Design and Evaluation of Parallel , Scalable ,Curve Based Processor over Binary Field

RAHILA BILAL [1], Dr.M.RAJARAM [2]
[1] Department of ECE, [2] Department of EEE
[1] Anna University, [2] Anna University of Technology
[1] Chennai , [2] Tirunelveli
[1] INDIA, [2] INDIA
[1] bilalrahila@yahoo.co.in, [2] rajaramgct@rediffmail.com

Abstract: - Implementing Public-Key cryptography systems is a challenge for most application platforms when several factors have to be considered in selecting the implementation platform. Elliptic Curve Cryptography is considered much more suitable than other public-key algorithms. It uses lower power consumption, has higher performance and can be implemented on small areas that can be achieved by using ECC. In this work, scalable and parallel framework of FPGA based ,Dual Field (Prime and Binary Field) ECC processor is explored.Using Altera –Quartus software tool, a 160 bit ECC processor core with four 32 bit Arithmetic Units is evaluated on EP3SE50F780C3 .Scalar multiplication is performed in 445 μsecs and occupies 9763 LUT's .

Key-Words: - Public-Key cryptography ,ECC, Prime Field , Binary Field, FPGA, Scalar Multiplication.

1 Introduction

The incredible improvements in ubiquitous computing, and its indispensable implications gives rise to its being an effective domain of interest. As the notion of ubiquitous computing is becoming more and more part of life, various applications consisting of this new technology can be encountered.

PKC is indispensable for secure digital communications in security systems including high performance applications (*e.g.* ATMs) and low power applications (*e.g.* smart cards and RFID tags). In order to satisfy the performance requirements of different public-key cryptosystems, the hardware has to support modular operations over GF(p) or GF(2^m) with different operation sizes .

Moreover, it is preferable to have scalability in performance, that is, when allocating more hardware resources, a higher performance should be obtained. In this work, a flexible and scalable datapath for ECC processor is proposed. Having flexibility in the controller block is also necessary for public-key crypto systems to support different computational sequences in RSA and curve-based cryptography.

ECC depends on hard number theoretic problem: Elliptic Curve Discrete Logarithms (ECDL). At the base of ECC operations is finite field (Galois Field) algebra which focuses on prime Galois Fields (GF(p)) and binary extension Galois Fields (GF(2^m)). It is Standardized by NIST, ANSI and IEEE: NIST, NSA Suite B, ANSI X9.62, IEEE 1363, etc.

P

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. These designs operate over a fixed finite field. For example, the multiplier designed for 155 bits cannot be used for any other field of higher degree. When a need for a multiplication of larger precision arises 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 amount of effort in coding. Therefore, a scalable hardware module specifically tailored to take advantage of the concurrency of the Montgomery multiplication algorithm becomes extremely attractive.

Even though prime and binary extension fields, GF(p) and GF(2^m),have dissimilar properties, the elements of either field are represented using almost the same data structures inside the computer. Also, the algorithms for basic arithmetic operations in both fields have structural similarities allowing a unified module design methodology. For example, the steps of the Montgomery multiplication algorithm for binary extension field $GF(2^m)$ given in [9] only slightly differs from those of the integer Montgomery multiplication algorithm . Therefore, a scalable arithmetic module, which can be adjusted to operate in both types of fields, is feasible, provided that this extra functionality does not lead to an excessive increase in area or a dramatic decrease in speed.

Performance, security, size and versatility of ECC systems are a function of : finite field selection, elliptic curve type ,point representation type ,algorithms used ,protocol ,key size, hardware only, software only or mixed hardware-software implementations ,memory available (table lookups) and area.

The Experimental results show that the processor over Binary Field has high throughput , high speed and is compact in area.

Main contributions in this paper are summarized below,

- Simplified Hardware Architecture for Arithmetic Unit is introduced.
- A New Scheduling Unit is developed.
- Montgomery Multiplication Unit is modified.
- An Efficient DATA PATH for the processor is presented.
- .We Analyse the Design considerations such as the effect of Time , Area ,Power , Number of Arithmetic Units in Parallel etc., by supplying implementation results obtained by Altera Quartus Synthesis Tools.

2 Literature Survey

PUBLIC-KEY cryptosystems provide robust data security for vital applications such as private communications and services. Among them, elliptic curve cryptography (ECC) [4],[6],[9] has been regarded mature, having higher security strength, compared with other conventional public-key cryptosystems (e.g., RSA), when considering the same key length. However, ECC involves complicated finite-field arithmetic, i.e., a sequence of modular multiplications and additions with large numbers. ECC is based on point operations on elliptic curves (ECs) over a finite field, either prime field GF(p) or binary field GF(2^m). Many ECC designs have been published over specified finite fields [2],[3],[7],[10],[11],[12], either GF(p) or GF(2^m), especially because ECC over a specific GF(2^m) is fast and compact due to its carry-propagation-free nature. Recently, there have been more and more dual-field ECC designs addressing flexibility and scalability for widespread applications [7] [11], [12]. In addition, parallel ECC architectures with multiple arithmetic units [7],[10],[11] have been proposed to effectively reduce operation time, compared with serial ones. In this brief, the previous work on the two-phase scheduling methodology and a parallel ECC architecture [7] is extended, addressing the hardware architecture for realistic chip implementation, measurement, and characterization, and performance analysis when integrated with a practical system platform. In addition, to full fill efficient system applications, such as the elliptic curve digital signature algorithm (ECDSA) [1] and data encryption/ decryption schemes were done. Point double and point addition of López's projective coordinate [8] over $GF(2^m)$ were done in previous work. For hardware efficiency, the word-based Montgomery multiplication [5] is adopted for fast modular multiplication.

3 EC Arithmetic Operations

ECC manipulates points on the given EC to add or double them. Our processor focuses on the ECs over GF(2^m) specified in the IEEE 1363 Standard Specifications for Public-Key Cryptography . The standardized EC over GF(2^m) is

$$y^2 + xy = x^3 + \alpha x^2 + \beta,$$

where $x, y \in GF(2^m)$ and $\beta \neq 0$.

The most common point operation of ECC *is the point scalar multiplication, i.e.,*

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ times}},$$

where k is a scalar and P is a point on EC. We adopt the addition-and-subtraction method for the point scalar multiplication, which consists of iteratively point double and/or point double with point addition/ subtraction. Lopez projective coordinate $(x,y,z) \leftarrow (x/z, y/z^2)$ is used for GF(2^m).

