RECONFIGURABLE ARCHITECTURES FOR CURVE-BASED CRYPTOGRAPHY ON EMBEDDED MICRO-CONTROLLERS

Lejla Batina[1], *Alireza Hodjat*[2], *David Hwang*[2], *Kazuo Sakiyama*[1] *and Ingrid Verbauwhede*[1,2]

[1]ESAT/SCD-COSIC,
Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, Leuven, Belgium
[2]El. Engineering Dept.
University of California
Los Angeles, CA 90095

ABSTRACT

This paper discusses architectures for embedded security to enable various cryptographic services at low cost. To realize the large bit-lengths and complex arithmetic on an 8-bit embedded micro-controller, several hardware acceleration options for Elliptic and Hyperelliptic Curve Cryptography (ECC and HECC) are studied and systematically evaluated. Two key factors influence the performance: one is the communication interface *i.e.* I/O transfers between processor and co-processor and the other one is the boundary between hardware and software. Our experiments are run on an 8051 and an AVR micro-controller with the crypto co-processors implemented on a FPGA.

1. INTRODUCTION

Public-key cryptosystems are present in almost all spheres of digital communication *e.g.* for financial, governmental and medical applications. They allow secure communications over insecure channels without prior exchange of a secret key and they also enable digital signatures. One of the most prominent examples is Elliptic Curve Cryptography (ECC), which was proposed in the mid 1980s by Miller [9] and Koblitz [7]. In 1988 Koblitz suggested to use the generalization of Elliptic Curves (EC) for cryptography, the so-called Hyperelliptic Curves (HEC). One advantage of HECC resides on the fact that they alow one to work in a smaller field. More precisely, while typical bit-lengths for ECC are at least 160 bits (providing the security of 1024-bit RSA), for HECC of the same level security we can use finite fields of 80 bits. This fact makes HECC a very good choice for platforms with limited resources.

Lejla Batina and Kazuo Sakiyama are funded by FWO projects (G.0450.04, G.0141.03) and FP6 project SESOC. This research has been also partially supported by the EU IST FP6 projects SCARD and ECRYPT, by IBBT and NSF.

At this stage we briefly introduce the design environment GEZEL [5] in which we model the co-designed system on an example of an 8051 micro-controller. In this application, we used the Dalton 8051 ISS to perform cycle-accurate simulations for our software-only implementation. For the hardware/software system, we designed our co-processor multiplier using GEZEL's hardware description language. The language syntax is primarily used to describe the FSMD (finite state machine plus datapath) system model. Thus, a datapath for the co-processor was designed and its corresponding control logic was also designed in the GEZEL language. After the design of the hardware co-processor, we attached the co-processor to the input/output ports using the GEZEL design environment, and then performed timing and functional verification. GEZEL gives us the ability to co-simulate in a cycle-exact manner. Upon verification of the functionality of the multiplier co-processor, the GEZEL code is automatically converted to VHDL and input into Synplicity for FPGA synthesis.

In this paper several hardware acceleration options are studied and systematically evaluated in order to deploy the large bit-lengths and complex arithmetic on an 8-bit embedded micro-controller. We notice that two key factors influence the performance: one is the communication interface between processor and co-processor, the other is the boundary between hardware and software. However, these two factors are interconnected *i.e.* moving the hardware/software boundary in a suitable way can overcome the drawback of a huge number of data transfers.

The remainder of this paper is organized as follows. In Sect. 2 some background information on curve-based cryptosystems is given including the algorithms required. Details of our work and case-studies for various micro-processors are given in Sect. 3. Results are discussed in Sect. 4.

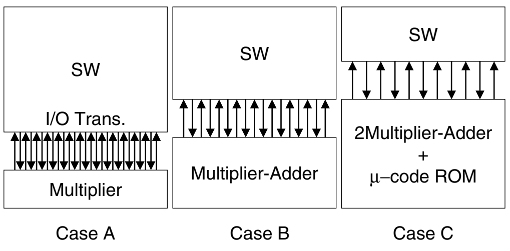

Fig. 1. The hardware/software partitioning in the cases A, B and C.

2. BACKGROUND ON ECC/HECC

Most of the public-key algorithms require the structure of an algebraic group. In the case of Elliptic Curve Cryptography, the group of points on an elliptic curve is used. The main operation in any curve-based primitive is the scalar multiplication. The hierarchical structure for operations required for implementations of curve-based cryptography is as follows. Point/Divisor multiplication is at the top level. At the next (lower) level are the point/divisor group operations *i.e.* addition and doubling. The lowest level consists of finite field operations such as addition, subtraction, multiplication and inversion required to perform the group operations. The only difference between ECC and HECC is in the middle level that in this case consists of different sequences of operations. Those for HECC are a bit more complex when compared with the ECC point operation, but they use shorter operands.

