Merging GF(p) Elliptic Curve Point Adding and Doubling on Pipelined VLSI Cryptographic ASIC Architecture

Adnan Abdul-Aziz Gutub
gutub@kfupm.edu.sa
King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

Summary
This paper merges between elliptic curve addition presents a modified processor architecture for Elliptic Curve Cryptography computations in Galois Fields GF(p). The architecture incorporates the methodology of pipelining to utilize the benefit of both parallel and serial implementations. It allows the exploitation of the inherited independency that exists in elliptic curve point addition and doubling operations using a single pipelined core. The processor architecture showed attraction because of its improvement over many parallel and serial implementations of elliptic curve crypto-systems. It proved to be efficient having better performance with regard to area, speed, and power consumption.

Key words:
Elliptic Curve Cryptography, Pipelined Crypto Architectures, Crypto Arithmetic Hardware, Efficient crypto ASIC Processor.

1. Introduction

Elliptic Curve Cryptography (ECC) is a public-key crypto-system proposed by Niel Koblitz and Victor Miller in 1985. The idea of ECC is based on the Discrete Logarithm problem over the points on an elliptic curve. Since 1985, the year ECC was introduced, no real breakthroughs have been made in determining security weaknesses in the algorithm [1-9]. Although evaluators are still unconvinced to the trustworthiness of this technique, several cryptographic applications have been developed lately using these properties. The main improvement of ECC when compared to other equal security cryptosystems (e.g. RSA) is found in the significant reduction in its key size [2,5,8], which results in a substantial faster system.

Several GF(p) ECC processors have been proposed in the literature [4,7,10,12,13]. The gain of using dedicated hardware as crypto-systems is that it results in a considerable speed improvement and power reduction when compared to software solutions on general purpose programmable processors. It also provides higher security than software solutions [10].

The proposed architecture considers representing the elliptic curve points as projective coordinate points in order to reduce the number of all inversion operations to one, to enhance the overall performance as adopted in many processors [4,6,12]. This design, however, differs from existing ones in departing from the current sequential and parallel approaches in the design of crypto processors to pipelining in a four-stage pipelined architecture. It is shown that pipelining will improve the speed and area over the sequential and parallel approaches, actually gaining the benefit of both, as will be proven by the AT characteristics.

In the next section, we give an idea of encryption and decryption using ECC. Then, in Section 3, we provide some background on the main arithmetic operations and its calculations as needed in ECC. The operations are introduced (in Section 3) in the normal two dimensional coordinates system known as affine coordinates. In Section 4, the arithmetic in affine coordinates is extended to projective coordinates to avoid the complexity of the inverse computations. Section 4 also maps the projective coordinate procedures into data flow graphs showing data dependencies. Section 5 provides the new pipelined hardware and discusses several aspects about implementation. In Section 6, we present the concluding comparisons.

2. Elliptic Curve Encryption & Decryption

There are many ways to apply elliptic curves for encryption/decryption purposes. In its most basic form, users randomly chose a *base point* (x, y), lying on the elliptic curve E. The plaintext (the original message to be encrypted) is coded into an elliptic curve point (x_m, y_m). Each user selects a private key 'n' and compute his public key $P=n(x,y)$. For example, user A's private key is n_A and his public key is $P_A = n_A(x, y)$.
For any one to encrypt and send the message point (x_m, y_m) to user A, he/she needs to choose a random integer k and generate the ciphertext $C_m = \{k(x, y) , (x_m, y_m)+ kP_A \}$. The ciphertext pair of points uses A's public key, where only user A can decrypt the plaintext using his private key.

To decrypt the ciphertext C_m, the first point in the pair of C_m, $k(x, y)$, is multiplied by A's private key to get the point: $n_A (k(x, y))$. Then this point is subtracted from the

second point of C_m, the result will be the plaintext point (x_m, y_m). The complete decryption operation is:

$$((x_m,y_m)+kP_A) - n_A(k(x,y))=(x_m,y_m)+k(n_A(x,y))-n_A(k(x,y))=(x_m,y_m)$$

In fact, the most time consuming operation in the encryption and decryption procedure is finding the multiples of the base point, (x,y)[10], hence, *scalar multiplication*. Scalar multiplication in the group of points of an elliptic curve is the analogous of exponentiation in the multiplicative group of integers modulo a fixed integer p. Computing $k.P$ can be done with the straightforward double-and-add approach based on the binary expression of $k=(k_{l-1},\ldots,k_0)$ where k_{l-1} is the most significant bit of k. However, several scalar multiplication methods have been proposed in the literature. A good survey is presented by Gordon in [15].

The double-and-add algorithm, so called binary method, performs by point doubling each time regardless to the bit value of k_i. The point addition is performed only if $k_i=1$, otherwise no addition will be performed. This could be shown in *FMSB-Alg.* for the most to least version and in *FLSB-Alg.* for the least to most version.