Equations (1) and (2),summarize the computation of **point double** and **point addition** using López's projective coordinate over $GF(2^m)$.

$$\begin{aligned} z_q &= z_0^2 \times x_0^2 \\ x_q &= x_0^4 + \beta \times z_0^4 \\ y_q &= \beta \times z_0^4 \times z_q + x_q \times (\alpha \times z_q + \beta \times z_0^4) \end{aligned} \quad (1)$$

$$\begin{aligned} A &= y_1 \times z_q^2 + y_q \\ B &= x_1 \times z_q + x_q \\ C &= z_q \times B \\ D &= B^2 \times (C + \alpha \times z_q^2) \\ z_2 &= C^2 \\ E &= A \times C \\ x_2 &= A^2 + D + E \\ F &= x_2 + x_1 \times z_2 \\ G &= x_2 + y_1 \times z_2 \\ y_2 &= E \times F + z_2 \times G \end{aligned} \quad (2)$$

3.1 Montgomery Multiplier

Our work is focussed in the binary field. In the case of $GF(2^m)$, we use polynomials of degree at most $m-1$ with coefficients from the binary field $GF(2)$ to represent the field elements. Given two polynomials

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a1x + a_0 \quad (3)$$

$$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b1x + b_0 \quad (4)$$

and the irreducible polynomial of degree m

$$p(x) = x^m + p_{m-1}x^{m-1} + .. + \cdots + p1x + p_0 \quad (5)$$

generating the field $GF(2^m)$,the Montgomery multiplication of $A(x)$ and $B(x)$ is defined as the field element $C(x)$ which is given as

$$C(x) = A(x) \cdot B(x) \cdot R(x)^{-m} \pmod{p(x)} \quad (6)$$

The Montgomery image of a polynomial $A(x)$ is given as $A^*(x) = A(x) \cdot x^m \pmod{p(x)}$. Similarly, before performing Montgomery multiplication, the operands must be transformed into the Montgomery domain and the result must be transformed back. These transformations are accomplished using the pre-computed variable

$R^2(x) = x^{2m} \pmod{p(x)}$ as follows:

$$\begin{aligned} A^*(x) &= \text{MonMul}(A, R^2) \\ &= A(x) \cdot R^2(x) \cdot R^{-1}(x) \\ &= A(x) \cdot R(x) \pmod{p(x)} \quad (7) \\ B^*(x) &= \text{MonMul}(B, R^2) \\ &= B(x) \cdot R^2(x) \cdot R^{-1}(x) \\ &= B(x) \cdot R(x) \pmod{p(x)} \quad (8) \\ C(x) &= \text{MonMul}(C^*, 1) \\ &= C(x) \cdot R(x) \cdot R^{-1}(x) \\ &= C(x) \pmod{p(x)}. \quad (9) \end{aligned}$$

4 Elliptic Curve Processor

4.1 Overall Architecture

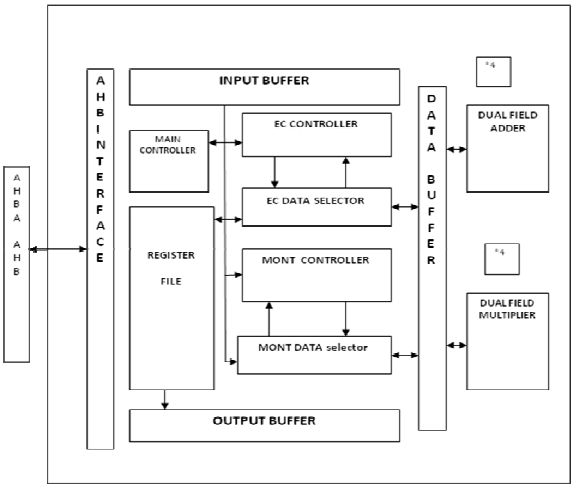

Fig 1 ECC PROCESSOR

The ECC instructions and data are fed into the input buffer through the standard advanced microcontroller bus architecture advanced high-

performance bus (AHB) interface. The main controller decodes the instructions that support comprehensive cryptographic functions, including the point coordinate conversion, point double, point addition, point scalar multiplication, Montgomery pre-/post processing, modular exponentiation, common finite-field arithmetic operations, and RSA basic operations. Then, the microinstructions are generated for the EC Controller to manipulate the data path, i.e., dual-field multipliers and adders. In addition, the Montgomery controller is used for efficient Montgomery multiplication. The dual-field multipliers and dual-field adders are capable of performing arithmetic over both the prime and binary fields by a unified hardware. Each intermediate variable during the EC operations stored in the register file. Finally, the output buffer stores the results, which can be accessed via the bus.

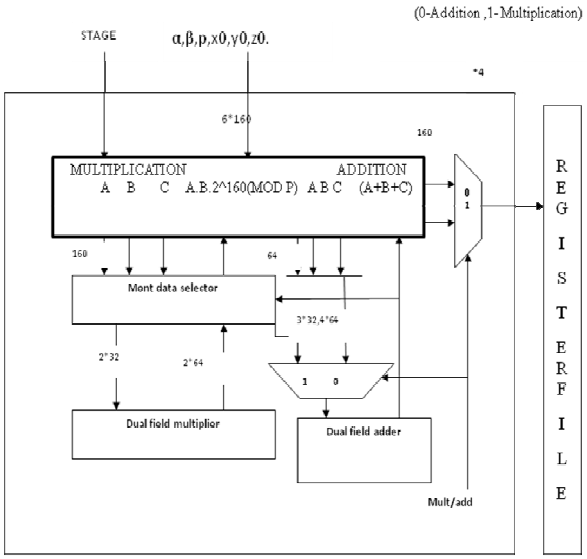

Fig 2 Data path of ECC Processor

The parallel architecture utilizes four 32-bit word-based multipliers and four 64-bit word-based adders. By applying the two-phase scheduling methodology the ECC throughput can be improved 2.5 times over GF(p) and 3.2 times over GF(2^m). Using the word-based approach, a scalable key size can be supported by simply extending the buffer size in the register file and the Montgomery data selector. Extending the key size requires the processor to complete the operation in more cycles. The area increases by about 28 K gates for the larger buffers. On the other hand, a larger word width can be used for the word-based arithmetic units. Let *m* be the data width of the field size and *r* be the data width of the multiplier; the number of cycles for the Montgomery multiplication is proportional to $(m/r)^2$. In addition, the number of cycles for EC operations (e.g., the EC point scalar multiplication and EC point addition) is proportional to that for the Montgomery multiplication. Therefore, the larger arithmetic unit can speed up the operation for the price of area overhead.

Fig 2 shows the proposed dual-field data path, which consists of the word-based dual-field adders, EC data selector, Montgomery data selector, and register file. The inputs include curve parameters α and β, the prime or irreducible polynomial *p*, and base point (x_0, y_0, z_0). The EC point scalar multiplication can be done by iteratively point double and/or point double with point addition. To accomplish point addition and doubling over the prime field and the same over the binary field, the EC controller decomposes the equations into a sequence of atomic operations with a single multiplication/addition. It manages the operation scheduling by control signal stage. For the addition phase, the EC data selector directly accesses the dual-field adders by control signal mul/add. The word-oriented partial results are then stored in the register file by mul/add as well. For the multiplication phase, the 160-bit operands are manipulated by the Montgomery data selector to perform the Montgomery multiplication, which consists of word-based multiplication and addition, via word index signal (w-index*)* from the Montgomery controller. At most seven 160-bit intermediate results are stored in the register file. The two levels of EC data selection and Montgomery data selection make the architecture highly scalable for different field sizes and word widths, and flexible for arbitrary EC parameters.