We now give a short mathematical background for hyperelliptic curves and we refer to the algorithms for efficient arithmetic in the group of points on elliptic curve (for ECC) and in the Jacobian group (for HECC) [3].

A hyperelliptic curve C of genus $g = 2$ over $\mathrm{GF}(2^n)$, which is given with an equation of the form: $C : y^2 + h(x)y = f(x)$ in $\mathrm{GF}(2^n)[x, y]$, where $h(x) \in \mathrm{GF}(2^n)$ is polynomial of degree at most g $(deg(h) \leq g)$ and $f(x)$ is a monic polynomial of degree $2g{+}1$ $(deg(f) = 2g{+}1)$. Some more conditions should be satisfied. For the genus 2, in the general case the following equation is used $y^2 + (h_2x^2 + h_1x + h_0)y = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$. For our implementation we used the so-called type II curves [4], which are defined with $h_2 = 0, h_1 \neq 1$. An elliptic curve is a hyperelliptic curve of genus $g = 1$.

We do not give any more details but we stress again that the crucial operation is finite field multiplication that has to be performed thousands of times for one scalar multiplication. Field addition is easy, as we work with binary fields and inversions can be avoided by use of projective coordinates. Actually those can be traded for more multiplications. Therefore, we need to find a way to deal with that many multiplications in order to obtain efficient low-cost implementations of curve-based cryptography. The point/divisor scalar multiplication is obtained by use of repeated divisor addition and doubling. We used the NAF algorithm [3] and we used the formulae from [4] for doubling and we applied the same approach to get formulae for addition.

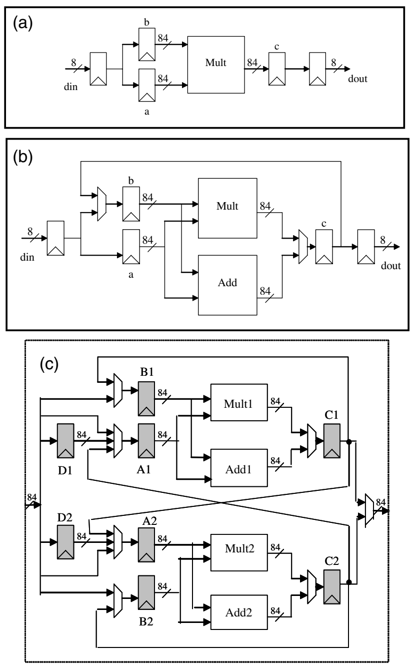

Fig. 2. (a) Data path for the initial design (case A). (b) Data path for the improved design (case B). (c) The co-processor's datapath (case C).

3. IMPLEMENTATION OPTIONS

In the following, three different HW/SW partitioning and the architecture of implementations are presented. We elaborate here on various architectural options. These options differ with respect to the change of the boundary between hardware and software and are shown in Figure 2. The general idea is given in Figure 1.

Case A: The first choice is to implement only the Galois field multiplier as a hardware accelerator. The Galois field multiplication is used frequently in the point/divisor's addition and doubling operations. Therefore, as Figure 2 shows, a hardware datapath includes a Galois field multiplier and it uses 8-bit data input and output interfaces attached to the micro-controllers.

Case B: The next level of HW/SW partitioning is to define an accelerator datapath that can perform all the Galois field operations in hardware. The key observation is that in the schedule of divisor's double and add operations for HECC there are many expressions of the following form: $d = a \cdot b + c$. For this purpose a hardware adder and a feedback line that can keep the result of the multiplication

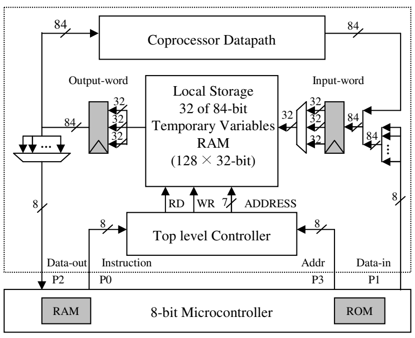

Fig. 3. The block diagram of the hardware architecture.

in hardware was added to the original data path and therefore, the number of I/O transfers decreased with not much of extra hardware. These two options were described in more detail in [2] for the case of an 8051 processor.

