HARDWARE/SOFTWARE CO-DESIGN OF AN ELLIPTIC CURVE PUBLIC-KEY CRYPTOSYSTEM

S. Janssens[1], J. Thomas[1] W. Borremans[1], P. Gijsels[1],
I. Verbauwhede*, F. Vercauteren, B. Preneel, and J. Vandewalle
K.U.Leuven, ESAT/COSIC, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
*UCLA EE Dept., Los Angeles, CA 90095-1594, U.S.A.

Abstract. This contribution discusses an implementation of an elliptic curve public-key cryptosystem on the Atmel FPSLIC, a system on a chip (SOC) that integrates a 40K FPGA with an AVR micro-controller and a set of peripherals. The FPGA is ideally suited for an efficient implementation of the underlying finite field arithmetic. The software benefits the global control. We use a standard basis representation for the field elements and projective coordinates to implement the group operation. The results for area are comparable with existing hardware implementations. Although no attempts have been made yet to reduce the critical path delay of the hardware part, we obtained promising results towards speed and throughput. A clock frequency of 10 MHz is realized, but a lot more must be possible after optimization.

1 INTRODUCTION

The goal of this work is to develop a high-speed hardware/software co-design for computing elliptic curve point multiplications with least development time, the lowest hardware cost and maximal flexibility. Elliptic curve cryptography is becoming increasingly common for implementing public-key protocols as the Diffie-Hellman key agreement. The security of these cryptosystems relies on the presumed intractability of the discrete logarithm problem on elliptic curves. Elliptic curve public-key cryptosystems (ECPKC) use smaller key sizes than other public key cryptosystems, such as RSA for the same level of security. Therefore ECPKCs are mainly used for applications where resources such as memory and computing power are limited, including smart cards and hand-held devices.

A considerable number of papers have been published on the implementation of elliptic curve cryptosystems either in software (see [DBV96], [DMPW98], [GP97] and [KT92]) or in hardware (see [AMV93] and [Ros98]). Most of the software implementations were dealing with problems concerning low performance/cost ratios and this because of word size mismatches, less parallel computation and algorithm/architecture mismatches. The main goal of the hardware implementations is to reduce the area needed to implement one bit of key length. As a hardware/software co-design offers the possibility to combine the advantages of both software and hardware, this results in systems with higher performances.

In this paper a hardware/software co-design for performing elliptic curve point multiplications is introduced and implemented on an Atmel FPSLIC (Field Programmable System Level Integration Circuit). The FPLSIC implements an AVR micro-processor, memory, peripherals and a FPGA on the same chip.

1. This paper includes results of Master's thesis [JT01] and [BG00], the research for which was mostly done during a visit to UCLA during the winters of 2001 and 2000, respectively.

0-7803-7145-3/01/$10.00 ©2001 IEEE

The remainder of the paper is organized as follows. In Section 2 we present a short introduction to public-key cryptography using elliptic curves over GF(2^n). The hardware/software design architecture is described and analysed in section 3. Section 4 discusses implementation results, but also the design methodology and tools used to realize the design in a fast and effective way. Finally, some conclusions and further work are presented in section 5.

2 THEORETICAL BACKGROUND

The idea behind public key cryptography is explained by the Diffie-Hellman key exchange protocol: two entities, A and B, can derive and share a common piece of secret information over an insecure communication channel [M93]. They can then use this secret as their key in a symmetric cryptosystem such as AES. The security of the Diffie-Hellman key exchange, based on elliptic curves, relies on the difficulty of the discrete logarithm problem (DLP): given the curve, the point P and the point multiplication $k.P$, it is hard to recover the integer k.