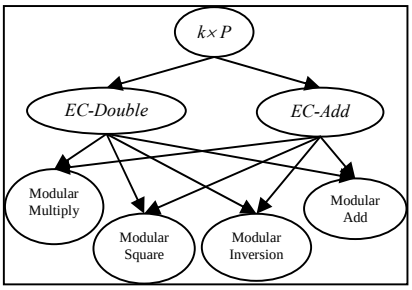

Figure 1: elliptic curve arithmetic hierarchy

FMSB-Alg.: Double-and-add from most significant bit Alg

1. *input k, P*
2. $Q \leftarrow P$
3. *for i from l-2 to 0 do*
   - 3.1. $Q \leftarrow 2Q$
   - 3.2. *if* $k_i = 1$ *then* $Q \leftarrow Q + P$
4. *output Q*

FLSB-Alg.: Double-and-add from least significant bit Alg

1. *input P, k*
2. $Q \leftarrow 0$
3. *for i from 0 to l-1 do*
   - 3.1. *if* $k_i = 1$ *then* $Q \leftarrow Q + P$
   - 3.2. $P \leftarrow 2P$
4. *output Q*

Both algorithms, *FMSB-Alg.* and *FLSB-Alg.*, compute the same final result, however, the *FLSB-Alg.* is preferred in our research because steps *3.1* and *3.2* are independent and can be performed in parallel. This case does not exist in *FMSB-Alg*., where step *3.1* is needed to be completed before step *3.2* is to start.

3. Affine Coordinate Arithmetic

An elliptic curve over $GF(p)$ is defined as the cubic equation:

$$E: y^2 \bmod p = x^3 + ax + b \bmod p.$$

With $a,b,x,y \in GF(p)$ and $4a^3 + 27b^2 \bmod p \neq 0$.
The set of solution $\{(x,y) \mid y^2 \bmod p = x^3 + ax + b \bmod p\}$ is called the points of the elliptic curve E. The elliptic curve (EC) point multiplication is computed by repeated point additions such as:

$$\underbrace{P + P + \ldots + P}_{k\,times} = k \times P$$

with $k \in N$ and $P \in E$.

The basic element of an elliptic curve cryptosystem is the calculation of the point $k.P$, since it needed in each encryption/decryption operation. The hierarchy of arithmetic for EC point multiplication is shown in Figure 1. The top level $k.P$ algorithm is performed by repeated EC-Add and EC-Double operations. The EC operations, in turn, are composed of the basic operations which include: Modular Multiplication, Modular Squaring, Modular Inversion (division) and Modular Addition.
The addition of two points on the elliptic curve is computed as shown below:

$(x_1 , y_1) + (x_2 , y_2) = (x_3 , y_3)$; where $x_1 \neq x_2$
$\lambda = (y_2 - y_1)/(x_2 - x_1)$
$x_3 = \lambda^2 - x_1 - x_2$
$y_3 = \lambda(x_1 - x_3) - y_1$

However, the addition of a point to itself (doubling a point) on the elliptic is computed as show below:

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

In both points addition and point doubling, we need an inversion step to calculate λ. The inversion is the most expensive operation [13]. However, there are designs that replace the inversion by several multiplication operations by representing the elliptic curve points as projective coordinates.

4. Projective Coordinate Arithmetic

The projective coordinates are to eliminate the need for performing inversion. For elliptic curve defined over $GF(p)$, the normal elliptic point (x,y) is projected to (X,Y,Z), where $x = X/Z$, and $y = Y/Z$ [1]. This transformation computation to projective coordinates is performed only twice: at the beginning and at the end, so they can be calculated in software or by the main processor. The form of procedures for point addition is shown in Figure 2. The form of procedures for point doubling is shown in Figure 3. The squaring calculation in

GF(p) is very similar to the multiplication computation. They both are noted as *M* (multiplication). The number of multiplication processes for adding two points is found to be *15M*, while the number of operations for doubling a point is found to be only *13M*.

$P = (X_1,Y_1,Z_1)$; $Q = (X_2,Y_2,Z_2)$;
$P+Q = (X_3, Y_3,Z_3)$; where $P \neq \pm Q$
$(x,y)=(X/Z, Y/Z) \Rightarrow (X,Y,Z)$

$\lambda_1 = X_1Z_1$	1M
$\lambda_2 = X_2 Z_1$	1M
$\lambda_3 = \lambda_2 - \lambda_1$	
$\lambda_4 = Y_1 Z_2$	1M
$\lambda_5 = Y_2 Z_1$	1M
$\lambda_6 = \lambda_5 - \lambda_4$	
$\lambda_7 = \lambda_1 + \lambda_2$	
$\lambda_8 = \lambda_6^2 Z_1Z_2 - \lambda_3^2\lambda_7$	5M
$Z_3 = Z_1Z_2\lambda_3^3$	2M
$X_3 = \lambda_8\lambda_3$	1M
$\lambda_9 = \lambda_3^2 X_1Z_2 - \lambda_8$	1M
$Y_3 = \lambda_9\lambda_6 - \lambda_3^3 Y_1Z_2$	2M
	---
	15 M

Figure 2: form of procedures for point addition

$P = (X_1,Y_1,Z_1)$;
$P+P = (X_3, Y_3,Z_3)$
$(x,y)=(X/Z, Y/Z) \Rightarrow (X,Y,Z)$

$\lambda_1 = 3X_1^2 + a Z_1^2$	3M
$\lambda_2 = Y_1 Z_1$	1M
$\lambda_3 = X_1 Y_1\lambda_2$	2M
$\lambda_4 = \lambda_1^2 - 8\lambda_3$	1M
$X_3 = 2\lambda_4\lambda_2$	1M
$Y_3 = \lambda_1(4\lambda_3 - \lambda_4) - 8(Y_1\lambda_2)^2$	3M
$Z_3 = 8\lambda_2^3$	2M
	---
	13 M

Figure 3: form of procedures for point doubling

The data flow of doubling a point over the elliptic curve is shown in Figure 4. It is made of sixteen multipliers and six adders. On the other hand, Figure 5 shows the data flow graph for adding two elliptic curve points. It is made of thirteen multipliers and four adders.

