Remodeling of Elliptic Curve Cryptography Scalar Multiplication Architecture using Parallel Jacobian Coordinate System

Adnan Abdul-Aziz Gutub aagutub@uqu.edu.sa
Center of Excellence in Hajj and Omrah Research,
Umm Al-Qura University P.O. Box: 6287,
Makkah 21955, Saudi Arabia

Abstract

In this paper, an improved parallel elliptic curve processor is designed and modeled. We adjusted the Jacobian coordinates system by interacting point double and point add operations. This modified coordinates is parallelized using four multipliers similar to older parallel architectures. We implemented the components of the proposed design using FPGA with parametric features, in terms of number of parallel multipliers, number of parallel adders and width of input operands. The remodeled design is compared to other similar designs i.e. parallel Jacobian coordinates and parallel standard projective coordinates yielding better performance. Results showed that this proposed modified Jacobian design gave higher speed and cost (AT^2) showing attractive research direction.

Keywords: Cryptography hardware, Elliptic curve cryptography; Jacobian coordinate system; Parallel multipliers architecture; Projective coordinate cryptosystems

1. INTRODUCTION

Elliptic curves were first proposed as a basis for public key cryptography in the mid 1980s independently by Koblitz [1] and Miller [2]. Elliptic curve cryptography (ECC) algorithm is practical than existing security algorithms [3,4]. Because of this fact, it showed real attraction to portable devices (handheld devices) manufacturers and the security of their systems. In fact, through these devices, any one can access either email, or do bank transaction or buy any thing on internet using credit cards with high security standards. Elliptic curve algorithm is promising to be the best choice of these handhelds or similar devices because of low computing power (low battery consumption) and fast execution. ECC further gives very high security as compared to similar crypto systems with less size of key. For example, 160 bit ECC system is believed to provide same level of security as 1024 bit RSA [5,6]. Also, the rate at which ECC key sizes increase in order to obtain increased security is much slower than the rate at which integer based discrete logarithm (DL) or RSA key sizes increase for the same level increase in security [7].

Elliptic curves provide a public key crypto-system based on the difficulty of the elliptic curve discrete logarithm problem, which is so called because of its similarity to the discrete logarithm problem (DLP) over the integers modulo a prime p [3,4,8]. This similarity means that most cryptographic procedures carried out using a cryptosystem based on the DLP over the integers modulo p can also be carried out in an elliptic curve cryptosystem. ECCs can also provide a faster implementation than RSA or DL systems, and use less bandwidth and power [9]. These issues are crucial in lightweight applications, i.e. smart cards [10].

An elliptic curve over a Galois field with p elements, GF(p), where p is prime and p > 3 may be defined as the points (x,y) satisfying the curve equation $E: y^2=x^3+ax+b \pmod p$, where a and b are constants satisfying $4a^3+27b^2 \neq 0 \pmod p$. In addition to the points satisfying the curve equation E, a point at infinity (ϕ) is also defined. With a suitable definition of addition and doubling of points [2], this enables the points of an elliptic curve to form a group with addition and doubling of points being the group operation, and the point at infinity being the identity element. We then further define scalar multiplication of a point P by a scalar k as being the result of adding the point P to itself k times (i.e. kP = P + P + P + · · · + P (k- times)). The elliptic curve discrete logarithm problem is then defined as to compute scalar k such that Q = kP; given the prime modulus p, the curve constants a and b, and two points P and Q. This problem is infeasible for secure elliptic curves [1,2], and thus scalar multiplication is the basic cryptographic operation of an elliptic curve. Scalar multiplication involves mainly three modular operations: addition, multiplication and inversion, where the modular addition operation is the simplest and least to be worried about [11].

The other two ECC scalar multiplication modular operations are inversion and multiplication. Inversion is known to be the complex and very expensive operation [7], its cost is reduced by converting the normal (x,y) affine coordinate system to projective coordinate system (X,Y,Z), which will add-up more modular multiplications to the process; i.e. it will increase the number of multiplications in both ECC point doubling and adding processes operations to reduce the inversion complexity. Thus, modular multiplication is considered to be the repetitive arithmetic ECC scalar multiplication operation to be focused on.

This work extends our previous work of parallelizing the modular multiplications operations within the elliptic curve scalar multiplication process using four modular multipliers. We remodeled the Jacobian coordinate system by interacting the two ECC point doubling and adding processes. We implemented the components of the proposed parallel scalar multiplication design using Field Programmable Gate Arrays (FPGA) with parametric features, in terms of number of parallel multipliers, number of parallel adders, and width of input operands. The results compared timing of the modular multiplication (digit serial), modular adder and total time needed to perform scalar multiplication for three designs i.e. parallel Jacobian coordinates, parallel standard projective coordinates, and this proposed remodeled hardware designs. Analysis showed that this proposed enhanced Jacobian model via four parallel multipliers and two adders gave better AT^2 cost than existing scalar multiplication designs.

The flow or the rest of the paper is as follows. The next section will give a short overview of several attempts and hardware ECC designs related to this work. Section 3 describes the ECC scalar multiplication algorithm in some details. Then, the ECC affine and projective coordinates systems are described in Section 4. In Section 5, we describe our proposed design that we extend in this paper. Sections 6 and 7 are for describing our proposed design for scalar multiplication in ECC and its components FPGA implementations. For displaying the results and analysis, we reserved Section 8. Finally, we concluded with the achievements remarks and future work in Section 9.

2. RELATED WORK

Several hardware implementations to compute ECC scalar multiplication have been reported in the literature. Every technique has its pros and cons and requires fitting based on the application need. Many designs were dedicated for $GF(2^m)$ computation since it does not suffer the carry propagation problem. For example, in 1993, Agnew et al. [12] implemented ECC over $GF(2^{155})$ normal basis finite field to be simple and gain efficient solution through an optimal multiplier. Their design used a programmable control processor that achieved high performance but limited to the finite field it is designed for. In 1998, Rosner [13] worked on his thesis to develop a reconfigurable ECC crypto engine. His thesis hardware was dedicated for Galois fields $GF(2^n)m$ in standard base representations implemented using FPGAs. His work proved that a full point multiplication on ECC can be implemented on FPGAs although it is built for GF$(2^n)^m$.

