A Run Time Reconfigurable Co-Processor for Elliptic Curve Scalar Multiplication

Miguel Morales-Sandoval, Claudia Feregrino-Uribe, René Cumplido and Ignacio Algredo-Badillo
National Institute for Astrophysics, Optics and Electronics
Computer Science Department
Luis Enrique Erro No. 1, Sta. Ma. Tonantzintla, 72840 Puebla, Mexico
{mmorales,cferegrino,rcumplido,algredobadillo}@inaoep.mx

***Abstract*—This paper reports a run-time reconfigurable co-processor for scalar multiplication in elliptic curve cryptography. By reconfiguration, the co-processor can support various finite field orders and hence, different security levels. This is a contribution to solve the current interoperability problems in elliptic curve cryptography. We report the co-processor hardware organization and the cost in terms of area and speed of the reconfigurable solution compared to a static implementation.**

I. Introduction

Elliptic Curve Cryptography (ECC) [1], [2] is a kind of public key cryptography that guarantees the security services of confidentiality, authentication, integrity and no-repudiation, which are necessary to protect information stored or transmitted.

ECC has the advantage of using shorter keys while provides the same security level than other widely used cryptosystems, for example RSA [3]. The use of shorter length keys implies less space for key storage, time saving when keys are transmitted and less costly arithmetic computations. These characteristics make ECC the best and default choice to provide data security [4]. Security services are provided for ECC cryptographic schemes like key agreement, digital signatures and bulk encryption [5].

An ECC cryptosystem is defined as the tuple $T = (GF(q), a, b, G, n, h)$, where $GF(q)$ is a finite field, a and b define an elliptic curve on $GF(q)$, G is a generator point of the elliptic curve, n is the order of G, that is, the smaller integer such that $nG = O$ (identity point in the additive group). h is called the co-factor and it is equal to the total number of points in the curve divided by n. An elliptic curve on the characteristic two field $GF(q)$ is defined by equation 1.

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

To make security applications based on ECC interoperable, they must agree the same tuple T and the same cryptographic schemes. However, several tuples T have been recommended by international standards like ANSI, ISO, IEEE, what has led to interoperability problems.

One of the most important values in tuple T is the finite field and its size. The size determines the security level of the cryptographic schemes. The greater this size the higher the security level and also the computational time of the cryptographic schemes. It is agreed that a field size equal or greater than 2^{163} provides enough security until 2010 [5].

The most time consuming operation in ECC cryptographic schemes is the scalar multiplication, an operation intrinsically related to the tuple T that involves elliptic curve and finite field arithmetic.

In this paper we propose a reconfigurable hardware accelerator attached to a microprocessor to perform the scalar multiplication. The co-processor has the ability to be reconfigured to support different tuples T and hence to provide interoperability while performing the operation faster compared with a software implementation. We present the general architecture of the reconfigurable co-processor, based on our work reported in [6]. We evaluate the co-processor architecture on the prototyping board ML403 [7], which includes hardware reconfigurable. We use the tuples SEC-113, SEC-131 and SEC-163 recommended by SECG [5]. The co-processor is parameterizable in the field order and could be implemented for other security levels. We give performance and area costs of the reconfigurable solution compared to its static implementation.

Next section introduces the concept of reconfigurable computing. Section 3 presents the basis of scalar multiplication operation and the algorithms involved in its computation. Section 4 shows the co-processor architecture. Details about the practical implementation are given in section 5 and the results are shown in section 6. Finally, concluding remarks and further directions are presented in section 7.

II. Reconfigurable computing

Traditionally there are two options to implement an algorithm. One of them is based on software, a set of instructions executed by a microprocessor. The other choice is the direct implementation of the algorithm in hardware. High parallelizing parts of an algorithm are suited to be implemented in hardware while the rest remains implemented in software, leading to have a functional Hardware/Software co-design, see figure 1 a). Co-processors have been implemented to

978-0-7695-3882-2/09 $26.00 © 2009 IEEE
DOI 10.1109/ENC.2009.57

IEEE computer society

speed up many intensive tasks found in image processing, high precision arithmetic and encryption operations. The drawbacks of hardware implementations are the lack of flexibility to switch to other implementation parameters. For example, if a hardware module performs an image convolution with a window size of three, and now it is required a convolution with a window size of nine, it would be necessary to design another hardware module that meets this new requirement. An approach studied in recent years combines the advantages of software (flexibility) and hardware (performance) in a new paradigm of computation named *reconfigurable computing* RC [8]. Reconfigurable computing involves computer science and electronic engineering areas. RC uses reconfigurable hardware for computation purposes. Run-time reconfiguration allows reconfigurable hardware to implement multiple functions which are separated temporally on the same silicon area and the configuration can be changed while the application is running. Also, hardware reconfiguration makes possible to implement different hardware modules in the same area of reconfigurable hardware that otherwise do not fit, see figure 1 b). The concept of reconfiguration can be used to implement several applications but a general design methodology is not applied for all cases. Reconfigurability makes possible hardware flexibility, that is, to have specialized silicon to perform some computational task faster compared with software but keeping flexibility. Reconfiguration have found applications in different fields, for further reading refer to [9].

