High Performance Elliptic Curve Point Operations with Pipelined GF(2^m) Field Multiplier

Turki F. Al-Somani¹ and Alaaeldin Amin²

¹Computer Engineering Department, Umm Al-Qura University, Makkah, Saudi Arabia
²Computer Engineering Department, King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia,

E-mail: ¹tfbsomani@uqu.edu.sa  ²amindin@kfupm.edu.sa

Abstract

This paper studies the effect of high performance pipelined GF(2^m) bit-serial multiplier on elliptic curve point operations. A 3-stage pipelined version of the Massy-Omura GF(2^m) normal basis multiplier for 160 ≤ m ≤ 256 was studied in terms of area overhead and throughput improvement. Simple gate area and delay models were used to estimate the throughput of the pipelined and the non-pipelined multipliers. The proposed pipelined architecture has been shown to have a significant improvement in throughput allowing a single 3-stage pipelined multiplier to have higher throughput than an architecture employing three parallel non-pipelined multipliers. The AT² performance metric has shown an even more significant improvement.

1. Introduction

Elliptic Curve Cryptosystems (ECCs) have been recently attracting increased attention [1]. Standards for ECCs have been adopted by IEEE, ANSI, NIST, SEC and WTLS. The ability to use smaller key sizes and the computationally more efficient ECC algorithms compared to those used in earlier public key cryptosystems such as RSA [2] and ElGamal [3] are two main reasons why ECCs are becoming more popular. They are considered particularly suitable for implementation on smart cards or mobile devices.

Since ECCs have been proposed in 1985, extensive research has been done on the underlying math, security strength and efficient implementations. Among the different fields that can underlie elliptic curves, prime fields GF(p) and binary fields GF(2^m) have shown to be best suited for cryptographic applications.

In particular, binary fields allow for fast computation in software as well as in hardware [4] [5].

Inversion operations are the most expensive operation over Finite Fields [6]. The approach adopted in the literature is to represent Elliptic Curve points in projective coordinate systems in order to replace the inversion operations with repetitive multiplications [6] [7]. Almost all recently reported ECC cryptoprocessors use projective coordinate systems. Several projective coordinate systems have been reported and are in common use [6] [7]. The selection of a particular projective coordinate system, however, has mostly been influenced by the number of arithmetic operations, mainly multiplications, required to perform elliptic curve point operations. This is expected due to the sequential nature of these architectures where a single multiplier is used.

For high performance servers, such sequential architectures are too slow to meet the ever increasing demand. For such servers, high-speed cryptoprocessors are becoming a necessity. One approach to meet this requirement is to exploit the inherent parallelism within Elliptic curve point operations in projective coordinate systems. Recently, several ECC cryptoprocessor architectures have been proposed where the choice of the projective coordinate system was influenced by the degree of its inherent parallelism [8 – 10]. The architectures proposed in [9] and [10] have reported using three parallel multipliers to efficiently compute point operations since, in such computations, multiplication is the most dominant operation and the most time consuming. These architectures have reported better (AT²) area-time complexity compared to other architectures that are based only on a single multiplier. Recently, Al-Somani et. al. [8] conducted a comparative study on sequential and parallel designs using different projective coordinate systems. Al-
Somani et al. analyzed the dataflow of point operations for each projective coordinate system by finding the critical path that has the lowest number of the multiplication operations. Accordingly, the number of multipliers needed to meet this critical path is found.

This paper studies the effect of a proposed a high-performance pipelined GF(2^m) multipliers on elliptic curve point operations. The rest of this paper is organized as follows: Section 2 gives a brief introduction to ECCs. Section 3 describes the proposed pipelined multiplier. Section 4 provides performance metrics for the pipelined versus the non-pipelined multipliers regarding their area overhead and system throughput. Section 4 also presents comparisons between the two multiplier architectures for different projective coordinate systems are also presented. Finally, Section 5 concludes this work.

