Efficient utilization of scalable multipliers in parallel to compute GF(p) elliptic curve cryptographic operations

ADNAN ABDUL-AZIZ GUTUB

King Fahd University of Petroleum & Minerals; Computer Engineering Department; Dhahran 31261; Saudi Arabia; Email: gutub@kfupm.edu.sa

ABSTRACT

This paper presents the design and implementation of an elliptic curve cryptographic core to realize point scalar multiplication operations used for the GF(p) elliptic curve encryption/decryption and the elliptic curve digital signature algorithm (ECDSA). The design makes use of projective coordinates together with scalable Montgomery multipliers for data size of up to 256-bits. We propose using four multiplier cores together with the ordinary projective coordinates which outperform implementations with Jacobean coordinates typically believed to perform better. The proposed architecture is particularly attractive for elliptic curve cryptosystems when hardware area optimization is the key concern.

Keywords: crypto hardware designs; elliptic curve cryptography; modulo arithmetic; projective coordinate cryptosystems; scalable multipliers

INTRODUCTION

A number of Elliptic Curve Cryptographic (ECC) schemes have been proposed based on the discrete logarithm problem over points on an elliptic curve as originated by Miller (1986) and Koblitz (1987). Although ECC is around 20 years old, its reliability is still suspect with no significant breakthrough in determining weaknesses in the algorithm (Blake et al. 1999, Raju & Akbani 2003). In fact, the ECC problem appears very difficult to crack, implying that key sizes can be shortened considerably and the search space reduced exponentially, particularly when compared to the key size used by other cryptosystems. This makes ECC a promising practical replacement to *RSA* (Rivest Shamir Adleman crypto system), one of the most popular public key methods known (Rivest *et al.* 1978). ECC offers the same level of security as RSA but with much smaller key size, *i.e. the 256-bit ECC method is better in security than the 1024-bit RSA* (Ors *et al.* 2003). This advantage of ECC has been recently recognized with its incorporation into many standards (Raju & Akbani 2003). In 1999, the Elliptic Curve Digital Signature Algorithm was adopted by ANSI and is now included in the ISO/IEC 15946 draft standards, IEEE P1363 and the Internet Engineering Task Force.

ECC systems can be implemented in software as well as hardware (Orton *et al.* 1986, Agnew *et al.* 1993, Royo *et al.* 1997, Hankerson *et al.* 2000, Huang *et al.* 2003, Ors *et al.* 2003, Gutub 2006). Hardware systems normally provide better speed and security (Ors *et al.* 2003, Dyka & Langendoerfer 2005, Morales-Sandoval & Feregrino-Uribe 2005, Park *et al.* 2005, Zhou *et al.* 2005, Gutub 2006). Software, on the other hand, provides flexibility in the choice of the key size (Hankerson *et al.* 2000). To benefit from both hardware and software advantages we use *scalable multipliers*. For cryptographic applications, it is more secure to handle the computations in hardware rather than software. Software-based systems can be interrupted and trespassed by intruders more easily than hardware, jeopardizing the whole application security (Michener & Mohan 2001).

Several ECC hardware processors have recently been proposed in the literature for Galois Fields including GF(p) and GF(2^k) (Michener & Mohan 2001, Huang *et al.* 2003, Ors *et al.* 2003, Dyka & Langendoerfer 2005, Morales-Sandoval & Feregrino-Uribe 2005, Park *et al.* 2005, Zhou *et al.* 2005, Somani *et al.* 2006). The design of these processors is based on representing the elliptic curve points as projective coordinate points (Orlando & Paar 2001, Ors *et al.* 2003) in order to eliminate division, hence inversion, operations. It is known that adding two points over an elliptic curve requires a division operation, which is the most expensive operation over GF(p) (Blake *et al.* 1999, Savas *et al.* 2005, Gutub 2007). There are several candidates for projective coordinate systems. The choice thus far has been based on selecting the system that has the least number of multiplication steps, since multiplication over GF(p) is a common operation and the next most time consuming process in ECC.

In this paper, we propose that the choice of the projective coordinate system should also depend on its inherent parallelism. High-speed crypto processors are crucial for today’s security applications (Michener & Mohan 2001). Parallelism will be investigated as a practical solution for meeting this requirement. We use scalable GF(p) multipliers as reported by Tenca and Koc (2003) since they lead to a wide range of hardware flexibility and trade-offs between area and time, as compared to conventional GF(p) multipliers. The scalable multipliers allow the hardware designer to choose between area and time as required by the application. Scalable multipliers are implemented in digit serial fashion such that the multiplication of any iteration does not begin before the multiplication operation of the previous iteration is completed. It should be noted that any ECC processor must implement the procedures of projective coordinates efficiently since they are the core steps of the point operation algorithm. The main contribution of this work can be viewed as efficient usage of parallelism within a projective coordinate procedure via available scalable Montgomery multipliers utilized to compute GF(p) ECC operations.

The outline of the paper is as follows. In Section 2, we provide a brief theoretical background to elliptic curve cryptography. Section 3 outlines the algorithm used for ECC multiplication which is the basic concept behind using elliptic curve in cryptography. The elliptic curve point addition and doubling are elaborated using projective coordinates in Section 4, followed by the description of the proposed hardware architecture in Section 5, which will present the modeling details, including the reasons of choosing scalable multipliers. The conclusion of the paper is given in Section 6.

ELLIPTIC CURVES OVER GF(P)

It will be assumed that the reader is familiar with the arithmetic over elliptic curves. For a good review the reader is referred to Blake *et al.* (1999). The elliptic curve arithmetic of *GF(p)* is the usual mod *p* arithmetic. The elliptic curve equation over *GF(p)* is:

$$y^2 = x^3 + ax + b \text{ ; where } p > 3,\ 4a^3 + 27b^2 \neq 0, \text{ and } x, y, a, b \in GF(p).$$

There is also a single element named the *point at infinity* or the *zero point* denoted '*O*'. By adding this point, the projective version of the curve is obtained. If *P* and *Q* are two points on the elliptic curve, a third point which is the intersection of the curve with the line through *P* and *Q* can be uniquely described. If the line is tangent to the curve at a point, then that point is counted twice; and if the line is parallel to the *y-axis*, we define the third point as the point *O (zero point)*. Exactly one of these conditions holds for any pair of points on an elliptic curve. If a point on the elliptic curve is to be added to another point on the curve or to itself, some special addition rules are applied, depending on the finite field and the coordinate system used.