In 2000, Torii and Yokoyama [6] used efficient hardware techniques to implement ECC on a digital signal processor (DSP). Their techniques improved modular multiplications based on Montgomery's multiplication method [14] but specified for pipeline processing on DSP. They devised an improved method for computing the number of multiplications and additions which enhanced computing the point doubling operation. Their ideas have been interesting but restricted to their targeted DSP hardware. In the same year, Bednara et al. [15] presented a focus on field multiplications hardware analysis for ECC FPGA hardware implementation. They analyzed Montgomery field multipliers utilizing lookup tables to gain more efficiency. Their study compared Massey-Omura multipliers with LFSR in terms of area and speed. They evaluated different curve coordinate representations with respect to the number of operations within the fields. The best coordinate system matching their FPGA design was reported.

In 2004, Saqib et al. [16] described a parallel architecture for Computing Scalar Multiplication using Hessian Elliptic Curves over F(2^{191}) on FPGA. The design aimed to be parallel in all levels and as general as possible without assuming any hardware type to gain the best possible speed. Their results have been interesting for GF(2^m) parallel architecture. A year later, in 2005, Dyke and Langendoefer [17] implemented ECC using Karatsuba's method. Their implementation used iterative hardware accelerator for polynomial multiplication with extended Galois fields (GF), which resulted in reducing the area consumption for recursive applications. Their approach reduces the energy consumption to 60% of the original approach. However, cost for all this achievement is the increased execution time. In 2006, Al-Somani and Ibrahim [18] proposed high performance GF(2^m) Elliptic Curve Crypto processor that is based on standard representation and uses three multipliers to perform parallel field multiplications. They used mixed coordinate systems in point operations to increase the performance. Their results showed better time complexity than existing designs by 76% when implemented on FPGA for GF(2^{173}). Al-Somani et al. in [19] further implements another ECC coprocessor using two multipliers only, with similar ideas that gained good results too. In 2007, Fan et al. [20] proposed parallel computing architecture for ECC scalar multiplication by using two-dimensional parallelism. They improved the performance by 26% and 32%, exploiting only vertical or horizontal parallelism, respectively. Different projective coordinates and recoding the scalar with Non-Adjacent Format (NAF) [14] represented further improvements in the performance with similar ideas. Since we focus our work in this paper on GF(p) ECC, the GF(2) ECC arithmetic and hardware implementations is less concentrated on.

Several ECC hardware designs were introduced for GF(p) scalar multiplications, for example, in 2001, Orlando and Paar [21, 22] proposed an architecture for computation of point multiplication for the ECC define over GF(p). Their architecture is scalable over area and speed and can easily be implemented on FPGA's. The processor used Montgomery multiplier (MM) for modular multiplications. The MM relied on the pre-computation of frequently used values and on the use of multiple processing engines. In 2003, Ors et al. [23] described a hardware implementation of an arithmetic processor suitable for RSA and ECC, due to the fact that they are the commonly used types of Public Key Cryptography. They implemented their processor efficiently for bit-lengths in a systolic array architecture that consists special operational blocks for all operations, for example Montgomery Modular Multiplication, modular addition/subtraction, EC Point doubling/addition, modular multiplicative inversion, EC point multiplier, projective to affine coordinates conversion and Montgomery to normal representation conversion. Their design is so generic and flexible that suffered engineering inefficiency in its speed, area, and power consumption.

In 2004, we [24] proposed a parallel architecture for GF(p) elliptic curve cryptographic processor. We used several multipliers adopting projective coordinates to reduce the inversion complexity within the ECC point operations. Our parallelization in [24] found that projecting ECC using Homogeneous Projective coordinates gave better results compared to Jacobian coordinates. Later, in 2005, Ansari, and Huapeng [25] introduced separating point addition and point doubling operations using two parallel processors to compute kP operations (scalar ECC multiplications). They used a buffer to hold results of point doubling while point addition is still in operation. They have shown that their parallel processors methods raised the operations speed by 90% compared to the single processor methods. However, this ratio in [25] is found dependant on the selection of ECC coordinate system and cannot be generalized. Sozzani and Turcato, in 2005 too, [26] proofed that ECC implementation in hardware is much faster than software implementation. They implemented ECC using hardware CMOS technology (VLSI HCMOS9 library, STMicroelectronics) using some level of parallelization which gave some specific improvement. In the same year, Chen et al. [27] presented a concurrent algorithm to speed up the point multiplication for the ECC based cryptosystem. They have used extra memory space to store intermediate points. The proposed algorithm achieved 100% hardware utilization that depends on the presented time schedule. Their work is found improving a 2001 paper [28], which saved around 32.2% delay for 256 bits computation. A year later, in 2006, Mishra [29] proposed pipelining scheme for implementing the ECC. This scheme enhances the computation of scalar multiplication significantly based on accessing a multiplier for each pipe stage. The pipelining scheme works based on a key observation i.e. to start the subsequent operation without waiting for the previous step to complete. The idea is interesting but needs further elaboration, which is taking place as in the work future study.

In 2007, Al-Khaleel et al. [7] introduced a technique for implementing ECC on FPGAs. It based its design on radix-4 modular multipliers, to allow for more efficient bit processing than radix-2. The hardware also exploited parallelism through proper scheduling and mapping of the algorithm. It presented area time tradeoff when adopting partitioning schemes and folded pipeline techniques. Chelton in 2008 [30] carried out a hardware development on ECC through application specific instruction set (ASIP) to gain high-performance using FPGA technology. They developed a combination of point-doubling operation and point addition operations based on the proposal in [31]. For gaining speed in the operation processes, the data path was pipelined allowing different levels of operation parallelism. The study showed the clock frequency increase and the optimal pipeline depth which changes by changing the FPGA platform.

In this work, we propose a remodeled scalar ECC multiplication architecture that extends our research in [24]. We benefited from the work scheme in [25] inverting it. Instead of separating the point addition and point doubling operations, we mix them benefiting from the scheduling thoughts of [7, 30, 31]. We found that Jacobian coordinates is more appropriate when mixing and parallelizing in four multipliers turning around one achievement of [24]. To build a fair study, we implemented the components used in all designs in FPGA to be used for the compareson. We, then, compared this modified mixed ECC Jacobian coordinates structure with similar parallel projective coordinates structures, i.e. original Jacobian coordinates and standard one. The comparison showed promising results that open directions for interesting research.