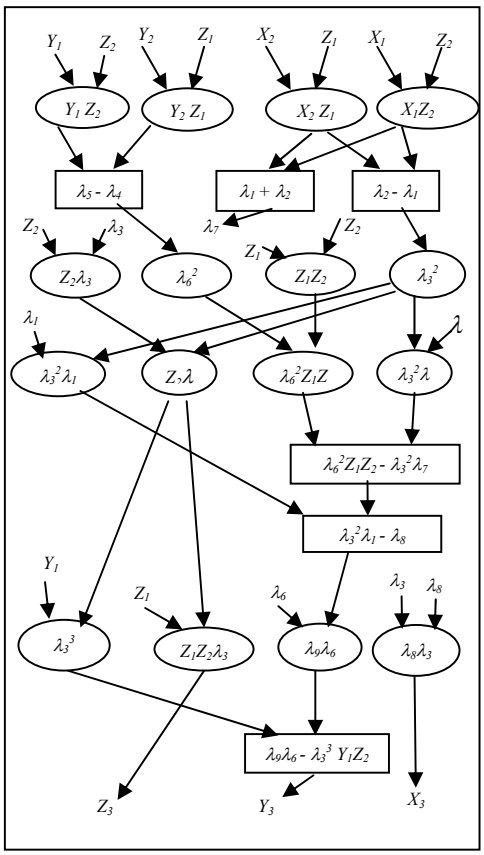

Figure 4: data flow graph for adding two elliptic curve points

It is clear that it is unpractical to implement the elliptic curve point operations as shown in the Figures 4 and 5, or completely sequential. The time needed to complete the operations is huge. We improved this implementation significantly using pipelining design approaches.

5. Pipelined Architecture

The pipelined design consists of basic unit, or core. The core used in point addition operation can be represented as shown in Figure 6. This core consists of a modular adder, a modular multiplier, a controller and register files. The adder and the multiplier will be pipelined. Moreover, the interconnection unit consists of mainly group of multiplexers.

Pipelining has the advantage of increasing the throughput, which is the number of results in time unit. However, pipelining will cause additional area and time because of the latching between the pipelined stages. The pipelined multiplier used in this design consists of four main stages, where each stage should have a well-defined input and output interfaces. Each stage independently processes its inputs and generates the outputs for the next

stage. In addition, the adder stages are equivalent to two stage of the multiplier (the worst case). Actually, addition is involved in multiplication where it is not lengthy compared to multiplication, as we can see in [13].

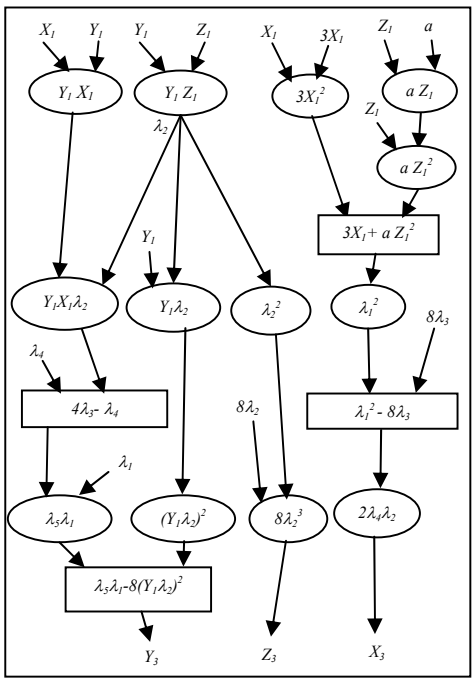

Figure 5: data flow graph for doubling elliptic curve point

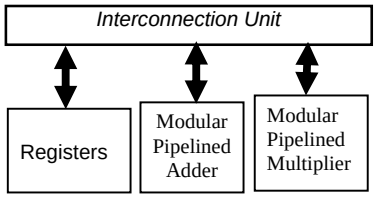

Figure 6: elliptic curve cryptography core design

The pipeline used for scheduling two points addition operation can be shown in Figure 7. The space component (pipelining stages) is the horizontal axes and time is the vertical one. We can see that the last stage shows in which register the result will be stored. The total number of registers needed for this pipeline to store intermediate values is seven registers. The total number of time units needed is 31. Registers R_1 , R_3 , and R_2 , contains the values of Z_3 , X_3 , and Y_3 , respectively.