4.2 Algorithm

The basic EC arithmetic, e.g., the point double, addition, or subtraction, consists of a heterogeneous variety of primitive finite field operations, such as addition, subtraction, multiplication, and inversion. The EC arithmetic with traditional affine coordinate involves finite field inversion which is much more expensive than multiplication and addition. Our design adopts Jacobian's projective coordinate $(x,y,z)\rightarrow(x/z^2,y/z^3)$ over GF(P) to effectively replace the field inversion with several field multiplications. Furthermore, López's projective coordinate is used over GF(2^m) because there are fewer field operations in López's projective coordinate $(x,y,z)\rightarrow (x/z,y/z^2)$ than those in Jacobian's over GF(2^m). Despite the inversion, the finite field multiplication is critical among primitive field operations. Montgomery algorithm is

a well-known fast modular multiplication algorithm. Our design adopts finely integrated operand scanning (FIOS) Montgomery algorithm , as shown in Algorithm . It is a word-based algorithm for both GF(P) and GF(2^m) . Let m be the number of bits of the prime or the irreducible polynomial, and r be the word width of the multiplier, then w=(m/r) ,which is the number of words in an operand. In our design 5=160/32. Different field size can be supported by changing w accordingly. When adopting projective coordinate representation and Montgomery multiplication, the pre/post-processing is needed. The post-processing requires a Montgomery multiplication with the unity value 1. The conversion of the EC point between affine and projective coordinates are also required. Converting a EC point from affine coordinate to projective coordinate can be easily done, i.e.,

$\{x \leftarrow X, y \leftarrow Y, z \leftarrow 1\}$.

However, field inversions and multiplications are needed to to convert the point in the projective coordinate to affine coordinate,i.e.,

$\{X \leftarrow x/z^2, Y \leftarrow y/z^3\}$ over GF(P),and

$\{X \leftarrow x/z, Y \leftarrow y/z^2\}$ over GF(2^m).

Word Based Montgomery Multiplication Algorithm

Input: a,b,p,q

Output: $c=a\times b\times 2^{-m} \pmod p$

Steps:

C=0;

For i=0 to w-1 by +1 do

z=0;

$\{z, c_0\} = c_0 + a_1 \times b_0$;

$t = c_0 \times q \pmod{2^r}$;

$\{z, c_0\} = \{z, c_0\} + t \times p_0$;

for j to w-1 by +1 do

$\{z, c_j\} = c_j + a_i \times b_j + z$;

$\{z, c_{j-1}\} = \{z, c_j\} + t \times p_j$;

end for;

$c_{w-1} = z$;

end for;

4.3 Proposed Montgomery Multiplier Unit

Fig 3 shows the data path for the modified Montgomery Multiplier Unit. Four 160 bit inputs are a,b,p,q .The control signals are fieldsel, clk, en. Splitter is used to divide 160 bit number into five 32 bit numbers to perform word –based multiplication algorithm. It involves series of preprocess, Dual Multiplication Unit (Dual mult unit), Field Multiplexer (Field Mux), Dual Field Adder(DFA) and Dual Field Multiplxer (DFA Mux). After completion of Five iterations, the result cout is obtained.

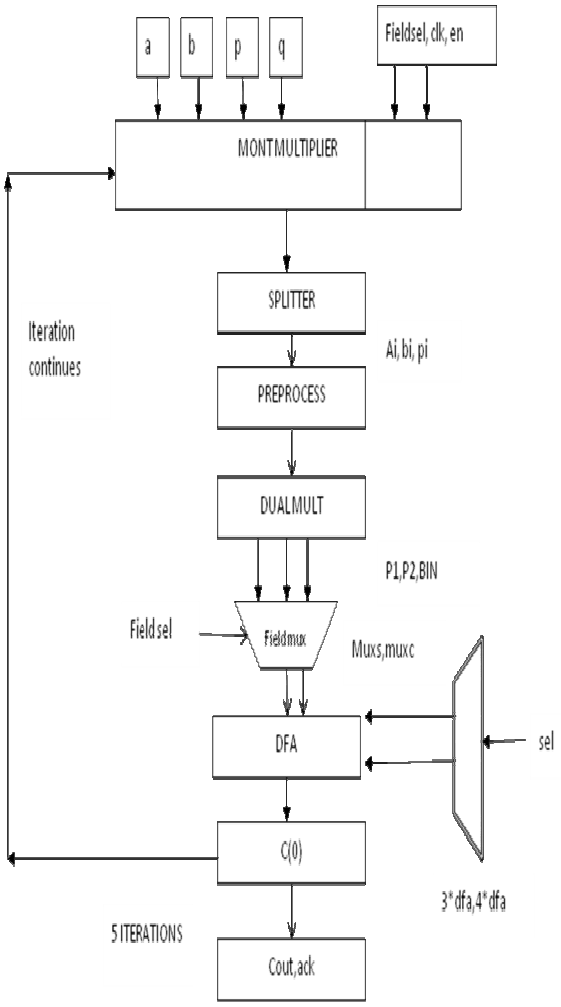

Fig 3 Modified Multiplier Unit

4.4 Proposed Arithmetic Unit

In this unit arithmetic operations are performed by either choosing Mont Multiplier or dfa_au unit. The inputs a,b,p,q are fed for both units. By using correct control signals sel, fieldsel, as_sel either Mulout or Addout is obtained.