3. SCALAR MULTIPLICATION ALGORITHM

The algorithm used for scalar multiplication is based on the binary method [34], since it is efficient for hardware implementation. The binary method algorithm is shown below:

```
Inputs:  k: a constant , P: point on the elliptic curve
Output: Q: another point on the elliptic curve, Q=k . P
Define: w: number of bits in k, where $k_i$ is the $i^{th}$ bit in k
---------------------------------------------------------------
If $k_{w-1}$:=1 then Q := P else Q := 0;
for i:= w-2 down to 0 do
    Q := Q + Q;    Point Doubling
    If kw-1=1 then  Q := Q + P;    Point Addition
Return Q;
```

Basically, the binary method algorithm scans the bits of the constant k, in our case, from most to least bit and doubles the current point Q each time. After each point double operation, if the current k bit is one, then the algorithm adds the current point Q to the base point P. Each point operation, double or add, involves three elementary operations: modular multiplication, modular addition and modular multiplicative inverse.

Finding multiplicative inverses in the field GF(p) is extremely slow, and is generally avoided as much as possible [7]. The use of coordinate systems other than the Affine coordinate system (will be illustrated later) greatly reduces the number of inversions required in the operations of the scalar multiplication on the expense of extra multiplications.

ECC use effectively point doubling and addition operations in arithmetic execution. From many years of research, optimize formulae are available for the operations. Especially, by eliminating the costly field inversion from the main loop of the scalar multiplication, fast operations is achieved by using projective coordinates [32]. However, as in [33], the operation in projective coordinate involves more scalar multiplication than in affine coordinate and ECC on projective coordinate will be efficient only when the implementation of scalar multiplication is much faster than multiplicative inverse operation. Therefore, transfer is needed from one coordinate to another for avoiding the inversion process cost. The following section is dedicated for illustration of the coordinate systems structure used for these purposes.

4. THE COORDINATE SYSTEMS

An elliptic curve can be represented by several coordinate systems [11]. Following are descriptions of three coordinates, i.e. affine coordinate, standard projective coordinate, and Jacobian projective coordinate procedures.

4.1 Affine coordinate:

Let E an elliptic curve over GF(p), has the following equation:
E: $y^2=x^3+ax+b$ (mod p) , where a and b are constants satisfying $4a^3+27b^2\neq 0 \ (mod\ p)$.
Let $P=(x_1,y_1)$, $Q=(x_2,y_2)$, and $P+Q=(x_3,y_3)$, be points of E(GF(p)) ,

Addition formula: $x_3=\lambda^2-x_1-x_2$, $y_3=\lambda(x_1-x_3)-y_1$, where $\lambda=(y_2-y_1)/(x_1-x_2)$
Doubling formula: $x_3=\lambda^2-2x_1$, $y_3=\lambda(x_1-x_3)-y_1$, where $\lambda=(3x_1^2+a)/(2y_1)$

Addition time = 3 M + 6 S + 1 Inversion
Doubling time = 3 M + 4 S + 1 Inversion

4.2 Standard Projective coordinate:

For standard projective coordinates, we set x=X/Y and y=Y/Z, giving the equation:

$$Ep : Y^2Z=X^3+aXZ^2+bZ^3 (mod\ p) ,$$

Let $P=(X_1,Y_1,Z_1)$, $Q=(X_2,Y_2,Z_2)$ and $P+Q=(X_3,Y_3,Z_3)$ be points of E(GF(p)) ,

Addition formula: $X_3= vA$, $Y_3= u(v^2X_1Z_2 - A) - v^3Y_1Z_2$, $Z_3=v^3Z_1Z_2$
where, $u = Y_2Z_1 - Y_1Z_2$, $v = X_2Z_1 - X_1Z_2$, $A = u^2Z_1Z_2 - v^3 - 2v^2Y_1Z_2$

Doubling formula: $X_3= 2hs$, $Y_3= w(4B-h) - 8s^2Y_1^2$, $Z_3=8s^3$
where, $w = aZ_1^2+3X_1^2$, $s = Y_1Z_1$, $B = X_1Y_1s$, $h = w^2-8B$

Addition time = 12 M + 2 S
Doubling time = 7 M + 5 S

4.3 Jacobian coordinate:

For Jacobian coordinates, we set $x=X/Z^2$ and $y= Y/Z^3$, giving the equation:

$$Ep : Y^2=X^3+aXZ^4+bZ^6 (\text{mod } p) ,$$

Let $P=(X_1,Y_1,Z_1)$, $Q=(X_2,Y_2,Z_2)$ and $P+Q=(X_3,Y_3,Z_3)$ be points of $E(GF(p))$,

Addition formula: $X_3=-H^3-2U_1H^2+r^2$, $Y_3=-S_1ZH^3+r(U_1H^2-X_3)$, $Z_3=HZ_1Z_2$
where, $U_1=X_1Z_2^2$, $U_2=X_2Z_1^2$, $S_1=Y_1Z_2^3$, $S_2=Y_2Z_1^3$, $H=U_2-U_1$, $r=S_2-S_1$

Doubling formula: $X_3=T$, $Y_3=-8Y_1^4+M(S-T)$, $Z_3=2Y_1Z_1$
where, $S = 4X_1Y_1^2$, $M=3X_1^2+aZ_1^2$, $T= -2S+M^2$

Addition time = 12 M + 4 S
Doubling time = 4 M + 6 S

5. PREVIOUS DESIGNS

In [24], the two projective coordinates, standard (Section 4.2) and Jacobian (Section 4.3), have been implemented using parallel architecture, as shown in Figure 1. This design uses four modular multipliers to process inputs according to a proposed specific data path. The data path for projective coordinates addition and doubling are shown in Figure 2 and Figure 4, respectively. Similarly, Figure 3 and Figure 5 show respectively, the data path for addition and doubling, when Jacobian coordinates are implemented. Since the addition/subtraction is very fast compared to multiplication, one adder is used in this design.

The two implementations of the coordinates were compared in [24] with respect to their critical paths and their hardware utilization. The study resulted in choosing the standard projective coordinate as the appropriate efficient choice for parallel designs.