The pipelining used for scheduling elliptic curve point doubling operation is shown in Figure 8. The number of registers needed here is six registers, and number of time units is 30. Register R_6 , R_3 , and R_1 , contains the values Z_3 , X_3 , and Y_3 , respectively. It can be noticed from Figure 7 and Figure 8 that both pipelines are partially utilized, because of the stalls existence.

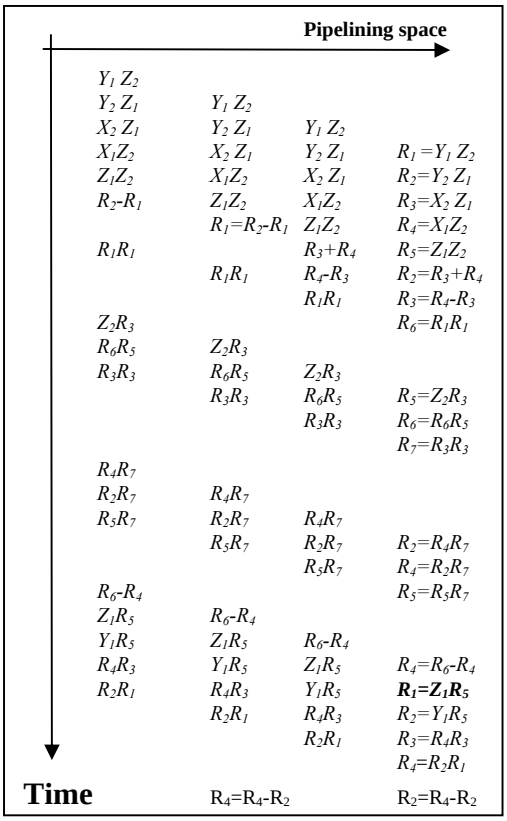

Figure 7: pipelining a two elliptic curve points addition

However, we can merge those two pipelines since we can do doubling and addition at the same time as observed in the FLSB-Alg. where steps 3.1 and 3.2 are independent (Section 2). The resulting pipeline is shown in Figure 9. The total number of registers needed is sixteen registers, and the number of time units is 45. Registers R_1 , R_3 , R_2 , R_{13} , R_{10} , and R_8 , contains the values of Z_{3a} , X_{3a} , Y_{3a} , Z_{3b} , X_{3b} ,and Y_{3b}, respectively. The index a points to the result of doubling operation and index b points to the addition operation.

In fact, the pipeline shown in Figure 9 represents the full word length operations. However, we can generalize the pipeline by introducing the size of the digit used in the multiplier: $C = 45(N/w)$, where, C is the total number of time units, N is the full word length, and w is the digit size. Therefore, the first four operations in the pipeline $(Y_1 Z_2)$, $(Y_2 Z_1)$, $(X_2 Z_1)$ and (X_1Z_2) will be repeated (N/w) times. The core suggested in Figure 6 is interfaced using the pins described in Table 1.

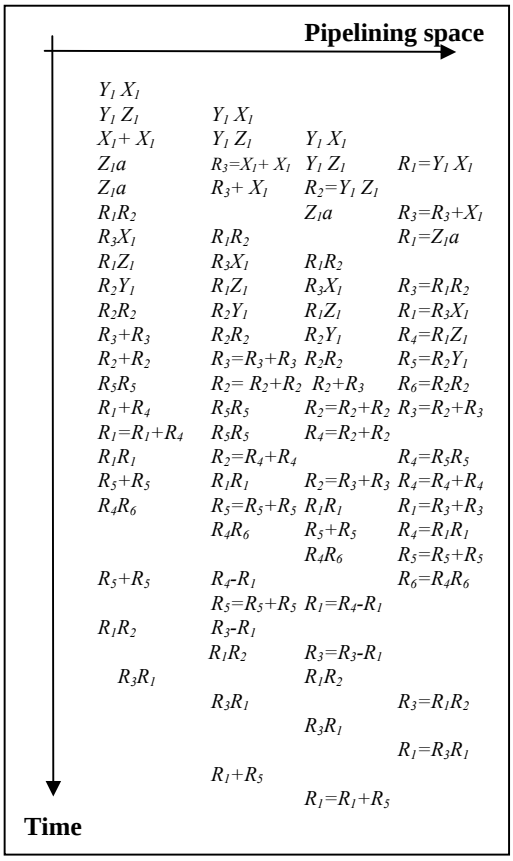

Figure 8: pipelining an elliptic curve point doubling operation

Table 1: pins description of the hardware core

Pins	***Type***	***Description***
p	Input	The modulus.
a	Input	A constant in the GF(p) elliptic curve equation to be used.
X_1 , Y_1 , Z_1, X_2 , Y_2 , Z_2	Inputs	The projective coordinates of the elliptic curve points.
Z_{3a}, X_{3a}, Y_{3a}, Z_{3b}, X_{3b}, Y_{3b}	Output	The added/doubled elliptic curve points in the projective coordinates.
Clk	Input	Clock input.
Start	Input	Active high signal; the input values are available.
Done	Output	Active high flag; that the results are available at their output pins.

The Interconnection unit (Figure 10) is constructed of twenty multiplexers. The multiplexers are to map the data between the sixteen registers and computation modules (the adder and the multiplier). Since the multiplier used will be a digit serial multiplier, the first operand of the multiplier is fed in parallel through register R_x. However, R_x itself is fed digit by digit while the pipelined multiplication process is taking place. The other operand is fed from the registers digit be digit.