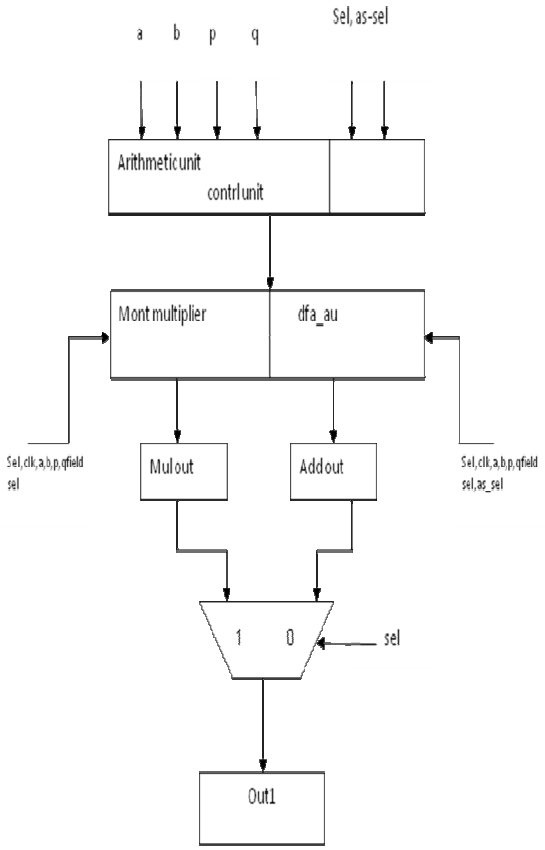

Fig 4 Proposed Arithmetic Unit

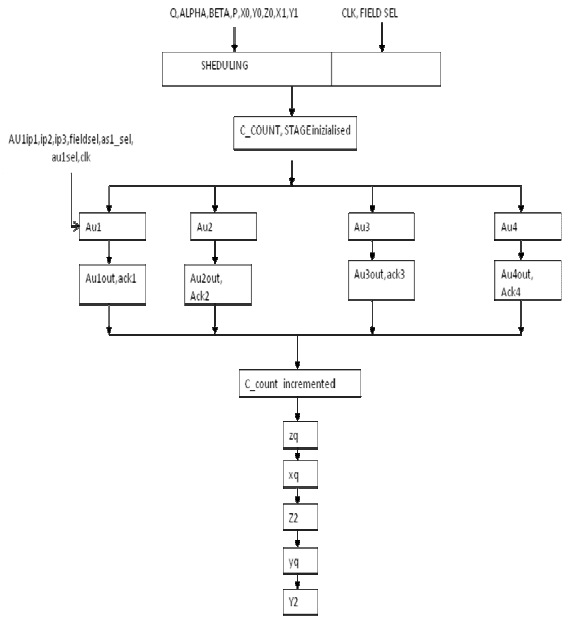

Fig 5 Proposed Scheduling Unit

In this scheduling unit,base point x0,y0,z0,x1,y1,curve parameters are fed to perform scalar multiplication.c_count and stage are incremented as operations are carried out in different Arithmetic Units (AU's) by their respective control signals.Final values obtained are listed in this Fig.5

5 Design Scheduling

The computation of ECC is decomposed into atomic finite field operations and optimized under the proposed parallel architecture. Based on the parallel architecture, scalable ECC processor with multiple AUs is presented.Once we have the specific hardware architecture, the design exploration can be done effectively with various design parameters, e.g., area, throughput, etc. Point scalar multiplication, the most crucial operation in our ECC processor, consists of repeated point double (PDBL) and point addition/subtraction (PADDSUB) that requires primitive finite field operations. Traditional serial ECC architectures utilized single finite field AU and addressed on its faster design. Recently, several parallel architectures tried to shorten the computation time with multiple AUs in a straightforward manner. A two-phase approach to schedule the primitive operations based on our parallel ECC architecture, which consists of the coarse-grained scheduling and fine-grained scheduling. With multiple AUs and the proposed methodology, successive iterations (i.e., PDBL-PDBL, PDBL-PADDSUB, or PADDSUB-PDBL) can be further folded up to reduce the operation time in the point scalar multiplication.

5.1. Coarse-Grained Scheduling

The coarse-grained (or global) scheduling is based on the data path scheduler using the integer linear programming (ILP, also known as LIP) technique, which can guarantee the optimal result under the given constraints. An example is used here for the illustration of coarse-grained scheduling approach. Suppose part of the EC point arithmetic over GF(p) is listed as follows:

$$\left.\begin{aligned} x_2 &= p-(x_0z_1^2+x_1z_0^2)(x_0z_1^2+x_1z_0^2)\text{^}2 \\ z_2 &= z_0z_1(x_0z_1^2-x_1z_0^2) \end{aligned}\right\} \quad (10)$$

Where $p_0=(x_0,y_0,z_0)$, $p_1=(x_1,y_1,z_1)$ and $p_2=(x_2,y_2,z_2)=p_0+p_1$. For the simplification only x_2 and z_2 are considered in the example, and we assume the intermediate value p in equation (10) is pre calculated and known in advance. The first step of the scheduling is to further decompose the EC arithmetic into atomic (or primitive) finite field

operations (e.g., the single multiplication, square, addition or subtraction). For this example, 11 atomic operations are obtained as shown in Table 1 . Suppose there are atomic field additions, subtractions, and multiplications in an EC arithmetic operation. Each can be labeled as O_i , where $1 \leq i \leq n$. Data precedence relation can be defined as $O_i \longrightarrow O_j$ if output of O_i is one of the inputs of O_j , i.e., O_i is the immediate predecessor of O_j. The start time (s_i) and require time (r_i) of ith atomic operation can be found by the data precedence relation. With each atomic operation taking one stage, several parameters are also defined:

1) N_s represents the number of stages.

2) N_{au} denotes the number of AUs in our ECC processor core.

3) $x_{i,j}$ is a zero-one variable. If O_i is scheduled in Stage j , $x_{i,j}$ =1, otherwise

$x_{i,j}$ =0.

Therefore, our scheduling becomes an ILP optimization subject to the following constraints:

$$\sum_{j=s_i}^{r_i} x_{i,j} = 1 \quad \forall 1 \leq i \leq n \tag{11}$$

$$\sum_{j=s_i}^{r_i} (j \times x_{i,j}) - \sum_{j=s_k}^{r_k} (j \times x_{k,j}) \leq -k \quad \forall o_i \rightarrow o_k \tag{12}$$

$$\sum_{i=1}^{n} x_{i,j} \leq N_{au} \quad \forall 1 \leq j \leq N_s \tag{13}$$

Equation (11) defines the mobility that O_i must be executed between the S_ith Stage and r_i th Stage. Equation (12) ensures that the precedence relations are preserved, where K is the number of stages required for executing O_i. We assume that each operation takes one stage and the K is assigned to be 1. Equation (13) constrains the maximum number of operations in each stage as the given number of AUs (i.e., N_{au}). Finally, our objective is to minimize N_s for the smallest number of stages (i.e.,T) with the given constraint of N_{au} (i.e.,A)—in other words, to obtain the highest throughput under the given parallel architecture. Our observation shows that when performing area or throughput optimization, the finite field addition and subtraction play a minor role as compared with the multiplication. Therefore our coarse grained scheduling focuses on the multiplications. Table 2 shows the data precedence relation of atomic operations in Table 1.

Table 1 Atomic operations of Equation 10

$o_1: q_0 = z_1^2$	$o_2: u_0 = x_0 \times q_0$	$o_3: q_1 = z_0^2$
$o_4: u_1 = x_1 \times q_1$	$o_5: w = u_0 \quad u_1$	$o_6: t = u_0 + u_1$
$o_7: q_2 = z_1 \times w$	$o_8: z_2 = z_0 \times q_2$	$o_9: q_3 = w^2$
$o_{10}: q_4 = t \times q_3$	$o_{11}: x_2 - p - q_4$	

Table 2 Data precedence relation of Table 1

$o_1 \rightarrow o_2$	$o_3 \rightarrow o_4$	$o_2 \rightarrow o_5$	$o_4 \rightarrow o_5$
$o_2 \rightarrow o_6$	$o_4 \rightarrow o_6$	$o_5 \rightarrow o_7$	$o_7 \rightarrow o_8$
$o_5 \rightarrow o_9$	$o_6 \rightarrow o_{10}$	$o_9 \rightarrow o_{10}$	$o_{10} \rightarrow o_{11}$