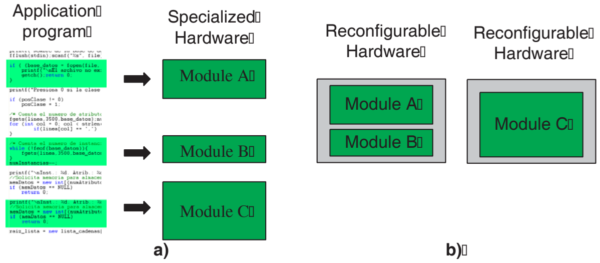

Figure 1. Hardware/Software co-design. a) Computational intensive tasks are implemented in hardware. b) Reconfigurable hardware allows flexibility to switch different modules in the same area.

III. Scalar multiplication

The most time consuming operation in elliptic curve cryptographic schemes is the scalar multiplication, which is intrinsically related to the tuple T. This costly elliptic curve operation is performed according the operations described in three layers:

At the top layer there are different methods for computing the scalar multiplication, which is defined as the result of adding the point P to itself $n-1$ times. That is,

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

This operation is performed using two kinds of sums: ECC-ADD to sum two different points ($P + Q$) and ECC-Double to sum the same point ($P + P$).

The middle layer refers to the coordinate system being used. Depending on this representation the ECC-ADD and ECC-Double are defined in a different way. The simplest is the affine representation (x, y) but most of the reported ECC implementations use projective coordinates because of point addition is free of finite field inversions, being this operation the most time consuming.

At the lower layer is the finite field arithmetic. The performance of the arithmetic units impacts the overall performance of the scalar multiplication and hence the performance of the ECC cryptographic schemes. Finite field operations are multiplication, inversion, squaring and adding. The rules to perform these operation depend on the finite field used.

While the number of ECC-ADD and ECC-Double operations depends on the method chosen in the top layer, the kind and number of operations in the finite field depends on the coordinates used in the middle layer.

Efficient hardware/software implementations of the scalar multiplication kP have been the main research topic on ECC in recent years. The main focus in the related works has been the fast computation of the kP scalar multiplication. Our approach is different. We aim to perform this operation as fast as possible while keeping a flexible architecture that can adapt to several security levels, defined by the size of the finite field $GF(q)$.

This implies a careful algorithms selection and implementation, that leads to a dedicated unit for scalar multiplication that performs well for several elliptic curves and finite fields.

Some works [10], [11], [12], [13], [14], [15], [16] have reported parameterizable arithmetic units for scalar multiplication but those architectures need to be reconfigured out of line. It would be desired a real time adaptation of the co-processor to different security levels. We aim to provide such architecture by using run-time hardware reconfiguration.

IV. The co-processor architecture

Due to its mathematical properties, the binary finite field $GF(2^m)$ has been widely used in elliptic curve cryptography as the finite field $GF(q)$. In this field the operations like multiplication, inversion and sum depend on a basis. Polynomial basis have been preferred due its advantages when implemented in hardware.

In polynomial basis, the elements of $GF(2^m)$ are viewed as $m-1$ grade polynomials $A(x)$ with coefficients in $GF(2) = \{0, 1\}$. A basis of $GF(2^m)$ is one of the form $\{1, t, t_1, t_2, t_{m-1}\}$, where t is a square of an irreducible m grade polynomial $P(x)$ (cannot be factored as two polynomials). Arithmetic in $GF(2^m)$ with polynomial basis is arithmetic of polynomials modulo $P(x)$.