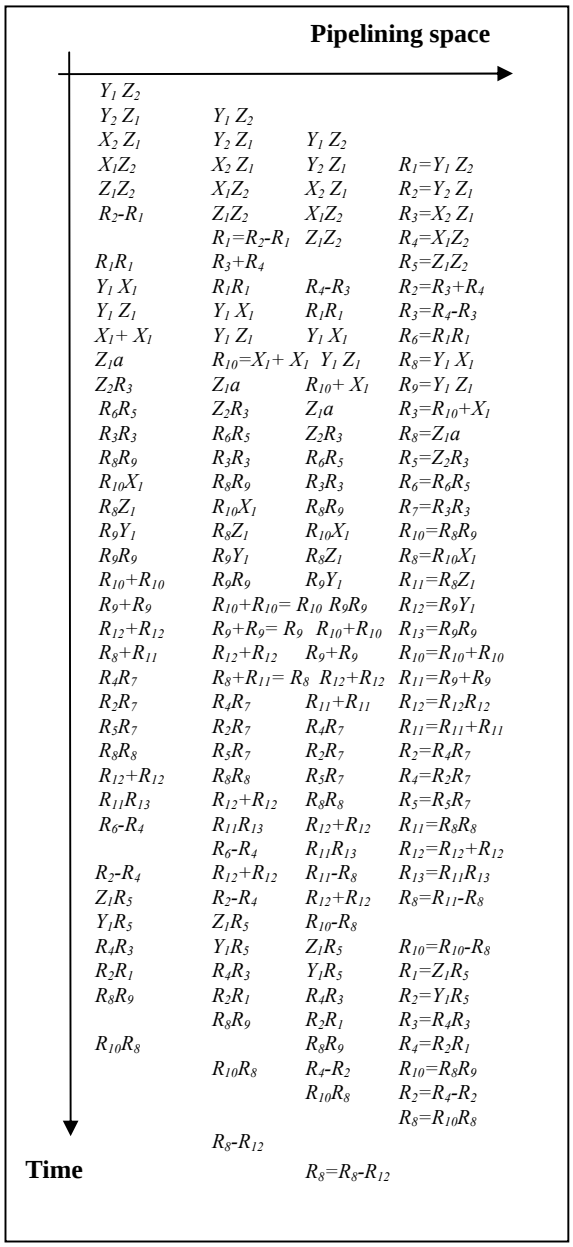

Figure 9: Pipeline for both elliptic curve point addition and doubling

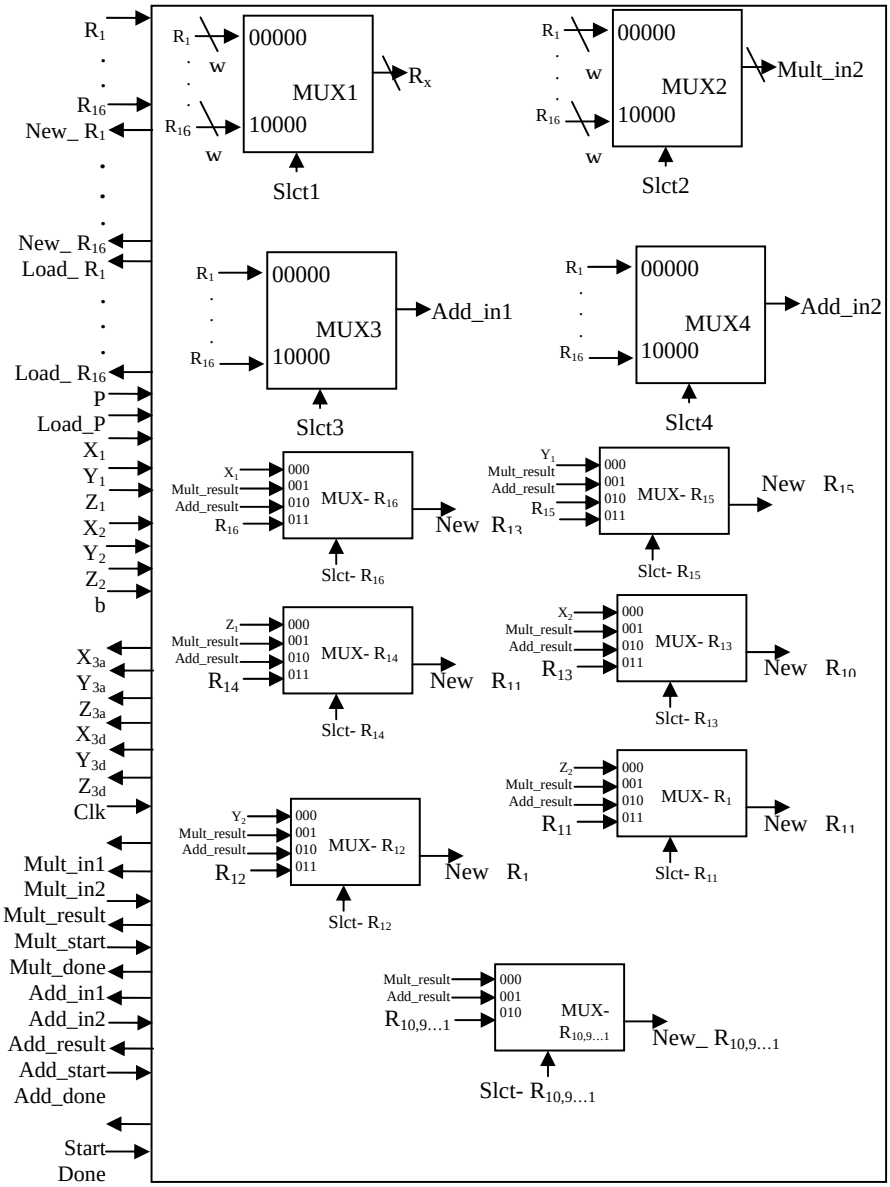

Figure 10: The interconnection unit details

5.1 Verification of the Pipeline

Pipelining is based on having independent operations that can be done in the same time. To deduce a pipeline for the set of operations needed in the ECC point operations, we need to prove it through the data flow of the operations and check the dependencies. If we take the first part of the data flow of Figure 4, we can see that there are four independent operations the can be achieved simultaneously. Because of the independence of those operations, we can process them in a pipeline of four different calculations. Therefore, as shown in Figure 9, in the first clock cycle, we plcae (Y_1 Z_2) into the pipeline first stage. In the next cycle, we move (Y_1 Z_2) to next stage and load (Y_2 Z_1) into the first stage, and so on.

5.2 Modular Pipelined Adder

The sum: *(X+Y) mod M*, can be defined as:

$$(X+Y)\bmod M = \begin{cases} X+Y-M & X+Y \geq M \\ X+Y & X+Y < M \end{cases}$$

The adder described in [14], exploits this fact. To obtain a pipelined implementation of this adder, it has been divided into two stages as shown in Figure 11.

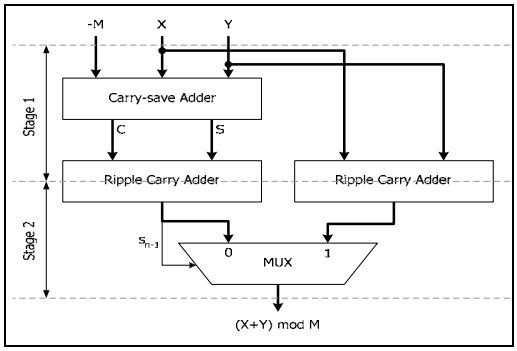

Figure 11: the modular adder pipelined into two stages

The two Ripple Carry Adders (RCAs) are divided into two stages using latches as shown in Figure 12. Table 2 compares the two adders, our two stage pipelined adder with the original version of [14], in terms of time and area with the following assumptions:

- N is the word-length in bits.
- Area is calculated in terms of simple gates, where:
  - AND, OR, NAND and NOR are simple gates.