2. Elliptic Curve Preliminaries

Elliptic curve cryptosystems, originally proposed by Niel Koblit and Victor Miller in 1985 [1], are considered as viable alternative to RSA cryptosystems but with much shorter key sizes. An ECC with key size of 128-256 bits has been shown to offer equal security to that of RSA with key size of 1-2K bits [1]. To date, no significant breakthroughs have been reported in determining weaknesses in the ECCs, which are based on the discrete logarithm problem over points on an elliptic curve. The fact that the problem appears so difficult to crack means that key sizes can be considerably reduced allowing ECC to challenge the dominating position enjoyed thus far by the RSA cryptosystems. Because of their short key sizes, ECC have gained popularity and are considered particularly suitable for implementation on smart cards and mobile devices.

An elliptic curve over the finite field GF(p) defined by the parameters a, b \( \in \) GF(p) with p > 3, consists of the set of points P = (x, y), where x, y \( \in \) GF(p), that satisfy the equation:

\[
y^2 = x^3 + ax + b
\]  

(1)

where a, b \( \in \) GF(p) and \( 4a^3 + 27b^2 \neq 0 \) mod p, together with the point at infinity \( O \) which is the additive identity of the group [1]. The number of points \( #E \) on an elliptic curve over a finite field GF(q=pm) is defined by Hasse’s theorem [4]. The set of discrete points on an elliptic curve forms an abelian group, whose group operation is known as point addition. Elliptic curve point addition is defined according to the “chord-tangent process”. Point addition over GF(p) is described as follows:

Let P and Q be two distinct points on an elliptic curve E defined over GF(p) with \( Q \neq -P \) (Q is not the additive inverse of P). The addition of the two points P and Q is the point R (R = P + Q), where R is the additive inverse of S, with S being the third point on E intercepted by the straight line through points P and Q.

The additive inverse of a point P = (x, y) is the point \( -P = (x, -y) \) which is the reflection of the point P with respect to the x-axis on E. When \( P = Q \) and \( P \neq -P \) the addition of P and Q is the point R (R = 2P), where R is the additive inverse of S with S being the third point on E intercepted by the straight line tangent to the curve at point P. This operation is referred to as point doubling.

The finite field GF(2^m) is of particular importance in cryptography since it leads to efficient hardware implementations. Elements of the field are represented in terms of a basis. Most implementations use either a Polynomial Basis or a Normal Basis [5]. Let GF(2^m) be a finite field of characteristic two. A non-supersingular elliptic curve E over GF(2^m) is defined to be the set of solutions (x, y) \( \in \) GF(2^m) \times GF(2^m) to the equation,

\[
y^2 + xy = x^3 + ax^2 + b
\]  

(2)

where a and b \( \in \) GF(2^m), b \( \neq 0 \), together with the point at infinity.

It is well known that E forms a commutative finite group, with O being the group identity, under the addition operation. Explicit formulas for the addition rule involve several field arithmetic operations (addition, squaring, multiplication and inversion) in the underlying finite field. The group operation in affine coordinate system involves finite field inversion, which is a very costly operation, particularly over prime fields. Projective coordinate systems are used to eliminate the need for performing inversion. Several projective coordinate systems have been proposed in the literature including homogeneous, Jacobian, and Lopez-Dahab coordinate system [6] [7]. For elliptic curve defined over GF(2^m), many different forms of formulas may be used for point addition and doubling. For the Homogeneous coordinate system, an elliptic curve point \( (x, y) \) takes the form \( (X/Z, Y/Z) \), while for the Jacobian coordinate system, a point takes the form \( (x, y) = (X/Z^2, Y/Z^3) \). The Lopez-Dahab coordinate system, on the other hand, takes the form \( x, y = (X/Z, Y/Z^2) \).

Adding a point P on the elliptic curve E to itself a number of times \( k \) is known as the scalar product (kP) of point P by the scalar k. Scalar multiplication is a basic operation for ECCs which, in the group of points of an elliptic curve, is analogous to exponentiation in the multiplicative group of integers modulo a fixed integer m. The scalar multiplication operation (kP)
yields a point on the elliptic curve which is the result of adding point $P$ to itself $k$ times. Several scalar multiplication methods have been proposed in the literature [12]. Computing $kP$ can be done using a straightforward binary method, the double-and-add method, based on the binary expression of the multiplier $k$. Computing $kP$ using the binary method is described as follows:

Let $k = (k_{m_{R1}}, ..., k_0)$, where $k_{m_{R1}}$ is the most significant bit of $k$, be the binary representation of $k$. The multiplier $k$ can be written as:

$$k = \sum_{0 \leq i \leq m_{R1}} k_i 2^i = k_{m_{R1}}2^{m_{R1}} + k_{m_{R2}}2^{m_{R2}} + \cdots + k_1 2 + k_0$$ (3)

Using the Horner expansion, $k$ can be rewritten as:

$$k = ((...(k_{m_{R1}}2^2 + k_{m_{R2}})2 + \cdots + k_1)2 + k_0)$$ (4)

Accordingly, $kP$ can be written as:

$$kP = 2(2(2(k_{m_{R1}}P + k_{m_{R2}})P + \cdots + k_1P) + k_0P)$$ (5)

The binary method algorithm is shown below:

**Algorithm 1 The Binary Algorithm: most-to-least version**

1. input $P$, $k$
2. $Q \leftarrow P$
3. for $i$ from $m$-2 downto 0 do
   3.1. $Q \leftarrow 2Q$
   3.2. if $k_i = 1$ then $Q \leftarrow Q + P$
4. end for
5. output $Q$

The binary scalar multiplication method is the most straightforward scalar multiplication method. It inspects the bits of the scalar multiplier $k$, if the inspected bit $k_i = 0$, only a point doubling operation is performed. If, however, the inspected bit $k_i = 1$, both a point doubling and a point addition operations are performed. The binary method requires $m$ point doublings and an average of $\frac{m}{2}$ point additions.

### 3. The Proposed Pipelined Multiplier

Instead of using parallel multipliers to improve performance, in this work we study the effect of using a pipelined multiplier to increase the system throughput and hence overall performance. This approach saves the area taken by the parallel multipliers at the cost of a slight area increase due to the added pipeline registers. This, however, is minimized in the proposed approach since only 3-pipeline stages are proposed with only two added register stages required. The multiplier used here is the Massey-Omura normal basis bit-serial multiplier [14] since it can be used with both types of the optimal normal bases (type I and type II) [16]. For a reasonable level of security, we will investigate private key sizes ($m$) in the range $160 \leq m \leq 256$. Results obtained, however, can be easily extended to larger key sizes.

Massey and Omura [14] have proposed an efficient normal basis bit-serial multiplier over $GF(2^m)$. The Massey-Omura multiplier requires only two $m$-bit cyclic shift registers and combinational logic. The combinational logic consists of a set of AND and XOR logic gates. The first implementation of the Massey-Omura multiplier was reported in [15]. The area complexity of the Massey-Omura multiplier is $(2m - 1)$ AND gates + $(2m - 2)$ XOR gates, while the time complexity is $T_A + \left[1 + \log_2(m-1)\right]T_X$, where $T_A$ and $T_X$ are the delay of one AND gate and one XOR gate respectively.

Figure 1 shows an example of a $GF(2^5)$ Massey-Omura bit-serial field multiplier. The combinational logic of the multiplier (Figure 1) consists of one level of AND logic gates and four levels of XOR logic gates. In general, the combinational logic of the Massey-Omura bit-serial multiplier over $GF(2^m)$ consists of one level of $(2m - 1)$ AND logic gates and $\left[1 + \log_2(m-1)\right]$ levels of XOR logic gates. For the range of $m$ considered in this work, $160 \leq m \leq 256$, the number of levels of XOR logic gates is 9 levels. Thus, the proposed pipelined Massey-Omura multiplier will be partitioned into three stages. The first stage (Stage0) contains one level of AND logic gates and three levels of XOR logic gates. The remaining two stages (Stage1 & Stage2), on the other hand, contain only three levels of XOR logic gates as illustrated in Figure 2.

![Figure 1: GF(2^5) bit-serial Massey-Omura multiplier.](image-url)
4. Performance Analysis

To compare the proposed pipelined multiplier with original non-pipelined Massey-Omura multiplier in terms of area, we use an area metric corresponding to the sum of individual element estimated areas. The estimated area of various elements correspond to the area of an equivalent number of simple logic gates. A simple logic gate is assumed to have a unit area of 1U. An AND logic gate is considered as one simple logic gate with one unit area 1U. An XOR logic gate has a typical area of two simple logic gates, i.e. 2U, while a Latch area is equivalent to four simple logic gates, or 4U. Accordingly, the area of the non-pipelined Massey-Omura multiplier of \((2^m - 1)\) AND gates + \((2^m - 2)\) XOR gates = \((2^m - 1) \times 1U + (2^m - 2) \times 2U = (6m - 5)U\).