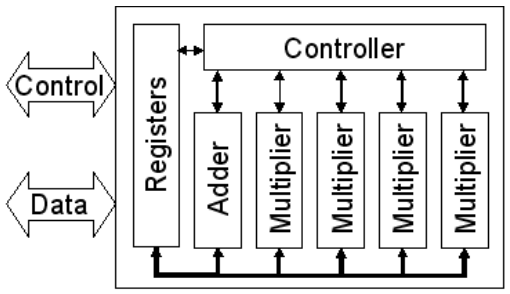

FIGURE 1: Elliptic Curve Processor Architecture [1]

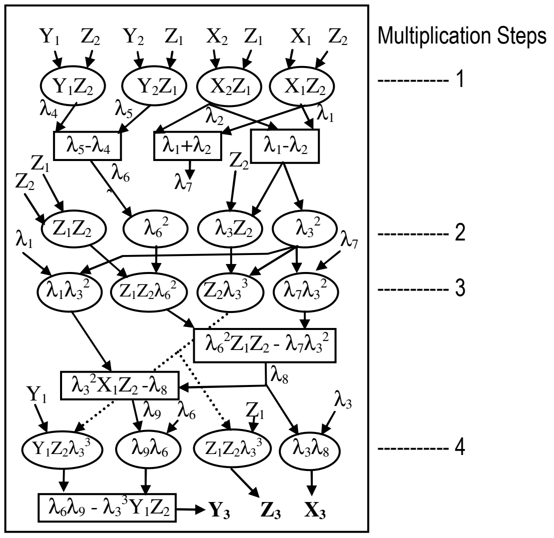

FIGURE 2: Projecting *(X,Y)* To *(X/Z,Y/Z)* Adding Two Points Data Flow

6. PROPOSED DESIGN

The work proposed here found some interesting benefit from improving the Jacobian coordinate system for parallel hardware designing. Since ECC point add operation is not needed all times, i.e. required based on the value of scalar k (Section 3); and this add operation involves more modular multiplications than point double, we propose to transform some of the modular multiplications needed by the point addition procedure to be pre-computed within the point doubling phase. The idea also makes the most usage of the hardware by allowing the multipliers not used in the last stages of Jacobian point double operation (Figure 5) to be fully utilized. In fact, the hardware of Figure 1 is modified by proper scheduling and adding one more adder makes the remodeled point doubling operation computed in three multiplication times, as shown in Figure 6. The hardware is modified but not much, i.e. the hardware components will be normal with regard to four modular multipliers but adjusted with two modular adders instead of one.

Our idea is to utilize the unused multipliers of the original parallel Jacobian procedure shown in Figure 5, to prepare some pre-multiplications, that may be needed later for the next operation, i.e. may be needed by if point addition operation is required. These pre-multiplications are $U_1=X_3Z_2^2$ and $U_2=X_2Z_3^2$, as shown in Figure 6. As the Jacobian point doubling is rescheduled on three multiplication steps, interestingly the related point adding is rescheduled to be performed in three multiplication times too, as shown in Figure 7.

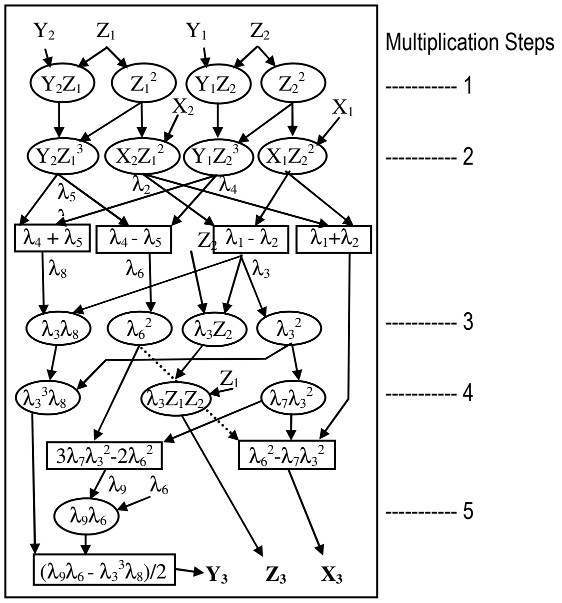

FIGURE 3: Jacobian Projecting *(X,Y)* To *$(X/Z^2,Y/Z^3)$* Adding Points Data Flow

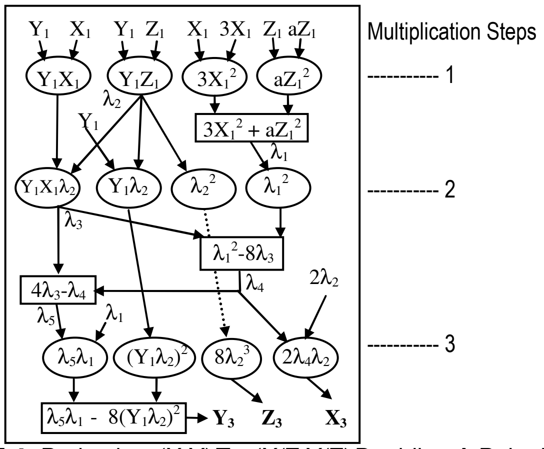

FIGURE 4: Projecting *(X,Y)* To *(X/Z,Y/Z)* Doubling A Point Data Flow

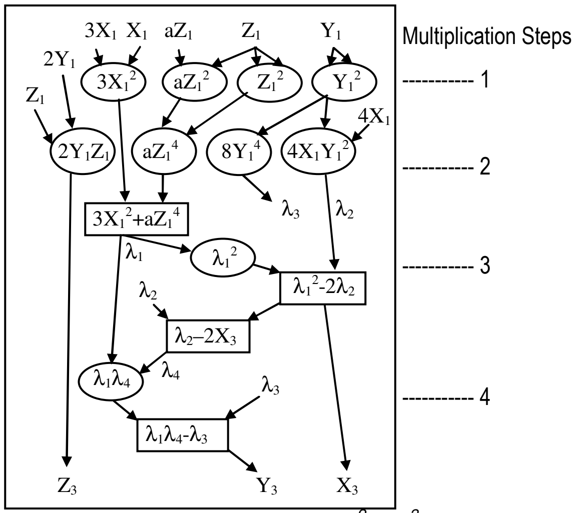

FIGURE 5: Jacobian Projecting *(X,Y)* To *(X/Z², Y/Z³)* Doubling A Point Data Flow