  - XOR = 2 gates
  - 1 Full-adder = 2 XOR + 3 gates = 7 gates
  - 1 Latch = 4 gates
- The multiplexer is implemented using 2 levels of gates, i.e. AND-OR implementation.
- The time is given in terms of a simple gate time.
- For the non-pipelined design, the given time is the total time needed to get the result.
- For the pipelined version, the depth of the pipeline (time for the longest stage) and the number of needed cycles, to get the output, are given.

Table 2: A Comparison between two Implementations of Modular Adders

Time		Area	
Non-Pipelined	**Pipelined**	**Non-Pipelined**	**Pipelined**
4N+6	2N+4 (2 cycles)	24N	36N+8

5.3 Modular Pipelined Multiplier

To obtain the value of *(XY) mod M*, one of two methods can be used [11]:

1. *Reduction after Multiplication*: Where the product *XY* is computed first and then it is divided over *M* to get the remainder.
2. *Reduction during Multiplication*: Where each partial sum is reduced modulo *M* before accumulating the next partial product.

However, it is not necessary to reduce the partial sum fully. It has been proven in [11] that it is much more efficient to restrict the partial sum to be n-bits wide rather than to be less than *M*. This can be done by truncating the partial sum and adding a pre-calculated correction to make up for that truncation, as shown in Figure 13.

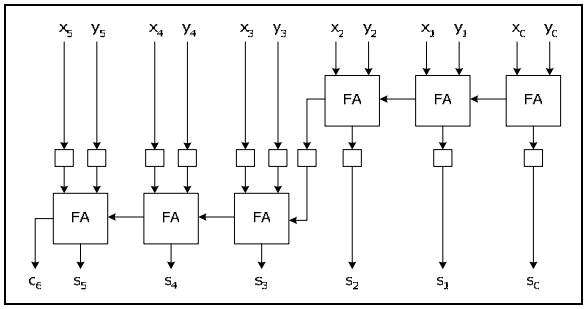

Figure 12: 6-bit Ripple Carry Adder divided into two stages

The architecture of the multiplier in [11] (for radix 2) is shown in Figure 14. To pipeline this design, each RCA is divided into 2 stages, which can be formed in a pipeline of 4 stages. Table 3 compares, in terms of area and time, between a non-pipelined and a pipelined multipliers based on the assumptions mentioned earlier.