Several efficient algorithms [BSS99, p. 57-73] are developed to decompose the point multiplication $k.P$ into basic operations on points of the elliptic curve, e.g. doubles, additions and subtractions. In turn, each of these operations consists of a series of additions, squarings, multiplications and inversions in the underlying field. Let GF(2^n) be a finite field of characteristic two. These fields are particularly interesting for hardware implementations since operations in this field lack carry propagation. The most straightforward representation of elements of GF(2^n) is used, namely a standard basis. A non-supersingular elliptic curve E over GF(2^n) is defined to be the set of solutions $(x,y) \in GF(2^n) \times GF(2^n)$ to the Weierstrass equation:

$$y^2 + xy = x^3 + ax^2 + b$$

where a and $b \in GF(2^n)$, $b \neq 0$, together with the point at infinity denoted by O. The use of elliptic curves in cryptography is based on the property that a group law can be defined on the set of points on an elliptic curve. More information about the group structure and the formulas for adding and doubling two points can be found in [BSS99, p. 29-39]. Based on [DV98], projective coordinates are used to represent the points on the elliptic curve, since these have the advantage that no inversions, which are time consuming operations, have to be performed in the underlying field.

3 DESCRIPTION OF ARCHITECTURE

The design hierarchy is shown in Fig. 1. The design consists of a Data-path, which performs the finite field arithmetic, and two finite state machines that each control a particular part of the functionality at a particular level of hierarchy. At the highest level the Software Controller is the master. It gives instructions to the Hardware Controller, which translates these instructions in a sequence of direct control signals for the operators in the Data-path.

The architecture is given in Fig. 2. Its main function is to compute the core operation of an elliptic curve cryptosystem, i.e. the point multiplication. It also provides communication with a Visual Basic program on a PC through a Communication module. Hence, it can be used as a fully functional demonstrator.

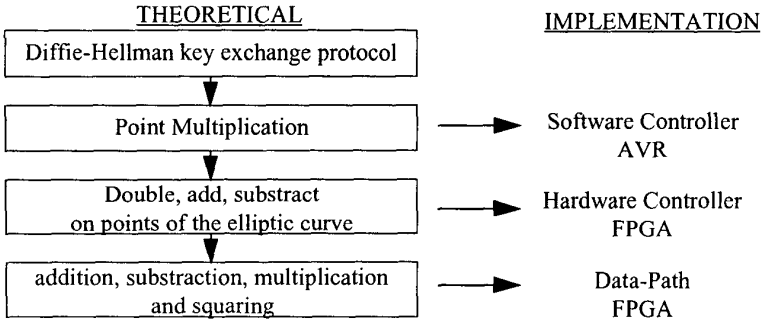

Figure 1 : Design hierarchy

3.1 Software Controller

Implementing control in hardware is possible but difficult, because it is error prone and very hard to modify. Thus the highest level of control is implemented on the AVR 8 bit micro-controller. The Software Controller has two main tasks to execute. First, the FSM Point Multiplication breaks down a point multiplication into individual group operations (double, add or subtract). The algorithm used for this is the so-called double-and-add/subtract algorithm, based on a ternary representation of the multiplier k, [IEEE99, Sect. A.10.3]. Second, the Communication Controller provides the mutual communication between the outside world (PC), FPGA and AVR. The communication between PC and AVR is realized via an RS-232 serial link; between the AVR and the FPGA via an 8 bit bidirectional internal data bus. Since the parameter size is typically of the order of 200 bits, the data items are divided in chunks of 8 bits, which are handled sequentially.

The Software Controller combines the FSM Point Multiplication and the Communication Controller into one finite state machine. This FSM sequentially reads in new data from the PC, sends the relevant data to the Data-path, controls the execution of the point multiplication, and reads out the result to the PC. The communication between the software controller and both the hardware controller and the Visual Basic Program on the PC are interrupt driven.

3.2 Hardware Controller

The Hardware Controller is a finite state machine that consists of an FSM Double Add Substract and an Address Controller. It receives instructions from the Software Controller for reading field and curve parameters, for reading data of the point P to be multiplied and the multiplier k, for starting a group operation and for making the result available to the outside world. The FSM Double Add Substract implements a single group operation in terms of individual field operations, according to the steps described in [IEEE99, Sect.A.10.6-7]. The Address Controller is responsible for the memory management of the two RAM-blocks in the Data-path.