The proposed co-processor architecture to compute scalar multiplications is depicted in figure 2. It is based on scalable finite field arithmetic modules developed previously [17] and finite state machines for control. The main blocks are a serial multiplier, a combinatorial squarer and a divider module, all them performing finite field arithmetic in polynomial basis and parameterizable for any m value. The control unit is a finite state machine that implements the binary method to compute the scalar multiplication kP (see algorithm 1). ECC-ADD or ECC-Double operations are performed accordingly the parsing of scalar k by sending control signal to the arithmetic modules and multiplexers. Both the scalar k and point P are entered to the architecture in groups of 32-bit words. Additionally to scalar multiplication, the architecture can perform an ECC-ADD operation, which is required in elliptic curve digital signature verification schemes.

```
**Algorithm 1: Binary method for scalar multiplication** $kP$
**Input:** $P = (x, y)$ $x, y \in GF(2^m)$, $k = (k_{m-1}, k_{m-2}, ..., k_0)$
**Output:** $R = kP$
$R \leftarrow (0, 0)$
$S \leftarrow P$
**for** $i$ **from** $0$ **to** $m - 1$
    **if** $k_i = 1$
        $R \leftarrow$ ECC-ADD$(R, S)$
    **end if**
    $S \leftarrow$ ECC-Double$(S)$
**end for**
```

```
**ECC-ADD algorithm**
**Input:** $P = (x_1, y_1)$, $Q = (x_2, y_2)$
**Output:** $R = P + Q = (x_3, y_3)$
$x_3 \leftarrow \lambda^2 + \lambda + x_1 + x_2 + a$
$y_3 \leftarrow \lambda(x_1 + x_3) + x_3 + y_1$
$\lambda \leftarrow (y_2 + y_1)/(x_2 + x_1)$
```

```
**ECC-Double algorithm**
**Input:** $P = (x_1, y_1)$
**Output:** $R = 2P = (x_3, y_3)$
$x_3 \leftarrow \lambda^2 + \lambda + a$
$y_3 \leftarrow x_1^2 + \lambda x_3 + x_3$
$\lambda \leftarrow x_1 + y_1/x_1$
```

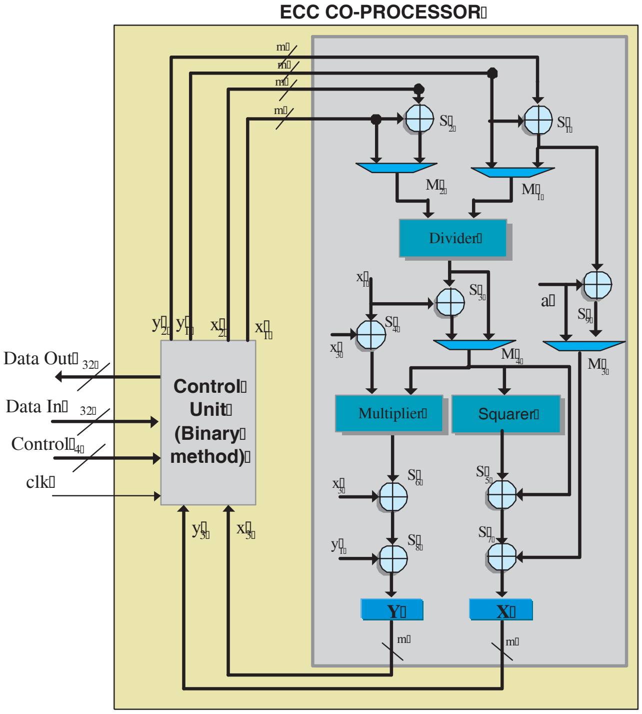

Figure 2. Scalar multiplication co-processor architecture.

The binary method checks each bit value k_i of scalar k from the least to the most significant bit. It performs an ECC-Double operation in each iteration and an ECC-ADD operation only if $k_i = 1$.

Although the binary method can perform both the ECC-ADD and ECC-Double in parallel, the control unit of the ECC-coprocesor performs those operations serially. The same arithmetic modules are used to perform both operation, different to the architecture presented in [6] where there are specialized units for each elliptic point addition. Due an scalar k in the field $GF(2^m)$ has $m/2$ bits equal to '1', the binary method performs m ECC-Double operations and $m/2$ ECC-ADD operation in the average for each kP scalar multiplication.

V. Practical implementation

We implemented the co-processor on a field programmable gate array (FPGA) Virtex-4 included in the prototyping board ML403. The co-processor was described in Very High Hardware Description Language (VHDL) and its functionality simulated in Active-HDL.

A. Static version

We first implemented the static version of the co-processor using ISE 8.2 tools. Then, we use EDK 8.2 to define the complete system (see figure 3), which includes the PowerPC microprocessor, local buses PLB and OPB, an universal asynchronous receiver/transmitter UART module, memory blocks for data and program and our co-processor. The original co-processor shown in figure 2 was wrapped with the IPIF EDK core and busmacros to interconnect it to the OPB bus.