These addition rules in the finite field GF(p) and affine coordinate system are as follows:

$O = -O,$

$(x, y) + O = (x, y),$ *and*

$(x, y) + (x, -y) = O.$

The addition of two different points on the elliptic curve is computed as shown below:

$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$; where $x_1 \neq x_2$,

$\lambda = (y_2 - y_1)/(x_2 - x_1)$,

$x_3 = \lambda^2 - x_1 - x_2$, *and*

$y_3 = \lambda(x_1 - x_3) - y_1$.

The addition of a point to itself (doubling a point) on the elliptic curve is computed as shown below:

$(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$; where $x_1 \neq 0$,
$\lambda = (3(x_1)^2 + a)/(2y_1)$,
$x_3 = \lambda^2 - 2x_1$, *and*
$y_3 = \lambda(x_1 - x_3) - y_1$.

We assume that the squaring calculation has the same complexity as multiplication. To add two different affine coordinates points in *GF(p)* we need six additions, one inversion, and three multiplication operations. To double a point we require four additions, one inversion, and four multiplication computations. The GF(p) point operations will be discussed for ECC crypto processors in section 5.

POINT MULTIPLICATION ALGORITHM

There are many ways to apply elliptic curves in crypto applications (Blake *et al.* 1999). The most time consuming operation in all ECC methods is finding the multiples of a point *P*, i.e. *nP*, which involve several point additions and doublings. The normal ECC algorithm used for calculating *nP* from *P* is based on the binary representation of *n*, since it is known to be efficient and practical to implement in hardware (Blake *et al.*

1999, Hankerson *et al.* 2000). This method, proposed originally for affine coordinates, is shown as the *Binary Algorithm*:

```
Define k: number of bits in n and  n_i: the i^th bit of n
Input:  P (a point on the elliptic curve).
Output:  Q = nP (another point on the elliptic curve).
1.  if $n_{k-1} = 1$, then $Q:=P$ else $Q:=0$;
2.  for $i = k-2$ down to $0$;
3.    { $Q := Q + Q$ ;
4.       if $n_i = 1$ then $Q := Q + P$ ; }
5.  return $Q$;
```

Basically, the binary algorithm scans the bits of *n* and doubles the point *Q* *k*-times. Whenever a particular bit of *n* is found to be one, an extra operation of point addition (*Q*+*P*) is needed. As shown in Fig. 1, every point addition or point doubling requires the three modulo operations of multiplication, inversion, and addition/subtraction as clarified earlier in Section 2.

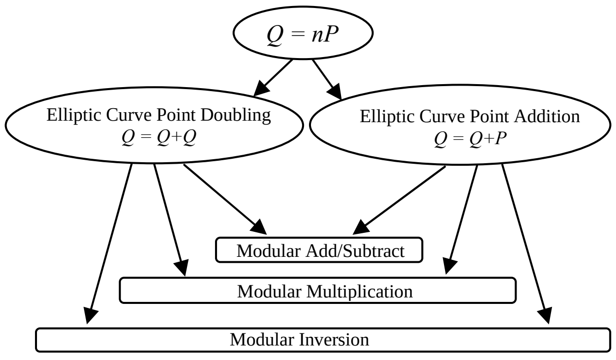

Figure 1: Elliptic curve arithmetic hierarchy

PROJECTIVE COORDINATES

Projective coordinates are used to eliminate the need for performing lengthy inversion as in the crypto processors of Orlando and Paar (2001). For an elliptic curve defined over *GF(p)*, two different forms of formula are available (Miyaji 1992, Blake *et al.* 1999,) for point addition and doubling. One form projects $(x,y)=(X/Z^2,Y/Z^3)$ (Blake *et al.* 1999), while the second projects $(x,y)=(X/Z,Y/Z)$ (Miyaji 1992). The two procedures for projective point addition of *P*+*Q* (two elliptic curve points) are shown below:

$P=(X_1,Y_1,Z_1);Q=(X_2,Y_2,Z_2);P+Q=(X_3,Y_3,Z_3);$ *where* $P \neq \pm Q$

$(x,y)=(X/Z^2,Y/Z^3) \Rightarrow (X,Y,Z)$		$(x,y)=(X/Z,Y/Z) \Rightarrow (X,Y,Z)$	
$\lambda_1 = X_1Z_2^2$	*2M*	$\lambda_1 = X_1Z_2$	*1M*
$\lambda_2 = X_2Z_1^2$	*2M*	$\lambda_2 = X_2Z_1$	*1M*
$\lambda_3 = \lambda_1 - \lambda_2$		$\lambda_3 = \lambda_2 - \lambda_1$	
$\lambda_4 = Y_1Z_2^3$	*2M*	$\lambda_4 = Y_1Z_2$	*1M*
$\lambda_5 = Y_2Z_1^3$	*2M*	$\lambda_5 = Y_2Z_1$	*1M*
$\lambda_6 = \lambda_4 - \lambda_5$		$\lambda_6 = \lambda_5 - \lambda_4$	
$\lambda_7 = \lambda_1 + \lambda_2$		$\lambda_7 = \lambda_1 + \lambda_2$	
$\lambda_8 = \lambda_4 + \lambda_5$		$\lambda_8 =\lambda_6^2 Z_1Z_2-\lambda_3^2\lambda_7$	*5M*
$Z_3 = Z_1Z_2\lambda_3$	*2M*	$Z_3 = Z_1Z_2\lambda_3^3$	*2M*
$X_3 = \lambda_6^2 - \lambda_7\lambda_3^2$	*3M*	$X_3 = \lambda_8\lambda_3$	*1M*
$\lambda_9 = \lambda_7\lambda_3^2 - 2X_3$		$\lambda_9 = \lambda_3^2 X_1Z_2 - \lambda_8$	*1M*
$Y_3 = (\lambda_9\lambda_6 - \lambda_8\lambda_3^3)/2$	*3M*	$Y_3 = \lambda_9\lambda_6 - \lambda_3^3 Y_1Z_2$	*2M*
	-----		-----
	16M		***15M***

Note that the 16M and 15M represent the total number of multiplication operations (multiplications count) for each procedure, respectively.

