic curve E defined over $GF(p^m)$ and k is a scalar. The security of elliptic curve cryptosystems is based on the elliptic curve discrete logarithm problem (ECDLP). This is the problem of determining k given P and Q. This problem is deemed computationally infeasible for an appropriate choice of system parameters such as those defined in [2].

The efficiency of an elliptic curve cryptosystem depends on the efficiency with which $Q = [k]P$ can be performed. Since elliptic curves were first proposed for use in cryptography by Koblitz [3] and Miller [4] many different algorithms for computing $Q = [k]P$ have appeared in the literature. It is also possible to use different coordinate systems to represent an elliptic curve point P. The choice of coordinates effects the performance of an elliptic curve cryptosystem. The finite field over which the elliptic curve is defined also effects the efficiency of the system. Finite fields of characteristic $p = 2$ are a popular choice as arithmetic over these fields can be performed efficiently in hardware.

⋆ This research was funded by the Embark Initiative Postgraduate Research Scholarship Scheme from the Irish Research Council for Science, Engineering and Technology (IRCSET).

Until recently most research on elliptic curve implementations focused on minimising the time per point multiplication. Recently, however, several implementations aimed at minimising power consumption for applications in constrained environments have appeared [5,6]. An ASIC based low power elliptic curve digital signature chip was presented in [7]. All of these ASIC based architectures use one particular algorithm and coordinate system to perform the point scalar multiplication. In [8] a low power elliptic curve processor based on an Atmel 8-bit microprocessor and a dedicated ASIC coprocessor to perform field multiplications is presented.

This work studies the effect that algorithm and coordinate choice have on the power consumption of a reconfigurable $GF(2^m)$ elliptic curve processor implemented on an FPGA, in order to identify the set of choices which minimises the power and energy consumption of the processor.

2 Characteristic 2 Finite Fields

The finite field $GF(2^m)$ consists of 2^m elements. In this work a polynomial basis is used to represent elements of $GF(2^m)$ i.e. $a = \sum_{i=0}^{m-1} a_i x^i \in GF(2^m),\ a_i \in \{0,1\}$. Addition of two elements is simply bitwise XORing of the elements. Multiplication, squaring and division is performed modulo an irreducible polynomial $f(x)$. The field is closed under the operations of addition, multiplication, squaring and division. The security of an elliptic curve cryptosystem depends on the size of the underlying field $GF(2^m)$ i.e. the size of m. For this work the field size $m = 163$ is used which is considered equivalent to 1024-bit RSA [9]. However, the processor presented in Section 4.2 can be used for any field size m.

3 Elliptic Curves

A non-supersingular elliptic curve over $GF(2^m)$ is defined by the equation:

$$E(GF(2^m)) : y^2 + xy = x^3 + ax^2 + b,\ b \neq 0 \tag{1}$$

An elliptic curve point $P \in E$ is defined as the pair of elements of $GF(2^m)$ (x, y) which satisfy equation (1). The points on E form a commutative finite group under the point addition operation. The special point φ, known as the point at infinity, is the additive identity of the group. Addition of two points on E is performed using the well known “chord and tangent” process [1]. The underlying operations used to perform point addition are $GF(2^m)$ arithmetic operations. Point doubling is a special case of point addition where the two input points are the same. Elliptic curve point scalar multiplication is performed by repeated point additions and doublings i.e. $Q = [k]P = P + P + \ldots + P$.

3.1 Coordinate System

Elliptic curve points can be represented using different coordinate systems. Affine coordinates, as seen in the previous section, represent each point using two

$GF(2^m)$ elements. Addition and doubling of affine points can be described by a set of algebraic equations [1]. Using these equations point addition can be performed in $1M + 1D + 1S + 8A$ (where M,D,S,A refer to $GF(2^m)$ multiplication, division, squaring and addition respectively). Similarly a point doubling can be performed in $1M + 1D + 1S + 6A$. Addition is simply bitwise XOR of the two operands and it can be implemented virtually for free (one clock cycle) in hardware relative to the other operations. Therefore the additions are neglected when discussing the cost of point operations.