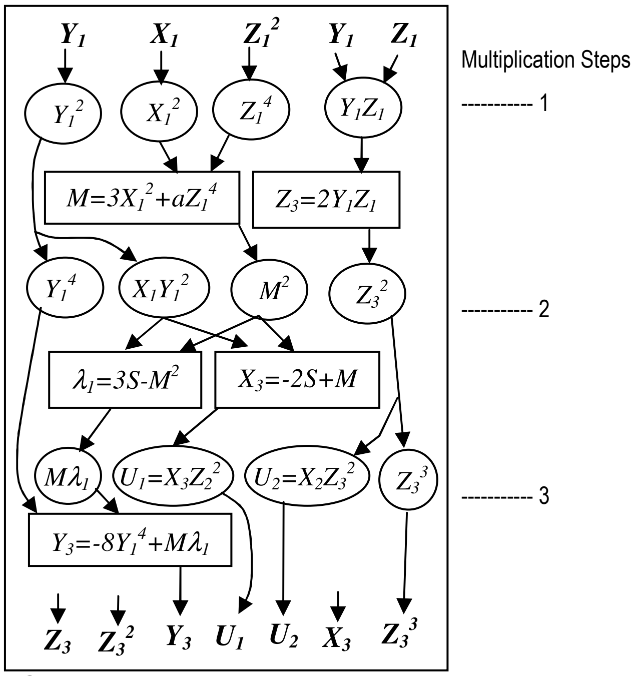

FIGURE 6: Proposed Data Flow For Doubling A Projective Point

The top level overview of the new ECC procedure and the complete data transfer is shown in Figure 8. Observe that all second point values (X_2, Y_2, Z_2, Z_2^2, Z_2^3) are initially not needed by the point doubling making them stored until their tern comes in the point addition. The F condition depends on the scalar value *k* to direct the procedure for point addition operation, if needed. The pre-computed values, U_1 and U_2, are needed only for point addition operation. If the *k* value directs the function F not to run point addition, the pre-computed values are to be ignored. Some variable are reallocated or reassigned to other registers after point doubling and point adding operations. These are found essential for proper mapping of data within the remodeled procedure to operate correct and efficient.

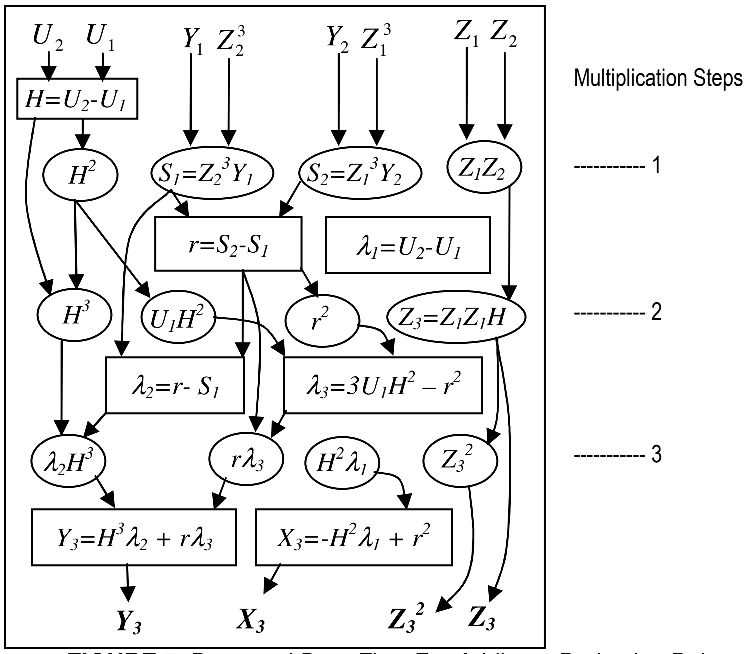

FIGURE 7: Proposed Data Flow For Adding a Projective Point

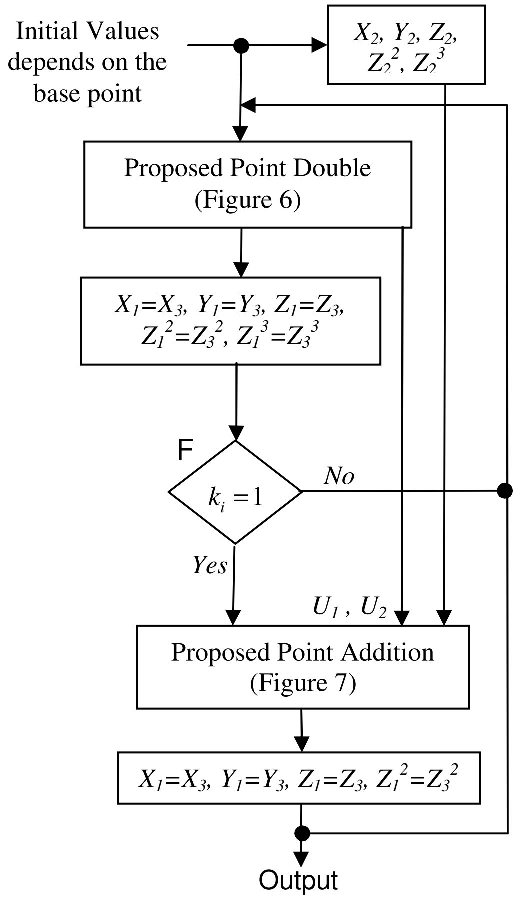
Initial Values depends on the base point
$X_2, Y_2, Z_2, Z_2^2, Z_2^3$
Proposed Point Double (Figure 6)
$X_1=X_3, Y_1=Y_3, Z_1=Z_3, Z_1^2=Z_3^2, Z_1^3=Z_3^3$
F
$k_i=1$
No
Yes
U_1, U_2
Proposed Point Addition (Figure 7)
$X_1=X_3, Y_1=Y_3, Z_1=Z_3, Z_1^2=Z_3^2$
Output

FIGURE 8: Overview of Proposed Design Flow

7. HARDWARE IMPLEMENTATION

The purpose of the hardware implementation is to give some common platform and fair comparison between our proposed architecture and similar previous designs. The focus in this study is not targeted toward the details of the architecture implementation; instead our aim is to extract the hardware time and area parameters of the main blocks to build a fair comparison study between the designs. Therefore, our implementation exploration here is going to be limited to the level needed to serve this comparison goal.

We will implement the basic blocks of hardware that are commonly used to build all studied designs, i.e. our model here as well as similar previous architectures. The major common components needed by all designs are modular multiplier and modular adder. We described these designs in VHDL and synthesized them for Xilinx Spartan-3 FPGAs. The implementation features of the two basic components are detailed in this section.

