VLSI CORE ARCHITECTURE FOR GF(P) ELLIPTIC CURVE CRYPTO PROCESSOR

Adnan Abdul-Aziz Gutub

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, SAUDI ARABIA
Email: gutub@ccse.kfupm.edu.sa

ABSTRACT

A novel GF(p) crypto processor core architecture is presented in this paper. The core is used to implement GF(p) Elliptic Curve Cryptosystem (ECC). The architecture is such that a single core can be used to implement ECC or alternatively a two core solution can be adopted. As a result, the core architecture allows the exploitation of the parallelism that exists in elliptic curve point addition and doubling. The core architecture results in several advantages over conventional implementations with regard to speed and power consumption.

1. INTRODUCTION

In 1985, Niel Koblitz and Victor Miller proposed Elliptic Curve Cryptosystem (ECC) [1-9], which has received considerable attention from mathematician around the world. Although critics are still skeptical as to the reliability of this method, to date no significant breakthroughs have been made in determining weaknesses in the algorithm, which is based on the discrete logarithm problem over points on an elliptic curve. The fact that the problem appears so difficult to crack means that key sizes can be reduced in size considerably, even exponentially [2,5,8], especially when compared to the key size used by other cryptosystems. This made ECC become a challenge to RSA, one of the most popular public key methods known, since ECC is showing to offer equal security to RSA but with much smaller key size (128-256 bits) [2]. Several encryption techniques have been developed recently using the properties of elliptic curve [9].

Several ECC processors have been proposed in the literature recently for GF(p) including GF(2^k) [4,7,13]. The advantage of using dedicated processors for encryption is that it results in a considerable power reduction when compared to a software solution on a general purpose programmable processor. It also provides higher security than software solutions.

It is well known that adding two points over an elliptic curve would require a division operation, which is the most expensive operation over GF(p) [14]. Many proposed processors are based on representing the elliptic curve points as projective coordinate points in order to eliminate division, hence inversion, operations [4,6,13]. This approach is also adopted in the proposed architecture.

The proposed architecture, however, differs from exiting designs in departing from the dominant approach in the design of crypto processors where the emphasis is placed on minimizing area through sequential implementation by using a single multiplier. It is worth noting that gate count is not a real constraint with current technology. Furthermore, area minimization is not the best approach for power reduction. In the proposed architecture two multipliers are used inside the core. In addition, more than one core can be used to implement the ECC. It is shown that such parallelism can be used either to improve speed, reduce power consumption, or achieve a compromise between the two by simply trading off source voltage with clock frequency.

2. ENCRYPTION AND DECRYPTION

It will be assumed that the reader is familiar with the arithmetic over elliptic curve. For a good review the reader is referred to [9]. There are many ways to apply elliptic curves for encryption/decryption purposes. In it 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 operations are:

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

The most time consuming operation in the encryption and decryption procedure is finding the multiples of the base point, *(x,y)*. The algorithm used to implement this is discussed in the next section.

3. POINT OPERATION ALGORITHM

The ECC algorithm used for calculating *nP* from *P* is the binary method, since it is known to be efficient and practical to implement in hardware [2,5,7,9,10]. This binary method algorithm is shown below:

Define k: number of bits in *n* and n_i: the i^{th} bit of *n*
Input: *P* (a point on the elliptic curve).
Output: *Q=nP* (another point on the curve).
1. if $n_{k-1} = 1$, then *Q:=P* else *Q:=0;*

2. for $i = k-2$ down to 0;
3. { $Q := Q+Q$;
4. if $n_i = 1$ then $Q := Q+P$; }
5. return Q;

Basically, the binary method algorithm scans the binary bits of n and doubles the point Q k-times. Whenever, a particular bit of n is found to be one, an extra operation is needed. This extra operation is $Q+P$.

As can be seen from the description of the above binary algorithm, adding and doubling elliptic curve points are the most basic operations in each iteration. As mentioned earlier, the points operations over elliptic curve requires inversion [9]. As in the crypto processor in [6,13], inversion is eliminated using projective coordinates as elaborated in the next section.

4. PROJECTIVE COORDINATES IN GF(P)

The projective coordinates are to eliminate the need for performing inversion. For elliptic curve defined over $GF(p)$, the forms of formulas are found in [9] for point addition and doubling. It projects $(x,y)=(X/Z^2,Y/Z^3)$. The procedure for projective point addition of $P+Q$ (two elliptic curve points) is shown below:

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

$$(x,y)=(X/Z^2,Y/Z^3) \rightarrow (X,Y,Z)$$

$\lambda_1 = X_1Z_2^2$	2M
$\lambda_2 = X_2Z_1^2$	2M
$\lambda_3 = \lambda_1 - \lambda_2$	
$\lambda_4 = Y_1Z_2^3$	2M
$\lambda_5 = Y_2Z_1^3$	2M
$\lambda_6 = \lambda_4 - \lambda_5$	
$\lambda_7 = \lambda_1 + \lambda_2$	
$\lambda_8 = \lambda_4 + \lambda_5$	
$Z_3 = Z_1Z_2\lambda_3$	2M
$X_3 = \lambda_6^2 - \lambda_7\lambda_3^2$	3M
$\lambda_9 = \lambda_7\lambda_3^2 - 2X_3$	
$Y_3 = (\lambda_9\lambda_6 - \lambda_8\lambda_3^3)/2$	3M
	---
	15 M

Similarly, the form of formulas for projective point doubling is shown below:

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

$$(x,y)=(X/Z^2,\ Y/Z^3) \rightarrow (X,Y,Z)$$

$\lambda_1 = 3X_1^2 + aZ_1^4$	4M
$Z_3 = 2Y_1Z_1$	1M
$\lambda_2 = 4X_1 Y_1^2$	2M
$X_3 = \lambda_1^2 - 2\lambda_2$	1M
$\lambda_3 = 8Y_1^4$	1M
$\lambda_4 = \lambda_2 - 2X_3$	
$Y_3 = \lambda_1\lambda_4 - \lambda_3$	1M

	10M

The squaring calculation over $GF(p)$ is very similar to the multiplication computation. They both are noted as M (multiplication). It is worth noting that any EC crypto processor must implement the procedures of projective coordinates efficiently since they are the core steps of the point operation algorithm of ECC.

5. PROPOSED CORE ARCHITECTURE

The architecture of the new core processor is shown in Figure 1. It has the following features:

- It has two serial multipliers and one adder. The multipliers and adder used are the same as those proposed in [13].
- It has FOUR busses for intra-core communications that can be configured dynamically to connect the core's multipliers with its own registers. Two of these are used for input operands and two for output operands.
- It has TWO busses for inter/intra-core communications that can be configured dynamically to connect a multiple of cores.

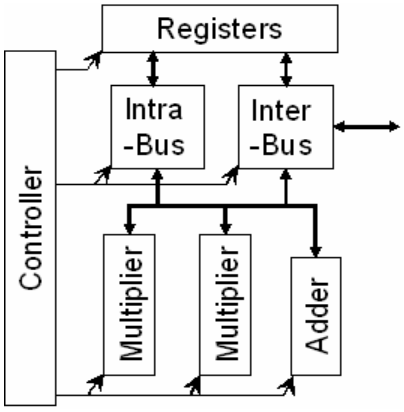

Figure 1. Architecture of the ECC core

The basic motivation behind the design of the proposed core is to exploit, as much as possible, the full parallelism that exists in ECC. Parallelism can be beneficial in two ways. It can be used for high speed execution needed for high data rates applications such as multimedia. It can also be used for low power consumption which can be achieved by reducing the clock frequency and hence consequently the source voltage. It is well known that reducing the source power is the most effective means of reducing power consumption. The benefits of parallel implementation of ECC on power consumption and its comparison with single multiplier implementations are discussed in more details in section 7. The mapping of the ECC on the proposed core is discussed in the next section.

6. MAPPINGS OF ECC ON PROPOSED CORE

In this paper we present two different implementations that lead to different trade-offs between time and power consumption. The first is based on two cores while the second is based on one core.

6.1. Two Core Implementation

Using two cores, as shown in Figure 2, will allow the use of four multipliers. As will be explained now, four multipliers are sufficient to exploit the full parallelism inherent in the projective coordinates discussed in section 4. Figures 3 and 4, show the dataflow of the elliptic curve point doubling and addition using the projective coordinates of section 4. The corresponding critical path of each dataflow diagram is effectively of 4 GF(p) multiplications and of 5 GF(p) multiplications, respectively. Here the time of GF(p) addition and subtraction is ignored since it is very small compared to multiplication. Therefore, the lower bound of the minimum computation time to perform one elliptic point operation in the calculation of nP is 9 GF(p) multiplications. It can be easily seen from Figures 3 and 4 that performing four multiplications in parallel will meet this lower bound, and any further

concurrent multiplications will not actually achieve any further reduction in the computation time.

Furthermore the utilization of the four multipliers is very high. As can be seen from Figures 3 and 4, all the four multipliers will be used in five out of the 9 steps, in one cycle three multipliers are used, and in only three out of the 9 cycles where a single multiplier is used. This indicates high utilization of hardware resources.