A projective elliptic curve point $P = (X, Y, Z)$ consists of three elements of $GF(2^m)$. To convert the affine point (x, y) to projective coordinates one simply sets $Z = 1$ i.e. $(x, y, 1)$. There are several different types of projective coordinates and they differ in how the projective point maps to an affine point. In an elliptic curve cryptosystem the result of a point scalar multiplication will usually need to be transmitted between two parties. As affine points are represented in $2m$ bits (and can be compressed further [10]) they are preferred for transmission. Therefore, projective coordinates are generally used for internal computations but the resultant projective point is converted to an affine point before being transmitted.

A Jacobian projective point $P = (X, Y, Z)$ maps to the affine point $P = (X/Z^2, Y/Z^3)$. Algebraic equations for Jacobian point addition and doubling can be found in [1]. Jacobian point addition and doubling can be performed in $15M + 5S$ and $5M + 5S$ respectively. The cost of converting a Jacobian point to an affine point is $3M + 1D + 1S$. When performing an elliptic curve point scalar multiplication it can be assumed that the input point P will always be in affine coordinates. Therefore when converted to Jacobian coordinates it will have $Z = 1$. In the special case that one of the input points to Jacobian point addition has $Z_1 = 1$ then the cost of addition can be reduced to $11M + 4S$. A further improvement to this can be obtained for the special case that the elliptic curve parameter $a = 1$. This saves $1M$ from the cost of Jacobian addition.

A Lopez-Dahab projective point $P = (X, Y, Z)$ maps to the affine point $P = (X/Z, Y/Z^2)$. This coordinate system and the corresponding equations for point addition and doubling are defined in [11]. Point addition and doubling cost $10M + 4S$ and $5M + 5S$ in the case $Z_1 = 1$. The cost of conversion back to affine coordinates is $2M + 1D + 1S$. The special case of $a = 0$ or 1 provides a saving of $1M$ for both point addition and doubling. Using the formula described in [12] a further $1M$ can be saved from the point addition at the cost of $1S$.

For this work the NIST pseudo-random curve over $GF(2^{163})$ [2] was used. Therefore $a = 1$. Table 1 summarises the cost of elliptic curve point addition and doubling for the various coordinate systems.

For both types of projective coordinates described here only one division is required at the end of a point scalar multiplication to convert the result back to affine coordinates. In Appendix $A.4.4$ of [10] an algorithm for computing the inverse of an element of $GF(2^m)$ using repeated multiplications and squarings is described. For the field $GF(2^{163})$ the inversion can be implemented in $9M$ and $162S$. In a hardware elliptic curve processor using projective coordinates this

Table 1. Cost of Point Operations $a = 1$

Coordinates	Addition	Doubling	Conversion	Conv. no divider
Affine	1M+1D+1S	1M+1D+1S	-	-
Jacobian ($Z_1 = 1$)	10M+4S	5M+5S	3M+1D+1S	12M+163S
Lopez-Dahab	8M+5S	4M+5S	2M+1D+1S	11M+163S

Algorithm 1. Binary Double and Add Point Multiplication

```
Input:      $P \in E(GF(2^m)), k = \sum_{i=0}^{t} k_i 2^i$
Output:     $[k]P$
Initialise: 1. $Q = P$
Run:        2. for $i = t - 1$ downto 0 do
            3.    $Q = [2]Q$          /* Point Double */
            4.    if($k_i = 1$) then
            5.       $Q = Q + P$      /* Point Addition */
            6.    end if
            7. end for
Return:     8. $Q = [k]P$
```

algorithm can be utilised to remove the need for a divider, hence reducing the area footprint of the processor.

3.2 Point Scalar Multiplication Algorithms

Computing $Q = [k]P$ is a special case of the general problem of exponentiation in Abelian groups and the shortest addition chain problem for integers. This problem can be stated as: starting from the integer 1, and at each step computing the sum of two previous results, what is the minimum number of steps required to reach k?