A program was written in C language that runs on the PowerPC microprocessor which is included as a hard processor in the Virtex-4 FPGA. The application enables the co-processor and sends the point P and scalar k through the

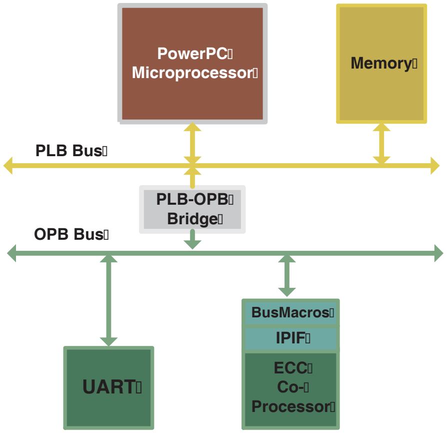

Figure 3. Reconfigurable co-processor attached to a microprocessor.

PLB bus as a group of 32-bit words. After reading the input parameters the co-processor starts the computation while the processor waits for the results. By asserting a signal, the co-processor notifies the end of the computation and then the application reads back the results and show them via the UART module in a terminal window.

B. Run time reconfiguration version

Then, we designed the run time reconfiguration version based on the design flows [18] and [19] to generate the static and reconfigurable modules. As described in those documents some changes were applied to the static design. We have to define the fixed part (that part of the system implemented in the FPGA that will not change) and the reconfigurable part (that part in the FPGA that will change at run time). In our design the fixed part is composed of all the modules in figure 3 except the ECC co-processor wrapped by the IPIF module, which is the reconfigurable part. All signals that connects the fixed and reconfigurable part must cross through busmacros. Also, global FPGA resources like the buffered clock must be locked and specified. All these changes are specified in a constraint file. We use PlanAhead 8.2. to generate this file. The distribution of the components in the FPGA are shown in figure 4.

We use the IMPACT Xilinx utility to download the full and partial FPGA configuration files (bitstreams). Using the software application we test the co-processor for the security level SEC113, SEC131 and SEC163, recommended in [5].

VI. Results

Area results of the co-processor are shown in table I. These results are fewer compared to [6] but at the expense of more clock cycles to perform the scalar multiplication.

The area results for the static system and the reconfigurable one are shown in tables II and III. More area is required for the reconfigurable co-processor when it is integrated to the complete system, due to the additional modules IPIF and busmacros.

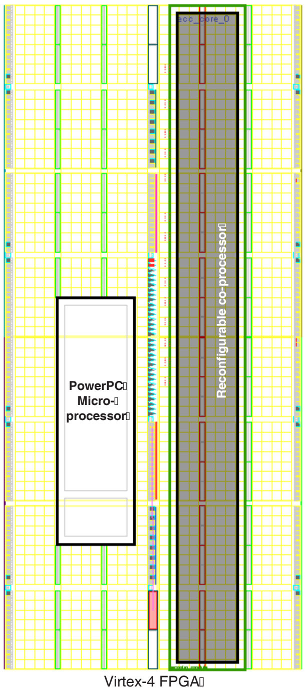

Figure 4. Virtex-4 overview and area assignment.

The time to perform the scalar multiplication is given in table IV. These results do not include the data transfer between the microprocessor and the co-processor. The co-processor uses the same clock frequency of the bus system, which is the same that the microprocessor clock, 100 MHz. Based on the timing shown in table, our co-processor requires 0.0067 ms per 1-bit security level.

Compared to an optimized software implementation for 270 bits security level which requires 196.71 ms on a dual-Xeon computer at 2.6 GHz [8]; our co-processor would perform that same 270 bit operation in approximately 1.8 ms. That is, it would perform 100 times faster while its clock speed is almost 25 times slower (100 MHz) than the Xeon processors.

Table IV
Time results for the reconfigurable kP co-processor.

Sec. level	Cycles/kP	Time
113	51,730	0.57 ms
131	68,887	0.69 ms
163	107,043	1.07 ms

All results we obtained were validated by comparing them against a software implementation that is a slight modification of the code available in [20].

Table I
AREA RESULTS FOR THE kP CO-PROCESSOR.

Hw Resouces	Security level (bits)		
	113	131	163
Flip-Flops	1,759	2,045	2,519
LUTs	3,968	4,625	5,272
Slices	2,529	2,852	3,088
Gate count	39,512	45,961	54,090

Table II
AREA RESULTS FOR THE NON RECONFIGURABLE SYSTEM.