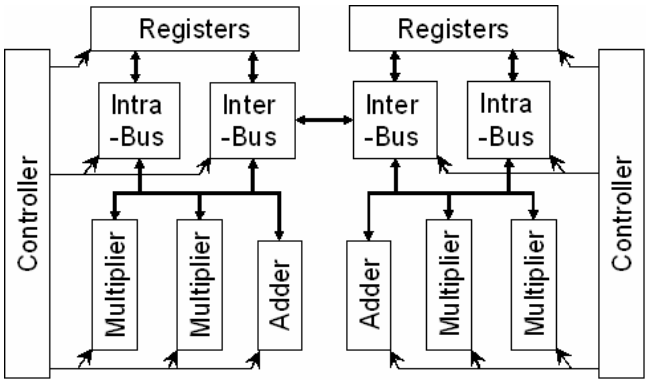

Figure 2. Implementation of ECC on two cores

In what follows we will show that two inter-core communication busses and four intra-core communication busses are sufficient to implement the dataflow in Figures 3 and 4 efficiently. From Figures 3 and 4, the operations inside the right doted box are implemented on the right core in Figure 2, while the operations within the left doted box are implemented on the left core in Figure 2. It can be easily seen from Figures 3 and 4 that there is a maximum of only two arrows that cross from one doted box to another for every multiplication/addition stage. This effectively requires a maximum of two inter-core busses which is the case in the proposed architecture.

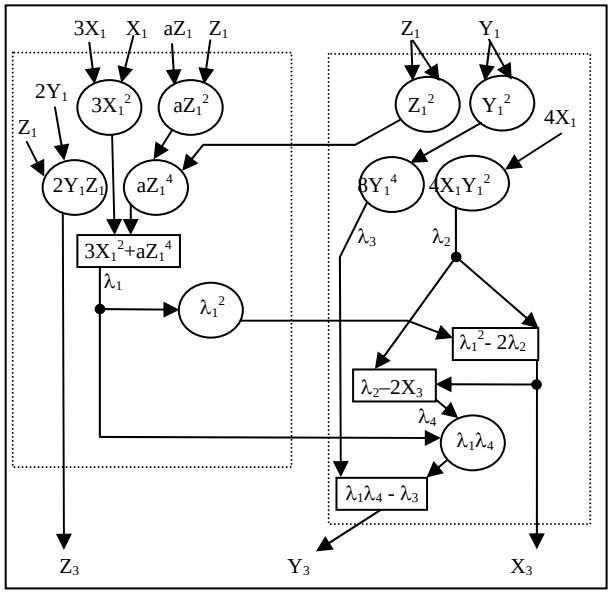

Figure 3. Doubling an elliptic curve point data flow graph

As mentioned earlier, each core has four intra-core busses; two busses being used for input operands and two busses for output operands. It is clear that the two intra busses for the output operands are sufficient since only two outputs are generated at any one stage by within each core.

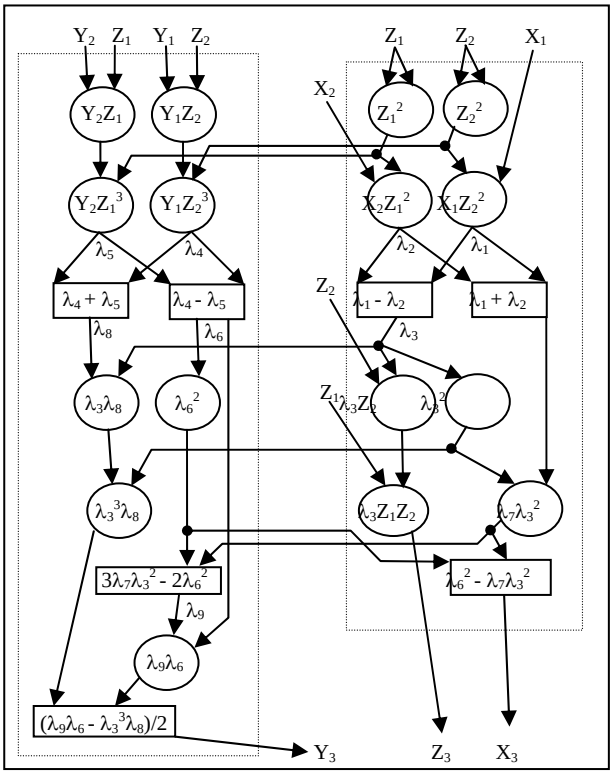

Figure 4. Data flow graph for adding two points

To justify the need for only two intra-core busses, a careful examination of Figures 3 and 4 are needed. For each core, four input operands are required. At every stage it can be verified by simple inspection that some of these input operands are shared, such that either:

(i) Two intra-busses plus two inter-busses are sufficient,
(ii) Four intra-busses are sufficient (note that inter-busses also support intra-bus operation), or
(iii) Three intra-busses and one inter-bus are sufficient.

In all cases therefore, there is no need to use more than two inter/intra-busses and two intra-busses. It should be pointed out that the busses must be configured dynamically in order to implement the dataflow shown in Figures 3 and 4.