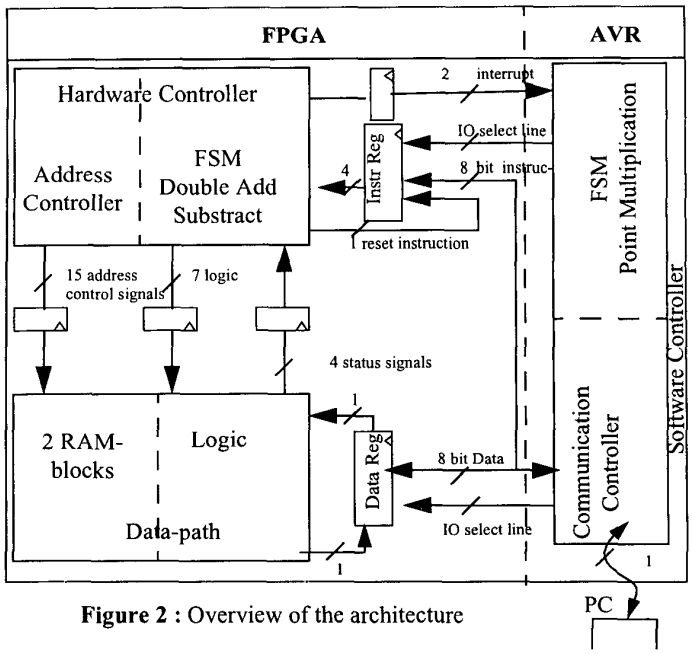

Figure 2 : Overview of the architecture

3.3 Data-path

Fig. 3 shows the architecture of the Data-path. The Data-path is made up of two main parts: a storage part with 13 memory locations, and a part with the arithmetic blocks to perform the basic operations in $GF(2^n)$. Four busses and two multiplexers provide the necessary connections between these two parts.

The input data enters the Data-path through the data register and the software part fills it byte by byte. After the data is shifted in the IOregister, it is put in both RAM-blocks. The two RAM-blocks mimic a single-write, dual-read RAM module. When a complete point multiplication is executed, the result is written from RAM-block1 in the IOregister and is then shifted byte after byte in the data register.

Each elliptic curve double, add or subtract is made up of additions, multiplications and squarings on elements of $GF(2^n)$. Therefore there are three operators in the Data-path. The output of RAM-block1 is connected with *bus1*, RAM-block2 is connected with *bus2*. These two busses provide the three operators with the right operands. When a multiplication or a square is executed and the result falls outside the scope of the finite field, the result must be reduced modulo the irreducible field-polynomial of the field $GF(2^n)$. Therefore *bus3* is implemented to connect the IOregister to the Data-path module. Since the IOregister is not used during the execution of a double, add or subtract instruction, it is used to store the *fieldpoly*. When one of these instructions starts, first the fieldpolynomial is read out of RAM-block1 and put in the IOregister.

Adder Addition of elements of $GF(2^n)$ is implemented simply as an array of XOR gates. This means that there is no carry propagation.