Table 3: Comparing between two Modular Multiplier Implementations

Time		Area	
Non-Pipelined	**Pipelined**	**Non-Pipelined**	**Pipelined**
$4N^2+16N+16$	N+6 (4N+4)	71N+71	143N+119

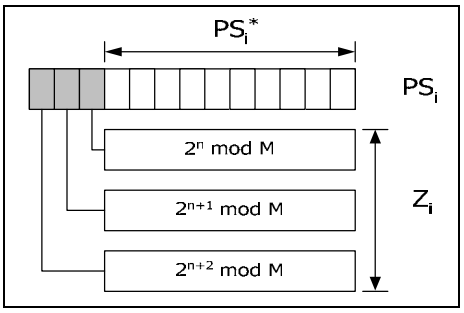

Figure 13: Partial Sum Truncation and Correction

6. Performance Evaluation & Comparisons

As mentioned earlier, the pipelined multiplier needs more area and time that the non-pipelined multiplier used in parallel design. In this part, we will compare the three

possible designs: sequential, parallelized and pipelined design. Tables 4 and 5 compare the area and time for the three designs. For the sequential design, we will need one adder and one multiplier. The parallelized design needs three adders and eight multipliers. Finally, the pipelined design needs only one multiplier and one adder. Table 6 shows the *AT* characteristics for the designs.

Table 4: Area Component for the three designs

Design	Add's Area	Mult's Area	Total Area
Sequential	24N	71N+71	95N+71
Parallelized	3(24N)	8(71N+71)	640N+568
Pipelined	36N+8	143N+119	179N+127

Table 5: Area Component for the three designs

Design	Add's Time	Mult's Time	Total Time
Sequential	10(4N+6)	$28(4N^2+16N+16)$	$112N^2+488N+508$
Parallelized	4(4N+6)	$4(4N^2+16N+16)$	$16N^2+80N+88$
Pipelined	---	$(45/4)(4N^2+28N+24)$	$45N^2+315N+270$

Table 6: AT characteristics for the three designs

Design	AT
Sequential	$10640N^3+50512N^2+80068N+36068$
Parallelized	$10240N^3+60288N^2+101760N+49984$
Pipelined	$8055N^3+62100N^2+88335N+34290$

It is clear from table 6 that the proposed pipelined design beats both the sequential and parallel design in terms of *AT*. For the pipelined design, the *AT* is almost 75% and 78% for the sequential and parallelized design, respectively, for high values of *N*. Figure 15 shows a graph of the relation between the *AT* and the number of bits *N*.

Moreover, the pipelined design has the advantage of having less registers for storing intermediate values. As shown earlier, the pipelined design found to have 16 registers whereas the parallelized design has three registers in each core, leading to 24 registers in four cores. In addition, the pipelined design has no inter-core communication as in parallelized design which means less power consumption in the pipelined design.

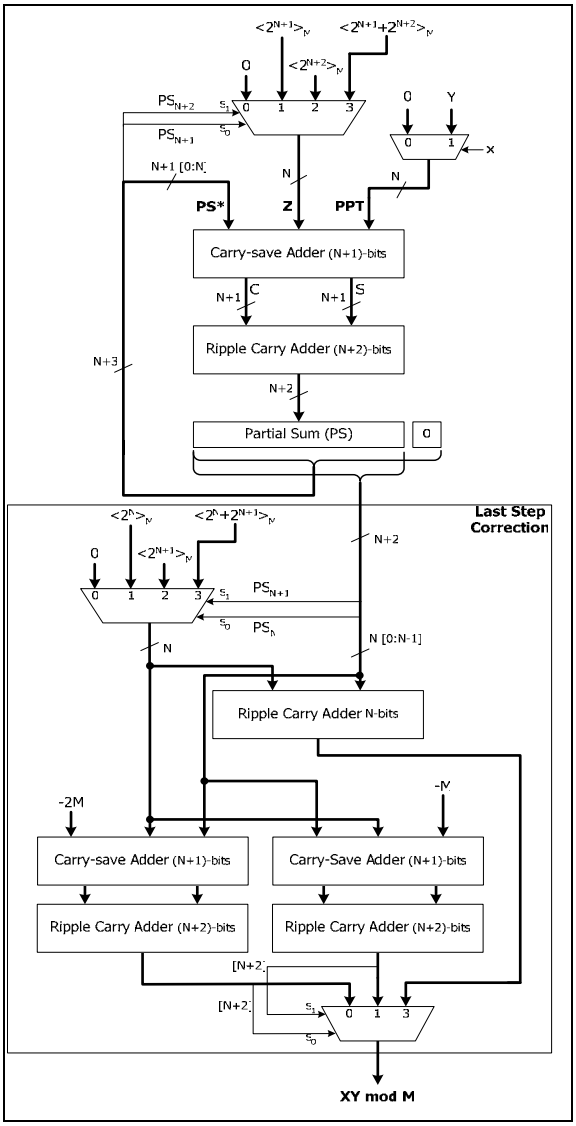

Figure 14: Radix 2 N-bit Modular Multiplier

7. Conclusion

This research proposed a pipelined processor architecture for GF(p) Elliptic Curve Cryptography computations. It exploited the inherited independency that exists in elliptic curve point addition and doubling operations using a single pipelined core made of 16-registers, 2-stage pipelined adder, and 4-stage pipelined multiplier. Both elliptic curve point addition and doubling are performed in at the same time, which was the benefit of using the scalar multiplication algorithm (binary double and add algorithm) that scans the bits starting from least significant bit. The processor architecture showed attractive results because of its improvement over parallel and serial

implementations. Our proposed pipelined hardware proved to be efficient. It showed better AT performance and interesting area, speed which made it a suitable choice for implementing elliptic curve crypto-systems.