The simplest method for computing $[k]P$ is the binary double and add method [1] presented in Algorithm 1. Algorithm 1 iterates through the binary expansion of k. On each iteration a point doubling is performed. If $k_i = 1$ then a point addition is performed. In general a binary expansion of k will have approximately m bits. On average half of these bits will be equal to one. Therefore the cost of implementing a point scalar multiplication using Algorithm 1 is given in equation (2).

$$N_{binary} = (m-1)N_{double} + (m/2 - 1)N_{add} \tag{2}$$

The basic binary double and add algorithm can be improved upon by using an addition-subtraction algorithm. In this case a signed digit expansion of $k = \sum_{i=0}^{l} s_i 2^i,\ s_i \in \{-1, 0, 1\}$ is used. Non-adjacent form (NAF) is a signed digit representation in which there are no adjacent non-zero digits. NAF has the fewest non-zero coefficients of any signed binary expansion of k. The NAF of a non-negative integer given in binary representation can be computed using

Algorithm 2. NAF Addition-Subtraction Chain Point Multiplication

```
Input:         $P \in E(GF(2^m)), k = \sum_{i=0}^{l} s_i 2^i$
Output:        $[k]P$
Initialise: 1. $Q = P$
Run:        2. for $i = l - 1$ downto 0 do
            3.    $Q = [2]Q$           /* Point Double */
            4.    if($s_i = 1$) then
            5.       $Q = Q + P$       /* Point Addition */
            6.    if($s_i = -1$) then
            7.       $Q = Q - P$       /* Point Subtraction */
            8.    end if
            9. end for
Return:    10. $Q = [k]P$
```

Algorithm $IV.5$ in [1]. It was shown in [13] that the expected weight of an NAF of length l is $l/3$.

Algorithm 2 presents the point scalar multiplication algorithm based on an NAF expansion of k. When $s_i = -1$ an elliptic curve point subtraction is performed. Point subtraction is performed by adding the negative of the point i.e. $P_0 - P_1 = P_0 + (-P_1)$. The negative of an affine elliptic curve point $P = (x, y)$ is given by $-P = (x, x + y)$. The negative of a projective point $P = (X, Y, Z)$ is given by $-P = (X, XZ + Y, Z)$. The cost of implementing a point scalar multiplication using Algorithm 2 is given in equation (3).

$$N_{NAF} = (l - 1)N_{double} + (l/3 - 1)N_{add} \tag{3}$$

Montgomery [14] proposed a different method for computing $[k]P$ which maintains the relationship $P_2 - P_1$ as invariant. Montgomery's method is presented in Algorithm 3. Algorithm 3 performs an addition and a doubling on each iteration of the loop regardless of the value of k_i. The advantage of Montgomery's method is that it only operates on the x coordinate in affine coordinates and the X and Z coordinates in projective coordinates. After the loop the y coordinate of the resultant point can be computed. The cost of Montgomery's method is given in equation (4). Algebraic equations for implementing Montgomery's method in both affine and Lopez-Dahab projective coordinates are given in [15].

$$N_{Montgomery} = N_{double} + (m - 1)N_{loop} + N_{compute\ y} \tag{4}$$

4 Hardware Implementation

4.1 $GF(2^m)$ Components

As described in Section 2, $GF(2^m)$ addition is simply bitwise XOR of the elements and so can be implemented in hardware using m XOR gates with a latency of one XOR gate delay. $GF(2^m)$ multiplication is implemented using the digit-serial algorithm described in [16]. This architecture performs a multiplication

Algorithm 3. Montgomery Point Multiplication

```
Input:           $P \in E(GF(2^m)), k = \sum_{i=0}^{t} k_i 2^i$
Output:          $[k]P$
Initialise: 1.   $P_1 = P$; $P_2 = [2]P$
Run:        2.   for $i = t - 1$ downto 0 do
            4.       if($k_i = 1$) then
            5.           $P_1 = P_1 + P_2$; $P_2 = [2]P_2$
            6.       else
            7.           $P_2 = P_2 + P_1$; $P_1 = [2]P_1$
            8.       end if
            9.   end for
            10.  compute $y_1$
Return:     11.  $Q = P_1 = [k]P$
```

in $n = \lceil \frac{m}{d} \rceil$ clock cycles, where d is the variable digit size of the architecture. A bit-parallel squaring architecture was used which has a purely combinational delay and computes the result in one clock cycle. Division is implemented using the division architecture described by Shantz in [17]. It computes the result in $2m$ clock cycles.