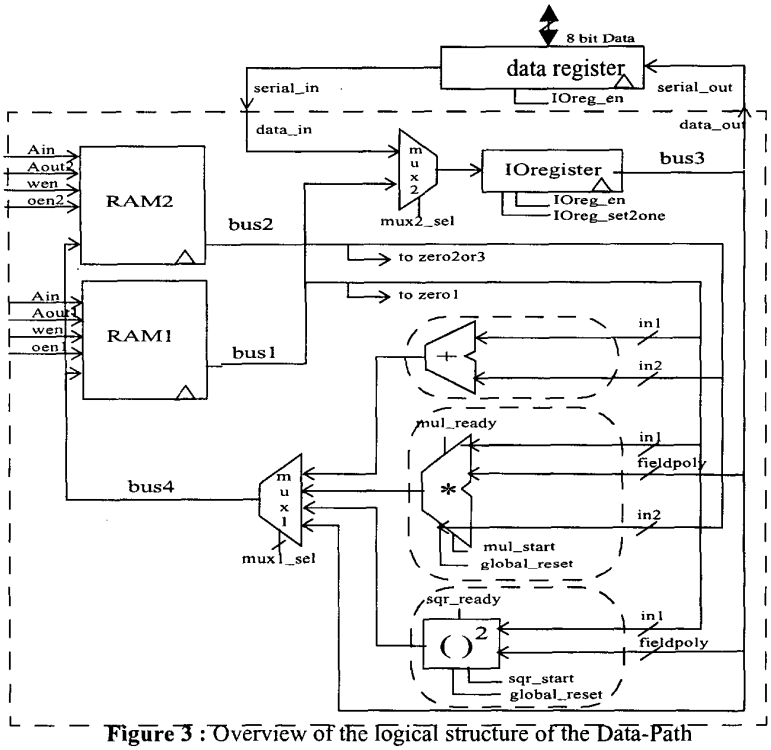

Figure 3 : Overview of the logical structure of the Data-Path

Multiplier The multiplier is based on the multiplier described in [LC83, p.161]. A multiplication in GF(2^n) consists of two steps: a multiplication of binary polynomials, and a reduction modulo the irreducible field polynomial. These two steps can be combined in an interleaved way. This results in a serial multiplier that consumes relatively little area. A complete multiplication is performed in n clock cycles. The simplicity and compactness of the multiplier is probably the most important advantage of a standard basis representation over an (optimal) normal basis. An optimal normal basis multiplier needs an equal number of clock cycles to find the result, but [Gei93] concludes that it is considerably more complex.

As can be seen from Fig. 4, a multiplier for GF(2^n) consists of a linear feedback shift register, where the coefficients a_i of one of the multiplicands can be added to the state of the register, depending on the bits b_i of the other multiplicand. Each cycle a new b-coefficient is shifted in, and the c-coefficients are shifted one bit to the left. A feedback originating from the most significant c-coefficient exists to do the reduction modulo the irreducible fieldpoly.

Our multiplier is slightly more complex since it allows for a programmable field polynomial. Therefore, instead of a fixed feedback path, an array of AND gates multiplies the feedback bits with the respective coefficients of the field polynomial, which are stored in the IOregister. Recall that a multiplication in GF(2) is just a log-

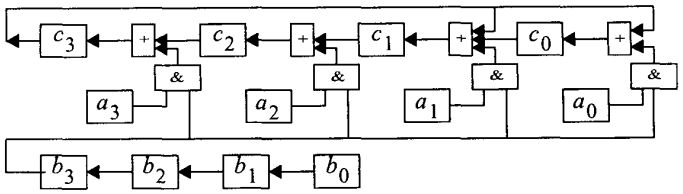

Figure 4 : Serial multiplier for GF (2^4) based on the irreducible field polynomial $x^4 + x + 1$

ical AND. With this multiplier, there is still room to trade speed for complexity. For instance, [BG89] describes a multiplier which is slightly more complex but which handles two bits at a time, and hence is approximately twice as fast.

Squarer The squarer we use in this design has almost the same complexity as our multiplier. A number of options are explored in [DPV99]. Since the *c*-coefficients are shifted over two positions each cycle, they are constructed in two rows: one for the pair and one for the unpair coefficients. The result is found in $[n/2]$ clock cycles. This is the case because during the first $[n/2]$ clock cycles, the data on the feedback path remains zero, and hence the $[n/2]$ most significant bits of a can be preloaded into the even coefficients of c in one clock cycle. Again, our implementation is slightly more complex to allow for a programmable field polynomial.