7.1 Modular Multiplier Implementation:

The modular multiplier is designed to run Montgomery multiplication in binary format, which is proven to be the efficient operation, similar in principle to the work in [21,22,23]. The Montgomery multiplier algorithm is expressed as the following:

```
$s_0 = 0$
$for\ \ i = 0\ \ to\ \ n\ \ do$
        $q_i = S_i \mod 2^k$
        $S_{i+1} = (S_i + q_i \cdot M)/2 + b_i \cdot A$
    $end$
```

The main operation running this Montgomery multiplication is simply modular addition. This made the multiplier algorithm implemented using two cascaded Carry save adders (CSA) connected as shown in Figure 9. We found that these CSA elements are the fastest components in our implemented system; which made the decision of adjusting our system clock. The clock is dominated to the run the signals through two cascaded CSA plus their registering delay. For example, the time needed for an $n \times n$ multiplier to operate is: $T_{mod\_mul} = n \times CLK + 3 \times CLK$; where the additional term: $3 \times CLK$ is to compensate for the final propagate adder. The study assumed that the hardware number of bits used to be *160-bits*, as needed by practical applications of ECC [5,6]. The results showed that for a 160×160 multiplier running with a clock period of *12ns*, the multiplication can be performed in around *2us*.

The *CLK* period is set to be equal to the longest path delay required by an iteration, which is:
$CLK = T_{iteration} = 2T_{CSA} + T_{REG}$
So, the total delay for an $n \times n$ modular multiplier becomes: $T_{total} = n(2T_{CSA} + T_{REG}) + T_{PCA}$

The multiplier is designed, synthesized and tested through VHDL. It is implemented on FPGA using flexible parameterizble features. The design word size is arranged to be an input parameter (*W*) to the synthesizer. The VHDL code is first complied using value of *W=4*, which is used to build a functional (behavioral) simulation platform. Then, a timing simulation (after place and route) is tested. After that, the design is complied for *W=160-bits*, and the time required for the multiplication result to be ready is computed. The multiplication time found through this process is *2.2usec*, i.e. for our *160-bits* experimentation.

7.2 Modular Addition Implementation:

During the ECC point add/double operations (Figures 6 and 7), an extra addition hardware module is needed beside the multipliers. This modular adder involves several hardware units, such as a controller, counter, shifter, Carry Propagate Adder (CPA), accumulator (ACC), and a multipliexor (Mux), connected as shown in Figure 10. Some occasions required the addition operations to involve adding more than two operands, which lead the adder design to be optimized as follows:

- First, the register of the accumulator (ACC) initializes its results to zero.
- Then, all the required operands are added into ACC.
- Finaly, ACC is reduced to accommodate the modular GF(p) requirement.

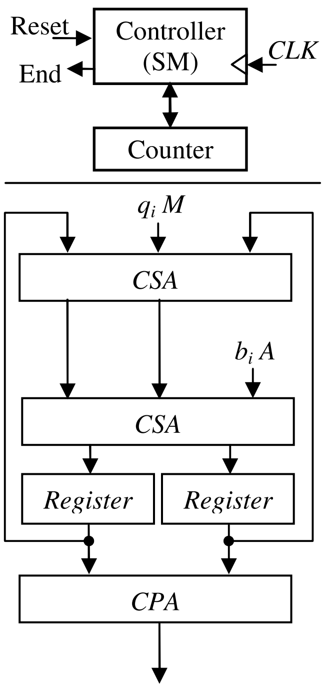

FIGURE 9: Block Diagram of the Modular Multiplier Unit

For example, when I want this modular addition unit to compute the following:
R = (a + 5×b) mod M
the following operation sequence are executed:

- ACC = 0 (at this stage registers of ACC are initialized to zero)
- ACC = ACC + *b* (at this stage ACC equals *b*)
- ACC = ACC+ *4 × b* (simply *b* shifted by two bits and added to *b* making it equal to *5b*)
- ACC = ACC + *a*
- Reduce ACC (compute ACC *mod M*)

By measuring and analyzing the waveform of the VHDL simulation for the *160-bits* modular adder in relation to the modular multiplication, we found that the time needed by the modular addition stage can be represented as: $T_{mod\_add}=0.18\times T_{mod\_mul}$

8. COMPARISONS AND ANALYSIS

Using the implemented multiplication and addition units (Section 7) three ECC designs are compared. We compared the proposed design with two existing similar designs are all studied in relation to their area and time, as showed in Table 1. Since the basic components are the same implemented in FPGA, the comparison is believed to be fair and very close to reality. The study considered the area and timing of the internal registers, which cannot be avoided; an *n-bits* CSA is identical to an *n-bits* register. To make our study consistent with the previous study in [24], we assume the basic hardware unit as the multiplier. All other units are quantified relative to this multiplier unit, as follows:

- One *n-bits* Register ≈ 0.13 *n×n* multiplier

- One *n-bits* Adder ≈ 0.40 *n×n* multiplier

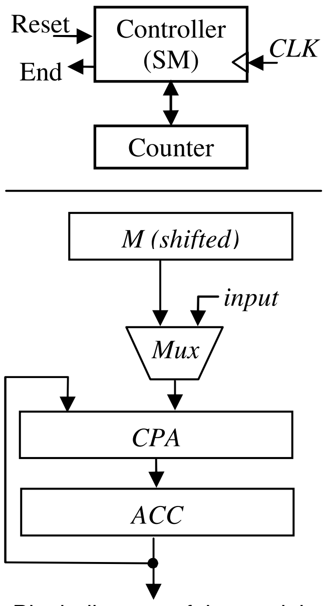

FIGURE 10: Block diagram of the modular addition unit

The timing (average cycles multiplication time) comparison counts the addition stages as well as the average multiplication number of cycles (see Table 1); i.e. the multiplication was the only factor in [24] where we added addition timing in this study for more realistic study. The average multiplication number of cycles is computed based on point addition and point doubling according to the ECC binary scalar multiplication algorithm described in Section 3. The multiplication time is computed by adding the total number of multiplications of point doubling procedure plus half the number of multiplications of point addition procedure. The point adding is half the point doubling assuming the value *k* in binary as half ones and half zeros as an average statistical data assumption (Section 3).

Design		Sequential Projective[6]	Parallel Standard Projective[1]	Parallel Jacobian Projective[1]	Design (One Adder)	Proposed Design (Two Adders)
Number of	Multipliers	1	4	4	4	4
	Adder	1	1	1	1	2
	Registers	6	3+3+6 =12	4+4+1 =9	5+5+6 =16	5+5+6 =16
Total Area (multiplier size)		2.18	5.96	5.57	6.48	6.88
Average Cycles (Multiplication time)		18.18	5+1.26 =6.26	6.5+1.35 =7.85	4.5+1.53 =6.03	4.5+0.9 =5.4
Cost (different figure of merit values)	AT	40	37	44	39	37
	AT^2	721	234	343	236	201
	A^2T	86.4	222	244	253	256

TABLE 1: Comparison Between Different Designs

In this study, the area is figured by a number related to the number of multipliers, i.e. the adders and registers are given an area estimate relative to the multipliers which totals up to an area factor used for comparison reasons. All architectures are designed for *160 bits* crypto calculations, which is the common number of bits needed by most applications [5,6].