Table 3 Scheduling for GF(2^m) with 3 AU's

S	AU1	AU2	AU3
1	$* p_0 = x_0^2$	$* p_1 = z_0^2$	$* p_4 = y_0^2$
2	$* p_2 = p_0^2$	$* p_3 = p_1^2$	$* z_q = p_0 \times p_1$
3	$* p_5 = \beta \times p_3$ $* x_q = p_2 + p_5$	$* p_6 = \alpha \times z_q$ $* h = p_4 + p_5 + p_6$	$q_0 = x_1 \times z_q$
4	$* p_7 = x_q \times h$ $* y_q = p_7 + p_8$	$* p_8 = p_5 \times z_q$ $b = q_0 + x_q$	$q_1 = z_q^2$
5	$q_2 = b^2$ $a = q_5 + y_q$	$c = z_q \times b$	$q_5 = y_1 \times q_1$
6	$q_3 = \alpha \times q_1$ $q_4 = c + q_3$	$q_8 = a^2$	$z_2 = c^2$
7	$q_7 = y_1 \times z_2$ $x_2 = q_8 + d + e$	$d = q_2 \times q_4$	$e = a \times c$
8	$q_6 = x_1 \times z_2$ $f = x_2 + q_6$	$g = x_2 + q_7$	
9	$q_9 = z_2 \times g$ $y_2 = q_9 + q_{10}$	$q_{10} = e \times f$	

similarly the scheduling was performed with 2,3,and 4 arithmetic Units .Simulation and synthesis for all has be done and the snap shots are shown. To study the Performance Comparison of Binary with the Prime field , the Scheduling for the prime filed is also performed.

Table 4 Scheduling for GF(2^m) with 2 AU's

S	AU1	AU2
1	$* p_0 = x_0^2$	$* p_1 = z_0^2$
2	$* p_3 = p_1^2$	$* p_4 = y_0^2$
3	$* p_2 = p_0^2$	$* p_5 = \beta \times p_3$
4	$* z_q = p_0 \times p_1$ $* x_q = p_2 \mid p_5$	$* p_6 = \alpha \times z_q$ $* h = p_4 + p_5 + p_6$
5	$* p_7 = x_q \times h$ $* y_q = p_7 + p_8$	$* p_8 = p_5 \times z_q$
6	$q_0 = x_1 \times z_q$ $b = q_0 + x_q$	$q_1 = z_q^2$
7	$q_2 = b^2$	$c = z_q \times b$
8	$z_2 = c^2$ $q_4 = c + q_3$	$q_3 = \alpha \times q_1$
9	$d = q_2 \times q_4$	$q_6 = x_1 \times z_2$
10	$q_5 - y_1 \times q_1$ $a = q_5 + y_q$	$q_7 = y_1 \times z_2$
11	$e = a \times c$ $x_2 = q_8 + d + e$ $f = x_2 + q_6$	$q_8 = a^2$ $g = x_2 + q_7$
12	$q_9 = z_2 \times g$ $y_2 = q_9 + q_{10}$	$q_{10} = e \times f$

Table 5 shows the scheduling result for data flow diagram given in Fig 6. It has been realized using two AU's and four stages. The coarse-grained scheduling is applied to optimize the PDBL and mix-coordinate PADDSUB simultaneously. The scheduling results of the PDBL with PADDSUB over GF(2^m) and for GF(p) is also performed.

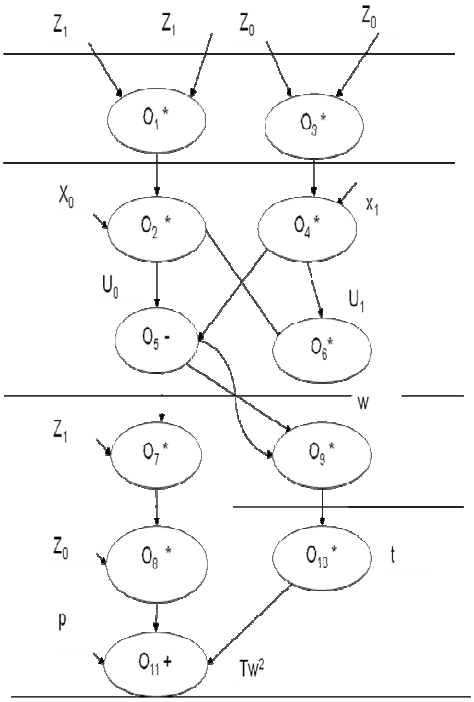

Fig 6 Data Flow Diagram

Table 5 Scheduling result for DFD.

S	AU1	AU2
1	$o_1: q_0 = z_1^2$	$o_3: q_1 = z_0^2$
2	$o_2: u_0 = x_0 \times q_0$ $o_5: w = u_0 - u_1$	$o_4: u_1 = x_1 \times q_1$ $o_6: t = u_0 + u_1$
3	$o_7: q_2 = z_1 \times w$	$o_9: q_3 = w^2$
4	$o_8: z_2 = z_0 \times q_2$ $o_{11}: x_2 = p - q_4$	$o_{10}: q_4 = t \times q_3$

Scheduling with more than four AUs cannot obtain further improvement. For a single stage, each AU (with one multiplier and one adder) performs one modular multiplication with at most two

modular additions and subtractions. We allow multiple additions and subtractions in a single stage because of their little cycle overhead as compared with the cycles of multiplication. The asterisk marks in the figures identify those operations belonging to the PDBL to produce (x_q,y_q,z_q). whereas the complete set of operations are to calculate the PDBL-PADDSUB, i.e.,(x_2,y_2,z_2) . The result has been summarized in Table 6

Table 6 comparison - coarse grained scheduling

FIELD	SHEDULING	N_{AU}			
		1	**2**	**3**	**4**
GF(P)	COARSE-GRAINED	**22**	**11**	**8**	**7**
GF(2^m)	COARSE-GRAINED	**24**	**12**	**9**	**7**

.

5.2. Fine-grained scheduling:

After the coarse-grained scheduling, several fine-grained (or detailed) scheduling techniques can be further applied, i.e.,Operand rescheduling, Atomic rescheduling and Loop folding.It is obvious that the computation of the PDBL is much simple than that of the PDBL-PADDSUB. For the illustration we redraw one iteration for the PDBL with four AUs (that perform atomic operations with asterisk marks), as shown in Table 7. The AUs are not fully utilized in this scheduling. For example,AU_3 and AU_4 are both idle at Stages 1, 3, 4, and 5 for the PDBL over GF(2^m) .