Similarly, the two formulae and their multiplication operation count for projective point doubling are shown below:

$P = (X_1,Y_1,Z_1);\ P+P = (X_3,Y_3,Z_3)$

$(x,y)=(X/Z^2, Y/Z^3) \Rightarrow (X,Y,Z)$		$(x, y) = (X/Z, Y/Z) \Rightarrow (X,Y,Z)$	
$\lambda_1 = 3X_1^2 + aZ_1^4$	*4M*	$\lambda_1 = 3X_1^2 + aZ_1^2$	*2M*
$Z_3 = 2Y_1Z_1$	*1M*	$\lambda_2 = Y_1Z_1$	*1M*
$\lambda_2 = 4X_1 Y_1^2$	*2M*	$\lambda_3 = X_1Y_1\lambda_2$	*2M*
$X_3 = \lambda_1^2 - 2\lambda_2$	*1M*	$\lambda_4 = \lambda_1^2 - 8\lambda_3$	*1M*
$\lambda_3 = 8Y_1^4$	*1M*	$X_3 = 2\lambda_4\lambda_2$	*1M*
$\lambda_4 = \lambda_2 - 2X_3$		$Y_3=\lambda_1(4\lambda_3-\lambda_4)-8(Y_1\lambda_2)^2$	*3M*
$Y_3 = \lambda_1\lambda_4 -\lambda_3$	*1M*	$Z_3 = 8 \lambda_2^3$	*2M*
	------		-----
	10M		***12M***

The squaring calculation over *GF(p)* is considered similar to the multiplication computation. They are both noted as *M* (multiplication). Here the time of addition and subtraction are ignored since they are negligible compared to multiplication (Blake *et al.* 1999). Since the number of projective point additions is taken to be, on an average, half the number of bits, it can be clearly seen form the above tables that the projective coordinate $(x,y) = (X/Z^2,Y/Z^3)$ has on the average 18 multiplication operations, while the projection $(x,y) = (X/Z,Y/Z)$ has on the average 19.5 multiplications. Considering the worst case scenario of having the number of point additions similar to the number of bits, the projective coordinate $(x,y) = (X/Z^2,Y/Z^3)$ has 26 multiplication operations, whereas the projection $(x,y) = (X/Z,Y/Z)$ has 27 multiplications. Clearly, the projective coordinate *(x,y) = (X/Z^2,Y/Z^3)* would be the projection of choice for sequential implementation as summarized in Table 1.

Table 1: Comparison between the different projective coordinate assuming sequential implementations

Procedure of Projecting	Average Number of Multiplication Cycles	Worst Number of Multiplication Cycles
(x,y) to *(X/Z^2,Y/Z^3)*	18	26
(x,y) to *(X/Z,Y/Z)*	19.5	27

Our basic motivation in this research is gained by taking full advantage of the parallelism that exists in the ECC and its projective coordinate operations. The two forms of projecting procedures *(x,y) = (X/Z^2,Y/Z^3)* and *(x,y) = (X/Z,Y/Z)* for projective point addition of *P+Q* are depicted in Figs. 2 and 3.

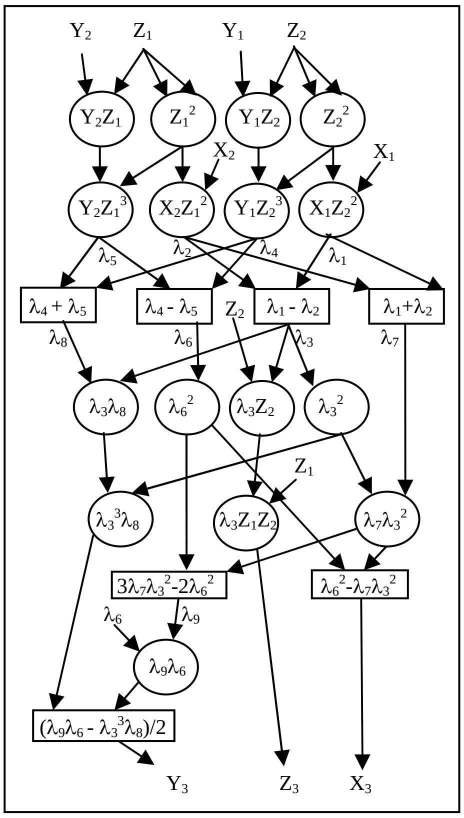

Figure 2: Addition data flow diagram for projection of *(x,y)* to *(X/Z^2,Y/Z^3)*