The area factor and timing average estimate will be multiplied together to generate different cost figures, as in Table 1. These cost figures are just simple figure of merit values to be used for evaluation reasons. For example, the cost AT (AT = A×T), assumes that time and area is having similar balanced importance to the application. When timing is more important, the cost figure of merit AT is assumed to be further multiplied by time T making it AT^2 (AT^2 = A×T×T). On similar concept but with allowing for the application to have more importance to area than time, we included in this study the cost A^2T, where the area is squared multiplied by the timing once. This new A^2T cost is believed to be needed for applications with very limited hardware area such as smart cards and small mobile devices, where area is more important than speed.

Based on the cost values, *AT*, AT^2, and A^2T, an appropriate design can be preferred. All cost figures for all designs are plotted in Figure 11, with some figures rescaled to fit in the graph. The benefit of the cost comparison is chose the preferred design and not in the cost figure value. Our proposed hardware is showing to have the best cost when time is having more priority over area, i.e. for AT^2. When the area and time are having the same importance (*AT* cost), our proposed design is having similar cost to the parallel standard hardware as our design in [24], which is also the lowest preferred value. Here the proposed design is a bit faster the old design but with the benefit I n speed compensated for in the hardware area.

The sequential hardware is preferred over all parallel designs when area is having more importance than time, as shown in A^2T cost plotting in Figure 11. This remark is also including our proposed design in this study, which can be a drawback of this parallel design whenever the hardware area is important much more than the speed.

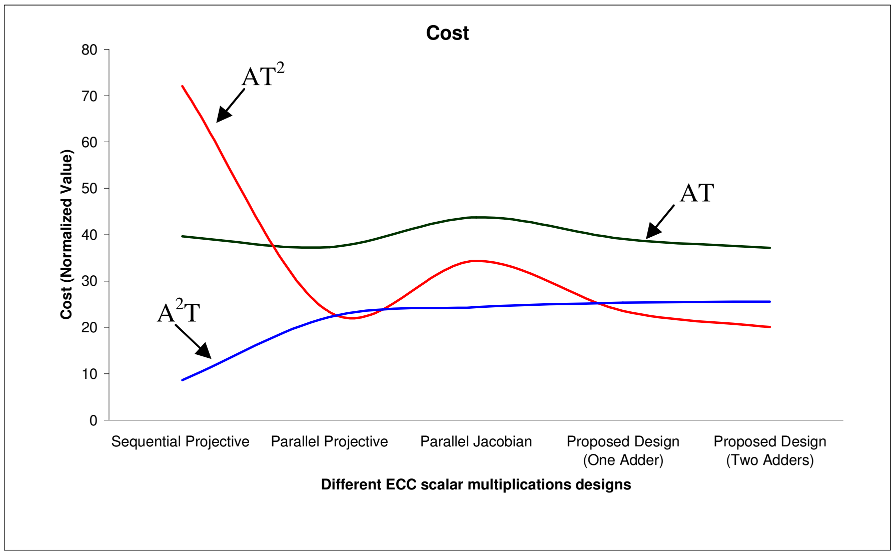

FIGURE 11: Different Cost Comparison of All Designs (rescaled to fit in the figure)

9. CONCLUSION

This study targeted speeding-up elliptic curve crypto (ECC) computations. We focused on ECC scalar multiplications adopting projective coordinates to reduce the inversion complexity effect. The study proposed remodeling Jacobian projective coordinate system tuned for parallel hardware implementation. We proposed merging ECC point adding and point doubling operations as a new modified method. The proposed hardware is similar to previous designs of four multipliers and an adder but with one more adder making it involve two addition units.

The new architecture is compared to existing scalar multiplication designs. All designs' basic units are implemented similarly on FPGA to insure fair comparison and area time cost analysis. The cost evaluation involved three studies, i.e. *AT*, AT^2 and A^2T. Our proposed design showed interesting performance results for *AT* and AT^2 costs. The clear improvement is shown in the AT^2 cost, where area is not as important as the computation timing. We concluded that implementing the proposed Jacobian coordinate using four multipliers and two adders, yields better AT^2 cost than existing scalar multiplication designs. The study is attractive for researchers to observe promising direction behind this research idea.

ACKNOWLEDGMENTS

The author is grateful to *Professor Mohammad K. Ibrahim* for all his beneficial ideas and engorgements. Thanks to the COE 509: Applied Crypto Systems students, i.e. *Mr Esa Alghonaim* for his contribution in VHDL coding and *Mr Aleem Alvi* for his proofreading and feedback related to this research. Thanks to King Fahd University of Petroleum and Minerals (KFUPM) and Umm Al-Qura University (UQU) for supporting all this work.

REFERENCES

[1] N. Koblitz, "Elliptic curve cryptosystems", In Mathematics of Computation, volume 48, pages 203–209, 1987.

[2] V. Miller, "Use of elliptic curves in cryptography", *Advances in Cryptology—CRYPTO'85*, Vol. 218 of Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1986.

[3] J.H. Cheon, H.J. Kim, S.G. Hahn, "Elliptic curve discrete logarithm and integer factorization", The Math Net Korea, Information Center for Mathematical Sciences (ICMS), February 7, 1999, http://mathnet.kaist.ac.kr/

[4] A Certicom Whitepaper, "The Elliptic Curve Cryptosystem", July 2000, http://www.certicom.com/

[5] Hitchcock, Yvonne Roslyn, "Elliptic Curve Cryptography for Lightweight Applications", *Institution Queensland University of Technology*, 2003. http://adt.library.qut.edu.au/adt-qut/public/adt-QUT20040723.150510/

[6] Naoya Torii and Kazuhiro Yokoyama, "Elliptic Curve Cryptosystem", *FUJITSU Sci. Tech. Journal*, Vol. 36, No. 2, pages 140-146, December 2000. www.fujitsu.com/downloads/MAG/vol36-2/paper05.pdf

[7] O. Al-Khaleel, C. Papachristou, F. Wolff, K. Pekmestzi, "An Elliptic Curve Cryptosystem Design Based on FPGA Pipeline Folding", *13th IEEE International On-Line Testing Symposium, IOLTS 07*, pages 71 – 78, 8-11 July 2007.