Therefore, the p_8 and y_q by AU_1 at the Stage 5 can be moved to Stage 4 and executed by while keeping the correct data precedence as shown in Table 8 .Therefore, the simple atomic rescheduling can reduce the stage number of the PDBL over from 5 to 4. As previously mentioned, the EC scalar multiplication consists of iteratively PDBL and PDBL-PADDSUB operations. We present here a loop folding technique to further improve the scheduling with four AUs. As shown in Table 7 after the atomic rescheduling, AU_3 and AU_4 are still idle at Stages 1, 3, and 4. If AU_3 and AU_4 at Stage 1 are used for the computation of P_0 and p_1 , as shown in Table 8, AU_3 and AU_4 at Stage 4 can also be used to compute the P_0 and p_1 of the next iteration no matter the successive iteration is the PDBL or PDBL-PADDSUB [i.e.,(PDBL)-(PDBL) or (PDBL)-(PDBL-PADDSUB)], because both of them also require only two AUs at Stage 1. Similarly, AU_3 and AU_4 at the last Stage of the PDBL-PADDSUB can be used to compute the P_0 and p_1 of the successive PDBL or PDBL-PADDSUB iteration [i.e., (PDBL-PADDSUB)-(PDBL) or (PDBL-PADDSUB)-(PDBL-PADDSUB)] over GF(2^m). Loop folding technique can be applied to the PDBL-PDBL and PADDSUB-PDBL over GF(p) with four AUs as well . The two consecutive iterations can be overlapped for one stage. This kind of loop folding technique, which is similar to the software pipelining, can efficiently improve the hardware utilization and throughput as long as no precedence violation occurs. For the scheduling with four AUs one stage can be effectively removed for each iteration. As a result, the minimal number of stages are obtained.

Table 7 Coarse grained scheduling for PDBL with 4 AU's over GF(2^m)

S	AU1	AU2	AU3	AU4
1	$*p_0 = x_0^2$			
2	$*p_2 = p_0^2$	$*z_q = p_0 \times p_1$	$*p_3 = p_1^2$	$*p_4 = y_0^2$
3	$*p_8 = \beta \times p_3$ $*x_q = p_2 + p_8$	$*p_6 = \alpha \times z_q$ $*h = p_4 + p_5 + p_6$		
4	$*p_7 = x_q \times h$			
5	$*p_8 = p_8 \times z_q$ $*y_q = p_7 + p_8$			

Table 8 Fine grained scheduling for PDBL with 4 AU's over GF(2^m)

S	AU1	AU2	AU3	AU4
1			$*p_0 - x_0^2$	$*p_1 = z_0^2$
2	$*p_2 = p_0^2$	$*z_q = p_0 \times p_1$	$*p_3 = p_1^2$	$*p_4 = y_0^2$
3	$*p_8 - \beta \times p_3$ $*x_q = p_2 + p_8$	$*p_6 = \alpha \times z_q$ $*h = p_4 + p_5 + p_6$		
4	$*p_7 = x_q \times h$	$*p_8 = p_8 \times z_q$ $*y_q = p_7 + p_8$	$*p_0 = x_0^2$	$*p_1 = z_0^2$

Darkened cells shows the beginning of next iteration. Thus in total, only 3 stages are required for point doubling operation with four AU's over GF(2^m).

6 Simulation Results

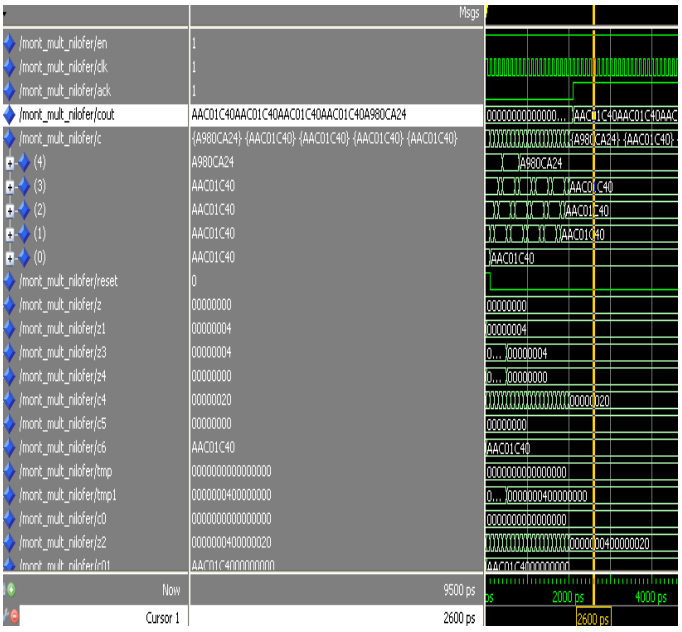

Fig 7 Montgomery Multiplier Unit(160 bit):

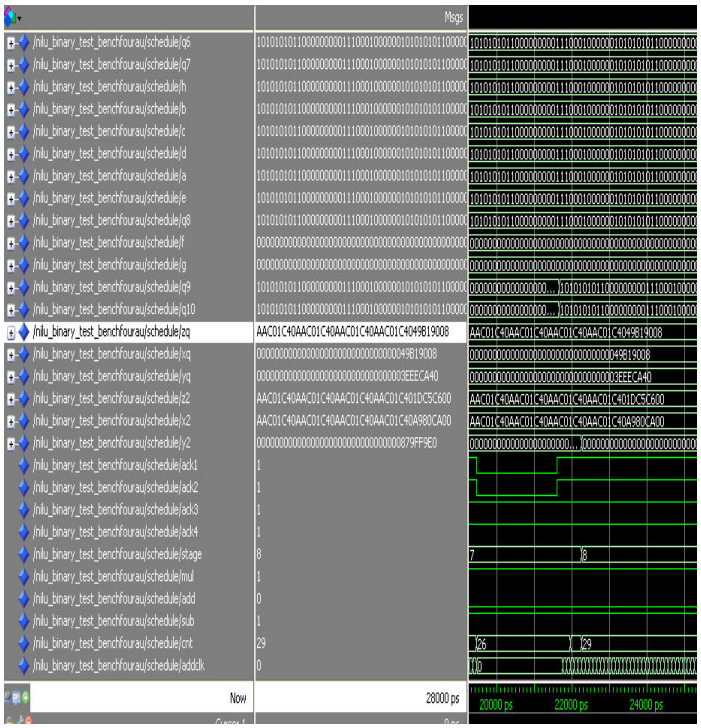

Fig 9 Scheduling with 4 AU-binary(160 bit):

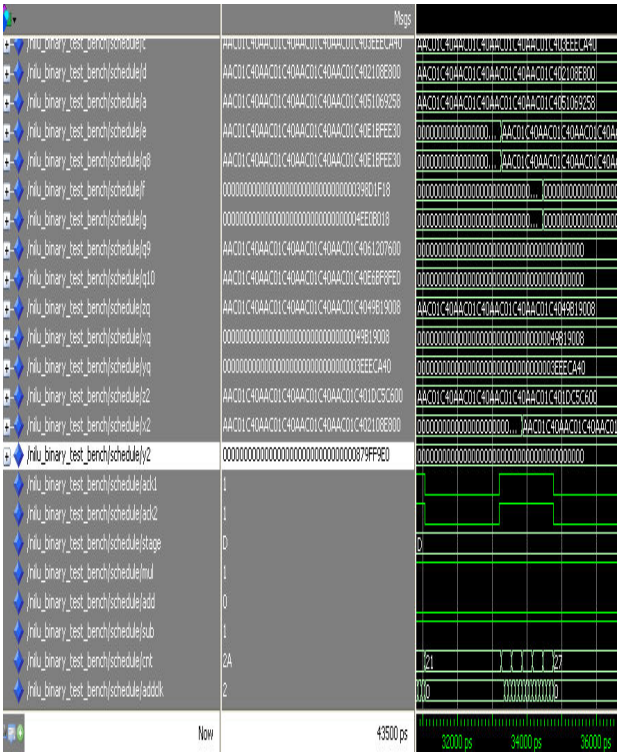

Fig 8 Scheduling with 2 AU-binary(160 bit):