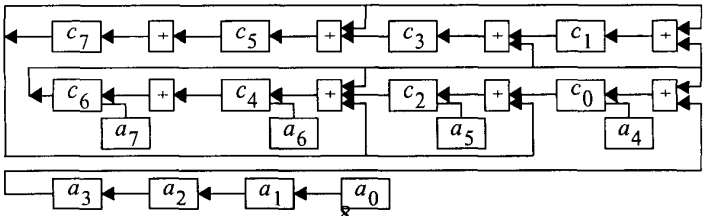

Figure 5 : Serial squarer for GF(2^8) based on the irreducible field polynomial $x^8 + x^4 + x^3 + x + 1$

Bit Slices The four busses in Fig. 3 all have the width of a field element. This number is programmable and is typically of the order of 200 bits. Busses of this width going over a long distance cause high line delays, routing problems and more. Therefore the Data-path is structured in a bit-sliced way. A number of different types of bit slices are needed to obtain a correct implementation of the Data-path [BG00]. In our case, the structure of the squarer imposes a distinction between the even and the odd bits and between the lower half and the upper half of the bits. It turns out that the complete Data-path can be constructed from 6 different bit slices, since there is also one for the least significant bit and the second least significant bit. A major advantage of using bit slices is that it also becomes easier to make the fieldsize n programmable.

4 IMPLEMENTATION RESULTS

To realise the design we followed to a large extent the design flow which is proposed by the Atmel System Designer software [Atmel]. It includes all the tools, data bases and flows for making a hardware/software co-design in an integrated way.

4.1 Software Design

The software part of the Design is developed in C and compiled with the IAR compiler. On average, *125n* instructions for the AVR are needed to read in data, perform the point multiplication and writing out the results. Since the embedded AVR core achieves throughputs approaching 1 MIPS per MHz, this corresponds with *125n* clock cycles.

4.2 Hardware Design

The critical path after synthesis is *33 ns* and is located in the Hardware Controller and not in the Data-path. The corresponding maximum clock frequency of nearly *30 MHz* is a fairly good result, keeping in mind that no attempt has been made to reduce the length of the critical path. Since timing reports show a critical path delay of *5 ns* for the Data-path only, a clock frequency of *200 MHz* can be reached by introducing pipelining and/or other optimizations in the Hardware Controller. The post-synthesis simulation gave the right results and showed no timing violations.

After place-and-route it turned out that *23 Combinatorial Logic Blocks* (CLBs) are needed per bit slice and *496 CLBs* for the hardware controller. Since there are only 2304 CLBs available on the FPGA, a design for a *keylength of 72 bits* can be implemented. These results are comparable with [Ros98], taking into account that the structure of one CLB of the Xilinx FPGA, used in [Ros98], is different from the Atmel FPGA. The Xilinx FPGA is using 3 look-up tables and 2 registers per CLB in comparison with 2 look-up tables and 1 register for the Atmel CLB.

4.3 Hardware/Software Co-design

The results above are used estimate the total time it takes to perform a complete point multiplication. Since the Software Controller is always waiting on an interrupt from the Hardware Controller before sending a new instruction, the Hardware Controller receives this new instruction almost directly after it has finished performing the former instruction. Therefore we can take only the number of clock cycles needed by the hardware part, which is in the average case $12n^2$ [JT01]. Table 1 presents the number of clock cycles and the corresponding throughput for different key lengths. Besides the realized throughputs, also the possible throughputs after optimization of the Hardware Controller are given. The mapping efficiency is represented with the number of CLBs used.

Value of n	CLB Usage	Clock Cycles	Point mult/sec @ 10 MHz	Point mult/sec @ 200 MHz
8	668	768	13020	260416
16	852	3072	3255	65104
72	2189	62208	160	3215
192	4907	442368	22	452

Tabel 1 : FPSLIC chip area utilization and throughput

5 CONCLUSIONS AND FURTHER WORK

The main merit of our work is a working demonstrator which computes a point multiplication on an FPSLIC. An interface is realized with a Visual Basic program on a PC. We also explored the field of the hardware/software co-design and conclude that if the Hardware Controller is optimized, clock frequencies up to 200 MHz must be possible.