[8] A.J. Menezes, T. Okamoto, S.A. Vanstone, S, "Reducing elliptic curve logarithms to logarithms in a finite field", *IEEE Transactions on Information Theory*, Volume 39, Issue 5, pages 1639 – 1646, Sept. 1993.

[9] T. Hasegawa, J. Nakajima, M. Matsui, "A practical implementation of elliptic curve cryptosystems over GF(p) on a 16-bit microcomputer", *In Public Key Cryptography – PKC '98, Proceedings*, volume 1431 of Lecture Notes in Computer Science, pages 182–194, Springer-Verlag, 1998.

[10] Scott Vanstone, "Crypto Column: The Importance of Good Crypto and Security Standards", Code & Cipher- Certicom's Bulletin of Security and Cryptography, Volume 1, Issue 4, 2004, http://www.certicom.com/codeandcipher

[11] A. Daly, W. Marnane, "Efficient Architectures for implementing Montgomery Modular

Multiplication and RSA Modular Exponentiation on Reconfigurable Logic", *Proceedings of the ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, pages: 40 - 49, Monterey, California, USA, 2002

[12] G.B. Agnew, R.C. Mullin, S.A. Vanstone, "An implementation of elliptic curve cryptosystems over F2^{155}", *IEEE Journal on Selected Areas in Communications*, Volume 11, Issue 5, pages 804 – 813, June 1993

[13] Martin Christopher Rosner, "Elliptic Curve Cryptosystems on Reconfigurable Hardware", *MS Thesis submitted to Electrical Engineering in Worcester Polytecnic Institute*, U.S.A., 1998.

[14] I. Blake, G. Seroussi, N.P. Smart., “Elliptic Curves in Cryptography”, London Mathematical Society, Lecture Note Series. Cambridge University Press, 1999.

[15] M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, J. Shokrollahi, "Tradeoff analysis of FPGA based elliptic curve cryptography", *IEEE International Symposium on Circuits and Systems, ISCAS 2002*, Vol. 5, pages 797 – 800, 26-29 May 2002.

[16] N.A. Saqib, F. Rodriguez-Henriquez, A. Diaz-Perez, "A Parallel Architecture for Computing Scalar Multiplication on Hessian Elliptic Curves", *International Conference on Information Technology: Coding and Computing (ITCC'04)*, Vol. 2, pages 546–552, Las Vegas, NV, USA, 2004.

[17] Z. Dyka, P. Langendoerfer, "Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba's method", *Proceedings of Conference on Design, Automation and Test in Europe*, pages 70 – 75, 2005.

[18] T.F. Al-Somani, M.K. Ibrahim, "High Performance Elliptic Curve GF(2m) Cryptoprocessor Secure Against Timing Attacks", *International Journal of Computer Science and Network Security - IJCSNS*, Vol. 6, No.1B, pages 177-183, January 2006.

[19] T.F. Al-Somani, M. Ibrahim, A. Gutub, "Highly Efficient Elliptic Curve Crypto-Processor with Parallel GF(2m) Field Multipliers", *Journal of Computer Science (JCS)*, Vol. 2, No 5, pages 395-400, 2006.

[20] J. Fan, K. Sakiyama, I. Verbauwhede, "Elliptic Curve Cryptography on Embedded Multicore Systems," *Workshop on Embedded Systems Security - WESS*, pages 17-22, 2007. http://www.cosic.esat.kuleuven.be/publications/article-937.pdf

[21] G. Orlando, C. Paar, "A scalable GF(p) elliptic curve processor architecture for programmable hardware", *Third International Workshop on Cryptographic Hardware and Embedded Systems - CHES*, pages 348-363, Paris, France, 14-16 May 2001.

[22] G. Orlando, "Efficient Elliptic Curve Processor Architectures for Field Programmable Logic", *Ph.D. Thesis, Worcester Polytechnic Institute*, March 2002.

[23] S.B. Ors, L. Batina, B. Preneel, J. Vandewalle, "Hardware implementation of an elliptic curve processor over GF(p)", *IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 433 – 443, 24-26 June 2003.

[24] A. Gutub, M.K. Ibrahim, "High Radix Parallel Architecture For GF(p) Elliptic Curve Processor", *IEEE Conference on Acoustics, Speech, and Signal Processing - ICASSP 2003*, pages 625- 628, Hong Kong, April 6-10, 2003.

[25] B. Ansari, Huapeng Wu,"Parallel scalar multiplication for elliptic curve cryptosystems", *International Conference on Communications, Circuits and Systems*, Vol. 1, pages 71-73, 27-30 May 2005.

[26] F. Sozzani, G. Bertoni, S. Turcato, L. Breveglieri, "A parallelized design for an elliptic curve cryptosystem coprocessor", *International Conference on Information Technology: Coding and Computing - ITCC 2005*, Vol. 1, pages 626 – 630, 4-6 April 2005.

[27] Jun-Hong Chen, Ming-Der Shieh, Chien-Ming Wu,"Concurrent algorithm for high-speed point multiplication in elliptic curve cryptography", *IEEE International Symposium on Circuits and*

Systems - ISCAS, pages 5254 – 5257, 23-26 May 2005.

[28] S. Moon, J. Park, and Y. Lee, "Fast VLSI Algorithms for High-security Elliptic Curve Cryptographic Application", *IEEE Trans. on Consumer Electronics*, Vol. 47, No. 3, pages 700-708, 2001.

[29] P.M. Mishra,"Pipelined computation of scalar multiplication in elliptic curve cryptosystems (extended version)", *IEEE Transactions on Computers*, Vol. 55, No. 8, pages 1000 – 1010, Aug. 2006.

[30] W.N. Chelton, M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 16, No. 2, pages 198-205, Feb. 2008.

[31] W. Chelton, M. Benaissa, “High-speed pipelined ECC processor on FPGA”, *IEEE Workshop Signal Process. Syst. (SiPS)*, pages 136-141, Banff, Canada, 2006.

[32] P. Longa, A. Miri, "Fast and Flexible Elliptic Curve Point Arithmetic over Prime Fields", *IEEE Transactions on Computers*, Vol. 57 , No. 3, pages 289-302, March 2008.

[33] M.S. Anoop, "Elliptic Curve Cryptography, An Implementation Guide", online Implementation Tutorial, Tata Elxsi, India, 5 January 2007 http://www.infosecwriters.com/text_resources/pdf/Elliptic_Curve_AnnopMS.pdf

[34] D. Hankerson, A. Menezes, S. Vanstone, “Guide to Elliptic Curve Cryptography”, Springer-Verlag, ISBN 0-387-95273-X, 2004.