6.2. Single Core Implementation

In applications where area becomes an important factor, the ECC can also be implemented using a single core. In this case, both the left and right doted boxes in Figures 3 and 4 are mapped on the same core. In this case, the order of execution is done such that the first stage of the right box is first executed followed by the first stage of the left box, followed by the second stage of the right box and so on. In effect, the implementation of the operations within the two doted boxes in each dataflow is interlaced on the same core.

7. PERFORMANCE EVALUATION AND COMPARISONS

The power consumption of using two cores and one core is compared in Figure 5 with that of using a single multiplier (such as that in [13]) for different execution times. Here time is computed as follows, time=No. of cycles x f_o. Power is given by $P=fCV_S^2$ and assuming that $V_S=kf_o$, where f_o is the maximum

operating frequency for the given V_s, *then* $P=kf^3C$. The capacitance is made proportional to Area.

In an existing design [13], a single multiplier and a single adder is used to perform all the multiplications needed in Figures 3 and 4. The reason is that using more than one multiplier is perceived to be too expensive. However, as can be seen from Figure 5, both of the proposed implementations in fact lead to lower power consumption than using a single multiplier for the same execution time. Furthermore one can achieve a better trade-off between time and power consumption when using two cores rather than using a single core. A single core implementation becomes more advantageous when the normalized execution time is higher than 75.

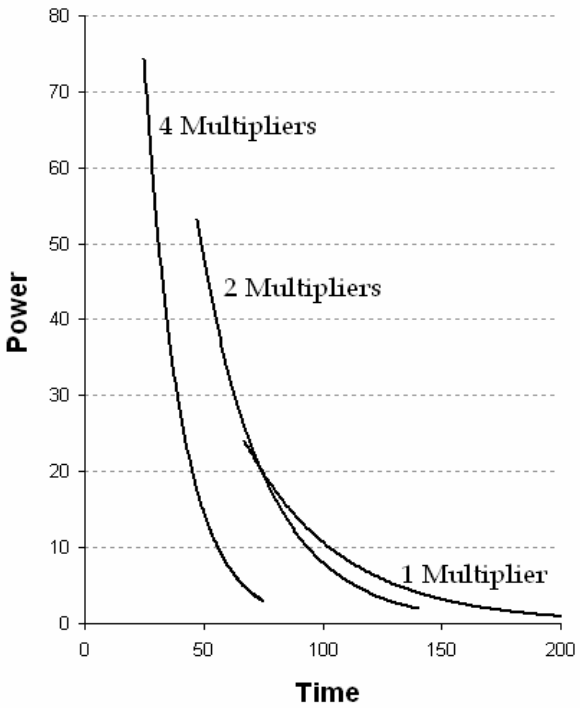

Figure 5. Power time comparison when using different number of multipliers in the complete ECC architecture.

It should also be pointed out that the proposed architecture can support a further reduction in power by dynamically switching off either a whole core, or one of the multipliers. As can be seen from Figures 3 and 4, not all the multipliers are needed in the final stages of computing elliptic point doubling and addition. In this case the control unit will simply ensure that either the entire core is switched off (in the case of two core implementation) and/or one of the multipliers is turned off such that there is no dynamic power consumption.

8. CONCLUSION

An innovative GF(p) elliptic curve crypto core processor is proposed in this paper. The new architecture results in considerable reduction in power consumption as well as offering users a range of trade-off between power and time. The basic feature of the new architecture is that it exploits the inherent parallelism in the computation of point doubling and addition over an elliptic curve. Performance evaluation shows a considerable advantage over conventional implementation of using a single multiplier in terms of power consumption and time. Finally, in addition to the discussed advantages of using two multipliers in the proposed core, this feature can also be exploited to lead to a fault tolerant implementation.

9. ACKNOWLEDGMENTS

The Author is grateful to Professor *Mohammad Ibrahim* and *Muhammad Elrabaa* for their beneficial ideas and observations related to this research. Thanks to King Fahd University of Petroleum and Minerals for supporting this work.

10. 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*, MA, USA, Aug 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: NY, 1999.

[10] Hankerson, D., Hernandez, J., and Menezes, A., "Software Implementation of Elliptic Curve Cryptography Over Binary Fields", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, August 2000.

[11] Orton, G. A., Roy, M. P., Scott, P. A., Peppard, L. E., and Tavares, S. E., "VLSI implementation of public-key encryption algorithms", *Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 277-301, Springer-Verlag, 1987.

[12] Scott, Norman R., "*Computer Number Systems and Arithmetic*", Prentice-Hall Inc., New Jersey, 1985.

[13] 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.

[14] Gutub, Adnan Abdul-Aziz, Tenca, A., and Koc, C., "Scalable VLSI architecture for GF(p) Montgomery modular inverse computation", *IEEE Computer Society Annual Symposium on VLSI*, pages 53-58, Pittsburgh, Pennsylvania, April 25-26, 2002.