Ignoring the wiring cost, the Latches that are inserted between the stages constitute the main area overhead. We estimate the required number of Latches between the stages, for a value of that \(m=256\) bits. Accordingly, the first level of logic gates consists of 511 AND gates. The first level of XOR logic gates, on the other hand, will have 255 XOR gates. Each of the remaining levels of XOR gates will contain half the number of XOR gates in the preceding level. This implies that a 64-bit register is required between Stage_0 & Stage_1 and an 8-bit register is required between Stage_1 & Stage_2. Thus, the area of the proposed pipelined multiplier is \((2m -1)\) AND gates + \((2m - 2)\) XOR gates + 72 register bits which is equivalent to an area metric of \((6m + 283)U\). Table 3 lists the area of the proposed pipelined multiplier and the non-pipelined Massey-Omura multiplier.

Table 1: Area Metrics of the proposed pipelined multiplier and the non-pipelined Massey-Omura multiplier.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Area (gates)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Proposed Pipelined</td>
<td>((6m + 283)U)</td>
</tr>
<tr>
<td>Non-Pipelined Massey-Omura</td>
<td>((6m - 5)U)</td>
</tr>
</tbody>
</table>

To compare the proposed pipelined multiplier with the original non-pipelined Massey-Omura multiplier in terms of throughput, we compute the clock period for both designs. Delay calculations are normalized in terms of the delay of a simple logic gate \(\tau\). AND gates are assumed to be a simple logic gate of delay \(\tau\), while the delay of an XOR gate is equal to \(2\tau\). Assumptions made regarding the relative delay and area figures have been verified through the study of several CMOS standard cell libraries for the concerned gates assuming \(1x\) drive. The Massey-Omura multiplier non-pipelined data path delay is given by \(T_A + [1 + \log_2(m-1)]T_X\). Accordingly, the clock period in this case for \(m=256\), is \((1 \tau) + ((1 + 8) \times 2 \tau) = 19 \tau\). For the proposed pipelined multiplier, on the other hand, the clock period is limited by the delay of the slowest stage, Stage_0 in this case, which is equal to \(T_A + 3T_X\) or \(7 \tau\).

Table 2: Number of Multiplications in Different Coordinate Systems [8]

<table>
<thead>
<tr>
<th>Projective Coordinates (Pr)</th>
<th>Jacobian Coordinates (J)</th>
<th>Lopez-Dahab Coordinates (L-D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Doubl</td>
<td>Add</td>
</tr>
<tr>
<td>16M</td>
<td>7M</td>
<td>15M</td>
</tr>
<tr>
<td>5M</td>
<td>14M</td>
<td>5M</td>
</tr>
</tbody>
</table>

To compute the scalar multiplication, each loop iteration of the basic scalar multiplication algorithm (Algorithm 1) performs one point doubling operation and an average of \(1/2\) point addition operation. Table 2 shows the required number of multiplications for the two types of point operations, i.e. point addition and point doubling, for different coordinate systems [8]. Following the approach detailed in [8] and considering various coordinate systems, the average number of multiplications per a single loop iteration of the basic scalar multiplication algorithm is given in Table 3 for cases where a single multiplier, 2 parallel multipliers, 3 parallel multipliers, and 4 parallel multipliers are available.

Table 4 compares the minimum clock period required for the proposed pipelined multiplier with that of the non-pipelined Massey-Omura multiplier. In this table, P and Non-P refer to the Pipelined and Non-Pipelined multiplier respectively. The comparison in Table 4 are based on the average number of multiplication cycles provided in Table 3.

---

1 Three libraries of different technologies were considered including TSMC 0.25µ technology, the AMS 0.8µ and the ES2 ECPD07 process.
The results of Table 4 show that using 1 pipelined multiplier provides higher throughput than using 3 non-pipelined multipliers in all coordinate systems. It also shows that using two pipelined multipliers provide higher throughput than using four non-pipelined multipliers. Furthermore, Table 5 compares the proposed pipelined multiplier with the non-pipelined Massey-Omura multiplier. The results of Table 5, as with Table 4, show that using 1 pipelined multiplier provide better $AT^2$ than using 3 non-pipelined multipliers and using 2 pipelined multiplier provide better $AT^2$ than using 4 non-pipelined multipliers, which represents better than 50% improvement.