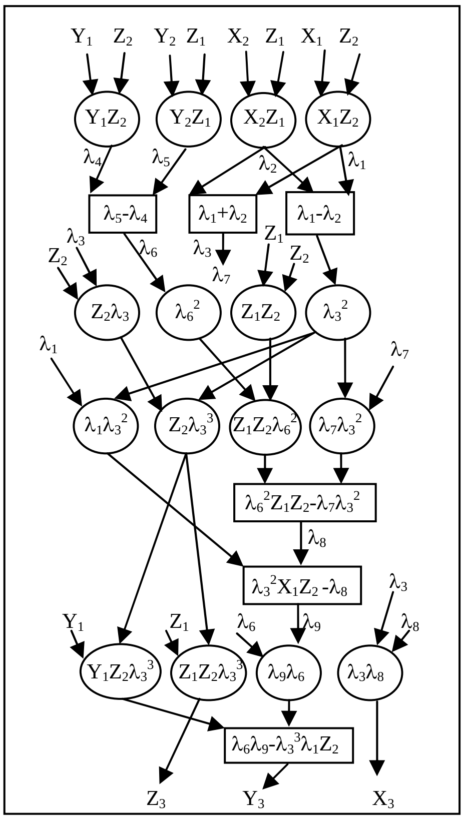

Figure 3: Addition data flow diagram for projection of *(x,y)* to *(X/Z,Y/Z)*

Similarly Figs. 4 and 5 show the procedures for projective point doubling of *P+P* in the different forms. Note that Figs. 4 and 5 assume that the product *aZ$_1$* can be computed without multiplication for small values of coefficient *a*. Through these figures, we highlight the dependency within the procedures. The figures show that both projective coordinate forms can be parallelized to the maximum possibility using four multipliers, but with different critical path stages, hence different number of

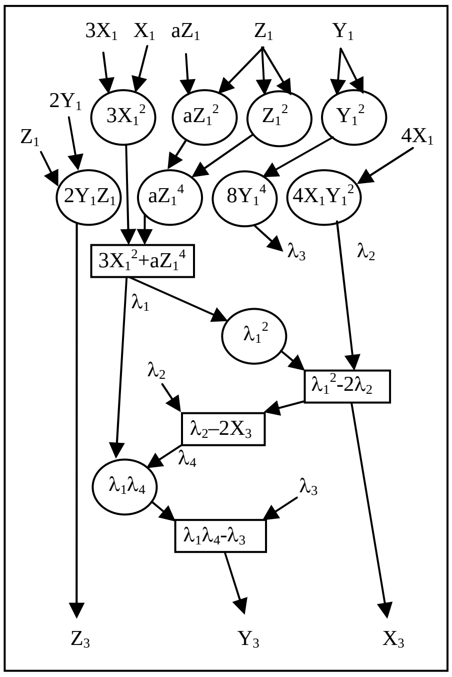

Figure 4: Doubling data flow diagram for projection of *(x,y)* to *$(X/Z^2,Y/Z^3)$*

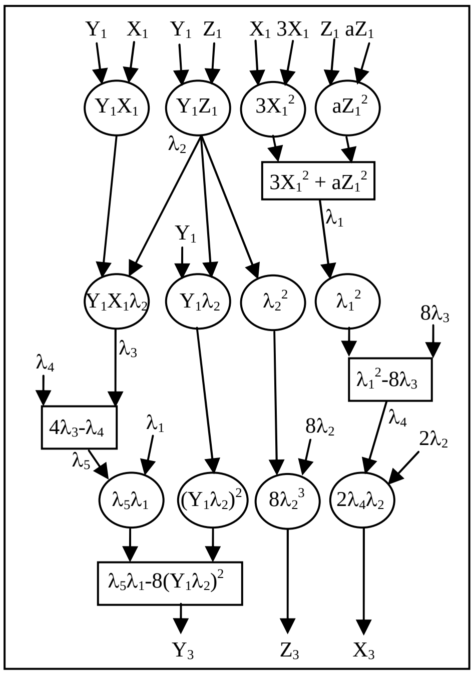

Figure 5: Doubling data flow diagram for projection of *(x,y)* to *(X/Z,Y/Z)*

multiplication cycles. It can be observed that the projective coordinate *$(x,y) = (X/Z^2,Y/Z^3)$* has on the average 6.5 multiplication cycles, whereas the projection *$(x,y) = (X/Z,Y/Z)$* has on the average 5 multiplications, using the common assumption of the number of point additions to be half the number of bits (*k/2*). Allowing for the worst case of having the number of point additions to be equal to number of bits (*k*), the projective coordinate *$(x,y) = (X/Z^2,Y/Z^3)$* has 9 multiplication operations, whereas the projection *$(x,y) = (X/Z,Y/Z)$* has 7 multiplications (Table 2).

Table 2: Comparison between the different projective coordinates assuming parallel implementations

Procedure of Projecting	Number of Multiplication Cycles		Multipliers Utilization	
	Average	Worst	Average	Worst
(x,y) to *$(X/Z^2,Y/Z^3)$*	6.5	9	69%	72%
(x,y) to *(X/Z,Y/Z)*	5	7	100%	100%

Since projection *(x,y) = (X/Z,Y/Z)* leads to fewer parallel multiplication steps, it would be the projection of choice for our implementation. A further benefit of the system is the 100% utilization of the four multipliers in all multiplication cycles, as seen in Figs. 3 and 5, which is not the case of the projection $(x,y) = (X/Z^2,Y/Z^3)$ in Figs. 2 and 4. Fig. 2 shows 80% utilization of the four multipliers. Only one multiplier is used in the final multiplication step, and in the next to last step, one multiplier is not utilized. Similarly or even worse is observed in Fig. 4 where all multipliers are utilized around 63%. The last two multiplication steps make use of one multiplier only, making three multipliers idle in 40% of each point doubling operation.