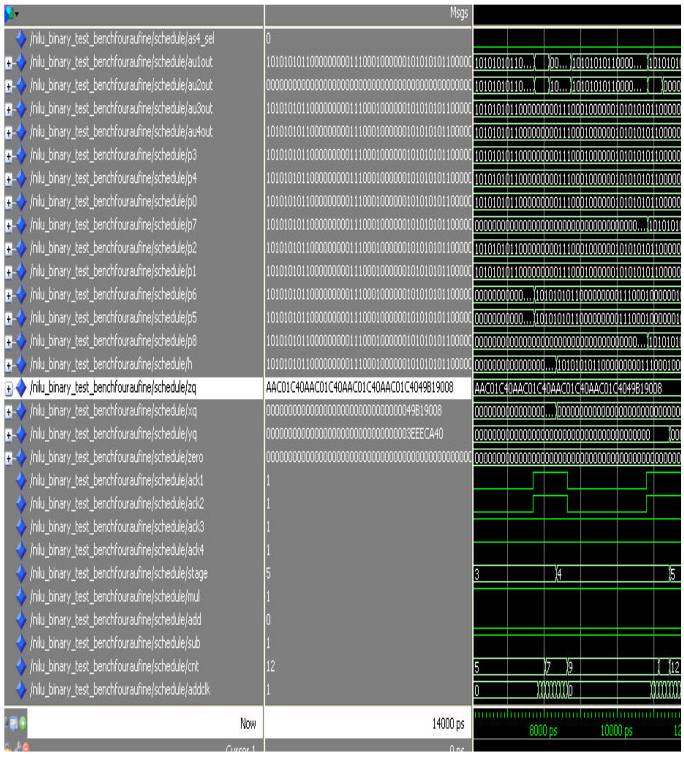

Fig 10 Scheduling with 4 AU(pd)-binary-fine grained(160 bit):

Table 9 Comparision of Various Designs - Scalar Multiplication

	OURS		A.SATOH AND K.TAKANO		K.SAKI YAMA		JYU-YUAN LAI	
PLATFOR	ALTERA-QUARTUS 10.1		0.13-µm CMOS		XILINX VIRTEX II PRO		XILINX VIRTEX II	
FIELD	160-BIT	GF (2^{160})	160-BIT	GF (2^{160})	160-bit	GF (2^{163})	160BIT	GF (2^{160})
CYCLES	27,257	30,028	282,000	177,000	103,594	83,901	74,021	54,319
FMAX	41.55	67.48	179	592	100	100	94.7	94.7
	656 µsec	445 µsec	1.71 ms	340 µsec	1.04 mse	574 µsec	782 µsec	574 µsec
PARALLEL	4, 32 BIT AU		1, 32-bit Mul		3, 160*4-bit MALU'S		4, 32bit AU	
AREA	14,075 LUT's	9763 LUT's	52,035 Gates	52,035 Gates	6 BRAM'S, 8954 SLICES		39,531 slices	39,531 slices

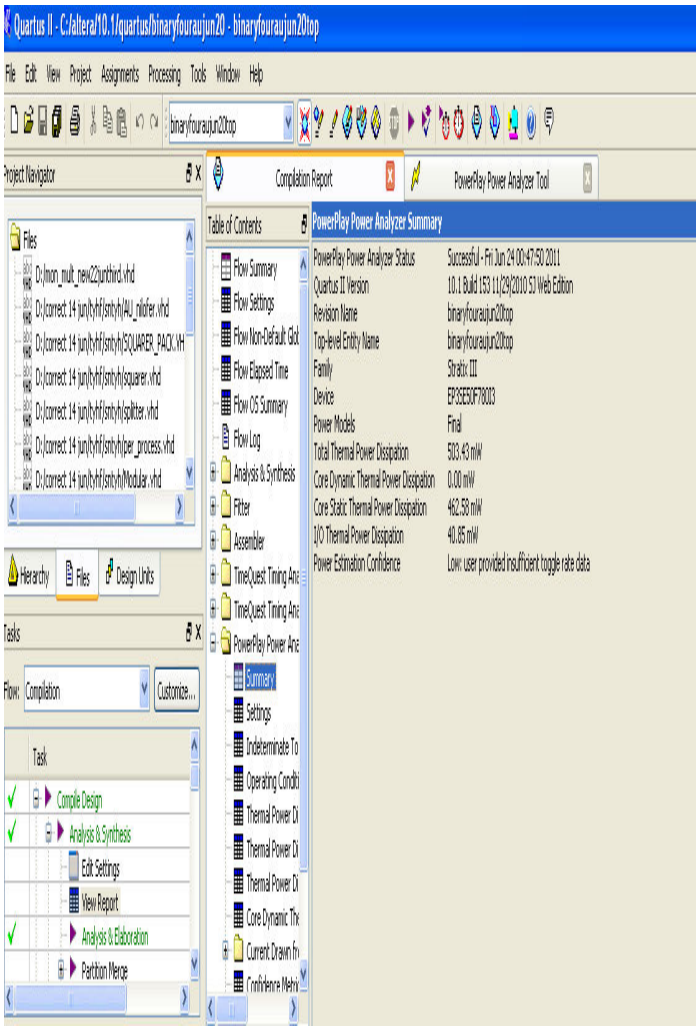

Fig 12 4 AU binary-power summary

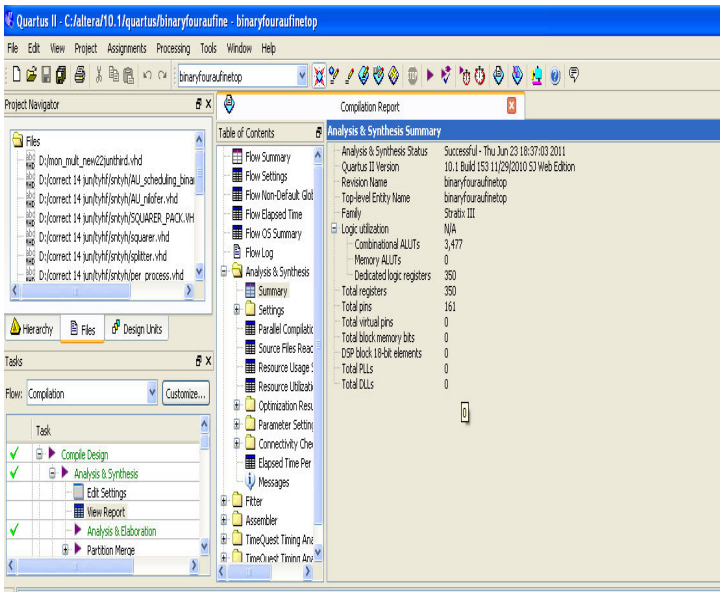

Fig 11 Au Binary-Fine Grain-Synthesis .