5. Conclusion

In this paper, the effect of a high performance pipelined GF($2^{256}$) bit-serial multiplier on elliptic curve point operations has been investigated. A 3-stage pipelined version of the Massy-Omura GF($2^m$) normal basis multiplier for $160 \leq m \leq 256$ was studied in terms of area overhead and throughput improvement. The proposed pipelined architecture has been shown to have a significant improvement in throughput allowing a single 3-stage pipelined multiplier to have higher throughput than an architecture employing three parallel non-pipelined multipliers. The $AT^2$ performance metric has shown an even more significant improvement.

References

### Table 3: Multiplication cycles for the coordinate systems using the non-pipelined Massey-Omura multiplier [8].

<table>
<thead>
<tr>
<th>Coord System</th>
<th>Average No. of Multiplication cycles</th>
<th>1 Multiplier</th>
<th>2 Multipliers</th>
<th>3 Multipliers</th>
<th>4 Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pr</td>
<td></td>
<td>15</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>J</td>
<td></td>
<td>12.5</td>
<td>7</td>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td>L-D</td>
<td></td>
<td>12</td>
<td>6.5</td>
<td>5.5</td>
<td>5</td>
</tr>
</tbody>
</table>

### Table 4: Minimum clock period for the pipelined versus the non-pipelined Massey-Omura multiplier in different coordinate systems.

<table>
<thead>
<tr>
<th>Coord System</th>
<th>Time (gate time)</th>
<th>1 Mult.</th>
<th>2 Mults</th>
<th>3 Mults</th>
<th>4 Mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>15τ</td>
<td>285τ</td>
<td>7τ</td>
<td>12τ</td>
<td>7τ</td>
</tr>
<tr>
<td>P</td>
<td>12.5τ</td>
<td>237.5τ</td>
<td>49τ</td>
<td>133τ</td>
<td>35τ</td>
</tr>
<tr>
<td>P</td>
<td>12τ</td>
<td>228τ</td>
<td>45.5τ</td>
<td>123.5τ</td>
<td>38.5τ</td>
</tr>
</tbody>
</table>

### Table 5: Comparisons results with different key sizes using the AT² performance metric.

<table>
<thead>
<tr>
<th>m (bits)</th>
<th>Coordinate System</th>
<th>AT²</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 Multiplier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P</td>
</tr>
<tr>
<td>160</td>
<td>Projective</td>
<td>13704075</td>
</tr>
<tr>
<td></td>
<td>Jacobian</td>
<td>9516719</td>
</tr>
<tr>
<td></td>
<td>Lopez-Dahab</td>
<td>8770608</td>
</tr>
<tr>
<td>180</td>
<td>Projective</td>
<td>15027075</td>
</tr>
<tr>
<td></td>
<td>Jacobian</td>
<td>10435469</td>
</tr>
<tr>
<td></td>
<td>Lopez-Dahab</td>
<td>9617328</td>
</tr>
<tr>
<td>200</td>
<td>Projective</td>
<td>16350075</td>
</tr>
<tr>
<td></td>
<td>Jacobian</td>
<td>11354219</td>
</tr>
<tr>
<td></td>
<td>Lopez-Dahab</td>
<td>10464048</td>
</tr>
<tr>
<td>220</td>
<td>Projective</td>
<td>17673075</td>
</tr>
<tr>
<td></td>
<td>Jacobian</td>
<td>12272969</td>
</tr>
<tr>
<td></td>
<td>Lopez-Dahab</td>
<td>11310768</td>
</tr>
<tr>
<td>256</td>
<td>Projective</td>
<td>20054475</td>
</tr>
<tr>
<td></td>
<td>Jacobian</td>
<td>13926719</td>
</tr>
<tr>
<td></td>
<td>Lopez-Dahab</td>
<td>12834864</td>
</tr>
</tbody>
</table>
Figure 3: $AT^2$ Comparison results for different key sizes.