IMPLEMENTING THE PROPOSED ALGORITHM

Several cryptographic architectures have been proposed in the literature (Ors *et al.* 2003, Dyka & Langendoerfer 2005, Morales-Sandoval & Feregrino-Uribe 2005, Park *et al.* 2005, Zhou *et al.* 2005). The conventional approach used in the design of these processors is to adopt serial computations at both the algorithmic level by using a single multiplier, as well as at the arithmetic level by using a serial multiplier. The reason for serial multiplier and sequential operation is that they lead to the lowest area for large word lengths, which is needed for secure encryption, i.e. 256 bits (Ors *et al.* 2003). The above approach reduces area at the expense of speed. The new architecture proposed in this paper has four parallel multipliers, an adder, registers and a controller, as shown in Fig. 6. The design is a straightforward implementation of the dependency graphs shown

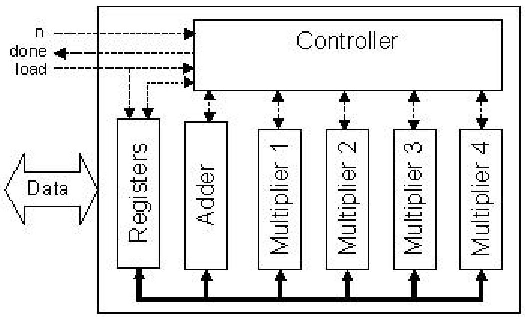

Figure 6: Proposed elliptic curve processor architecture

in Figs. 3 and 5. Its controller is constructed of a state machine to direct the flow of data to conduct the required projective point operation depending on the binary algorithm described previously in Section 3. The scalable multipliers use Montgomery multiplication to gain the best performance (Tenca & Koc 2003).

The improvement in our crypto-processor, other than the parallelism (seen in Figs. 3 and 5), is in the basic GF(p) multiplier. The designs proposed in Orton *et al.* (1986) and Orlando and Paar (2001) use multiplier hardware limited by the number of intended bits. If the number of bits need to be increased for some application, the complete hardware must be replaced. Furthermore, if the number of bits is much less than the intention of the VLSI design, the unnecessary bits will be considered as zeros and they will be included in the computation causing the same delay as if all bits are essential. These weaknesses motivated the adoption of special *scalable multipliers* instead of conventional ones.

Scalable multipliers

An arithmetic unit is called scalable if it can be reused or replicated in order to generate long precision results independently of the data path precision for which the unit was originally designed. To speed up the multiplication operation, various dedicated multiplier modules were developed in (Agnew *et al.* 1993, Royo *et al.* 1997). These designs operate over fixed finite fields. For example, the multiplier designed for 155 bits (Agnew *et al.* 1993) cannot be used for any other field of higher degree. When a need for multiplication of larger precision appears, a new multiplier must be designed.

Another way to avoid redesigning the module is to use software implementations and fixed precision multipliers. However, software implementations are inefficient in utilizing inherent concurrency of the multiplication because of the inconvenient pipeline structure of the microprocessors being used. Furthermore, software implementations on fixed digit multipliers are more complex and require excessive effort in coding.

Therefore, a scalable hardware module specifically tailored to take advantage of the concurrency of the multiplication algorithm has become extremely attractive (Tenca & Koc 2003). Also, computation of elliptic point doubling, addition and the algorithm of computing multiples of the base point is such that the multiplication of one stage must be completed before starting the multiplication of the subsequent stage. Therefore, pipelining the digits to further stages is not applicable, and even if fast digit serial multipliers are used, the throughput of such multipliers cannot be exploited since the next multiplication operation cannot begin until the multiplication operations in the previous stage have been completed.

Another benefit of this scalable multiplier is the flexibility in choosing the number of processing elements (*PE*) and their sizes, of course, with the price compensated from the number of clock cycles to complete the ECC computation. This trade-off between area and speed provides a hardware area range to fit in very limited areas such as smart cards. An outline diagram of the scalable multipliers is shown in Fig. 7. We will use this scalable multiplier as a block within our architecture. The details of this multiplier are found in Tenca and Koc (2003).

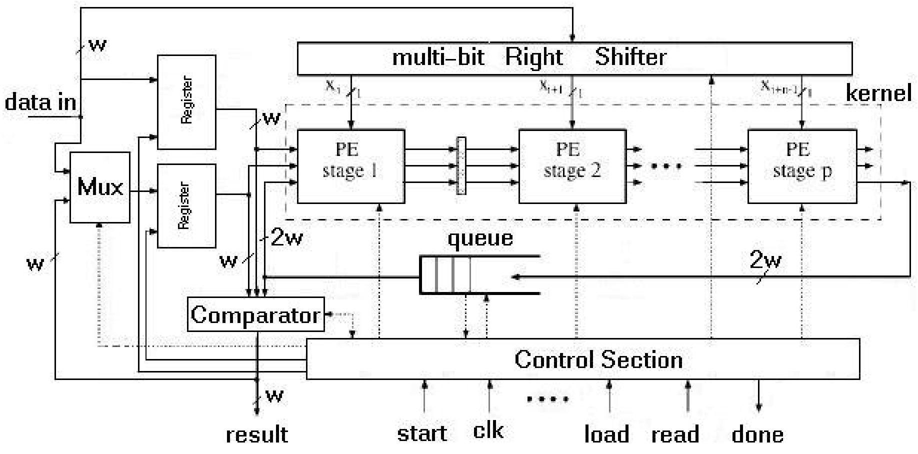

Figure 7: Organization of the scalable multiplier

Area comparison

The proposed scalable design was modeled with three main parameters, namely n_{max}, *PE* and w, which define several hardware configurations. The parameter *PE* is the number of processing elements. Each *PE* is processing a word of w bits. The maximum number of bits is noted as n_{max}.

The exact area of any design depends on the technology and minimum feature size. For technology independence, we use the number of equivalent gates as an area measure. The scalable multiplier is the unit to make the difference between our proposed design and any conventional one of single multiplier. Thus, the scalable multiplier is considered the main factor in calculating the speed (computation time) and area (gate counts), allowing our area analysis of the proposed architecture to be based solely on the size of the multipliers.

A CAD tool from Mentor Graphics (Leonardo) was used by Tenca and Koc (2003) to design and simulate the scalable multiplier. Leonardo takes the VHDL design code and provides a synthesized design with its area and longest path delay using the library provided in the ASIC Design Kit (ADK). It has to be mentioned here that the ADK was developed for educational purposes and cannot be thoroughly compared to technologies adopted for marketable ASICs. However, ADK provides a framework to contrast the different scalable hardware designs.