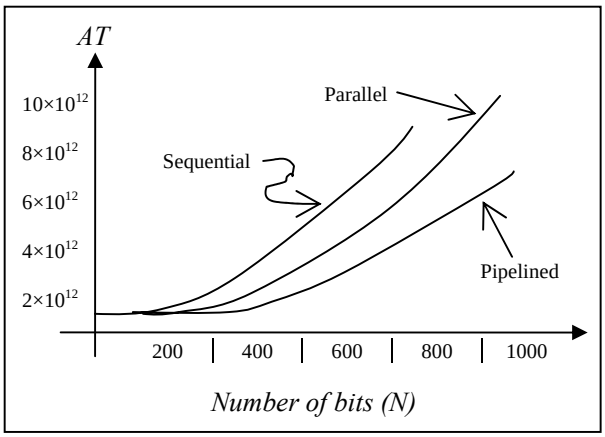

Figure 15: comparison results: AT vs. N

Acknowledgments

The author is grateful to *Professor Mohammad K. Ibrahim* for all his beneficial ideas and feedback. Thanks to the student *Mr Ahmad Kayali* for his effort related to this research. Thanks to King Fahd University of Petroleum and Minerals (KFUPM) for supporting this work.

References

[1] Miyaji A., : ' Elliptic Curves over F_P Suitable for Cryptosystems ', Advances in cryptology- AUSCRUPT'92, Australia, December 1992.

[2] Stallings, W., : ' Cryptography and Network Security: Principles and Practice ', Second Edition, Prentice Hall Inc., New Jersey, 1999.

[3] Chung, J., Sim, S., and Lee, P., : ' Fast Implementation of Elliptic Curve Defined over GF(p^m) on CalmRISC with MAC2424 Coprocessor ', Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Massachusetts, August 2000.

[4] Okada, S., Torii, N., Itoh, K., and Takenaka, M., : ' Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on an FPGA ', Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Massachusetts, August 2000.

[5] Crutchley, D. A., : ' Cryptography and Elliptic Curves ', Master Thesis under Supervision of Prof. Gareth Jones, submitted to the Faculty of Mathematics at University of Southampton, England, May 1999.

[6] Orlando, G., and Paar, C., : ' A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m) ', Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Massachusetts, August 2000.

[7] Stinson, D. R., : ' Cryptography: Theory and Practice ', CRC Press, Boca Raton, Florida, 1995.

[8] Paar, C., Fleischmann, P. and Soria-Rodriguez, P., : ' Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents ', IEEE Transactions on Computers, Vol. 48, No. 10, October 1999.

[9] Blake, I., Seroussi, G., and Smart, N., : ' Elliptic Curves in Cryptography ', Cambridge University Press: New York, 1999.

[10] Gutub, Adnan Abdul-Aziz and Ibrahim, Mohammad, : ' 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.

[11] Mekhallalati, M., Ibrahim, M. K. and Ashur, A, : ' Radix Modular Multiplication Algorithm ', Journal of Circuits and Systems, and Computers, Vol.6, N0.5, pp547-567, 1996

[12] Orlando, G., and Paar, C., : ' A scalable GF(p) elliptic curve processor architecture for programmable hardware ', Cryptographic Hardware and Embedded Systems, CHES 2001, May 14-15, 2001, Paris, France.

[13] Gutub, Adnan Abdul-Aziz, : ' VLSI Core Architecture For GF(p) Elliptic Curve Crypto Processor ', IEEE 10th International Conference on Electronics, Circuits and Systems (ICECS 2003), pages 84-87, University of Sharjah, United Arab Emirates, December 14-17, 2003.

[14] Piestrak S. J., : 'Design of High-Speed Residue-to-Binary Number System Converter Based on Chinese Remainder Theorem ', Proceedings of the Int'l Conf. Computer Design (ICCD '94), pp. 508-511, Oct. 1994.

[15] Gordon, D., : 'A Survey of Fast Exponentiation Methods', Journal of Algorithms, 1998, pp. 129-146.

Adnan Abdul-Aziz Gutub received his Ph.D. degree in June 2002 from the Department of Electrical and Computer Engineering at Oregon State University. He received his B.S. (1995) degree from the Electrical Engineering Department and M.S. (1998) degree from the Computer Engineering Department in King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia. Adnan's research interests are in modeling, simulating, and synthesizing VLSI hardware for computer arithmetic operations. Adnan is currently an assistant professor in the Computer Engineering Department at King Fahd University of Petroleum & Minerals in Saudi Arabia.