4.2 Reconfigurable Elliptic Curve Processor

Figure 1 shows the reconfigurable elliptic curve processor that was designed to implement the various algorithms described in Section 3 for any field size m. The processor is based on one each of the four $GF(2^m)$ computational units described in the previous section. The processor also contains two storage elements to store the m-bit inputs, outputs and variables required during the computation of $[k]P$. The computational units and memory are connected by an m-bit bi-directional data bus. To minimise power consumption each computational unit is only enabled when it is required to perform a calculation.

The processor is controlled by a Finite State Machine (FSM), counter and ROM. The ROM contains a sequence of 32-bit instructions which control the data transfer between the computational units and memory to perform the required operations to implement elliptic curve point addition and doubling, and conversion from projective to affine coordinates (if necessary). The FSM enables the counter to iterate through the ROM instructions. The counter contains several load pins. These allow the state machine to jump to various predefined locations within the ROM instruction set. This means that the ROM need only contain one set of instructions for doubling, adding and converting a point. The master counter tracks how many bits of the scalar k have been processed and indicates when the loop in the point scalar multiplication algorithm has been completed.

This processor architecture can be used to implement the various coordinates and algorithms described in Sections 3.1 and 3.2. The only changes required are to the instructions contained in ROM and to the FSM controller. The number of m-bit memory locations required also varies depending on the algorithm used.

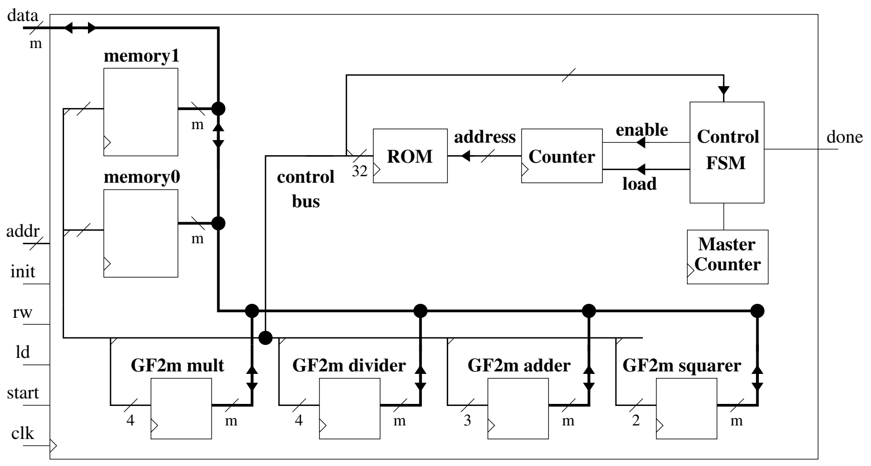

Fig. 1. Reconfigurable $GF(2^m)$ Elliptic Curve Processor

Table 2. Clock Cycles Per Stage

Algorithm	N_{Setup}	N_{Double}	N_{Add}	$N_{convert}$
Binary Affine	10	$2m + n + 44$	$2m + n + 48$	-
Binary Jacobian	12	$5n + 70$	$10n + 138$	$2m + 3n + 41$
Binary Jac no div	12	$5n + 70$	$10n + 138$	$12n + 823$
Montgomery Affine	$2m + 16$	$4m + 28$		$2m + 2n + 39$
Montgomery Lopez-Dahab	20	$6n + 79$		$10n + 111,\ D \leq 3$ $5n + 2m + 58,\ D > 3$
Montgomery L-D no div	20	$6n + 79$		$19n + 905$
NAF Affine	16	$2m + n + 44$	$2m + n + 48$	-
NAF Jacobian	18	$5n + 70$	$10n + 138$	$2m + 3n + 40$
NAF Jac no div	18	$5n + 70$	$10n + 138$	$12n + 819$

Table 2 details the clock cycles required for each stage of the computation for the various algorithm and coordinate combinations implemented. As mentioned in Section 3.1, only one division is required when using projective coordinates. To this end, for each of the algorithms using projective coordinates two processors were implemented, one with and one without a $GF(2^m)$ divider. In the case of no divider the division is implemented using repeated multiplications and squarings.