Using the maximum number of bits n_{max} and varying PE and w provide different scalable designs with tradeoff between speed and area. As detailed in Tenca and Koc (2003), the PE area (in equivalent gates) depends only on the word size w. The experimental results obtained with the synthesis tools mentioned above allows an estimation of the PE area as: $Area_{PE} = 50w+25$, which includes the local control logic. The area of each interstage latch was obtained as $Area_{latch}=34w$ and, therefore, the area of a pipeline with PE units may be approximated with the Scalable-Multiplier-Area *(SM-AREA)* formula:

$$SM\text{-}AREA \approx 84wPE + 25PE - 22w$$

The *SM-AREA* last term of $-22w$ accounts for simplifications in the last PE in the Scalable-Multiplier pipeline (Tenca & Koc 2003). The area of our proposed architectures and the regular single multiplier ones are compared in Fig. 8. All the designs of Fig. 8 are built for $n_{max}=256$ *bits*, as a practical example (Blake *et al.* 1999). Note that our proposed four-multiplier hardware can be smaller in area than the single multiplier designs when w increases to 4-bits and above, has been tested changing n_{max} and gave the same observation of area increases by increasing PE or w.

Computation time comparison

The total computation time is the product of three terms: the average number of multiplication steps, the number of clock cycles each multiplication takes, and the clock period of the VLSI hardware. The number of clock cycles each multiplication takes is estimated in Tenca and Koc (2003) as:

$$Number\text{-}of\text{-}Cycles\ per\ Multiplication = \lceil (n_{max}+1)/PE \rceil (\lceil (n_{max}+1)/w \rceil +1) \ -1 +2(PE-1).$$

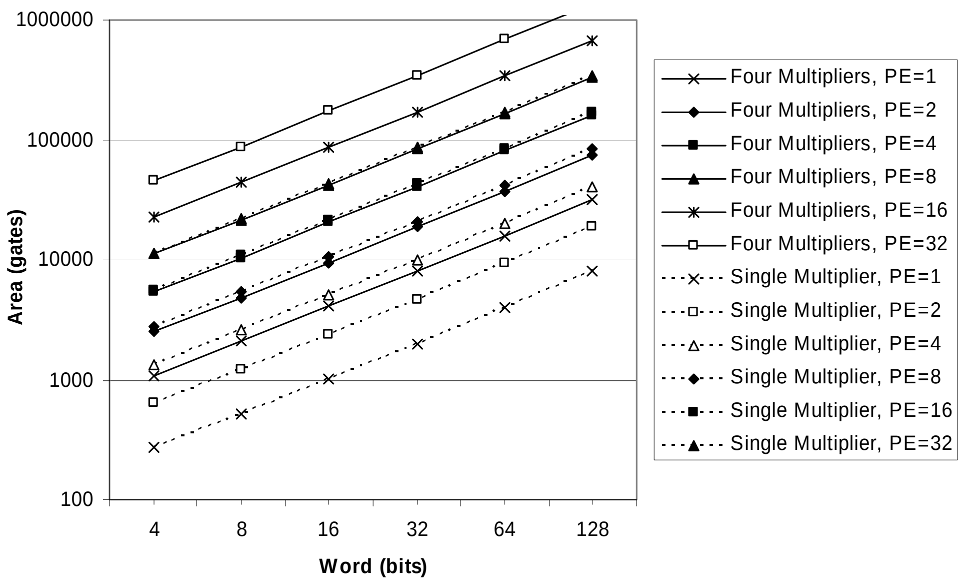

Figure 8: Area of different designs with $n_{max}=256$ bits

The clock period is assumed to be the exact longest path delay in order to get the most efficient frequency. The clock period is generated by the CAD tool, Leonardo (Tenca & Koc 2003), which changes with the value of *w* as listed in Table 3.

Table 3: Scalable multiplier clock cycle periods *(nanoseconds)*

Word (w-bits)	4	8	16	32	64	128
Clock Period	4.9	5.2	5.8	6	6.3	6.8

The average number of multiplication steps (ANMS) differ depending on the projective coordinate procedure employed. ANMS in our proposed design uses the projective coordinate of $(x,y) = (X/Z, Y/Z)$ which leads to a better ANMS of *5* (Table 2), while ANMS for the single multiplier hardware is taken as *18* where the projective coordinate preferred is $(x,y) = (X/Z^2, Y/Z^3)$ as clarified earlier in Table 1. The computation time of all different designs is shown in Fig. 9 for $n_{max}=256$ *bits*. It is interesting to note that some single multiplier designs can be faster than some parallel designs. For example, a parallel hardware with one *PE* is slower than all single multiplier designs with *4-PEs* or more. Note also increasing the word number of bits (*w*) in the architectures having *32-PEs* results in the total computation time becoming very similar to having *16-PEs*. This gives a clear indication that increasing the number of *PEs* and increasing the number of bits per word (*w*) should not exceed a higher boundary.

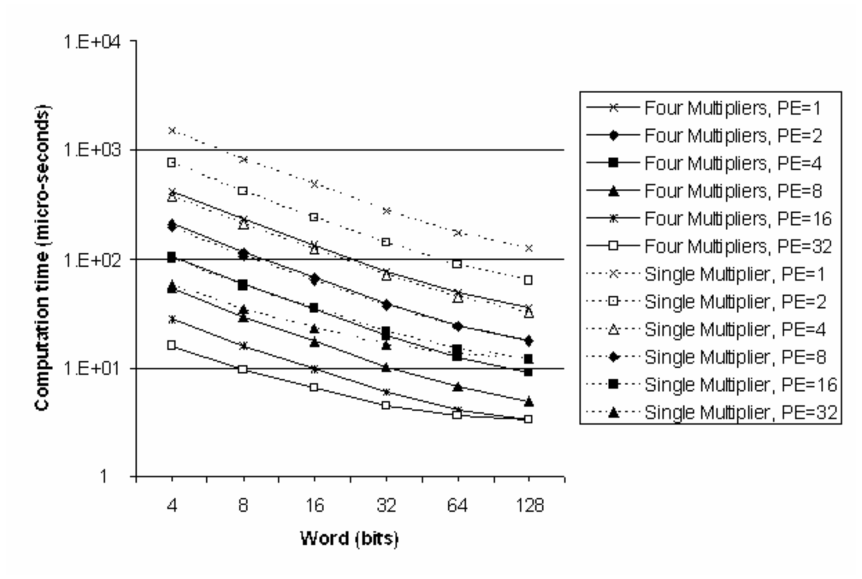

Figure 9: Computation time comparison of different designs with $n_{max}=256$ bits

Area time tradeoff

Depending on the importance of speed and area, the appropriate design is chosen. In fact, as seen from Figs. 8 and 9, as we pay in terms of area we gain in most of the cases in speed. But is the speed gained worth the area expenditure?