Case C: In this case the point/divisor operations are implemented in hardware where these operations are designed using the micro-code instructions. Each line of the divisor addition and doubling operations algorithms can be represented by one of these micro-code instructions. The scalar multiplication algorithm is implemented using the C code on the CPU.

Figure 3 shows the block diagram of the hardware architecture which is used in all three cases mentioned above. The coprocessor includes a datapath, local storage and the top controller. There are four 8-bit ports that are used for communication between the micro-controller and the coprocessor. Two of them are for the input and output data and the other two are for coprocessor's instructions and the address to access the local storage. Every data transfer to the local storage (RAM) is through the input-word and the output-word registers that are 84 bits wide (the word length of the operation in the coprocessor's datapath).

The software is divided into two main sections. One is the routines of micro-code instructions that are used to program the coprocessor. These routines implement the divisor's operations (doubling, addition, and subtraction). Also the routine that converts the final projective result to the affine coordinate is implemented using the micro-code instructions based on the Fermat's inversion algorithm. The second section of the software is the C code which is finally compiled to micro-controller's assembly code. The scalar multiplication algorithm is implemented in software and calls the routines of divisor's operations described above.

The local storage unit consists of 128 memory locations of 32-bit width. The 32-bit word-length is chosen due to the fact that the maximum word-length of each Block RAM of the FPGA is 32 bits wide. One temporary variable of the $GF(2^{83})$ field require four memory 32-but memory locations. Therefore, there are total of 32 temporary variables that can store the elements of $GF(2^{83})$. The memory address bus is 7 bits wide to cover the 128 locations (32 variables) and the coprocessor controller asserts the required values for the memory read (RD) and write (WR) signal. Also notice that the input into and out of the local RAM has to go through the input-word and output-word registers.

Table 1. Results for HECC on the 8051 and AVR at 12 *MHz*.

Impl.	Perf. [*M Cycles*]	Perf. [*s*]
SW only: 8051	1,800	149.8
SW only: AVR	99.8	8.31
8051 - Case A	61.8	5.15
AVR - Case A	2.61	0.218
8051 - Case B	29.8	2.48
AVR - Case B	2.14	0.78
8051 - Case C	7.87	0.656
AVR - Case C	0.938	0.078

4. RESULTS

In this section we give results of the case studies mentioned above and we discuss these further. The 8-bit micro-controllers that we used were 8051 and AVR. The hardware accelerators are attached to the I/O ports of the 8-bit micro-controllers. These I/O ports are available as "memory-mapped" interfaces to the hardware coprocessor. Using the memory-mapped software instructions of the micro-controller, the required instruction opcode can be assigned to each I/O port and then used to program the hardware accelerator. This communication is a one way handshaking process. Upon receiving the instruction opcode, the coprocessor performs the task and waits for the next instruction to come. Input and output data values are also transferred from software to the hardware in the same fashion.

Pure software figures are included to prove the option not feasible. Namely, we concluded that although our software implementations could possibly be further optimized, it would still be difficult to achieve an efficient ECC/HECC implementation.

Table 1 gives the results for HECC for the cases of 8051 and AVR for all 3 architectural options. The known fact that AVR is much faster than 8051 can be observed. In Table 2 our results for ECC and HECC on an architecture that includes 8051 micro-controller are compared. The results are in favor of HECC and this platform is the first one showing that HECC are preferred to ECC in some architectural solutions. The FPGA platform used is Virtex2.

Table 3 compares the delay of scalar multiplication of a point/divisor for various ECC and HECC implementation options using the 8051 platform. FPGA area is given in number of LUTs without XRAM and ROM which are specified separately. As it is seen in this table, there is a significant increase in performance when the field multiplication is moved into hardware. An additional timing reduction occurs after the point operation signal flow graphs are analyzed and manipulated, and the new "multiply-and-add" operation is used. This reduction is a bit bigger for the case of HECC

Table 2. ECC/HECC comparison using the 8051.

Impl.	FPGA [#LUTs]	RAM [Byte]	ROM [Byte]	Perf. [*s*]
ECC: SW	3,300	980	7,597	144.5
HECC: SW	3,300	1,186	13,926	149.8
ECC: HW/SW (Case A)	3,868	980	7,597	5.52
ECC: HW/SW (Case B)	4,210	910	8,739	3.97
HECC: HW/SW (Case A)	3,600	927	12,789	5.15
HECC: HW/SW (Case B)	3,781	936	11,524	2.49

Table 3. Comparison with other related work.