The total number of clock cycles given in Table 2 includes clock cycles for calculation as well as control overheads such as data transfer between the components. At the beginning of each computation of $[k]P$ a certain number of clock cycles are required to setup and initialise the processor, hence the setup stage (N_{setup}). The convert stage is required to convert the projective result into affine coordinates and, in the case of the processors based on the Montgomery method, to compute the y coordinate of the result.

5 Results

Each of the algorithms listed in Table 2 was implemented using the architecture described in the previous section. Two different digit sizes of the underlying $GF(2^m)$ multiplier were used. A digit size of $d = 1$ means that a $GF(2^m)$ division costs the same as two $GF(2^m)$ multiplications. A digit size of $d = 16$, giving a D/M ratio of thirty two, was also implemented. The designs were prototyped on a Xilinx Spartan3L $xc3s1000L$ $ft256-4$ FPGA using Xilinx ISE 9.1.01i. The memory required for each processor was implemented using the FPGA Block RAM resources. The power consumption of the elliptic curve processor for each different choice of algorithm was measured using XPower. The minimum post place and route clock frequency reported for all the designs over both digit sizes was $80MHz$. Therefore all the following results are for a clock frequency of $80MHz$. The results also assume an average value for the scalar k.

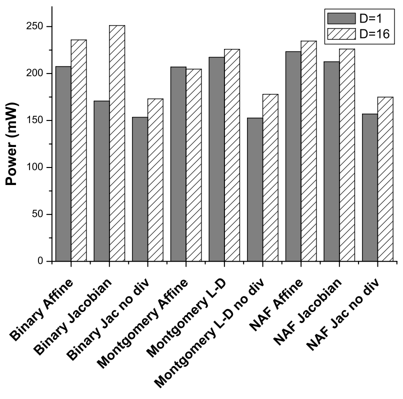

Fig. 2. Power Consumption of Various Algorithms

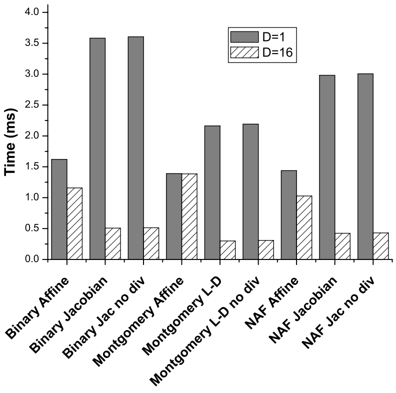

Fig. 3. Point Multiplication Time of Various Algorithms

Figure 2 illustrates the power consumption values of the various algorithms. It is noted at this point that these power values are meant to compare the effect of algorithm choice on power consumption. The architecture has not been optimised to minimise power consumption.

The minimum power consumption for a digit size of one was found to be for the Montgomery algorithm with Lopez-Dahab coordinates and no divider. For a digit size of sixteen the minimum power is reported for the Binary Algorithm using Jacobian coordinates and no divider. It is noted that removing the divider from the circuit reduces the power consumption of all the projective algorithms implemented. This is due to the reduced circuit area. The penalty is a slightly increased computation time as seen in Figure 3.

All of the algorithms implemented have different times per point multiplication. Therefore it is relevant to discuss the energy expended per point multiplication as illustrated in Figure 4. It can be seen that increasing the digit size from

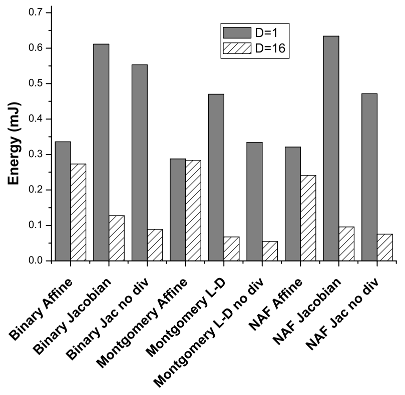

Fig. 4. Energy Usage of Various Algorithms per Point Multiplication

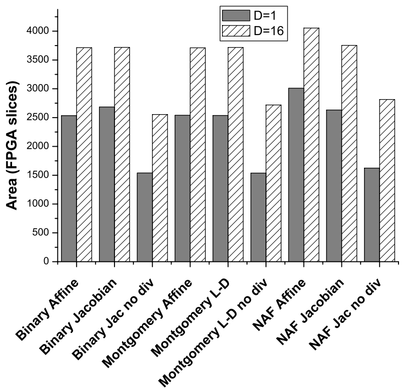

Fig. 5. Area of Various Algorithms

one to sixteen reduces the energy per point multiplication. This occurs because increasing the digit size reduces the computation time. The exception to this is the Montgomery affine processor which has virtually identical computation time and energy consumption for both digit sizes. This is because no multiplications are required in the dominant loop section of the algorithm (see Table 2).