To estimate an evaluation standard that relates area and time, two different options are used as *figures of merit*. One *figure of merit* is AT^2 (Area×Time×Time) that gives the time much more importance over the area. The AT^2 values with respect to the number of bits per word (*w*) for all the designs built for $n_{max}=256$ bits are shown in Fig. 10. The other *figure of merit* is *AT* (Area×Time). AT is a standard that assumes the time and the area have equal weight. The AT values with respect to the number of bits per word (*w*) for all the designs built for $n_{max}=256$ bits can be seen in Fig. 11.

Considering AT^2, Fig. 10 demonstrates that our proposed design (of four multipliers) gives the best AT^2 values when the *PE* number is ≥ 8. In fact, the best design has *16-PEs* and $w = 64$*-bits*. Note that as the number of bits per word (*w*) changes, the proper design selection varies. For example, Fig. 10 shows that when $w \leq 16$ bits, the proposed architecture with *32-PEs* is best followed by *16-PEs* and then *8-PEs*. These results indicate that large area and high number of stages (*PEs*) become a burden that cases delay instead of speedup. A similar result is observed for the design with *16-PEs* when $w \geq 128$ *bits*.

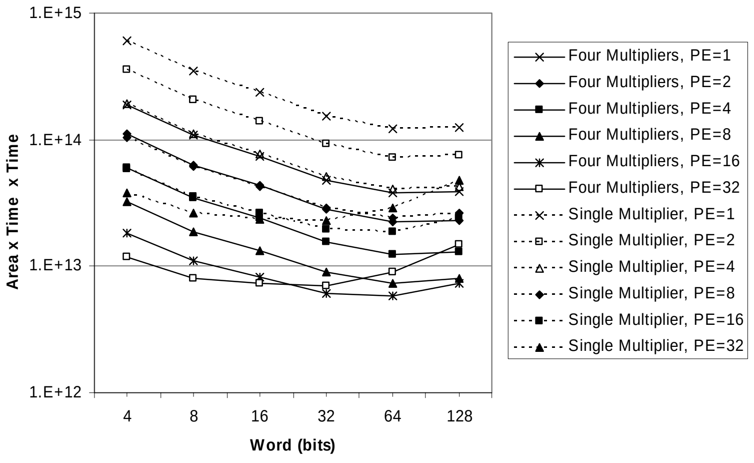

Figure 10: AT^2 of different designs with n_{max}*=256* bits

If the area and time have similar importance and *AT* is considered as the standard, Fig. 11 shows that our four multipliers designs give the best *AT* values when the number of *PEs* is reduced (minimizing the area). The best proposed design with parallel multipliers has only one *PE*. It should be mentioned that with this *AT* figure of merit standard, time is not very important, and a single multiplier hardware will be preferred over the parallel design with one *PE*. However, as the number of *PEs* and number of bits per word (*w*) change, the suitable design selection differs. For example, when $w >$ *32* bits, the proposed parallel architecture with *2-PEs* is very similar to a single multiplier hardware with *4-PEs*.

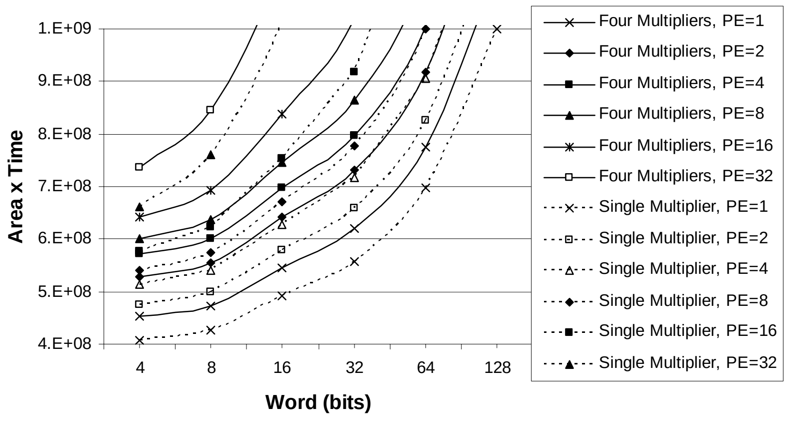

Figure 11: AT of different designs with n_{max}*=256* bits

CONCLUSIONS

This paper presented a new hardware model of a procedure used in the computation of elliptic curve cryptographic operations. The original idea is the utilization of the inherited parallelism of multiplication steps found in the projective coordinate algorithms that convert the inverse operation into consecutive multiplications. Two available projective coordinates forms where studied. Comparing projecting *(x,y)* to *$(X/Z^2,Y/Z^3)$* requires fewer multiplications than projecting into *$(X/Z,Y/Z)$*. The latter uses one less multiplication operation in adding two different elliptic points, however, it uses two more multiplication operations in doubling an elliptic point. For sequential implementation, i.e. using a single multiplier, projecting *(x,y)* into *$(X/Z^2,Y/Z^3)$* has always been the candidate of choice for implementing ECC since it has the minimum number of multiplication operations.