Ref.	Field	Platf.	*f* [MHz]	t [*ms*]
[10]	GF(2^{83})	ARM7	80	71.56
[1]	GF(2^{80})	ARM7	80	374
[8]	GF(2^{163})	AVR	4	113
[6]	GF(2^{160})	8051	12	4580
our	GF(2^{83})	8051	12	2488
our	GF(2^{163})	8051	12	3970

(40%) because the particular formulae for the divisor operations allowed taking more benefit of this option than for the case of 163-bit ECC (28%). Comparing ECC and HECC figures, we observe that HECC not only provides better performance, but also deploys a smaller hardware module. This result presents a unique observation among all previously published work. Hence, HECC offers some benefits especially on embedded platforms mainly due to shorter operands' bitlengths. Also that fact results in less extra hardware because the multiplier for HECC is half the size of the one for ECC. The only figure that still favors ECC is the amount of ROM, which is due to more complex divisor operations.

Additionally, we have calculated figures for the co-processor's usage for the 8051-case and the actual time distribution that is spent on I/O accesses (Fig. 4). The co-processor usage is defined as the number of clock cycles for which datapath is activated divided with the total number of clock cycles. The co-processor's usages in total number of cycles are 0.979%, 2.031% and 4.154% for 8051 and 23.189%, 28.282% and 34.855% for AVR for cases A, B and C respectively. For the 8051 case this means that the performance is actually achieved with a very low hardware utilization. Namely, by adding some more hardware one could achieve a speed-up in performance and still maintain a low-cost and low-power solution. The conclusion is that allocating more hardware resource in the co-processor, we can reduce the number of I/O transfers and therefore we obtain higher performance. Due to the fact that the 8051 uses the clock division 1:12, the co-processor usage in this case is much worse than the one of AVR.

5. CONCLUSIONS

This paper discusses various architectural options for embedded security to enable public-key cryptographic services

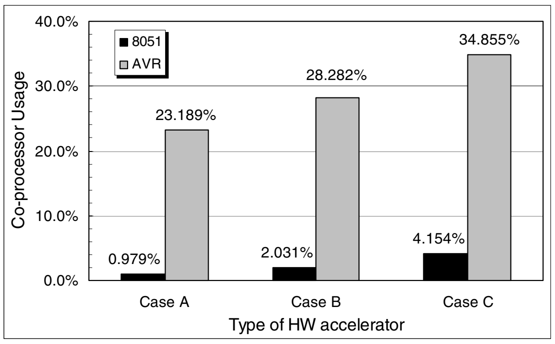

Fig. 4. The co-processor usage for HECC implementations.

at low cost. For this purpose several hardware acceleration options are studied and systematically evaluated. The results show that two crucial factors influence the performance: one is the communication interface between processor and co-processor, the other is the boundary between hardware and software.

6. REFERENCES

[1] S. Baktır, J. Pelzl, T. Wollinger, B. Sunar, and C. Paar. Optimal Tower Fields for Hyperelliptic Curve Cryptosystems. In *Proceedings of 38th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, USA, November 7-10 2004.

[2] L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede. Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 μP. In J. R. Rao and B. Sunar, editors, *Proceedings of 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 3659 in Lecture Notes in Computer Science, pages 106–118. Springer-Verlag, 2005.

[3] I. Blake, G. Seroussi, and N.P. Smart. *Elliptic Curves in Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

[4] B. Byramjee and S. Duquesne. Classification of genus 2 curves over F_{2^n} and optimization of their arithmetic. Cryptology ePrint Archive: Report 2004/107.

[5] GEZEL. `http://rijndael.ece.vt.edu/gezel2/`.

[6] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In M. Joye and J.-J. Quisquater, editors, *Proceedings of 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 3156 in Lecture Notes in Computer Science, pages 119–132, 2004.

[7] N. Koblitz. Elliptic curve cryptosystem. *Math. Comp.*, 48:203–209, 1987.

[8] S. Kumar and C. Paar. Reconfigurable instruction set extension for enabling ECC on an 8-bit processor. In *Proceedings of International Conference on Field-Programmable Logic and Applications (FPL) 2004*, Antwerp, Belgium, August 30-September 1, 2004.

[9] V. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology: Proceedings of CRYPTO'85*, number 218 in Lecture Notes in Computer Science, pages 417–426. Springer-Verlag, 1985.

[10] J. Pelzl, T. Wollinger, and C. Paar. *Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation*, chapter in Embedded Cryptographic Hardware: Design and Security. Nova Science Publishers, 2004.