Hw Resouces	Security level (bits)		
	113	131	163
Flip-Flops	2852	3138	3615
LUTs	5017	5649	6745
Slices	3589	4028	4853
Dual Port RAMs	188	188	188
Shift registers	53	53	53
RAMB16s	16	16	16
Equivalent gate count	1,119,196	1,125,492	1,136,304

Table III
AREA RESULTS FOR THE RECONFIGURABLE SYSTEM.

Hw Resources	Fixed part	Reconfigurable part		
		Security level (bits)		
		113	131	163
Flip-Flops	942	1,939	2,225	2,702
LUTs	891	4147	4,779	5,875
Slices	1107	2,272	2,442	2,971
Dual Port RAMs	216	0	0	0
Shift registers	64	0	0	0
RAMB16s	16	0	0	0
Gate count	1,079,972	42,062	48,358	59,170

VII. CONCLUDING REMARKS AND DIRECTIONS

We presented a co-processor to compute the most time consuming operation in elliptic curve cryptography, the scalar multiplication. To support different security levels and allows interoperability, the co-processor can be updated to support different security levels and reconfigured at runtime. Further improvements should be done to optimize area and allows to implement greater security levels. Additionally, self-reconfiguration of the architecture using the PowerPC processor and the ICAP interface could be pursued.

Acknowledgments

First author thanks the National Council for Science and Technology (CONACyT) for financial support throw the scholarship number 171577.

REFERENCES

[1] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, November 1987.

[2] V. Miller, "Use of Elliptic Curves in Cryptography," in *Proc. of Advances in Cryptology, CRYPTO'85*, August 1985, pp. 417–426.

[3] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[4] K. Lauter, "The advantages of elliptic curve cryptography for wireless security," *IEEE Wireless Communications*, pp. 62–67, 2004.

[5] Certicom Research, "Elliptic curve cryptography: Standards for efficient cryptography," 2000, http://www.secg.org.

[6] M. Morales-Sandoval and C. Feregrino-Uribe, "Hardware architecture for elliptic curve cryptography and lossless data compression," in *15th International Conference on Electronics, Communications and Computers (CONIELECOM'05)*. IEEE Computer Society, 2005, pp. 113–118.

[7] Xilinx Inc., "ML401/ML402/ML403 Evaluation Platform," User Guide, UG080., May 24 2006, www.xilinx.com.

[8] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2005.

[9] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2006, pp. Article ID 56 320, 19 pages, 2006, doi:10.1155/ES/2006/56320.

[10] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich, "Reconfigurable implementation of elliptic curve crypto algorithms," in *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2002, pp. 284–291.

[11] M. Ernest *et al.*, "A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2^n)," in *Proc. of the 4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES'2002*, ser. LNCS, vol. 2523. Springer, August 2002, pp. 381–399.

[12] T. Kerins *et al.*, "Fully Parameterizable Elliptic Curve Cryptography Processor over GF(2^m)," in *Proc. of 12th International Conference on Field Programmable Logic and Application, FPL'2002*, ser. LNCS, vol. 2438. Springer, September 2002, pp. 750–759.

[13] R. Cheung, N. Telle, W. Luk, and P. Cheung, "Customizable elliptic curve cryptosystems," *IEEE Trans. on VLSI Systems*, vol. 13, no. 9, pp. 1048–1059, Sept. 2005.

[14] G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)," in *Proc. of the Second International Workshop on Cryptographic Hardware and Embedded Systems, CHES'2000*, ser. LNCS, vol. 1965. Springer, August 2000, pp. 41–56.

[15] N. Mentens, S. Berna, and B. Preneel, "An FPGA Implementation of an Elliptic Curve Processor GF(2^m)," in *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, 2004, pp. 454–457.

[16] A. Hodjat, D. D. Hwang, and I. Verbauwhede, "A scalable and high performance elliptic curve processor with resistance to timing attacks," in *ITCC'05: International Conference on Information Technology: Coding and Computing*, vol. I, 2005, pp. 538–543.

[17] M. Morales-Sandoval and C. Feregrino-Uribe, "$GF(2^m)$ arithmetic modules for elliptic curve cryptography," in *3rd International Conference on ReConFigurable Computing and FPGAs (ReConFig06).* IEEE Computer Society, September 2006, pp. 176–183.

[18] Xilinx Inc., "Early access partial reconfiguration." User Guide, UG208., March, 6. 2006, www.xilinx.com.

[19] Xilinx Inc., "Two flows for partial reconfiguration: Module based or difference based," Application Note 290, XAPP290., September, 9. 2004, www.xilinx.com.

[20] M. Rosing, *Implementing Elliptic Curve Cryptography*. Greenwich, CT, USA: Manning Publications, 1999.