Although the proposed architecture can implement the procedures of both projective coordinate forms, the analysis of the critical paths of both projective coordinates indicates that for parallel implementation, projecting *(x,y)* to *$(X/Z,Y/Z)$* requires fewer cycles and hence it is faster than projecting *(x,y)* to *$(X/Z^2,Y/Z^3)$*. Projecting *(x,y)* to *$(X/Z,Y/Z)$* leads to the need for four parallel multipliers, which enjoys 100% utilization compared to the other projective system of 30% utilization of all four multipliers.

An important comment about the implementation of our proposed architecture is that we propose to use scalable multipliers which depend on digit serial multiplications. Digit serial computation is more suitable for the elliptic curve crypto-algorithm discussed above since in computation of elliptic point doubling, addition and multiplication of the base point is such that multiplication of one stage must be completed before starting the multiplication of the subsequent stage. Therefore even if a pipelined bit-parallel multiplier is used, the throughput of such a multiplier cannot be exploited since the next multiplication operation cannot commence until the multiplication operation in the previous stage has completed. The scalable multiplier is flexible to give different hardware versions of the same basic multiplier depending on the number of processing elements (*PE*) and the number of bits (*w*) each *PE* is handling.

The attraction of this work is that using the proposed architecture with projections of *(x,y)* to *(X/Z,Y/Z)* leads to the best performance by utilizing the inherited parallelism of the projective coordinate arithmetic.

ACKNOWLEDGMENTS

Thanks to King Fahd University of Petroleum & Minerals (KFUPM) for its unlimited support for this work and all research work.

REFERENCES

Agnew, G., Mullin, R., & Vanstone, S., 1993. An implementation of elliptic curve cryptosystems over $F2^{155}$. IEEE Journal on Selected Areas in Communications. **11(5):**804–813.

Blake, I., Seroussi, G., & Smart, N., 1999. Elliptic curves in cryptography. Cambridge University Press, NY, USA.

Dyka, Z. & Langendoerfer, P., 2005. Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba's method. IEEE Conference of Design, Automation and Test in Europe. **3:**70 – 75.

Gutub, A., 2006. Fast 160-bits GF(p) elliptic curve crypto hardware of high-radix scalable multipliers. International Arab Journal of Information Technology (IAJIT), **3(4):**342-349.

Gutub, A., 2007. High speed hardware architecture to compute GF(p) Montgomery inversion with scalability features. IET (IEE) Proceedings Computers and Digital Techniques, **1(4):** 389-396.

Hankerson, D., Hernandez, J., & Menezes, A., 2000. Software implementation of elliptic curve cryptography over binary fields. Second International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Worcester, MA, USA, Pp. 243-267.

Huang, C., Lai, J., Ren, J., & Zhang, Q., 2003. Scalable elliptic curve encryption processor for portable application. Proceedings of the 5th International Conference on ASIC. **2:**1312-1316.

Koblitz, N., 1987. Elliptic curve cryptosystems. Math Computing. **48:**203–209.

Miller, V., 1986. Use of elliptic curves in cryptography. Proceedings of Advances in Cryptology (CRYPTO), Lecture notes in computer sciences, Santa Barbara, California, USA, Pp. 417–426.

Miyaji, A., 1992. Elliptic curves over Fp suitable for cryptosystems. Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, Lecture Notes In Computer Science, **718:** 479 – 491.

Michener, J., & Mohan, S., 2001. Internet watch: clothing the E-Emperor, computer – innovative technology for computer professionals. IEEE Computer Society. **34(9):**116-118.

Morales-Sandoval, M., & Feregrino-Uribe, C., 2005. A hardware architecture for elliptic curve cryptography and lossless data compression. 15th IEEE International Conference on Electronics, Communications and Computers, Puebla, Mexico, 28 February - 02 March 2005, Pp. 113 – 118.

Orton, G., Roy, M., Scott1, P., Peppard, L., & Tavares, S., 1986. VLSI implementation of public-key encryption algorithms. Advances in Cryptology (CRYPTO). Santa Barbara, California, USA, **263:**277-301.

Orlando, G., & Paar, C., 2001. A scalable GF(p) elliptic curve processor architecture for programmable hardware. Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Paris, France, May 14-16, 2001, Pp: 348 - 363

Ors, S.B., Batina, L., Preneel, B., & Vandewalle, J. 2003. Hardware implementation of an elliptic curve processor over GF(p). IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP), The Hague, The Netherlands, 24-26 June 2003, Pp. 433 – 443.

Park, J., Hwang, J., & Kim, Y., 2005. FPGA and ASIC implementation of ECC processor for security on medical embedded system. International Conference on Information Technology and Applications (ICITA). **2:**547- 551.

Raju, G., & Akbani, R., 2003. Elliptic curve cryptosystem and its applications. IEEE International Conference on Systems, Man and Cybernetics, **2:**1540 – 1543.

Rivest, R., Shamir, A., & Adleman, L., 1978. A Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. **21(2):**120–126.

Royo, A., Moran, J., & Lopez, J., 1997. Design and implementation of a coprocessor for cryptography applications. European Design and Test Conference Proceedings, Paris, France, 17-20 March 1997, Pp. 213–217.

Savas, E., Naseer, M., Gutub, A., & Koc, C., 2005. Efficient unified Montgomery inversion with multi-bit shifting. IEE Proceedings Computers and Digital Techniques, **152(4):**489 – 498.

Somani, T., Ibrahim, M., & Gutub, A., 2006. Highly efficient elliptic curve crypto-processor with parallel GF(2m) field multipliers. Journal of Computer Science (JCS), **2(5):**395-400.

Tenca, A., & Koc, C., 2003. A scalable architecture for modular multiplication based on Montgomery's algorithm. IEEE Transactions on Computers. **52(9):**1215-1221.

Zhou, J., Jiang, X., & Chen, H., 2005. An efficient architecture for computing division over GF(2^m) in elliptic curve cryptography. 6th International Conference on ASIC (ASICON). **1:**321- 325.