Also, it might be useful to look for descriptions of the group operation that result in more efficient hardware. One option is to allow the arithmetic blocks to operate in parallel. This might be particularly useful for the multiplier, since there is a rather high degree of independence between the consecutive field multiplications in a group operation.

ACKNOWLEDGEMENTS

We would like to thank the Atmel support FPSLIC-team, especially Hing Kai Lo and Itsu Wang. This work was partially sponsored by Atmel, Panasonic and UC Micro #00-097. This work is dedicated to Erik De Win († April 2001), who originally started this research.

REFERENCES

[AMV93] G. Agnew, R. Mullin, and S. Vanstone. An implementation of elliptic curve cryptosystems over F_2^{155}. *IEEE Journal on Selected Areas in Com,* 11(5):804-813, 1993.

[Atmel] www.atmel.com/atmel/products/prod39.htm

[BG89] T. Beth and D. Gollman. Algorithm engineering for public-key algorithms. *IEEE Journal on Selected Areas in Com.,* 7(4):458-466, 1989.

[BG00] W. Borremans and P. Gijsels. *A hardware implementation of elliptic curve public-key cryptosystems.* Master's thesis, K.U.Leuven, 2000.

[BSS99] I. Blake, G. Seroussi, and N. Smart. Elliptic Curves in Cryptography. In London Mathematical Society, Lecture Note Series 265 Cambridge, 1995

[DBV96] E. De Win, et al., A fast software implementation for arithmetic operations in $GF(2^n)$. In K. Kim and T. Matsumoto, eds., *Advances in Cryptology, Proc. of Asiacrypt'96,* LNCS 1163, pg 65-76. Springer-Verlag, 1996.

[DMPW98] E. De Win, S. Mister, B. Preneel, and M. Wiener, On the performance of signature schemes based on elliptic curves. In J. P. Buhler, ed., *Algorithmic Number Theory Symposium III,* LNCS 1423, pg 252-266. Springer-Verlag, 1998.

[DPV99] Erik De Win, Bart Preneel, and Ingrid Verbauwhede. A fast serial squarer for $GF(2^n)$. draft paper, 1999.

[DV98] E. De Win, I. Verbauwhede, Technical Report: A Hardware Implementation of Elliptic Curve Public Key Cryptosystems, internal report, EE Dept. UCLA, 1998.

[Gei93] W. Geiselmann. *Algebraische Algorithmenentwicklung am Beispiel der Arithmetic in endlichen Korpern.* PhD thesis, University of Karlsruhe, 1993.

[GP97] J. Guajardjo and C. Paar, Efficient algorithms for elliptic curve cryptosystems. In W. Fumy, ed., *Advances in Cryptology, Proceedings of Eurocrypt'97,* LNCS 1233, pages 342-356. Springer-Verlag, 1997.

[JT01] Sven Janssens and Johan Thomas. *Hardware/software co-design of elliptic curve cryptography.* Master's thesis, K.U.Leuven, 2001.

[IEEE99] IEEE P1363/D13: Standard specifications for public key cryptography. working draft, November 1999.

[KT92] K. Koyama and Y. Tsuruka. Speeding up elliptic cryptosystems by using a signed binary window method. In E. Brickell, editor, *Advances in Cryptology, Proceedings of Crypto'92,* LNCS 740, pg 345-357, Springer-Verlag, 1992.

[LC83] Shu Lin and Daniel J. Costello, Jr. *Error control Coding : Fundamentals and applications,* Prentice-Hall, 1983.

[M93] A. Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, pages 1-5. 1993.

[Ros98] Martin Rosner. *Elliptic curve cryptosystems on reconfigurable hardware.* Master's thesis, Worcester Polytechnic Institute, 1998.