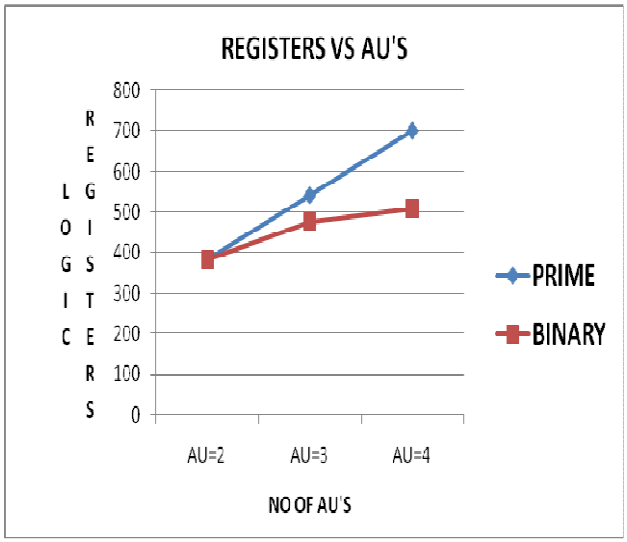

Fig 13 Scalar multiplication Space Complexity analysis

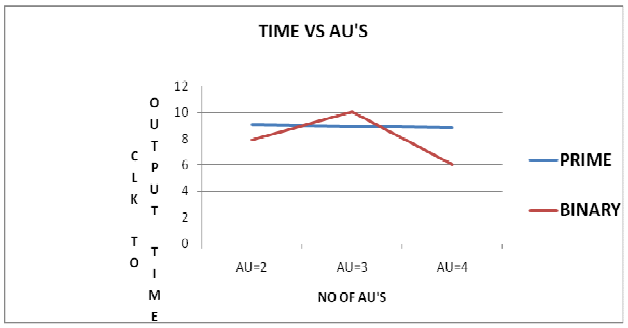

Fig 14. Clk to output time vs AU's

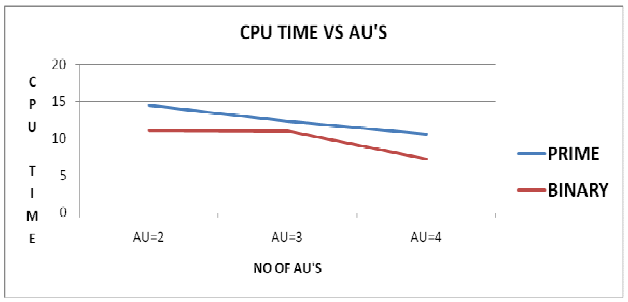

Fig 15 . CPU TIME vs AU'S

7 Conclusion

This paper presents a high-throughput Binary field elliptic-curve based crypto (ECC) processor that features all ECC functions with the programmable field and curve parameters over both prime and binary fields. The proposed ECC processor outperforms other ECC hardware designs in terms of functionality, scalability, performance, cost effectiveness, and power consumption.

The scalable ECC architecture and unified data path for both the prime and binary fields has been presented. In addition, to the basic EC arithmetic operations, i.e., point coordinate conversion, point double, point addition, and point scalar multiplication, this processor has been extended to form parallel architecture with 2, 3, 4 AU's. Scheduling is performed with coarse grained and fine grained scheduling.

All functional block units have been realized using VHDL language and simulated using Altera Modelsim 6.0 and synthesized on Quartus software tools. Simulated output waveform windows are shown. Synthesis summary window, Comparison Results obtained is plotted and shown .

Results show that, throughput of 4 AU system is increased when compared to processor with two or three number of AU's by reduction in cycle count. Various parameters were taken to compare Binary vs Prime field system. It shows that binary system is more area efficient and time efficient when compared to prime field system. Maximum throughput is further achieved by introducing fine grain scheduling to coarse grain scheduling. Cycle count decreases to 11,643 from 16,743 (@115.47MHz) in Fine grain scheduling to achieve high throughput.

As number of AU's increases ,CPU time decreases with some area overhead. On analysis of parameters like ALUT's, logic registers, clock to output time, CPU time, cycle count, power for different number of AU's , 4 AU system achieves Optimum result. Our design is compared with various designs given in Literature survey. This design in Altera-quartus platform (Target device-EP3SE50F780C3) Outperforms the other designs . This design can further be extended to CMOS platform.

References:

[1] *ANSI X9.62-1998: Public Key Cryptography for The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*,Sep. 1998, Washington, DC: Amer. Nat. Standards Inst. (ANSI).

[2] G. Chen, G. Bai, and H. Chen, "A high-performance elliptic curve cryptographic processor for general curves over *GF*(*p*) based on a systolic arithmetic unit," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 412–416, May 2007.

[3] J. Goodman and A. P. Chandrakasan, "An energy- efficient reconfigurable public-key cryptography processor," *IEEE J. Solid-State Circuits*, vol. 36,no. 11, pp. 1808–1820, Nov. 2001.

[4] IEEE, *IEEE 1363 Standard Specifications for Public- Key Cryptography*, Jan. 2000, Piscataway, NJ: IEEE Standards Dept.

[5] Ç. K. Koç and B. S. Kaliski, Jr., "Analyzing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33,Jun. 1996.

[6] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48pp. 203–209, 1987.

[7] J.-Y. Lai and C.-T. Huang, "Elixir: High-throughput cost-effective dualfield processors and the design framework for elliptic curve cryptography," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 11, pp. 1567–1580, Nov. 2008.

[8] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in *GF*(2*m*)," in *5th Annu. Int. Workshop SAC*. New York: Springer-Verlag, Aug. 1998, vol. 1556, pp. 201–212.

[9] V. S. Miller, "Use of elliptic curve in cryptography," in *Proc. Crypto*, 1986, pp. 417–426.

[10] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "Multicore curve- based cryptoprocessor withreconfigurable modular arithmetic logic units over *GF*(2*n*)," *IEEE Trans. Comput.*, vol. 56, no. 9, pp. 1269–1282, Sep. 2007.

[11] K. Sakiyama, E. D. Mulder, B. Preneel, and I. Verbauwhede, "A parallel processing hardware architecture for elliptic curve cryptosystems," in *Proc. IEEE ICASSP*, Toulouse, France, May 2006, vol. 3, pp. 904–907.

[12] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, Apr. 2003.