The minimum energy for a digit size of one is $0.288mJ$ reported for the Montgomery affine processor. For $d = 16$ the minimum energy is $0.055mJ$ reported for the Montgomery Lopez-Dahab processor with no divider. These results coincide with the fastest processors for both digit sizes. This indicates that while power consumption does not vary dramatically for the different algorithms the time per point multiplication does. Therefore the algorithms with minimum computation time tend to have the minimum energy per point multiplication when implemented on an FPGA.

6 Conclusions

A reconfigurable $GF(2^m)$ elliptic curve processor architecture was presented in this paper. The architecture was used to implement several different algorithms and coordinates for elliptic curve point scalar multiplication. The effect of algorithm and coordinate choice on power and energy consumption of the processor implemented on an FPGA was then investigated. It was found that for small D/M ratio the Montgomery method with affine coordinates uses the least energy per point multiplication. For large D/M ratio the Montgomery method with Lopez-Dahab projective coordinates minimises the energy required to perform a point multiplication. Future work includes examining the energy consumption of additional methods for improving the performance of the point scalar multiplication. The architecture could also be optimised to minimise the absolute power and energy consumptions.

References

1. Blake, I.F., Seroussi, G., Smart, N.P.: Elliptic curves in cryptography. London Mathematical Society Lecture Note Series, 265, Cambridge University Press, (1999)
2. National Institute of Standards and Technology (NIST): Recommended elliptic curves for federal government use. NIST Special Publication (1999)
3. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48, 203–209 (1987)
4. Miller, V.: Uses of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
5. Ozturk, E., Sunar, B., Savas, E.: Low-power elliptic curve cryptography using scaled modular arithmetic. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 107–118. Springer, Heidelberg (2004)
6. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: An elliptic curve processor suitable for RFID-tags. Cryptology ePrint Archive, Report 2006/227 (2006)
7. Schroeppel, R., Beaver, C.L., Gonzales, R., Miller, R., Draelos, T.: A low-power design for an elliptic curve digital signature chip. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 366–380. Springer, Heidelberg (2003)
8. Bertoni, G., Breveglieri, L., Venturi, M.: Power aware design of an elliptic curve coprocessor for 8-bit platforms. In: Proceedings Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06), pp. 337–341. IEEE, Los Alamitos (2006)
9. National Institute of Standards and Technology (NIST): Recommendation for key management-part 1: general (Revised). NIST Special Publication 800–857 (2006)
10. IEEE P1363: Standard Specifications for Public Key Cryptography. IEEE Std 1363–2000 (2000)
11. Lopez, J., Dahab, R.: Improved algorithms for elliptic curve arithmetic in $GF(2^n)$. In: Tavares, S., Meijer, H. (eds.) SAC 1998. LNCS, vol. 1556, pp. 201–212. Springer, Heidelberg (1999)
12. Al-Daoud, E., Mahmod, R., Rushdan, M., Kilicman, A.: A new addition formula for elliptic curves over $GF(2^n)$. IEEE Transactions on Computers 51(8), 972–975 (2002)
13. Morain, F., Olivos, J.: Speeding up the computations on an elliptic curve using addition-subtraction chains. Theoretical Informatics and Applications 24, 531–543 (1990)
14. Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorisation. Mathematics of Computation 48, 243–264 (1987)
15. Lopez, J., Dahab, R.: Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
16. Song, L., Parhi, K.: Low energy digit-serial/parallel finite field multipliers. Kulwer Journal of VLSI Signal Processing Systems 19(2), 149–166 (1998)
17. Shantz, S.C.: From Euclid's GCD to Montgomery multiplication to the great divide. Technical Report TR-2001-95, Sun Microsystems (2001)